Building an Open Source Platform for Forensic Medical Documentation of Human Rights Violations

by

Felipe Monsalve

B.S., Computer Science and Engineering and Mathematical Economics
Massachusetts Institute of Technology, 2020

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2022

Author . . . .
Department of Electrical Engineering and Computer Science
August 12, 2022

Certified by . . . .
Boris Katz
Principal Research Scientist
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Building an Open Source Platform for Forensic Medical Documentation of Human Rights Violations

by

Felipe Monsalve

Submitted to the Department of Electrical Engineering and Computer Science
on August 12, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

MediCapt is a virtual platform for secure documentation of medical records, currently deployed by Physicians for Human Rights (PHR) in Kenya and the Democratic Republic of the Congo to forensically and clinically document sexual violence. Its uses in documenting of sexual violence have allowed for the collection of court-admissible evidence for use in the prosecution of perpetrators, even in areas with limited access to healthcare and low levels of literacy, where perpetrators might otherwise go free due to a lack of registered physical evidence. We have rewritten an early version of MediCapt with the guidance of PHR, with the aim of building the first widely deployable, open-source, in-the-field health data collection platform that focuses on forensic and clinical documentation of sexual violence, and other human rights violations, in remote areas. To this end, we developed a new automated server-side infrastructure and frontend for MediCapt that scales automatically to any foreseeable level of demand, is compliant with the latest security and privacy regulations and best practices, and will require little to no maintenance in the coming years. Locally, we designed a secure caching system for offline functionality that is easy to integrate, modular and secure, which ensures proper functioning of the platform in low connectivity environments where MediCapt will be deployed.

Thesis Supervisor: Boris Katz
Title: Principal Research Scientist

Acknowledgments

I would like to thank my advisor, Dr. Andrei Barbu, for his dedication to this project and for his guidance throughout my graduate studies. Andrei, it was a pleasure learning from you and helping you build what I'm sure will be a deeply impactful project.

I would also like to thank Physicians for Human Rights for their work and guidance in building this system, and for taking the technical work we did and testing and deploying it where it's most useful.

Finally, I would like to thank my parents, Pablo and Kyra, for your constant support through this journey. Your company and encouragement, whether far or close, made this possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sexual violence is a major public health issue, with an estimated 35% of women worldwide having experienced intimate partner sexual and/or physical violence, or non-partner sexual violence at some point in their lives [27]. While a heinous human right violation, convictions rates for these crimes are painfully low, which not only denies victims justice for their crimes, but also often discourages reporting of crimes. This in turn results in less data on sexual violence, and therefore a poorer understanding of the issue.

The situation is especially dire in conflict zones, where systematic rape is more common, and simultaneously channels for reporting poorer. In Kenya and Democratic Republic of the Congo (DRC), where Physicians for Human Rights–a US-based human rights NGO–runs the Program on Sexual Violence in Conflict Zones, sexual violence is rampant [24]. In response, Physicians for Human Rights (PHR) created MediCapt, a platform for in-the-field forensic medical documentation of sexual violence for deployment in Kenya and DRC. Over the past few years, however, scaling and maintaining the aging platform with limited development resources, and given evolving needs, has become a burden for PHR. Therefore, we set out to rewrite MediCapt from the ground up, with the aim of developing an open-source system that would be flexible, adaptable to different contexts, and incredibly future proof, requiring minimal development and maintenance over the next decade.

We have developed MediCapt 3.0, the first open-source, distributed, in-the-field health collection platform built to forensically document human rights violations, which is scalable, ultra-maintainable, and compliant with the latest privacy and security regulations and best practices. We first redefined the key components and requirements for the MediCapt design, laying out the requirements for a platform for medical documentation that is geared towards reporting human rights violations, based on previous work done by PHR on the field, and on new stakeholder feedback. We also identified key concepts and defined different user types by segmenting key stakeholders and their responsibilities in the platform. Based on this preliminary work done along with PHR, we designed a system architecture to fullfil all of the functionality requirements while being secure, scalable, and ultra-maintainable. In the client-side, we developed a caching system for offline use to be deployed in low connectivity environments, that is simple to integrate, modular, and secure, in keeping with our mission for the system.

The remainder of this thesis is organized as follows. In Chapter 2, we will do a review of previous work related to our project. We first review the use of mobile technology on the fields of healthcare, human rights work, and data collection; then we review the evolution of MediCapt under PHR, and the work they have done to strengthen prosecution of human rights violations in conflict zones. Chapter 3 then discusses the design of MediCapt, from defining key components and required functionality of the system, to laying out the server and client-side architecture, and justifying the key design decisions in the context of the goals for the system. In Chapter 4, we introduce the caching system used for offline functionality in certain parts of the platform. We explain the motivations for designing it, the key requirements for it, and its general system architecture. We summarize our contributions and lay out the future work in Chapter 5.

Chapter 2

Previous Work

The review of previous work related to the development of an open source platform for forensic medical documentation will focus on two main areas. In Section 2.1, we will discuss previous tools or projects developed with similar purposes to MediCapt. We will discuss previous work in the use of mobile devices in the areas of medicine and human rights work, before focusing on mobile data collection tools that have been used in these fields. Section 2.2 will then briefly summarize PHR's motivation in building MediCapt, as well as discuss work on earlier versions of the app, providing some context onto how the vision for MediCapt evolved from a focused, single use-case tool into an general open-source platform for forensic documentation of human rights violations.

2.1 Existing Technology

The use of information technology for data collection in the medical or human rights field is not new. For decades, the use of electronic health records has been proposed as an alternative to paper health records [18, 34, 6], with proponents of e-health (the use of information technology for medical purposes) arguing that these electronic records present a whole host of benefits (from decreasing errors from handwriting to easing physical storage and sharing of records [29]). Since then, electronic data collection has drastically changed, with the spread of mobile devices and their high levels of penetration in the developing world offering interesting opportunities in the healthcare and human rights fields. The medical community, which had wide adoption of eHealth tools in the developrd world, has seen the birth and subsequent growth of mHealth (the use of mobile devices as a tool for medical purposes), specifically in the developing world. Similarly, as the medical community has embraced mobile phones in their mission to expand healthcare coverage, so has the human rights field done so in the search of justice.

In this section, we explore the user of mobile technology in the healthcare and human rights fields through past or ongoing projects, before surveying the state of mobile technology for structured data collection in these fields through the tools available. Section 2.1.1 will explore the promise, benefits and challenges with the push towards use of mobile devices in the healthcare sector (mHealth), specifically in developing economies. Section 2.1.2 will then discuss the use of mobile devices in the human rights field, through a few examples of tools currently deployed on the field. Section 2.1.3 will finalize by considering the state of mobile technology for structured data collection, by comparing existing tools widely used for data collection in the medical and human rights fields.

2.1.1 Mobile Health in the Developing World

mHealth, or mobile health, is a branch of eHealth which focuses on the use of mobile devices in the field of healthcare. While it dates back over 30 years, unprecedented mobile technology penetration in the developing world has seen it surface as an interesting solution to bring healthcare to areas of the world that have traditionally been underserved. Mobile devices' low price-point, efficient power consumption, small size, portability and and generally modest infrastructure needed to support them [33] make them well positioned to tackle one of the hardest problems in global healthcare: structural barriers to access [52]. The spread of mobile technology in developing countries–by 2016 there were over 40 mobile users per 100 people in the developing world (with the figure sharply rising year over year as seen in Figure 2-1) [36], and in 2014 it was estimated that over 80% of rural areas in the developing world had ac-

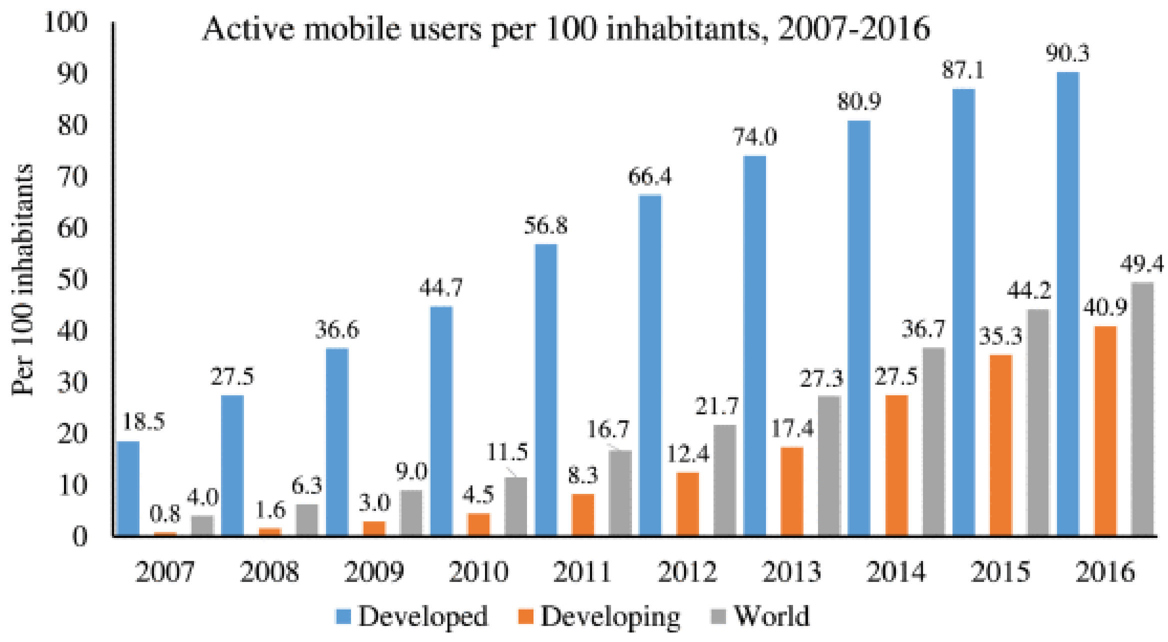


Figure 2-1: Mobile penetration in developed and developing world between 2007 and 2016, measured by the International Telecommunication Union [36]

cess to mobile networks [2]–has therefore resulted in many mHealth projects tackling issues in healthcare over the last decade or two.

By 2009, a study by the United Nations Foundation and Vodafone Foundation already listed over 50 mHealth pilot programs in 26 different countries. These programs were divided into six main areas, depending on their approach to improving healthcare: education and awareness, remote data collection, remote monitoring, resources for health workers, disease outbreak tracking, and diagnostic and treatment support (described more in detail in Table 2.1) [13]. That year, it was already estimated that some of mHealth applications had reached over 10 million people in the developing world [32]. Given the dramatic increase in mobile penetration in developing countries since then, we reasonably expect the number of people benefiting from mHealth solutions to be much larger now.

Despite their promise, however, more recent studies into the state of mHealth programs have shown that there is still a long way to go before mHealth reaches its full potential. Different surveys of the mHealth landscape have found that there are few evaluation frameworks for measuring results of mobile health technology [11], and

mHealth Area	**Description**	**Examples**
Education & Awareness	Send important health information to isolated populations, through SMS campaigns or voice-based hotlines.	Text to Change, Uganda; HIV Confidant, South Africa
Remote Monitoring	Connect medical providers to patients facilitating monitoring for conditions, medication and appointments.	MedicMobile; Colecta-PALM, Peru
Resources for Health Workers	Trains health professionals and connects them with sources of information that help them do their job effectively.	HealthLine, Pakistan; UHIN, Uganda
Disease Outbreak Tracking	Quickly capture and transmit data on disease incidence to contain outbreaks.	FrontlineSMS; GATHER, Uganda
Diagnostic & Treatment Support	Provide diagnosis and treatment advice to health workers through access to medical databases or medical staff.	M-DOK, Philippines; Mobile Telemedicine System, Indonesia
Remote Data Collection	Facilitate accurate data gathering and analysis among isolated communities and in challenging environments.	Magpi; Epicollect5; Nokia Data Gathering, Brazil

Table 2.1: Sample of mHealth programs along with country where they were deployed (unless they have been deployed more broadly) grouped by their different approach to improving healthcare, as categorized by a 2009 report from the United Nations Foundation and Vodafone Foundation Partnership [13].

there are legitimate questions about the quality of many of the mHealth services that have been deployed [1]. In addition to these questions, surveys of mHealth literature have found that while a lot of different pilots have been ongoing, the majority of them only utilize basic phone functions such as text messaging, while very few use powerful native SDKs that would give them access to cameras, sensors, or phone metadata useful for health tools [20].

The main challenge to overcome for mHealth's future, however, is likely the difficulty successful pilots have seen to reach scale (an issue often referred to as Pilotitis) [5]. Between 2008 and 2009, for example, a study on mHealth in Uganda found that 23 of the 36 mHealth initiatives tracked failed to advance from the pilot stage [52]. This issue, which plagues many of the mHealth pilots, has many causes, among which lack of funds or engineering capacity, outdated tech-stacks, lack of key stakeholder buy-in, and inconsistent evidence of success at the pilot stage are key [51]. While frameworks for evaluating and scaling mHealth technologies have been developed, and are well positioned to tackle pilotitis [11, 25, 42], the issue of scaling remains a difficult one for mHealth projects.

2.1.2 Mobile Technology in Human Rights Work

Like the healthcare sector, human rights communities in developing countries have embraced mobile technology as a tool in bringing about justice and helping them in their job. While scant literature exists on the use of mobile technology among human rights groups, leaders in the field appear to generally agree that there are three different categories of tools being used currently on the field: those that help activists communicate securely, and help provide them protection; those that aid activist in collecting and securely storing information to expose violations; and those that collect, analyse and preserve evidence that could be admissible in court [39, 41].

The first category–that of mobile technology helping activists communicate securely–includes messaging platforms using secure messaging protocols (like the Signal protocol), such as WhatsApp, Secure Chat and Signal [16]. Arguably more interesting (and certainly more targeted to the human rights community) Amnesty International's Panic Button app stands out as an example of an app that provides protection for human rights activists, by connecting them–and sending location information–to community members as a distress signal when triggered, allowing for a quick response by community members and giving them more information to act on [50].

The second category–the one used for aiding activist in collecting and securely storing information meant to expose human rights violations–includes Bentech's Martus, used by activists to securely collect and store human rights data, both on desktop computers as well as Android devices. Martus, which allows the storage of structured data through creation of templates and records on top of these templates, also supports attachments for supporting documents, while encrypting all of this data, and backing it up to severs offsite [8]. It has been used by human rights defenders in over 50 countries, including Uganda and Guatemala, to document over 245,000 cases of human rights violations [7].

The final category–used for to collect, analyze and preserve court-admissible evidence of human rights abuses–includes eyeWitness to Atrocities' eyeWitness system, in addition to the Guardian Project and WITNESS' InformaCam project. Both the eyeWitness system as the InformaCam project help activists in capturing video documentation of human rights abuses that might serve as evidence in court. The eyeWitness system, which is more focused, provides a video recording app for activists–which collects metadata for verification–and deals with uploading and storing the videos and metadata while maintaining a secure chain of custody, so they can be passed along to the bodies best suited to act on the data. It works with over 40 partners globally and has processed over 27,000 photos or videos [17, 19]. The more general InformaCam project, on the other hand, focuses on developing a core services library for building "verified mobile media" (media that is augmented by metadata from the device it was recorded in and maintains a secure chain of custody over the media and metadata) [46, 54, 55]. Based on the InformaCam platform, the Guardian Project developed Android app CameraV for capturing media that is augmented by metadata, and encrypted on the device [44, 45].

Unfortunately, like mHealth apps, mobile technology in human rights work has struggled to reach scale, because of similar factors as mHealth. Among other things, outdated tech stacks, along with lack of funding or development capacity to modernize them, has proved a difficult challenge to overcome for human rights mobile technology. From the examples discussed above, Amnesty International's Panic Button and Bentech's Martus are no longer actively supported, after struggling to find funding to continue with development efforts [9, 51]. While no official communication from the Guardian Project or WITNESS could be found on the state of the InformaCam project, no development activity since 2015 on InformaCore or 2017 on CameraV on GitHub would suggest InformaCam is also not being actively supported.

2.1.3 Mobile Data Collection for Social Ventures

In 1989, a paper titled "Revolutionising health data capture: use of hand-held computers" by K.C. Lun et al. appears to have been the first to propose health data collection on mobile devices (which back then consisted of pocket organizer computers) [35]. Since then, mobile data collection for social ventures has come a long way. In this space (and in particular in the healthcare and human rights fields) the spread of mobile technology has made a large impact in replacing error prone and hard-to-collect paper data, in favor of mobile applications that facilitate data collection and aggregation. In particular, the last few decades have seen a crop of new open-source or open access [53] platforms for structured data collection from mobile devices. While the field of mobile data collection is very large, we will center our discussion on those platforms which are commonly used by social ventures: focused, easy to use, open platforms built for resource-constrained environments [40], as they represent the main potential alternatives to MediCapt.

Given the above constraints, we will be discussing the state of mobile data collection for social ventures through five popular platforms in the healthcare and human rights fields: Epicollect5, developed by the Centre for Genomic Pathogen Surveillance from Oxford's Big Data Institute [21]; Open Data Kit (ODK), developed by Nafundi and the ODK community [4, 30]; KoBoToolbox, developed by the Harvard Humanitarian Initiative, the Brigham and Women's Hospital, and the Kobo community [31]; SurveyCTO, developed by Dobility, Inc. [15]; and Magpi, developed by the company by the same name: Magpi (previously DataDyne) [37]. The previously discussed Martus [8], also a good example of a mobile data collection for mobile ventures, is no longer actively supported so we will leave it out of our current discussion.

Tools such as FrontlineSMS [26] or EyeWitness [17] (also discussed previously), which have on occasion been used for mobile data collection, are also excluded since their main function–notably SMS messaging or media file (as opposed to structured data) collection, respectively–is different from mobile collection of structured data and additional effort is often required to have them accomplish this task [14]. Furthermore, we will also leave out of our discussion broader electronic medical record systems and patient monitoring platforms, such as OpenMRS [56] and MedicMobile [12], since structured data collection, though supported, is a small subset of the functionality, which makes these systems more complex and less directly comparable to MediCapt.

To understand the state of mobile data collection for social ventures, then, we will compare approaches by the platforms specified to access and customizability of the platform, deployment of the platform, and collecting structured data.

Access and Customizability

While all of the data collection platforms relevant for social ventures are, to some extent, free to use (likely given the resource-constrained nature of social work), their approaches to access and customizability are quite different. While ODK and KoBoToolbox are open-source, allowing anyone to contribute and modify the code to meet specific needs, the other platforms we're discussing are not. In fact, while Epicollect5 is free to use without limits, SurveyCTO and Magpi both have free tiers, but charge at higher levels of usage, or for more advanced features.

Deployment

Approaches to deployment among mobile data collection platform vary somewhat, though all platforms try to make it as easy as possible. Epicollect5, SurveyCTO and Magpi, being closed-source, are hosted by the respective organization, so usage doesn't involve deploying the platforms. Instead, users create accounts on the hosted platforms, and can begin using them (though SurveyCTO's team seems to be willing to allow self hosting in specific cases for paying customers [49]). In these cases costs of hosting the platform are covered by paying users (except in the case of Epicollect5, where the costs are covered by project backers). ODK and KoBoToolbox, meanwhile, offer self hosting with detailed instructions as is typical in open source projects, though both of them also have hosted offerings. In ODK's case, hosting costs are covered by a subscription for ODK Cloud (their hosted offering), while in KoBoToolbox's case–like with Epicollect5–costs are covered by project backers (though limits exist for those that aren't working for an organization providing humanitarian assistance).

Data Collection

Data collection among the different platforms share a lot of similarities. As they all focus on gathering structured data, all of them offer a form designer, in which some users can design form templates upon which records are based on (i.e. forms with fields to be filled out by people collecting data), to make it easy to encode many different forms and easily collect desired data. Focusing on the social venture sector, all of the platforms offer an Android app for data collection (though only Epicollect5, SurveyCTO and Magpi offer an iOS app), and support offline data collection, which is synced to the server when connection is reestablished. These apps all display forms created and available to a user, so they can be filled out to create a record. Additionally, they all collect some metadata on when the record was created and by whom, while only ODK, KoBoToolbox and SurveyCTO offer encryption from the time the record is store on the device [49].

2.2 MediCapt Evolution

While there are a host of form-based mobile data collection platforms and plenty of mobile technology projects for the healthcare or human rights fields–with significant development efforts behind them–the development of MediCapt responded to the unique requirements of a platform that is meant to document forensic evidence that can be admissible in court, with specific stakeholders in mind. Developed alongside clinicians, lawyers, officials and civilians in DRC and Kenya [39], MediCapt has evolved significantly since its inception. From a simple tool PHR intended to use to increase successful sexual violence prosecution rates in DRC–through the digitization of a single standardized medical intake form–to a general platform for forensic documentation of human rights violations–building features that make it scalable and future-proof while catering to very specific stakeholders–Medicapt (including its paper form predecessor) has undergone a few iterations, quite different from one another.

This section will briefly discuss these iterations, from the development of the Standard Sexual Violence form for DRC under PHR's Program on Sexual Violence in Conflict Zones (in Section 2.2.1), through the digitization of the form (in Section 2.2.2), to the broadening of its scope and change in vision for the app (in Section 2.2.3).

2.2.1 DRC Standard Sexual Violence Form

In 2011, Physicians for Human Rights (PHR) launched the Program on Sexual Violence in Conflict Zones with the goal of strengthening prosecution in sexual violence cases, particularly in conflict zones [24]. Working primarily in Kenya and Democratic Republic of the Congo (DRC), as both countries have suffered from systematic conflict-related sexual violence, PHR identified that low conviction rates were often tied to weak forensic evidence of sexual violence crimes, with incomplete or poorly documented medical investigations [10, 47]. Through multi-sectoral training in Congo, where PHR has convened over 400 health workers, law enforcement, and legal professionals–who traditionally worked in silos–to improve data collection and collaboration in sexual violence cases, PHR developed the DRC Standard Sexual Violence Form, also known as the Medical Certificate [3].

Developed with the input of human rights scholars, health workers, law enforcement, and legal professionals, to make sure that it adhered to international standards while conforming to Congolese rule of law, the Medical Certificate–precursor to MediCapt–improved quality of evidence of sexual violence primarily by standardizing medical record collection [28]. Previous non-standard medical records, which contained long swaths of text unlikely to be admitted in courts, were replaced by a guided evaluation that collected forensic evidence, by asking short questions or giving simple instructions to health professionals, such as checking a box or drawing on a pictogram [41].

2.2.2 Digitization of the Standard Form

While the Standard Sexual Violence Form in DRC improved the quality of forensic evidence collected by standardizing the medical intake form used for evaluations, fundamental challenges remained in collecting high quality evidence and using it in court. Challenges with the lack of photographic evidence to back up claims in the form, with healthcare professionals skipping important parts of the form they felt ill-equipped to fill out, and with securely storing and retrieving records, as well as establishing a trusted chain of custody for them after they were completed seemed like particularly harrowing problems [22]. In response to these challenges, in 2012 PHR decided to begin development on MediCapt, as a single-purpose app to digitize the standard form, but additionally allowing media capture to complement the form, guiding healthcare professionals through the form while validating its completeness, and securely storing and transmitting the completed records to relevant authorities while maintaining a verifiable trusted chain of custody.

By late 2013, after the requirements for the app had been decided, an initial prototype was developed by digitizing the Standard Sexual Violence Form in Magpi (a platform for structured mobile data collection described earlier in Section 2.1.3) and using InformaCam (described in Section 2.1.2) for media capture to complement records [23]. This prototype, MediCapt 1.0, would later be tested with Congolese healthcare providers in early 2014.

2.2.3 Broadening Scope in Response to Feedback

In response to feedback received early 2014 on the MediCapt 1.0 prototype, which included features deemed necessary for future versions (such as photo capture capability inside the app, encryption of data, secure chain of custody, printing of records and writable pictograms in forms), PHR worked with Main Street Computing–a development company–to conduct a needs assessment after which it was determined that no existing technology met the needs determined for MediCapt 2.0 [41]. Therefore, PHR decided to build MediCapt from scratch. Using this opportunity to future-proof MediCapt, and to better justify the development efforts that would go into building MediCapt, the vision for MediCapt as a single-purpose platform tied to the DRC Standard Sexual Violence Form was broadened into a data collection tool for general human rights documentation, as well as other data collection efforts. To this end, a robust form creation and management backbone had to be built into the app, which allowed some users to encode different forms, and made MediCapt into a general structured data collection platform similar to those discussed in Section 2.1.3 (though with domain-specific features that make it mainly geared to sexual violence and other human right abuses reporting).

On top of the broadening scope, MediCapt 2.0 built in those features deemed necessary by stakeholders into the app. Among other things, the new Android app for healthcare providers included photo capture based on InformaCore (open source technology behind InformaCam) and encryption on device, a web portal was built for relevant authorities to access records with verified chain of custody information, and significant development effort was expended into exporting records into PDF's for easy printing.

Chapter 3

MediCapt Design

While development efforts–dating back to 2014–to build MediCapt 2.0 were ambitious in the expanded scope for the platform, significant challenges in maintaining the platform have arisen. Due to aging infrastructure and lack of development efforts over the last few years, eight years in, MediCapt 2.0 has faced considerable trouble scaling. Additionally, given PHR's limited development resources, maintenance has become a significant burden. In response to these issues, we set out to build MediCapt 3.0–rewriting MediCapt from scratch once again–with the objective of building the first secure, scalable and ultra-maintainable open source platform for forensic documentation of human rights violations, which is compliant with the latest privacy and security regulations and best practices, and will require minimal development or maintenance over the next decade.

To achieve this, we identified key concepts to build MediCapt around, and defined the required functionality for a robust platform for forensic documentation of human rigths violations, with the help of PHR and feedback from stakeholders. We were then careful in designing a backend infrastructure that was simple and secure, but limited the amount of custom code that we would need to maintain. Among other things, security and privacy compliance also led us to build a platform where it was exceedingly clear where data could be stored and how it could be accessed. For the MediCapt frontent, which required a significant amount of custom code to maintain, we focused maintainability efforts on reusing as much of the logic across separate client-side side portals, and building all of them on a single tech stack.

This chapter will discuss the required functionality for MediCapt, as well as the key design decisions we landed on in our efforts to make MediCapt 3.0 as future-proof as possible. Section 3.1 discusses some main concepts MediCapt 3.0 was built around, and which provide some context for the rest of the chapter. Section 3.2 then explains the main functionality that MediCapt supports through the various portals for different kinds of users with varying responsibilities in the platform. Section 3.3 and Section 3.4 close the chapter by discussing the key architectural decisions for the server and client side of the platform, respectively.

3.1 Main Concepts

When designing MediCapt 3.0, we identified a few key concepts to build the system around: forms, records, locations, and users. These concepts–most of which have evolved through different versions of MediCapt, but which we redefine here–were central to our design. We provide some context on them, before discussing the expected functionality and architecture of our design.

3.1.1 Forms

While the first prototype of MediCapt (MediCapt v1.0), simply converted the Standard Sexual Violence form–developed by PHR to standardize forensic medical data collection in DRC–into a digital format using Magpi [48], the scope in subsequent versions of MediCapt has widened to accommodate many different forms. With our aim of building a general open-source platform for documenting forensic evidence of human rights violations, it became critical to easily accommodate different forms built for varying legal frameworks, languages, and cultures.

Forms templates in MediCapt are therefore meant to encode standard medical forms (either digitized physical forms, or forms built directly for the MediCapt platform), and are built out of form components built into MediCapt, as well as supported validation/ flow logic for more sophisticated medical forms. Forms creation is built into the platform, so that specific types of users can create forms and make them available to medical providers with no development effort. Form templates are then stored and made available to relevant users through the platform.

3.1.2 Records

As a platform to collect forensic medical data, the secure creation, storage, and sharing of records is the fundamental function of MediCapt. Records in MediCapt are always created and stored when trained medical providers collect medical forensic evidence, based on a form template. Once saved, records serve two main purposes on the platform: to aid in prosecution of perpetrators by serving as court-admissible forensic evidence, or to be de-identified and aggregated for analysis by researchers, who aim to identify patterns in crimes and using available data to prevent mass-crimes.

Because forms contain very sensitive personally identifiable information (PII) and protected health information (PHI), the treatment of records must be delicate. Only relevant medical providers must have access to a record, other authorities must only have access to records explicitly shared with them, and the records researchers have access to must be carefully de-identified so they can't be personally identifiable. While MediCapt is not targetted for deployment in the US, we use HIPAA compliance [38] as a benchmark to ensure we're taking proper care of patient's protected health information, or personally identifiable health information.

Besides privacy, records also need to maintain strict chains of trust, so that they remain court-admissible. To this end, all records are versioned, with modifications to a record not overwriting the record on the server, but creating a new version. More importantly, though, we built into MediCapt the ability to seal records, which prevent further modifications to the record (new versions of the record being created). Instead, all future amendments to a sealed record result on new records, which are listed as an associated record to the original.

3.1.3 Locations

Location are a logical entity defined in Medicapt, which typically represent a specific clinic or health center where records are collected. Given the sensitive nature of the data MediCapt stores, forms and records are associated with a location, while users have a list of locations they have access to. Naturally, users can only see data from locations they have access to. Furthermore, not only do locations serve as a security feature, preventing users from accessing data not relevant to their work, but they are also critical as MediCapt expands beyond a few clinics, preventing users from being flooded with irrelevant data.

3.1.4 Users

Users are direct participants that have a MediCapt account and a specific role in the functioning of the platform. MediCapt was designed with 5 different user types in mind, briefly described in Table 3.1, each with it's own portal and responsibilities within the system. Given that each user type has its own portal, with different interfaces specific to each user type's role, every user in the system belongs to exactly one user type. The different user types, and especially the role they play in the platform will be discussed more in depth in the upcoming Section 3.2.

3.2 Expected Functionality

The functionality that MediCapt must support is divided into the different types of users in the platform, each with different responsibilities in the platform. Built with the specific objective of recording forensic and medical data to aid in prevention and prosecution of sexual violence and other human rights violations, MediCapt offers the different participants in the platform different functionality. To this end, we built five different portals, for the five different user types described in Table 3.1. Sections 3.2.1-3.2.5 will discuss each user type in detail, by describing the functionality built into each user portal.

User Type	Description	Permissions
Form designer	Collaborators in charge of designing medical forms to be made available to medical providers.	Read, write forms (Specified locations)
Medical provider	Trained personnel that create medical records during forensic evaluations and share the records with relevant users.	Read forms; Read, write, share records (Specified locations)
Associate	Relevant authorities that can view medical records shared with them.	Accept, reject, view records shared
Manager	Users in charge of managing locations by giving or revoking users access to a location or approving new forms for said location.	Create users; Modify user permissions and attributes; Approve created forms (Specified locations)
Researcher	PHR researchers given access to system and patient record data, for work in identifying patters in human rights violations and preventing mass-crime.	Read only access to aggregate, anonimized records and system statistics.

Table 3.1: Different user types in mediCapt–each of which has different roles within the platform, and therefore have access to different portals–listed along with a short description of their role and permissions on the platform.

3.2.1 Form designer portal

As discussed in Chapter 2, a critical requirement for MediCapt is to support many different medical forms for reporting of sexual violence and other human rights violations. For this purpose, we built a robust form creation and management interface into MediCapt, accessible by form designer users. Form designers are therefore responsible for encoding forms, such as the Standard Sexual Violence form, into MediCapt for medical providers to fill out when performing a forensic medical evaluation. The form designer portal provides form designers with an interface which allows them to build forms with pre-built components which are rendered based on the the type of data that needs to be collected in a form section. It also supports more sophisticated flow logic, to modify latter parts of the form based on previous response, and validation logic so forms can prevent invalid data from going into a record. Once form designers build a form, they get to publish it to locations they have access, though the form will have to be approved by a manager before it is accessible to medical providers.

Because form designing is done sparingly, and doesn't need to happen on remote locations, and given the relative complexity of designing a form compared to filling one out, the portal for form designing is built exclusively for web, is meant to be accessed from a personal computer, and assumes internet access during the whole workflow. Because forms are currently written as structured text files (YAML), we also assume a relative sophistication in the form designer users, to build more complicated forms, understanding the custom flow/ validation logic that can be used in building forms (though we do offer an interactive view that shows the resulting form real-time). We believe, after close work and consultation with PHR, that these are reasonable assumptions.

3.2.2 Healthcare provider portal

Healthcare providers are the trained medical professionals who collect forensic data, following a form, to build records in MediCapt. Given the central role they play in MediCapt's success–their buy-in is critical since they collect all the medical data that goes in the platform–their feedback has been central in shaping how the platform looks today. For providers, we built a portal accessible from Android devices (as a mobile application) and web browsers, which lets them browse forms and records available in their locations, and create, modify or share records. Record creation and modification is done through interaction with the form components specified by form designers, filling out text fields, drawing on pictograms, or selecting an option from a list, among others. Sharing is then also built into the platform, as long as the relevant authorities have an associate account.

Given the varying degrees of sophistication in providers, and its deployment in remote areas with spotty connections, the provider portal was built with simplicity and reliance in mind, heavily focusing on making the record creation and modification flows as simple as possible, and building in local caching (discussed in Capter 4) to limit reliance on a working network. Many domain specific features were also built in at the request of medical providers. For example, the printing feature, which generates a printable PDF of virtual records, was added to the platform after providers deemed it necessary to have a physical manifestation of data stored in MediCapt for storage in facilities and as a receipt for patients [39].

3.2.3 Associate portal

Associates consist of other parties, aside from medical providers, that should have access to specific, identifiable medical records through MediCapt. Examples include police officers, lawyers, or judges, who should have access to full unredacted records for an effective prosecution of the crime. Associates only have access to records explicitly shared with them (as opposed to providers who have access to all records in locations they have access to), and can accept or reject shared records, and browse through records they have access to. Also in contrast to providers, their access to records is read-only and time-limited. Because of the similar focus on specific records, the associate portal is quite similar to the provider portal, though is generally more limited. Additionally, after consultation with PHR, we have disabled access to the associate portal through the mobile app, and have not built in the caching system for offline use. Therefore, like the form designers, associates are expected to access their portal through a web browser with an active internet connection.

3.2.4 Manager portal

Managers are critical to the functioning of the MediCapt, especially as it is deployed more widely, since they are in charge of creating users, enabling or disabling them on the platform, and giving them permission to certain locations (managers themselves can only give permission to a location they have access to). We have built a portal where managers will have access to one or a group of locations, and will authenticate and then add accounts for form designer so they can create forms, for new medical providers so they can begin creating records, or for associates so that records can be shared with them. The portal also gives managers the ability to approve forms in locations they have access to, and to review access logs to records, to make sure record access isn't being abused by providers or associates.

In addition to traditional managers, which use the portal as described above, a tiny subset of manager users are designated as administrators. Administrators, who access MediCapt through the same portal as traditional managers, have two main differences in functionality to other managers. First, administrators aren't associated to particular locations; instead, they have access to all locations. Second, administrators have the additional functionality of being able to reidenitfy anonimized records, as a "breakglass" mechanism for cases in which reidentification is critical. Because of the broad visibility that administrators have over the whole platform, the small number of them needed (among other things) to create other manager users, are hardcoded into MediCapt infrastructure, to prevent privilege escalation of a compromised manager account by malicious actors.

3.2.5 Researcher portal

While the immediate challenge that MediCapt targets is gathering court-admissible forensic medical data to improve conviction rates of sexual violence, the data gathered in the platform presented us with another important opportunity for mass-crime prevention. The research platform, which will be built in the near future, was therefore conceived as a tool for PHR researchers to access aggregate data from medical reports, for identifying patters in reports, and be able to quickly respond to them. Because records contain sensitive medical data and identifiable information, it is important that all data accessible from the researcher portal has been aggregated and de-identified. The research portal will then present this aggregate data, as well as system statistics, to provide researchers with a general understanding of the data for prevention and response to mass crimes and identification of possible improvements in data collection.

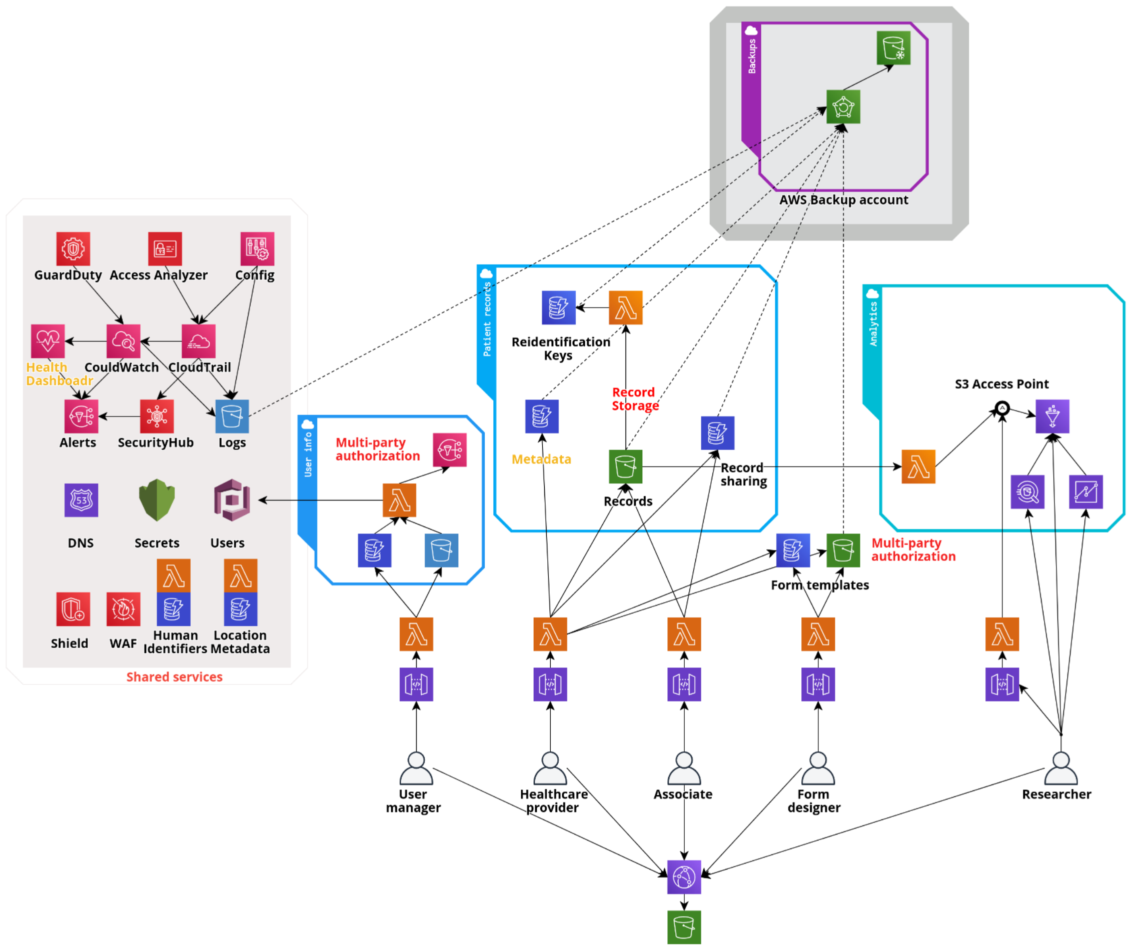


Figure 3-1: Server-side infrastructure diagram with services provisioned and the interactions between them. The services are labeled with their purpose, and what managed AWS services are used for different purposes.

3.3 Server-side Architecture

To build a highly scalable, future-proof application that would not need much maintenance into the future, we opted to deploy MediCapt on AWS. Making it highly maintainable therefore meant building the app as a collection of microservices, taking advantage of AWS managed services that reduced logic we would need to maintain, and limiting necessary custom services to small, stateless functions as much as possible. We did this while maintaining a focus on security and privacy compliance, which pushed us to keep the the system design (and importantly data flows) as clean and understandable as possible. The backend infrastructure diagram, seen in

Figure 4-2, outlines the key interactions between different services in the MediCapt backend. Key design decisions illustrated, that support our goals for maintainability, scalability and security, include different user pools and API gateways per user type, separate storage for different data types, distinct access points per data permission level, decoupled metadata, data, and associated files for form and record types, and use of presigned URLs for associated file downloads and uploads. We will explain these decisions more in depth, and justify why they were made in the context of our maintainability, scalability and security goals for the platform.

3.3.1 Different gateways per user type

As discussed previously, Medicapt functionality is divided into different portals for varying participants with differing roles in the platform. To prevent privilege escalation attacks that might compromise the whole platform, different user types are stored in separate Cognito pools, with identities for every role completely unrelated to one another. Furthermore, we built separate API gateways for every role, with separate endpoint handlers, each of which is authenticated against a specific pool so that e.g., a compromised associate account can't be used to call manager endpoints. Each endpoint then uses custom logic to enforce role-specific restrictions to data, such as location-based access for providers, or manually sharing of records for associates.

3.3.2 Separate storage for different data types

While separate endpoints for different user types are useful in preventing malicious users from getting access to data through privilege escalation, we also aim to reduce the surface for other kinds of attacks and prevent developers from inadvertently giving users access to the wrong types of data–both of which would undermine our push for security–by storing different data types separately (and giving services permissions only to resources they need). Therefore, we store user information, record metadata, and form metadata in different Dynamo tables, and different types of data in different S3 buckets. Shared record information, which is accessed by additional roles compared to general record metadata, is also stored separately, as are re-identification keys (which require administrator access and are therefore not accessible through any of the gateways). In fact, as an extra layer of security, different data types are often stored in separate virtual private clouds, as seen in Figure 4-2. This separate storage reduces the surface of attacks for these databases or buckets, since less endpoint handlers will have permissions to access data from each individual store. For instance, no endpoints from the form designer gateway have access to any record information.

3.3.3 Distinct access points per data permission level

When the same data can be accessed at different permission levels, we reduce the possibility for accidental or malicious access of a higher permission level by building different access points per permission level, which enforce these permissions. Health record data, for example, is accessible at different permission levels, since as protected health information it must be de-identified for users beyond the healthcare providers (such as researchers). Instead of duplicating the data at the different permission levels, which could be done using a trigger that runs when a new record is added–to de-identify it and store the anonymized record on a separate S3 bucket–we make use of an S3 access point with an attached lambda function that automatically de-identifies the data (which serves essentially the same function, but anonymizes the data on every request). Since the researcher endpoint handler doesn't have permission to access the records S3 bucket directly, but instead has to go through the access point with the attached de-identification function, the access point serves the same function as separating the storage of personally identifiable records from the de-identified ones (with the added benefit that we can circumvent the anonymization in extreme cases, by providing the function the record's re-identification key).

3.3.4 Decoupled metadata, data, and associated media

Because of database limitations and scalability concerns, we cannot store whole forms or records–which can become very large–in Dynamo. However, storing them in S3 buckets without a database would severely affect performance when filtering for specific forms or records by attributes other than their key. Instead, in designing MediCapt, we decided to separate metadata and data, storing enough metadata for users to be able to query or filter the records or forms in Dynamo, while storing the whole form or record data as S3 objects.

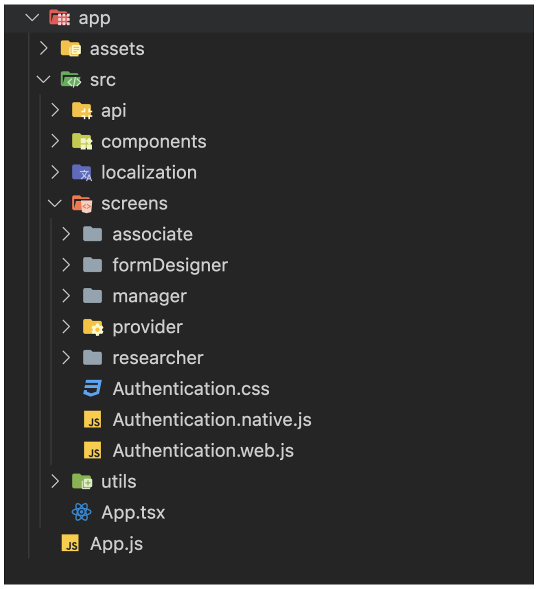
Furthermore, instead of storing the whole form or record data (included associated media) as a single S3 object, form and record data is decoupled from any associated files. Both the form or record data and all of its associated files are stored as separate objects in S3, with their hashes–prefixed by the form or record's UUID– as S3 keys (which prevents the same file across form or record versions from being stored twice). In order to link a form or record metadata to its data and associated files, the data and all of the files are listed, with their hash, in a main manifest file (also stored in S3), which has a root pointer to the data object. This manifest's hash, which is also its S3 key (prefixed by the form or record UUID), is then stored within the metadata in Dynamo.

3.3.5 Presigned links for media downloads and uploads

Given that forms and records are often uploaded from slower or unreliable networks, and they might be quite heavy, with many associated media files, it was important that sending or receiving forms or records wasn't done in a single payload. Therefore, we allow clients to upload and download forms in several requests (roughly one request per file in the manifest). To limit the number of endpoints and custom code we needed to maintain, and the compute resources we spend in downloads of uploads, we opted for using presigned URLs for users to download or upload files from or to S3.

While the gateway handlers have access to all rows in databases they have permissions for, access to S3 objects are handled more granularly. In MediCapt, by default, the handlers have no access to any S3 objects. In the case of a download or upload request, only once a custom handler logic verifies the user who made the request has access to the resulting forms or records in Dynamo, is the handler given permission to access S3 objects prefixed by the resulting forms or records unique identifiers (through creation of an AWS IAM role created on the fly). Once the handlers have permissions for these S3 objects, they can generate presigned S3 URLs with a short expiration date, so the clients can upload or download files from the links. The metadata and manifest (with the presigned links included), is then returned to the client.

3.4 Client-side Architecture



```
app
  assets
  src
    api
    components
    localization
    screens
      associate
      formDesigner
      manager
      provider
      researcher
      Authentication.css
      Authentication.native.js
      Authentication.web.js
    utils
    App.tsx
  App.js
```

Figure 3-2: Client-side file structure for project with all portals built in. The provider platform, which contains code which will be built for Android deployment lives among all the other portals, and shares components, utilities and other files with them.

As stated previously, a key guiding principle in the development of MediCapt was making it highly maintainable. On the client side, the maintainability effort focused on building the portals for the different user types on a single codebase with the same tech stack, and using well-supported libraries that worked across platforms to reduce the amount of custom code written. This was made particularly challenging given the heterogeneous platform requirements for the different portals (the provider portal, in particular, needed to be built as an Android app for low connectivity environments, while other portals needed to be built for web browsers and could assume reliable internet).

We solved this issue by writing a single React Native application and leveraging React Native for Web[1], which allowed us to build the same codebase for both Android and web. Nativebase[2], the component library we used to speed up development, recently added support for React Native for Web, which meant standard components were standard across all portals, regardless of the platform each portal was built for. In the rare scenario where separate code for web and native build targets was necessary, we added platform checks in code, or used React Native's ability to choose between a `.web` and `.native` file depending on build target. This approach allowed us to use a single codebase with plenty of reused code across protals, and kept the total amount of code to maintain at a minimum.

To ensure that each user sees only the portal that is relevant to their role, users are prompted to select what portal they wanted to sign in for and Cognito authenticates the credentials they send in against the user pool they selected. The authenticated user object returned to the client contains the user type, which the client then uses to display the relevant portal. The authenticated token the client received, meanwhile, belongs to a specific user pool, and therefore can only be used to make requests on the gateway corresponding to the selected user type. Finally, using a platform check, we made sure only enable logging in to portals that were built for the platform being used (e.g. we only enable the provider portal on Android).

[1]https://necolas.github.io/react-native-web/
[2]https://nativebase.io

Chapter 4

Caching System

While most of the MediCapt platform was built as a web application, to be accessed through a personal computer with internet connectivity, medical record collection had the specific challenge of needing to be accessible in remote areas with poor connectivity and access only to mobile phones or tablets. To tackle this challenge–of making the healthcare provider portal usable in low connectivity areas–we developed a caching system that enabled offline use for the parts of the platform that used it. In keeping with the goals for the platform, we aimed to build a caching system that is simple to integrate, modular, and secure.

We have developed an encrypted, multi-layered caching system that exposes an API similar to that used for network requests, while utilizing the device's local storage–through the key-value store MMKV[1]–and data structures computed at app startup to provide offline functionality and limit the network usage. While the caching mechanism is currently only in use on MediCapt's medical provider portal, it can be expanded to other user portals, to make more efficient use of network bandwidth and provide offline functionality across MediCapt.

This chapter will discuss the design of the caching system, along with the motivation for an interface modeled after the backend API, before discussing each individual system layer, with their responsibilities and subtleties that arose in building them.

[1]https://github.com/Tencent/MMKV

4.1 System Interface

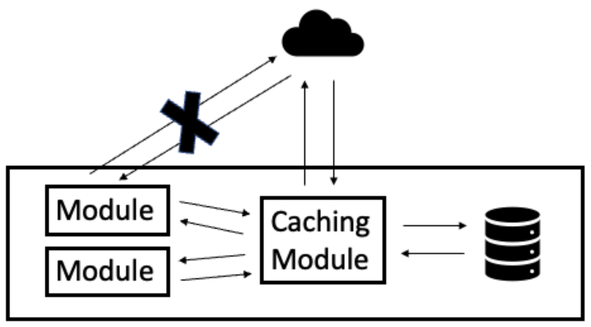


Figure 4-1: Interaction between a black-box caching module and the rest of the client. We can see calls from the healthcare provider portal to the server were swapped for calls to the caching module, which can itself calls the server, or resolves the request locally from storage.

The caching system used in the MediCapt client's provider portal serves two main purposes: to store copies of relevant objects for offline reading, and to store modified objects when offline, for forwarding to the server once a connection is established (an architecture inspired by store and forward techniques from delay-tolerant networks in telecommunications [43]). Because maintainability was a key consideration when building MediCapt, we decided to build this caching mechanism into a module that exposed an API similar to the module that made web requests, and transparently dealt with caching objects. This way, the client code across all portals is kept relatively uniform, with the provider-specific code swapping out the the server requests module with the caching system module.

This almost one-to-one swap of server calls to caching system calls essentially allows the caching system to intercept what would've been all calls to server. As seen in Figure 4-1, the caching system has two alternatives: if it's offline, or for some other reason wants to resolve the call locally, it emulates the request locally by reading or writing to local storage–where in the writing case it will result on dirty data on disk– or it can forward the request to the server. Because it intercepts the server response as well, when forwarding the request to server, the caching system uses the server response to update the resources on local storage, keeping the values in storage as current as possible.

4.2 Cached Object Types

MediCapt's client side cache was built specifically for the provider portal, which had the requirement of working well on low-connectivity environments. Because of this, the cache was designed around the data types the provider gateway has access to, and the types of operations it can perform on them. In particular, providers have read-only access to users, locations and forms, and read-write access to records (as well as record sharing). As discussed in the description of the caching system interface on Section 4.1, when resolving a request locally, the caching module emulates the request locally. In this section, we will discuss each data type individually, and explain how we approach locally emulating the operations that the provider gateway can perform on them.

4.2.1 Users & Locations

Form and record metadata contains references to user and location unique identifiers (UUIDs) to keep track, among other things, of their creator or the locations they should be available at. The provider gateway therefore includes a few endpoints that have read-only access to user and location metadata, allowing the provider portal to link a user or location's UUID to their name or any other relevant information contained in their metadata for the purpose of displaying it in the portal (in connection to a form or record). While on the backend users are versioned, supporting different user versions locally is unnecessary, since forms and records are linked to unversioned users. Furthermore, there are no endpoint on the provider gateway for users or locations that take a version as a parameter: when a provider gateway endpoint requests user or location metadata from sever, the most recent one is always returned. Therefore, locally it suffices to have only the latest version of any user or location seen.

To support emulating the read-only accesses to user or location metadata, therefore, any metadata object returned from a successful server request is placed in storage, indexed only by its unique identifier (and not by version). Any future request for that user or location metadata, where the network is unavailable, will find the previously cached metadata object in storage. The request for the list of all users, available on the provider platform, is emulated by iterating over all keys and filtering for the ones whose values are user metadata objects (in practice, we will see in Section 4.3 that this expensive operation will be done only on app startup or login, and its result will be kept in memory).

4.2.2 Forms

Although the provider gateway only has read-access to forms, their more complicated structure that includes metadata, data, and associated files (discussed in Section 3.3.4)–compared to users and locations which only contain metadata–make form storage on the caching system more complicated than that of users and locations. Additionally, unlike users and locations, records are linked to specific versions of forms, which make it important to store keep different versions cached locally. Therefore, form metadata objects returned from successful server requests are stored locally, indexed by the form UUID and the version. The form manifest, and form data plus associated files once they are downloaded, are also cached locally, indexed just by their hash. To avoid unnecessary downloads, after receiving–from a server request to get a form–the form metadata along with the form's manifest with download links for its files (as described in Section 3.3.5), the caching system only downloads files whose hashes are not already in storage.

To emulate read-only access to forms, when resolving a request for a form locally, the caching system takes one of three approaches:

1. If a specific form version is requested, the request is successful if the form metadata–indexed by the form UUID and version requested–is cached, as well as the manifest file indexed by the hash specified in the form metadata, and any files listed in the manifest indexed by their hashes (which are specified in the

manifest). A small difference in the interface for the caching system compared to the server here is that instead of returning the manifest with download links, it returns it with base64 encodings of the downloaded files.

2. If no specific form version is requested, the caching system iterates over all the values in storage, filtering for those indexed with the specific form UUID. If any value indexed by that form UUID exists, the one with the latest version is chosen, and then operation succeeds as before: if the chosen metadata's manifest is in storage, as well as all of the files listed in the manifest. As before, a manifest with base64 encodings for the files listed is returned, which is a slight change from the server interface.

3. Finally, to emulate the request for a list of all the forms, the caching system iterates over all the values in storage, and filters out for form metadata objects, choosing the latest version for each unique form UUID. Like the server, it returns a list of form metadata objects.

As before, we will see in Section 4.3 that the expensive operation of iterating over all the values in storage–used for emulating a few forms operations–will be done only once on app startup or login.

4.2.3 Records

Records share the more complicated structure, and a need to support accessing different versions locally, with forms while additionally needing to support emulating record writes locally. Therefore, supporting emulating the operations that are performed on records require some additional logic as compared to emulating operations on forms. In particular, whenever records are modified locally, a different "dirty" record metadata object is saved to storage. Dirty record metadata objects don't change the record version, or some of the other server-generated metadata, from their "clean" counterpart. Therefore, instead of indexing "dirty" record metadata by record UUID and version, like with "clean" records, they are indexed by the record UUID and a modification timestamp.

When a network connection is reestablished, the caching system iterates through all the values in storage, filtering for these “dirty” records, and attempts to upload them. Successful uploads of “dirty” records return “clean” records which are saved to storage, with an additional field (missing from records that didn’t come from a local “dirty” record upload) that indicates the timestamp of the “dirty” record which was uploaded. By iterating through all the “clean” records, therefore, we will be able to determine the timestamp of the last uploaded “dirty” record (and the clean record that corresponds to that upload).This timestamp of the last uploaded “dirty” record will be very important for emulating read-only operations locally.

The handling of these read-only operations goes along the same lines as that of forms. As with forms, to emulate read-only access to records, the caching system takes one of three approaches.

1. If a specific form version is requested, the operation succeeds if a “clean” record metadata indexed by the record UUID and version requested, plus chosen metadata’s manifest and all of the files listed in the manifest are in storage. Only “clean” records are considered as only they are assigned record versions by the server. As with forms, the caching system interface is slightly different from the server interface, returning manifests with base64 encodings for the files, instead of download links.

2. If no specific version is requested, the caching system will iterate over all values in storage, filtering for versions of the record metadata object with the UUID requested, to determine three things: the latest “clean” record (as determined by versions), the latest “dirty” record (as determined by timestamps), and the timestamp of the last uploaded “dirty” record (described above). It then returns the latest “clean” record if there’s not “dirty” record or if the timestamp of the last uploaded “dirty” record is the same as the timestamp for the latest “dirty” record, or the latest “dirty” record otherwise. The request succeeds if a metadata object, with its corresponding manifest and files are in storage, and returned the metadata object along with the manifest with base64 encodings of the files.

3. If the list of all records is requested, a set of all of the record UUIDs in storage is created, and for each of them, approach (2) is followed to decide whether to return the latest "clean" or "dirty" version for the UUID. The request then returns the metadata object selected for all of the record UUIDs in storage.

In the upcoming Section 4.3 on the architecture of the caching system, we will see how these operations that need to scan through all the values in storage are made more efficient through in-memory caching.

4.3 Caching System Architecture

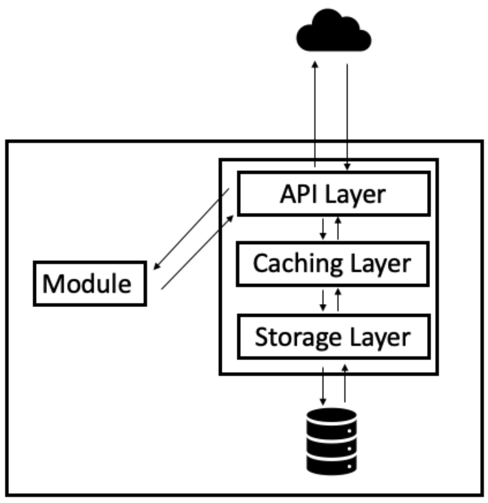


Figure 4-2: Caching system architecture diagram, illustrating interactions between the caching layers, and with other components of the platform. We can see how the rest of the client interacts with the API layer, which also interacts with the server API, while the storage layer interacts with the database directly.

To design a modular and secure caching system that exposed an API almost equal to that of the provider backend API–emulating the requests locally when not forwarding the request to the backend–but at the same time was efficient, robust, and tolerant to failure (all while running on a simple lightweight key value store that doesn't support transactions) we built the caching system into a 3-layered system, with each layer exposing a progressively more sophisticated API. This layered system allowed us to build a modular and simple to understand, yet efficient system that exposes a powerful API, out of very simple building blocks. We will explore each of the layers more in detail, starting from the lower layer, which interacts directly with the database, and moving up to the highest, which receives the client requests and decides whether to forward them to the server, or resolve them locally.

4.3.1 Storage Layer

The storage layer is the lowest layer in the caching system, and is the layer that interacts directly with storage (through the MMKV key-value store), and exposes an interface that is better suited for the needs of the caching system. Because storage is shared among users, it's also the layer that filters objects in storage by user, and because all objects put in storage are encrypted, also the layer that deals with encrypting and decrypting objects. Finally, because MMKV only allows mappings between strings, the storage layer is responsible of serializing and parsing keys and objects in storage, and enforcing a key type to value type mapping.

The storage layer, therefore, exposes a set of keys and value types, and exposes the functions `contains(key)`, `get(key)`, `put(key, value)` and `getKeys(user)`, to store these value types indexed by the key types. As stated before, the storage layer enforces mappings between specific types of keys and values (as described in Table 4.1).

4.3.2 Cache Layer

The cache layer, or the middle layer, computes and caches (in memory) derived state to make certain computations more efficient, and maintains this derived state consistent by updating it on every operation. This layer exposes a `createCache(user)` method which should be called on login or app startup, and which creates a cache object which maintains the following data structrues in memory: (1) a map between form UUIDs and the latest version of the form in storage, (2) a map between record UUIDs and the latest version of a clean record in storage, (3) a map between record

Data Type	Storage Key	Storage Value
Form	`{ user, type: ‘form’, formUUID, version }`	`{ formMetadata }`
Clean Record	`{ user, type: ‘clean-record’, recordUUID, version }`	`{ recordMetadata, uploadOf }`
Dirty Record	`{ user, type: ‘dirty-record’, recordUUID, modifiedTime }`	`{ recordMetadata, modifiedTime }`
Location	`{ user, type: ‘location’, locationUUID }`	`{ locationMetadata }`
Location	`{ user, type: ‘user’, userUUID }`	`{ userMetadata }`
File	`{ user, type: ‘file’, hash}`	`string`

Table 4.1: Data types that the caching system storage layer API supports, along with the shape of the key and corresponding value objects enforced by the storage layer.

UUIDs and the latest dirty record in storage, and (4) a map between the record UUIDs and the latest uploaded dirty record in storage. These data structures, computed on login or app startup along with the cache object, on a single pass of all the objects in storage, have to be kept up to date whenever anything is added storage. For instance, on every operation that adds a form to storage, we check if the form being added has a later version than the previous latest version for that form. If it does, we update the cache data structure to reflect the new latest form version in storage.

We keep all these in memory instead of storage, and incur the cost of recalculating them every time the app starts, to make the caching system more fault tolerant. Since the whole system is built on top of a very lightweight key value store that doesn’t support transactions, any operations should write into storage just once, or we run the risk of ending up in an inconsistent state if there is a crash between two writes belonging to the same operation. For simplicity and understandability of the system, then, we limit the system to write the bare minimum to disk (i.e. the values described in Table 4.1), and any derived storage is kept in memory and has to be recomputed on app start.

Finally, the cache layer exposes mainly a simple get/put API for each data type, but when getting forms or records, abstracts away the complexity of finding the latest form or record if none is specified (made easy by its cache object) and in the case of records deals with deciding whether to return a "clean" or "dirty" record, abstracting that decision away from the API layer.

4.3.3 API Layer

The API layer, which is the highest level layer in the caching system, exposes a simple API designed after the server requests API for any part of the app to use, and is responsible of implementing the high level logic to emulate the server requests if they need to be resolved locally (using the cache layer API as building blocks). Unlike the lower layers, which are all synchronous, this layer functions asynchronously, since it often makes network requests. To decide whether to resolve a request locally, it generally attempts to forward the request to the server first, before defaulting to local resolution (though to accommodate for parts of the app that might not need fresh data, API layer functions take a flag to force the request to resolve locally). The API layer also exposes a method that has to be called by the client on login or app startup, that calls the `createCache(user)` function from the cache layer, and additionally starts a background task that checks for dirty records in the cache for upload to the server, and attempts to upload them if there is an internet connection.

Chapter 5

Contributions and Future Work

This thesis has discussed the development of the first open-source, in-the-field health collection platform for use in the forensic documentation of human rights violations, adding to the literature on the use of mobile technology in the fields of healthcare, human rights work, and data collection. With MediCapt 3.0, we introduced a platform that aims to avoid the common issue mobile technology platforms in healthcare and human rights work have had moving beyond their pilot phase, by focusing on future-proofing, while building on top of many years of iterative work with key stakeholders by Physicians for Human Rights.

The discussions on building a platform that scales to any reasonable level of demand and adapts to different contexts, that requires little to no maintenance or development for a significant period of time, and that is compliant with the latest security and privacy regulations and best practices should be critical in any technology platform that aims to expand beyond a small number of communities. Furthermore, discussions on the definition of key concepts and expected functionality, and discussions about the development of the MediCapt server and client-side architectures, as well that of the client-side caching system, should serve as case studies on the development of technical platforms in the field of social ventures.

5.1 Future Work

While a preliminary version of MediCapt 3.0 is in the hands of Physicians for Human Rights, and it's actively being tested, at its current stage it still has some ways to go before critical features are finished, it gets open-sourced, and is fully deployed. This section presents a roadmap for the next steps in the development of MediCapt, before it's ready for wide deployment.

5.1.1 Researcher Portal

An important first step in finishing the development of the MediCapt platform is finalizing the design and development of the researcher portal, which requires significant work. While the objective for the platform, its general system architecture and some required infrastructure work are complete, much of its feature set is underdefined. Because of its potential for prevention and fast response to mass crimes, its development should be a priority.

5.1.2 User tests

While a lot of feedback from stakeholders on previous versions of MediCapt has gone into the development of MediCapt 3.0, little user testing has been done directly on this new platform, as PHR has just started testing it out. Once stakeholders are more comfortable with the new tools, and some of the more critical issues are ironed out, user testing will be critical for the completion of the platform. Feedback on the usability, features and many other important factors will lead the next stage of development.

5.1.3 Security audit

A key objective with MediCapt was making it compliant with the latest privacy and security regulations and best practices, which give stakeholders confidence in how we deal with patient data, and would aid in making the platform available different jurisdictions. While security and privacy are at the core of the platform design, a security audit in the future is critical to get feedback, validate our design, and ensure the system is compliant.

5.1.4 Improved localization

While MediCapt was initially built to be deployed by PHR primarily in Kenya and DRC, this new version is intended to adapt to different contexts, which includes different regions and languages. Some infrastructure for localization (support for different languages in particular) was built into MediCapt already, but there's still much to do in this area. A lot of work is needed, for example, to make MediCapt usable in any language other than English.

5.1.5 Open sourcing

MediCapt 3.0 was always meant to be open sourced, to increase transparency (allowing independent verification of the code) and collaboration (allowing other people to build on top of it and spreading the work of maintaining it among a wider community). Before open sourcing the project, however, there's significant work to be done on documentation, code-cleanup, and determining contributing guidelines.

Bibliography

[1] John; Akter, Shahriar; D'Ambra and Pradeep Ray. User perceived service quality of mhealth services in developing countries. In *ECIS 2010 Proceedings. 134*, 2010. `https://aisel.aisnet.org/ecis2010/134`.

[2] Arwa Fahad Albabtain, Dina Ahmad AlMulhim, Faisel Yunus, and Mowafa Said Househ. The role of mobile health in the developing world: a review of current knowledge and future trends. *Cyber Journals: Multidisciplinary Journals in Science and Technology [JSHI]. Journal of Selected Areas in Health Informatics42*, pages 10–15, 2014.

[3] Ted Alcorn. Responding to sexual violence in armed conflict. *The lancet*, 383(9934):2034–2037, 2014.

[4] Yaw Anokwa, Carl Hartung, Waylon Brunette, Gaetano Borriello, and Adam Lerer. Open source data collection in the developing world. *Computer*, 42(10):97–99, 2009.

[5] Skoll Foundation Archives. How do we cure mhealth pilotitis? `https://archive.skoll.org/debate/how-do-we-cure-mhealth-pilotitis-critical-lessons-in-reaching-scale/`. Accessed August 5, 2022.

[6] Omar Ayaad, Aladeen Alloubani, Eyad Abu ALhajaa, Mohammad Farhan, Sami Abuseif, Ahmad Al Hroub, and Laila Akhu-Zaheya. The role of electronic medical records in improving the quality of health care services: Comparative study. *International Journal of Medical Informatics*, 127:63–67, 2019.

[7] Inc. Beneficent Technology. Benetech and human rights. `https://benetech.org/story/benetech-human-rights/`. Accessed August 5, 2022.

[8] Inc. Beneficent Technology. Martus documentation. `https://www.martus.org/resources/documentation.html`. Accessed August 5, 2022.

[9] Inc. Beneficent Technology. Martus sunset. `https://benetech.org/blog/martus-sunsets-human-rights-data-collection/`. Accessed August 5, 2022.

[10] W Brown and S Varanasi. Mobile technology to improve data collection after sexual violence. *The Lancet Global Health*, 3:S19, 2015. Consortium of Universities for Global Health, 6th annual conference.

[11] William Brown, Po-Yin Yen, Marlene Rojas, and Rebecca Schnall. Assessment of the health it usability evaluation model (health-ituem) for evaluating mobile health (mhealth) technology. *Journal of Biomedical Informatics*, 46(6):1080–1087, 2013. Special Section: Social Media Environments.

[12] Vosloo Steven Castillo, Nathan M. Medic mobile: Case study by unesco-pearson initiative for literacy. 2017. `https://unesdoc.unesco.org/ark:/48223/pf0000260597`. Accessed August 5, 2022.

[13] Vital Wave Consulting. mhealth for development: The opportunity of mobile technology for healthcare in the developing world. *Washington Dc and Berkshire, UK*, 2009.

[14] Nagasushma Devarapalli and Silvia Figueira. Leveraging existing tools to help social enterprises: A case study. *Procedia Engineering*, 107:90–99, 2015. Humanitarian Technology: Science, Systems and Global Impact 2015, HumTech2015.

[15] Inc. Dobility. Surveycto. `https://five.epicollect.net`. Accessed August 5, 2022.

[16] Ksenia Ermoshina, Francesca Musiani, and Harry Halpin. End-to-end encrypted messaging protocols: An overview. In *International Conference on Internet Science*, pages 244–254. Springer, 2016.

[17] eyeWitness to Atrocities. eyewitness. `https://www.eyewitness.global/our-work`. Accessed August 5, 2022.

[18] Gunther Eysenbach et al. What is e-health? *Journal of medical Internet research*, 3(2):e833, 2001.

[19] Eleanor Farrow. *eyeWitness to Atrocities: Verifying Images with Metadata*, pages 143–155. Springer International Publishing, Cham, 2018.

[20] Maddalena Fiordelli, Nicola Diviani, and Peter J Schulz. Mapping mhealth research: A decade of evolution. *J Med Internet Res*, 15(5):e95, May 2013.

[21] Centre for Genomic Pathogen Surveillance. Epicollect5. `https://five.epicollect.net`. Accessed August 5, 2022.

[22] Physicians for Human Rights. Medicapt. `https://phr.org/issues/sexual-violence/medicapt-innovation/`. Accessed August 5, 2022.

[23] Physicians for Human Rights. Phr wins 2013 tech challenge for atrocity prevention with mobile app. `https://phr.org/news/phr-wins-2013-tech-challenge-for-atrocity-prevention-with-mobile-app/`. Accessed August 5, 2022.

[24] Physicians for Human Rights. Program on sexual violence in conflict zones. `https://phr.org/issues/sexual-violence/program-on-sexual-violence-in-conflict-zones/`. Accessed August 5, 2022.

[25] Jennifer Franz-Vasdeki, Beth Anne Pratt, Martha Newsome, and Stefan Germann. Taking mhealth solutions to scale: enabling environments and successful implementation. *Journal of Mobile Technology in Medicine*, 4(1):35–38, 2015.

[26] Frontline. Frontlinesms. `https://www.frontlinesms.com`. Accessed August 5, 2022.

[27] Claudia García-Moreno, Christina Pallitto, Karen Devries, Heidi Stöckl, Charlotte Watts, and Naeema Abrahams. *Global and regional estimates of violence against women: prevalence and health effects of intimate partner violence and non-partner sexual violence*. World Health Organization, 2013.

[28] Lindsey Green, Suzanne Kidenda, Roseline Muchai, and Brett D. Nelson. Centering survivors in technology for addressing sexual violence: The case of medicapt. `https://www.svri.org/blog/centering-survivors-technology-addressing-sexual-violence-case-medicapt`. Accessed August 5, 2022.

[29] Tracy D Gunter and Nicolas P Terry. The emergence of national electronic health record architectures in the united states and australia: models, costs, and questions. *Journal of medical Internet research*, 7(1):e383, 2005.

[30] Get ODK Inc. Okd: Collect data anywhere. `https://five.epicollect.net`. Accessed August 5, 2022.

[31] Kobo Inc. Kobotoolbox. `https://five.epicollect.net`. Accessed August 5, 2022.

[32] Gautam Ivatury, Jesse Moore, and Alison Bloch. A doctor in your pocket: health hotlines in developing countries. *Innovations: Technology, Governance, Globalization*, 4(1):119–153, 2009.

[33] Sharmin Jahan and M Mozammel Hoque Chowdhury. mhealth: a sustainable healthcare model for developing world. *American Journal of Modeling and Optimization*, 2(3):73–76, 2014.

[34] Chris Kimble. Electronic health records: Cure-all or chronic condition? *Global Business and Organizational Excellence*, 33(4):63–74, 2014.

[35] Donald E. Knuth. Mobile electronic health surveys and data collection: History and practical points to consider. In Vipan Nikore Juan Sebastian Osorio Kenneth Paik Leo Anthony G. Celi, Hamish S. F. Fraser, editor, *Global Health Informatics: Principles of eHealth and mHealth to Improve Quality of Care*, chapter 29. MIT Press, 2017.

[36] Siddique Latif, Rajib Rana, Junaid Qadir, Anwaar Ali, Muhammad Ali Imran, and Muhammad Shahzad Younis. Mobile health in the developing world: Review of literature and lessons from a case study. *IEEE Access*, 5:11540–11556, 2017.

[37] Magpi. Mobile data collection guide. `https://www.magpi.com/what-is-mobile-data-collection`. Accessed August 5, 2022.

[38] Erin Brisbay McMahon and Tracy Lee-Huber. Hippa privacy regulations: practical information for physicians. *Pain physician*, 4(3):280, 2001.

[39] Ranit Mishori, Michael Anastario, Karen Naimer, Sucharita Varanasi, Hope Ferdowsian, Dori Abel, and Kevin Chugh. mjustice: preliminary development of a mobile app for medical-forensic documentation of sexual violence in low-resource environments and conflict zones. *Global Health: Science and Practice*, 5(1):138–151, 2017.

[40] Nareesa A Mohammed-Rajput, Dawn C Smith, Burke Mamlin, Paul Biondich, Brad N Doebbeling, Open MRS Collaborative Investigators, et al. Openmrs, a global medical records system collaborative: factors influencing successful implementation. In *AMIA annual symposium proceedings*, volume 2011, page 960. American Medical Informatics Association, 2011.

[41] Karen Naimer, Widney Brown, and Ranit Mishori. Medicapt in the democratic republic of the congo: The design, development, and deployment of mobile technology to document forensic evidence of sexual violence. *Genocide Studies and Prevention: An International Journal*, 11(1):25–35, 2017.

[42] World Health Organization et al. *The MAPS toolkit: mHealth assessment and planning for scale.* World Health Organization, 2015.

[43] C. E. Palazzi, A. Bujari, S. Bonetta, G. Marfia, M. Roccetti, and A. Amoroso. Mdtn: Mobile delay/disruption tolerant network. In *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6, 2011.

[44] Guardian Project. Camerav app and the informacam system. `https://guardianproject.github.io/informacam-guide/en/InformacamGuide.html`. Accessed August 5, 2022.

[45] Guardian Project. Camerav: Secure verifiable photo video camera. `https://guardianproject.info/archive/camerav/`. Accessed August 5, 2022.

[46] Guardian Project. Informacam: Verified mobile media. `https://guardianproject.info/archive/informacam/`. Accessed August 5, 2022.

[47] Aviva Rutkin. Collecting horror stories. *New Scientist*, 223(2977):16, 2014.

[48] Joel Selanikio et al. Episurveyor/magpi. https://lib.digitalsquare.io/bitstream/handle/123456789/77558/episurveyormagpi.pdf. Accessed August 5, 2022.

[49] Markus Steinberg, Sirko Schindler, and Friederike Klan. Software solutions for form-based, mobile data collection – a comparative evaluation. In Holger Meyer, Norbert Ritter, Andreas Thor, Daniela Nicklas, Andreas Heuer, and Meike Klettke, editors, *BTW 2019 – Workshopband*, pages 135–144. Gesellschaft für Informatik, Bonn, 2019.

[50] Amnesty International Panic Button Team. More than an app: the panic button, one year on. https://www.amnesty.org/en/latest/campaigns/2015/07/panic-button-one-year-on/. Accessed August 5, 2022.

[51] Amnesty International Panic Button Team. Panic button: Why we are retiring the app. https://www.theengineroom.org/panic-button-retiring-the-app/. Accessed August 5, 2022.

[52] Mark Tomlinson, Mary Jane Rotheram-Borus, Leslie Swartz, and Alexander C Tsai. Scaling up mhealth: where is the evidence? *PLoS medicine*, 10(2):e1001382, 2013.

[53] Neil Versel. Episurveyor creator selanikio shakes up international development. *MobiHealthNews*. https://www.mobihealthnews.com/10645/episurveyor-creator-selanikio-shakes-up-international-development. Accessed August 5, 2022.

[54] WITNESS. Is this for real? how informacam improves verification of mobile media files. https://blog.witness.org/2013/01/how-informacam-improves-verification-of-mobile-media-files/. Accessed August 5, 2022.

[55] WITNESS. Video as evidence. https://vae.witness.org. Accessed August 5, 2022.

[56] Benjamin A Wolfe, Burke W Mamlin, Paul G Biondich, Hamish SF Fraser, Darius Jazayeri, Christian Allen, Justin Miranda, and William M Tierney. The openmrs system: collaborating toward an open source emr for developing countries. In *AMIA annual symposium proceedings*, volume 2006, page 1146. American Medical Informatics Association, 2006.