Investigating Decentralized Management of Health and Fitness Data

by

Jason Paulos

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

May 2020

Author .....
Department of Electrical Engineering and Computer Science
May 12, 2020

Certified by .....
Lalana Kagal
Principal Research Scientist
Thesis Supervisor

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

Investigating Decentralized Management of Health and Fitness Data

by

Jason Paulos

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2020, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Electronic healthcare records are the new standard for storing healthcare data due to their ability to be easily and quickly accessed. Additionally, a new class of fitness records have been created in recent years due to the rise of wearable devices by companies like Fitbit, Apple, and Google. Yet these fitness records are all stored in different formats and can be difficult to extract from the proprietary systems in which they are stored. There are great potential benefits for individuals, healthcare professionals, and researchers to combine this new source of fitness data with traditional patient records in a secure way. The Solid project offers a solution to this problem by allowing individuals to store and manage their health data through the use of personal data stores. The main contributions of this thesis are extending Solid libraries to support the development of mobile Solid applications, developing the functionality to integrate sensor data from phones and wearables into Solid and model it using the FHIR RDF specification, and creating Solid Health, a proof-of-concept decentralized mobile health application.

Thesis Supervisor: Lalana Kagal
Title: Principal Research Scientist

Acknowledgments

I would like to thank my family and friends for supporting me throughout my academic journey. Without you, none of this would have been possible or worthwhile.

I would also like to thank my advisor, Lalana Kagal, for providing the valuable support, direction, and feedback that made this project possible.

Contents

1 Introduction **11**

2 Background **15**

2.1 Solid 15

2.1.1 Server 15

2.1.2 Client 16

2.2 Linked Data 17

2.2.1 Resource Description Framework 18

2.2.2 Ontologies 21

2.2.3 SPARQL 22

2.3 React Native 24

2.3.1 Web APIs 25

3 Related Work **27**

3.1 Healthcare Solutions 27

3.1.1 MedRec 27

3.1.2 Medicalchain 28

3.2 Distributed Data Platforms 28

3.2.1 InterPlanetary File System 29

3.2.2 Blockstack 29

3.3 Mobile Linked Data Applications 30

3.3.1 DBPedia Mobile 30

3.3.2 RDF on the Go 31

4 System Overview **33**

4.1 Purpose 33

4.2 Environment 34

4.3 Functionality 35

4.3.1 User Authentication 35

4.3.2 Data Collection 37

4.3.3 Data Sharing 42

4.4 Evaluation 43

4.4.1 Solid Health Features 43

4.4.2 Porting an Existing App 47

5 Future Work **49**

5.1 Native Authentication 49

5.2 Direct Data Transmission 49

5.3 Offline Support 50

5.4 iOS Support 50

5.5 Extending Solid Health 50

6 Conclusion **51**

List of Figures

2-1 The graph of an RDF triple 18
2-2 A Turtle RDF document 19
2-3 An equivalent Turtle RDF document 20
2-4 A subset of the FOAF ontology 22
2-5 A subset of the FHIR ontology 23
2-6 A SPARQL select query 24

4-1 Google Fit architecture 38
4-2 A FHIR RDF record for distance walked 41
4-3 Solid Health application screens and features 44
4-4 Mark Book applications for browser and mobile 47

Chapter 1

Introduction

In an increasingly digital world, electronic healthcare records have come to replace paper documents [1]. This opens up new possibilities for healthcare, as multiple institutions can now share and collaborate on medical records with ease. It is clear that when data becomes easier to create and easier to share, many new things are possible.

While this increased collaboration between medical institutions is beneficial, the current system is far from perfect. Between 2009 and 2019, healthcare data breaches have resulted in the theft or loss of over 230 million healthcare records in the United States [2]. Additionally, in order to improve the usefulness of electronic medical records, it is necessary to give patients more authority to decide who to trust with access to their highly sensitive records [3]. Allowing patients to have authority and unrestricted access to their medical records would be a profound step toward maximizing the security and utility of healthcare records.

A variety of systems have been proposed with the goal of addressing this problem. More often than not, they feature a highly technical solution to make healthcare data widely available but still secure, involving the use of a decentralized blockchain, cryptographic keys, or both. While these solutions may address the issue of data availability and security, they do not necessarily make this data easy and simple to use, since it may be difficult for patients and institutions to efficiently navigate such complex systems.

There exist other decentralized solutions that are much easier to comprehend. The Solid project has created a distributed data platform that allows users to exhibit real ownership of their data without the need for a blockchain or related technologies [4]. This is done through the use of Solid pods, which are personal data storage servers that individuals can host for themselves, or delegate to a trusted third party. Solid is designed to let users decide exactly who else should have access to this data on a granular level, so that trusting someone with access to data is not an all-or-nothing dilemma. Solid has been used in a growing number of Web applications that protect users' privacy by taking advantage of its unique data ownership abilities [5].

One area that Solid is not yet capable of operating in is native mobile applications. Over the past few years, mobile devices have emerged as the most popular way for users to access the Web [6]. It is therefore essential to provide a mobile-first solution to address the issue of health data ownership and usability. In this thesis, I present Solid Health as a proof-of-concept mobile application to manage, record, and share health and fitness data in a secure and useful way. Through the use of smartphone sensors and wearable devices, Solid Health can make health observations and upload them to a user's Solid pod. These records are formatted using an open standard for healthcare records, and the user has the option to share some of their data with others, such as family, coaches, and doctors on a case by case basis.

In this thesis, I discuss the following contribution I have made:

1. Extended Solid libraries to support the development of Solid apps in React Native.

2. Developed functionality to integrate sensor data from phones and wearables into Solid and model it in FHIR RDF.

3. Created Solid Health, a proof-of-concept decentralized mobile health application.

The motivating use case for this project is a distributed electronic health record storage system that can be used by researchers to conduct studies involving the col-

lection of individuals' health and fitness data while preserving the privacy of all participants. The SNAPSHOT study is an example of an existing study which has collected a variety of health data about users in order to examine relationships between mood, stress, social interactions, sleep, and other factors related to well-being [7]. Participants in the SNAPSHOT study used a wearable device to measure sleep-wake patterns and installed a mobile application that recorded information about their text messages, phone calls, and application usage. Enabling such a study to be conducted with guaranteed privacy measures would significantly reduce the potential for data misuse and hopefully incentivize more users to participate in such studies that require sensitive information but provide benefits to individuals and society at large.

Chapter 2

Background

2.1 Solid

Solid is a proposed set of conventions and tools for building decentralized social applications [4]. Solid builds on the standards and protocols of the Web in order to offer users secure storage and sharing of information. In the Solid ecosystem, a user is represented by a WebID, which is a Uniform Resource Identifier (URI) that can be accessed using the Hypertext Transfer Protocol (HTTP). A user's WebID represents a retrievable document containing that user's public profile, which may include their name, picture, and other public information about them [8]. This document also indicates the location of their Personal Online Datastore, or pod, which is a server that stores all of that user's information in the Solid ecosystem. For example, my WebID is `https://jas0n.solid.community/profile/card#me`, and my Solid pod is located at `https://jas0n.solid.community`. The Solid project has defined a set of protocols that determine how Solid pod servers and Solid client applications interact. The details of each of these components are discussed below.

2.1.1 Server

The decentralized nature of Solid originates from the ability for a user to host their Solid pod anywhere that is accessible on the Web. For instance, a pod can be hosted

on a Raspberry Pi[1] in the user's home, on a virtual machine rented from a data center by the user, or on a multi-user pod provider that is professionally managed by an organization; the possibilities are not limited in any way, and each user has the ability to change where their pod is hosted at any time without consequence. This is in contrast to existing platforms such as Facebook which store all user data in a centralized database, leaving users without any control over how their data is managed. Regardless of where a pod is hosted, that pod must communicate with clients according to well defined protocols. The Solid project maintains an open source reference implementation of a pod server called `node-solid-server` that is freely available for anyone to use to host their own Solid pods [9]. Additionally, the Solid project also maintains a list of third party pod providers, which are organizations that host Solid pods for users on infrastructure that the organization manages [10].

A user's pod acts as a repository of data for that user and their Solid applications. Data in a Solid pod is arranged as a file system, where each directory and each file has permissions that specify which Solid users can perform certain actions, such as reading or modifying a file. Solid pods use the Web Access Control specification, which is a decentralized cross-domain access control system that enables the owner of a pod to have precise control over who has access to each part of their pod [11]. This use of strict access control is how Solid secures a user's data and enables users to control exactly who they choose to share their data with.

2.1.2 Client

The security benefits of storing data in a Solid pod would be meaningless without a way to generate and apply that data. That is the purpose behind the `solid-auth-client` JavaScript library, which allows Web applications running in a user's browser to read and write data residing in that user's pod [12]. This is possible through the use of the WebID-OIDC authentication specification created by Solid [13]. This specification is based on the OAuth 2.0 and OpenID Connect protocols, which are

[1] A Raspberry Pi is a credit card–sized computer used for education and digital making. More information can be found at `https://www.raspberrypi.org/documentation/`.

open standards for identify verification on the Web [14, 15]. WebID-OIDC differs in that it is designed to operate in a fully decentralized ecosystem like Solid, where there are many different identify providers and resource servers that must work together.

In order to authenticate a user, a Solid Web application offers users the ability to sign in using Solid, similar to how websites like Google and Facebook can act as OAuth identity providers to third party websites. However, since Solid is a distributed ecosystem, there is no single Solid server that can authenticate every user. Rather, users must also specify the location of their pod provider that will be able to perform authentication for them. Once a user specifies and is authenticated by their pod provider, the application receives session data for the current user. This data includes the user's WebID as well as cryptographic tokens that are used by `solid-auth-client` to prove the identify of the user and application when accessing resources on a Solid pod.

When a user's pod is the only storage destination for an application, that user is able to control all of the data that the application has created. This gives the user the ability to easily export and import application data at any time between any two compatible applications, thereby avoiding walled garden ecosystems that deprive the freedom to choose preferred applications or devices. For example, consider the situation where two messaging applications on Solid use the same ontology to store data (ontologies are described in Section 2.2.2). As a result, if a user decides they no longer prefer their current messaging application, they can switch to the other one without losing any contacts, message history, or existing conversations. Furthermore, this change would not even be noticeable to other users in their conversations. When users are able to choose applications based on their personal preferences without consequence, everyone benefits.

2.2 Linked Data

Linked Data is a set of principles that can be used to transform the web from a distributed file server into a distributed database [16]. Currently, it is common for

resources on the Web to reference other resources by their URI. For example, the homepage of a website written in HTML may contain links to the website's other pages, links to images, and a link to a style sheet defining the appearance of the site. This scheme allows the Web to be viewed as a graph of interconnected resources, where each node is a URI that represents a retrievable resource. This graph does not provide much information about the relationship between resources, only that they either exist or they do not. In contrast, Linked Data resources describe exactly their relationships to other resources [17]. These relationships are defined by URIs as well. Ontologies define the URIs that can be used as relationships between resources, as well as the meaning of these relationships. The Resource Description Framework is a language used to describe these concepts and their relationships [18]. Linked Data graphs can be queried using the SPARQL Protocol and RDF Query Language [19].

2.2.1 Resource Description Framework

The Resource Description Framework (RDF) is an abstract framework for creating a directed, labeled graph of information on the Web [18]. RDF documents contain a set of statements, where each statement consists of a subject, a predicate, and an object. These statements are called triples. The presence of a triple is an assertion that the subject has a one-way relationship with the object, and this relationship is described by the predicate, as shown in Figure 2-1. A predicate must be a URI, while subjects and objects may be a URI, a literal, or a blank node. A literal is a definite value, such a number, string, or date. A blank node is an embedded RDF document that contains more triples describing it. The power of Linked Data lies in the fact that each URI used as a subject or object represents a document on the Web that can be retrieved to further expand the graph of information created by the RDF document.

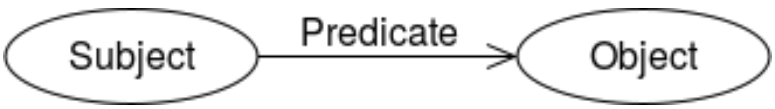


Figure 2-1: The graph of an RDF triple [18]

Serializations

The RDF specification describes the type of data that is contained in an RDF document, but not the exact format of this data. This is because RDF can be used in many different contexts, each of which might have a convenient but different way to represent RDF data. As a result there are several serialization formats for RDF, some of which are:

- RDFa, which embeds RDF triples into HTML documents [20].
- JSON-LD, which represents triples in JavaScript Object Notation [21].
- Turtle, a format designed to be compact and natural [22]. An example of a Turtle document is shown in Figure 2-2.

```
<#me> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://xmlns.com/foaf/0.1/Person>.
<#me> <http://xmlns.com/foaf/0.1/name> "Jason Paulos".
<#me> <http://xmlns.com/foaf/0.1/image> <me.jpeg>.
```

Figure 2-2: A Turtle RDF document

The remainder of this thesis will focus on Turtle, since that is the most common RDF serialization used by Solid. In Turtle, triples are formed by statements that end with a period. The Turtle document from Figure 2-2 asserts three RDF triples. The subject of these triples is the relative URI `#me`. Assuming the document is accessed from `https://jas0n.solid.community/profile/card`, the absolute URI that the subject refers to is `https://jas0n.solid.community/profile/card#me`[2]. The first triple asserts that the object `#me` conforms to the type `http://xmlns.com/foaf/0.1/Person`. This is a type from the Friend of a Friend (FOAF) ontology, discussed in Section 2.2.2. Using this same ontology, the second triple asserts the name of `#me` is `Jason Paulos`, and the last triple asserts that `https://jas0n.solid.community/profile/me.jpeg` is an image representing `#me`.

[2]This URI is my WebID, as shown in Section 2.1. In fact, Figures 2-2 and 2-3 are a subset of the triples asserted by my Solid profile.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

<#me>
  a foaf:Person;
  foaf:name "Jason Paulos";
  foaf:img <me.jpeg>.
```

Figure 2-3: An equivalent Turtle RDF document

Figure 2-2 is a basic example of three triples, but it is quite repetitive. Figure 2-3 is a more compact Turtle document that contains the exact same triples. This compact forms takes advantage of a few shortcuts in the Turtle syntax:

- A namespace can be defined using the keyword `@prefix`, followed by a name for the namespace and the URI it is describing. The name of a namespace is arbitrary, and elsewhere in the document, the namespace can be referenced using its name followed by first a colon, then an identifier. This reference is equivalent to the URI formed by the namespace's URI followed by the identifer, so `foaf:Person` is equivalent to `http://xmlns.com/foaf/0.1/Person`.

- The predicate `"a"` may be used as a shortcut for the URI `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`. This predicate states that the subject conforms to the type indicated by the object. This predicate is extremely common, so using `"a"` generally improves the readability of a Turtle document.

- If multiple triples have the same subject, they can be combined into a compound statement. This statement begins with the common subject, then the predicate and object of the first triple, followed by the predicates and objects of the remaining triples. A semicolon indicates the end of one triple in the compound statement and the beginning of another. After the final triple, a period marks the end of the compound statement.

2.2.2 Ontologies

Ontologies, also known as vocabularies, specify the predicates and classes that RDF documents can contain. Classes are types that have specific attributes or uses. An instance of a class is any RDF subject in a triple where the predicate is `http://www.w3.org/1999/02/22-rdf-syntax-ns#type` and the object is the class itself. Attributes are predicates defined by an ontology to assert relationships where the subject is an instance of a class. Ontologies are typically domain specific, and many ontologies can be used together in an RDF document.

Friend of a Friend

The Friend of a Friend (FOAF) ontology defines relationships and classes useful in a social network [23]. All of its definitions follow the prefix `http://xmlns.com/foaf/0.1/`. Figure 2-4 shows a subset of the classes and attributes defined by the FOAF ontology. Note that attributes may be applied to classes multiple times. For example, multiple triples with the same `foaf:Person` as the subjects, `foaf:knows` as the predicates, and different `foaf:Person`s as objects express that the subject knows multiple other people.

Fast Healthcare Interoperability Resources

The Health Level Seven Fast Healthcare Interoperability Resources (HL7 FHIR, or just FHIR) specification is a standard for representing and exchanging electronic healthcare records [24]. The FHIR specification defines an RDF ontology that can be used to represent electronic healthcare records [25]. This ontology follows the prefix `http://hl7.org/fhir/`. Figure 2-5 shows a subset of the classes and attributes defined by the FHIR ontology. The most relevant part of the ontology for the remainder of this thesis is the `fhir:Observation` class, which records a measurement made about a patient. Observations are extremely flexible and can represent arbitrary information, and they typically have an associated time or duration which describe when the observation took place.

FOAF ontology

- **Person**: represents a real person
  - *name* (string) the name of the person
  - *img* (Image) an image of the person
  - *age* (integer) how old the person is in years
  - *mbox* (a mailto URI) an email that the person owns
  - *page* (Document) a web page that the person owns
  - *knows* (Person) a friend of acquaintance of the person
  - ...
- **Document**: the URI of a document on the web
- **Image**: a subclass of Document; must represent an image
- ...

Figure 2-4: A subset of the FOAF ontology. Classes are **bolded** and attributes are *slanted*.

2.2.3 SPARQL

The SPARQL Protocol and RDF Query Language is a general-purpose query language for RDF, and it has a similar syntax to Structured Query Language (SQL) [19]. SPARQL queries can filter and select specifc parts of triples from an RDF document, making it possible to form powerful and precise queries. Figure 2-6 shows an example of a simple SPARQL `SELECT` query that can be used to find the title for a book. In addition to querying triples, SPARQL can also be used to modify the set of triples in an RDF document, through the use of `INSERT` and `DELETE` queries [26]. This functionality is useful for application which store state as RDF data, as it allows an RDF document to be used similarly to a SQL database. In Section 2.2 I mentioned that one of the benefits of using Linked Data is that the Web becomes a distributed database of information. SPARQL offers the ability to query and add new information to that database.

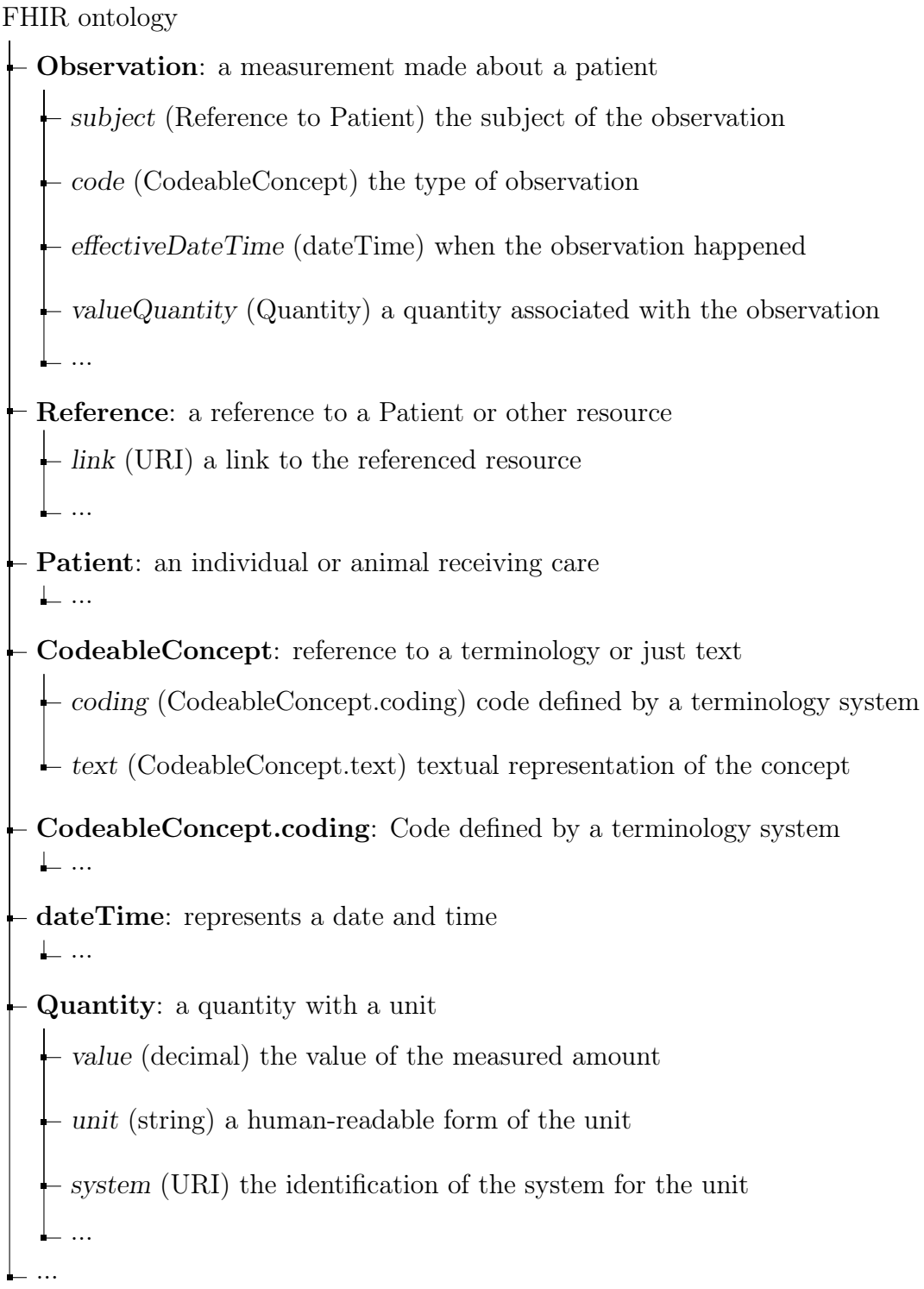


Figure 2-5: A subset of the FHIR ontology. Classes are **bolded** and attributes are *slanted*.

RDF triple:

```
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title>
    "SPARQL Tutorial" .
```

SPARQL query:

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title>
      ?title .
}
```

Result:

```
title="SPARQL Tutorial"
```

Figure 2-6: A SPARQL select query

2.3 React Native

React Native is an open source framework created by Facebook for building cross-platform native applications [27]. React Native allows developers to write iOS and Android application using JavaScript and React, another open source framework created by Facebook for creating user interfaces (UI). React Native applications have an embedded JavaScript runtime where this cross-platform JavaScript code is executed, but unlike traditional Web applications, the user interface of React Native applications is rendered with native UI components, not HTML. As a result, React Native applications are often indistinguishable from completely native applications. Since core application logic is often written in JavaScript for React Native apps, the framework exposes many native APIs and components to the JavaScript runtime. Additionally, to increase compatibility with JavaScript applications and libraries that run in web browsers, the framework also supports a limited number of Web APIs in its JavaScript runtime as well.

In addition to the native APIs that React Native exposes by default, the framework also allow applications to incorporate 3rd party native modules, which are libraries written using the platform's native environment and APIs, such as Objective-C in iOS

or Java in Android [28, 29]. Native modules expose JavaScript bindings to their native code, allowing it to be called from the JavaScript runtime. This allows React Native apps to access all resources available to traditional native apps, thereby removing any limitations from a React Native app.

2.3.1 Web APIs

React Native provides some existing browser APIs to its JavaScript runtime. Among them are:

- HTTP Requests: React Native provides support for the XMLHttpRequest API and the Fetch API, which are existing standards for executing network requests from JavaScript [30, 31]. This is vital for compatibility with browser JavaScript, as many existing browser application and libraries make extensive use of these APIs.

- Sockets: React Native supports the WebSocket Protocol [32]. This protocol allows applications to open a two way communication channel with a WebSocket server. This channel can stay open indefinitely and can encode text and binary data.

Notice that React Native does not support the Document Object Model (DOM) API. This API defines an extensive list of types, functions, and events related to the rendering and structure of Web pages [33]. This API is not compatible with the native UI components React Native uses to render applications. This means that any user interface code in an existing web application is incompatible with React Native. Therefore, to port most web applications to React Native would require at least rewriting all of the code related to its user interface.

Chapter 3

Related Work

3.1 Healthcare Solutions

This section focuses on current attempts at distributed medical records systems with access control and how they compare to the distributed storage guarantees offered by Solid.

3.1.1 MedRec

MedRec is a proof-of-concept decentralized record management system that uses an Ethereum blockchain as a means of keeping track of electronic health records [34]. The system is designed to integrate with institutions that already have a large number of medical records to allow them to exchange records securely on behalf of patients, such as when a patient switches to a new doctor. A blockchain is used to provide an immutable record of where data is located and who has access to it, and smart contracts are used to enforce cooperation between institutions; the records are still stored in institutions' databases. Patients are given access to their records through the use of a custom client, which downloads a copy of the blockchain to look up where their records are located, then queries the institutions that house the requested records. MedRec solves the complex problem of getting separate medical institutions to exchange health records quickly and securely, yet the records are still stored by

institutions, not the patients. As a result, if an institution's database becomes unavailable or compromised, it could result in data theft or even complete data loss. This also leaves institutions as a large target for potential data breaches. In contrast, Solid offers users the ability to store their data in their own pods, which are not tied to large institutions.

3.1.2 Medicalchain

Medicalchain is a decentralized medical platform [35]. It offers the ability to store and manage medical records, as well as a system to bill patients for medical services. It uses operates using two blockchains: the first is a Hyperledger Fabric blockchain used to store medical records, and the second is an Ethereum blockchain used for billing and smart contracts. The storage blockchain stores encrypted medical records and offers a way for the record's owner to share encryption keys with others, such as doctors or family members. There is also a way to revoke a user's access, which is done by re-encrypting records with a new key. This means that past records may always be available to someone even if they were only given access for a brief period of time. In contrast, access control in Solid is able to stop others from accessing data immediately after the user has decided to stop sharing with them. Medicalchain is additionally complicated by the MedToken, which is a cryptocurrency used in Ethereum smart contracts for patients to pay for medical services. The use of a cryptocurrency may be a high barrier to entry for patients and hospitals to adopt the Medicalchain system.

3.2 Distributed Data Platforms

This section focuses on distributed platforms for data storage and how they compare to Solid.

3.2.1 InterPlanetary File System

The InterPlanetary File System (IPFS) is a distributed data storage network that prioritizes data availability and immutability [36]. This is achieved through redundant data storage and content hash verification. IPFS also provides the ability for stored data objects to link to other objects, thereby creating the ability to form a large network of related information. However it is important to note that data objects can be distributed among any number of nodes in the IPFS network. This redundancy greatly increases the availability of objects, but this means that updating data is more complicated than simply editing and saving an existing document. In contrast, Solid pods are the single source of truth for a user's data; this makes updating data extremely simple. Solid also offers a way for clients to subscribe to real-time notifications of changes to documents using WebSockets, so that updates can immediately be noticed. The same cannot be said of IPFS at this time.

3.2.2 Blockstack

Another framework for creating decentralized applications is Blockstack [37]. Blockstack applications use personal data lockers, similar to Solid pods, to store users' data. The user has complete control over where their locker is hosted, and they have the ability to move their locker and add or remove apps from accessing their locker at any time. A blockchain is used to record references to users' lockers, which facilitates new users joining the system and users changing where their lockers are located. Unlike Solid pods which store data in plaintext, Blockstack has support for full encryption of the contents of lockers. In fact, this is how access control is enforced: plaintext data is readable by everyone, and data must be encrypted for it to be private. This may incur performance issues, especially on mobile devices, as searching for specific objects or files could require an application to download every object from a locker, decrypt them, then verify if the documents contain what the application is searching for. In contrast, because Solid uses a robust permissions system in place of data encryption for access control, it allows for efficient analysis and filtering of remote

files for clients with appropriate access rights through the use of SPARQL queries. In terms of interoperability features, Blockstack differs from Solid in that it does not prioritize a way to supported Linked Data or similar schemes. Collaboration between groups of users is possible through the use of a service called Radiks, however this introduces a centralized component to the system. In these groups, one administrator has complete control over membership, and each group exists in a completely isolated environment. This means that if a user wishes to share the same file with multiple groups, they need to copy the file into each group's space and encrypt it using each group's public key. Because Solid does not rely on encryption to regulate access control, it has no such restrictions.

3.3 Mobile Linked Data Applications

This section explores the use of Linked Data in past mobile applications and how they compare to Solid mobile application.

3.3.1 DBPedia Mobile

DBPedia Mobile is a client application for mobile devices that allows users to explore geospatial Linked Data objects on a map [38]. Locations for points of interest are obtained from DBPedia, which contains a collection of Linked Data information gathered from Wikipedia. The application makes use of a device's GPS sensors to show a map of the nearby area and any points of interest from DBPedia which fall in that area. The user can filter which type of locations they are interested in seeing and the map will show new points of interest that match that filter. By selecting a specific point of interest, more details are shown about it, including a description and pictures. All of the data in DBPedia Mobile is populated with Linked Data obtained a remote server that executes SPARQL queries. DBPedia Mobile shows how the utility of a data set increases greatly when it is easier to access. It is a great example of how mobile devices and sensors bring utility and convenience to extensive Linked Data data sets. However, DBPedia Mobile runs in a mobile device's Web browser,

and the only way it can obtain sensor data is when a companion application passes in readings from the device's sensors when starting DBPedia Mobile. In contrast, Solid mobile applications are in a better place to benefit from the combination of local sensors and Linked Data because they are able to access device APIs directly at any time.

3.3.2 RDF on the Go

RDF on the Go is an Android application that implements a full-fledged RDF datastore and SPARQL query processor [39]. RDF on the Go showcases this by showing a map of points of interest that are stored in its internal RDF datastore. Like DBPedia Mobile, the user's location is obtained through the device's GPS sensors so that nearby points of interest are displayed. But unlike DBPedia Mobile, querying for nearby points of interest is done directly on the device. RDF on the Go also allows users to directly compose and execute SPARQL queries locally. This is particularly useful because it minimizes the application's reliance on central servers to operate, as well as allows the application to continue to work without network access. Additionally, RDF on the Go shows that mobile device processors are powerful enough to manipulate and contribute to RDF datastores. Many years later, Solid mobile applications take this idea even further by having a ready-to-use remote location to export data: Solid pods. Both RDF on the Go and Solid show that since mobile applications are so prevalent and have a range of onboard sensors, they have the potential to greatly increase the content and quality of existing Linked Data data sets.

Chapter 4

System Overview

My main contributions in this thesis are:

1. Extended Solid libraries to support the development of Solid apps in React Native.

2. Developed functionality to integrate sensor data from phones and wearables into Solid and model it in FHIR RDF.

3. Created Solid Health, a proof-of-concept decentralized mobile health application.

The following sections discuss and evaluate these contributions in detail. Additionally, the source code for the Solid Health app is available in the repository `https://github.com/jasonpaulos/solid-health`.

4.1 Purpose

Solid Health is a decentralized application that can record and manage a user's health and fitness activity. Solid Health uses the Solid framework to store all health data in a user's Solid pod. However, unlike previous Solid applications, Solid Health does not run in a user's browser, but rather is installed as a native application on a user's phone. As a result, Solid Health can access platform-specific APIs and interact with

low-level system components that would otherwise be impossible to do from a web browser. This enhanced execution environment makes it possible for Solid Health to gather health and fitness data from the sensors in a user's mobile device and any connected smartwatches or fitness trackers.

The purpose of the Solid Health application is to provide a decentralized way to record and manage a user's health and fitness activity. All data collected from a user is stored on that user's Solid pod according to the FHIR RDF specification for interoperability and ease of use [25]. This is consistent with the central ideas of Solid, as it gives the user complete control over the location and accessibility of their data. Additionally, since the fitness data is stored in an open format, Solid Health avoids vendor lock-in so individual users are free to incorporate other apps or service that read or modify their fitness data, unlike the proprietary formats of other mobile fitness services.

Currently Solid Health is able to collect the number of steps walked per day, the distance walked per day, and the heart rate of a user. Since this data is generally unable to be collected from existing Web APIs, Solid Health must be implemented as a mobile application in order to function. As a result, it is possible for the application to collect any health or fitness data that is available to the user's device.

4.2 Environment

Solid Health has been developed to run as a mobile application on an Android phone. It has been tested on a Google Pixel phone running version 10 of the Android operating system. Android was chosen as the target platform because of its widespread use and open development standards. In contrast, the development of iOS applications must take place on macOS and requires a paid Apple developer account to test on devices.

I have chosen to use the React Native framework to develop Solid Health. React Native exposes some Web APIs and allows portions of an application to be written in JavaScript. It is possible to use existing JavaScript Web libraries in React Native,

although there are many restrictions since React Native does not provide every API available in a modern web browser. Despite this, React Native serves as a great starting point to using existing Solid JavaScript libraries on a mobile device.

As a result of using React Native to develop Solid Health, many of the existing JavaScript libraries used by Solid Web applications can be used directly by a mobile application. This was preferable to using Java or Kotlin to develop a native Android application, since I would have had to rewrite all of the existing Solid libraries in a new language, and each Solid Web application that wanted to be ported to a mobile application would also have to be rewritten in another language.

4.3 Functionality

This section describes the three major functions that make up the Solid Health application.

4.3.1 User Authentication

It is essential for any Solid application to obtain permission to read and write data to a user's pod. This is done through the Web-OIDC authentication protocol, in which the Solid application asks the user to specify their pod provider, and then requests permission to read and write data to the user's pod from this provider. This is often done by redirecting the user to their pod provider's website, having the user log in with their pod provider credentials, then confirming that they want to trust this Solid app to access their pod. Once that is complete, the pod provider redirects the user back to the application and providers the WebID of the user and tokens that can be used to perform that actual reading and modifying of resources on the pod.

Problem

Solid provides a library called `solid-auth-client` that allows users to authenticate with their Solid provider and grant an application access to their pod [12]. Unfortunately, the authentication mechanisms used by `solid-auth-client` are unable to

be used in a React Native environment, since the library relies on opening a popup browser window where the user is redirected to their pod provider. Furthermore, it is not advisable to modify an existing OAuth library to comply with Web-OIDC because the this protocol is under active development and may change at any time. Therefore, it is beneficial to be able to use the original `solid-auth-client` library so that future updates are incorporated into applications as easily and quickly as possible.

Workaround

In order to authenticate users from a mobile application, I chose to implement authentication through a proxy website. In this scheme, when a user wants to sign in to Solid Health, the application opens a web browser directed to a proxy website. For Solid Health, the proxy website is `https://jasonpaulos.github.io/solid-health/`. This website uses the `solid-auth-client` library to authenticate as a normal website would. Once authentication is successful and the user's session information, including their WebID and authentication tokens, is available to the proxy website, the website redirects the user back to Solid Health using deep linking[1], and the session information is included in this redirect.

Consequences

This method of authentication works with an unmodified version of the `solid-auth-client` library running on the proxy website, which makes any future updates easy to incorporate. However, it does have some drawbacks. Mainly, every app must have a hosted proxy website in order to perform authentication. This is contrary to the decentralized nature of Solid, although because of free static website hosting by GitHub[2] it is not as much of a burden as it would have been years ago.

[1]Deep linking is the ability for web pages to open links in a supported native application. See `https://developer.android.com/training/app-links` for more details.

[2]GitHub offers free static website hosting for public repositories through a feature called GitHub Pages. See `https://pages.github.com` for more details. This is how the proxy website for Solid Health is hosted.

A larger drawback is the issue of security. There is a separate set of OAuth best practices for native applications [40]. These recommendations are put in places to ensure that attackers cannot gain access to sensitive information during the authentication redirect back to the application. Unfortunately, this workaround does not follow these guidelines. As an additional measure, security could be increased by the application generating a public and private key pair, then including the public key in the redirect to the proxy website. Once the proxy website obtains the user's session information, it could encrypt it with the public key before sending it to back to the application. This would prevent any other apps intercepting the redirect with session information, but does not prevent other apps from arbitrarily redirecting to the app and injecting unwanted session information.

4.3.2 Data Collection

The most important feature of Solid Health is the ability to use Android APIs to collect and transmit fitness data to a user's Solid pod. This section discusses how the fitness data is collection from the user, transmitted to the user's pod, and the format in which it is stored. I have tested Solid Health's data collection abilities with a Fossil Sport Smartwatch running Wear OS by Google. This watch has GPS and heart rate sensors that are able to produce data that is eventually recorded as health observations in the Solid Health app.

Collection

Solid uses the Google Fit APIs to access fitness information for the current user of the device. The Google Fit APIs are part of Android's Google Play services and are supported in Android 2.3 and higher [41]. The APIs are able to collect raw data from a variety of sensors on and off the device, then process this data into concrete fitness actions and store it on Google's servers, as shown in Figure 4-1. Solid Health uses the `react-native-google-fit` native module, which exposes several Google Fit APIs as JavaScript functions [42]. The Solid Health application is able to call these methods

to read the number of steps taken and the distance walked by the user on a given day, as well as the user's heart rate values for a time range.

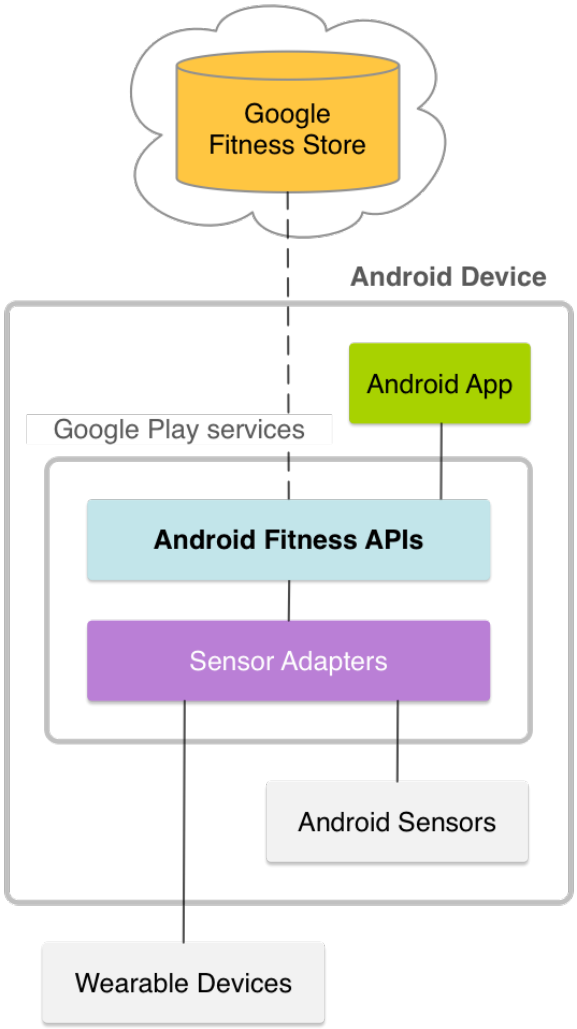


Figure 4-1: Google Fit architecture [41]

Google Fit APIs collect fitness data for a user in several ways, as described below:

- The user's phone detects fitness activity through its sensors. This can include accelerometer and GPS sensors which detect motion when a user is walking or running. This data is then used to calculate how many steps the user has taken in a specific time frame as well as the actual distance travelled.
- A dedicated fitness device records activity. If the user has paired a device with additional sensors, such as a smartwatch, that device's sensors will be used

to record fitness data as well. Many smartwatches have an accelerometer or even GPS senors that can provide more information for the Google Fit APIs to estimate the user's movements. Additionally, some smartwatches have the ability to measure heart rate, and when possible this data is also collected by the Google Fit APIs.

- One of the user's applications submits fitness records, with the user's permission. For instance, the user might use an exercise app to log runs taken without their device.

- The user manually enters fitness records through the official Google Fit app or website.

Since most phones have senors capable of measuring movement, Solid Health is able to collect the number of daily steps taken and distance travelled for most users. For users who also have a wearable device capable of measuring heart rate, Solid Health is able to collect those heart rate measurements as well.

Transmission

Once the Google Fit APIs have collected fitness data about a user, the Solid Health application must query the APIs to read this data.

Currently, Solid Health queries the APIs for new data every time it is launched. It asks the Google Fit API for records within the current month, and if the application sees that there is data it has not already stored in the user's pod, it converts the data to a valid RDF document and uploads it so the user's pod. Once the data for that month has been uploaded, the app asks the Google Fit APIs for records from the previous month and then uploads any new data from that month, continuing until it finds a month with no new data. The next section describes the exact format and location of the uploaded data in a user's pod.

Since Solid servers require the use of TLS, all pod locations should be only accessible through HTTPS. As a result, uploading the fitness records to a user's Solid pod

is secure because it happens in an encrypted channel. However, Solid Health cannot guarantee the security of any data transferred from a smartwatch to the user's Android device. It is possible for the data to be transferred over Bluetooth, WiFi, or a cellular connection. Bluetooth specifically may be vulnerable to snooping and Man-in-the-Middle exploits. Unfortunately Solid Health has no way to secure this transfer. It can only trust the Google Play services in the Android operating system to perform this transfer as securely as possible. However, even if such a breach were to occur, an attacker would only become aware of data currently being sent from the smartwatch; the user's entire fitness history would not be compromised.

Storage

Once the Solid Health application has collected fitness data, it must convert the data to RDF and it must find the correct location in the user's pod to store the data.

The data is converted to RDF so that, like many other Solid resources, it can add to the user's Linked Data network. For the highest level of interoperability with other healthcare applications and data, Solid Health serializes fitness data as objects that adhere to the FHIR ontology. Specifically, every point of data that the application collects is made into an instance of the RDF class with URI `http://hl7.org/fhir/Observation`, as required by the FHIR specification. As a result, other healthcare applications that understand FHIR `Observation`s will be able to read and operate with the fitness data created by Solid Health. This makes the utility of the application almost limitless, as there are no restrictions on how the user is able to share or consume their fitness data outside of the application.

Figure 4-2 shows an example of single fitness record encoded into a FHIR RDF `Observation`. The record measures the distance walked on March 3rd, 2020, and has a value of nearly 3457 meters.

Once the data has been serialized in the proper format, Solid Health must save the data to a specific location in the user's pod. Since users' preferences may vary with regard to where they want their health information to live on their pod, Solid Health consults the user's Type Index Registry to determine where it should up-

```
@prefix : <#>.
@prefix fhir: <http://hl7.org/fhir/>.
@prefix rd: <http://loinc.org/rdf#>.
@prefix XML: <http://www.w3.org/2001/XMLSchema#>.
@prefix c: </profile/card#>.

:distance_20200309
    a fhir:Observation;
    fhir:nodeRole fhir:treeRoot;
    <http://hl7.org/fhir/Observation.code>
            [
                <http://hl7.org/fhir/CodeableConcept.coding>
                        [
                            a rd:41953-1;
                            <http://hl7.org/fhir/Coding.code>
                            [ fhir:value "41953-1" ];
                            <http://hl7.org/fhir/Coding.display>
                            [ fhir:value "Distanced walked" ];
                            <http://hl7.org/fhir/Coding.system>
                            [ fhir:value "http://loinc.org" ];
                            fhir:index 0
                        ];
                <http://hl7.org/fhir/CodeableConcept.text>
                [ fhir:value "Distanced walked" ]
            ];
    <http://hl7.org/fhir/Observation.effectiveDateTime>
    [ fhir:value "2020-03-09"^^XML:date ];
    <http://hl7.org/fhir/Observation.status> [ fhir:value "final" ];
    <http://hl7.org/fhir/Observation.subject> [ fhir:link c:me ];
    <http://hl7.org/fhir/Observation.valueQuantity>
            [
                <http://hl7.org/fhir/Quantity.code> [ fhir:value "/d" ];
                <http://hl7.org/fhir/Quantity.system>
                [ fhir:value "http://unitsofmeasure.org" ];
                <http://hl7.org/fhir/Quantity.unit> [ fhir:value "m/d" ];
                <http://hl7.org/fhir/Quantity.value>
                [ fhir:value 3456.911376953125 ]
            ].
```

Figure 4-2: A FHIR RDF record for distance walked

load health records. The Solid Type Index Registry is a map of resources types to locations in the user's pod [43]. The resource type that Solid Health looks for is `http://hl7.org/fhir/Observation`, since the data it creates always has this value as its class type. If this resource type is mapped to a file location, then Solid Health will append its health data to the file through the use of a SPARQL-based HTTP PATCH request, which is a SPARQL `INSERT` or `DELETE` query that runs in a user's Solid pod. If the resource type is mapped to a directory location, then Solid Health will append its health data to the file `fitness.ttl` within that directory, creating it if it does not exist. If the resource type is not registered in the Type Index Registry, then Solid Health will add it to the registry with the location being the folder `/private/health/`. The application will also create this folder as well as a file called `fitness.ttl` inside of the folder.

Since there are currently no best practices with regard to storing health records in Solid pods, I created the above approach specifically for the Solid Health application. I believe this approach is reasonable, since multiple applications following it can coexist without losing any data, as files are never overwritten and existing triples are never deleted. Since SPARQL-based PATCH requests are executed on the server, appending records happens atomically and data races are not a concern. Additionally, since this approach allows the user to customize the location of their health records in their pod, I believe users will find it reasonable as well.

4.3.3 Data Sharing

The `Observation`s that Solid Health stores in a user's Solid pod are, by default, viewable only by the user. However, because these observations are in a Solid pod, users are able to take advantage of Solid's data sharing features. This means that users can give trusted individuals such as coaches, doctors, or family members permission to read their Solid Health observations. Once permission has been given, those permitted will see the latest observations as soon as they are uploaded. Additionally, read permission can be easily revoked from anyone at any time, allowing the user to have the most control possible of their data. It is even possible to give users the permission

to write observations as well, in case two users want to store their fitness data in a single combined location. Thanks to the social capabilities of Solid, all of these scenarios are possible without modifying or reconfiguring the Solid Health application.

Furthermore, other users are not forced to use the Solid Health app in order to view shared observations. They are free to use any Solid application or website that is able to understand FHIR `Observation`s. The choice to store fitness data in an open format allows Solid Health's data to be as interoperable and impactful as possible.

4.4 Evaluation

In this section, I evaluate the contributions of this thesis:

1. Extended Solid libraries to support the development of Solid apps in React Native.

2. Developed functionality to integrate sensor data from phones and wearables into Solid and model it in FHIR RDF.

3. Created Solid Health, a proof-of-concept decentralized mobile health application.

Section 4.4.1 evaluates contributions 2 and 3, and Section 4.4.2 evaluates contribution 1 by converting an existing Solid browser application to a mobile application.

4.4.1 Solid Health Features

This section evaluates contributions 2 and 3 of this thesis by showcasing the features of the Solid Health app.

Figure 4-3 shows the many different screens of the Solid Health app and how the user navigates between them. Each screen and its features are described below:

A. This page is displayed when no user is logged into Solid Health. When the app is first installed, this is what users see. Pressing "Sign in with Solid" will open

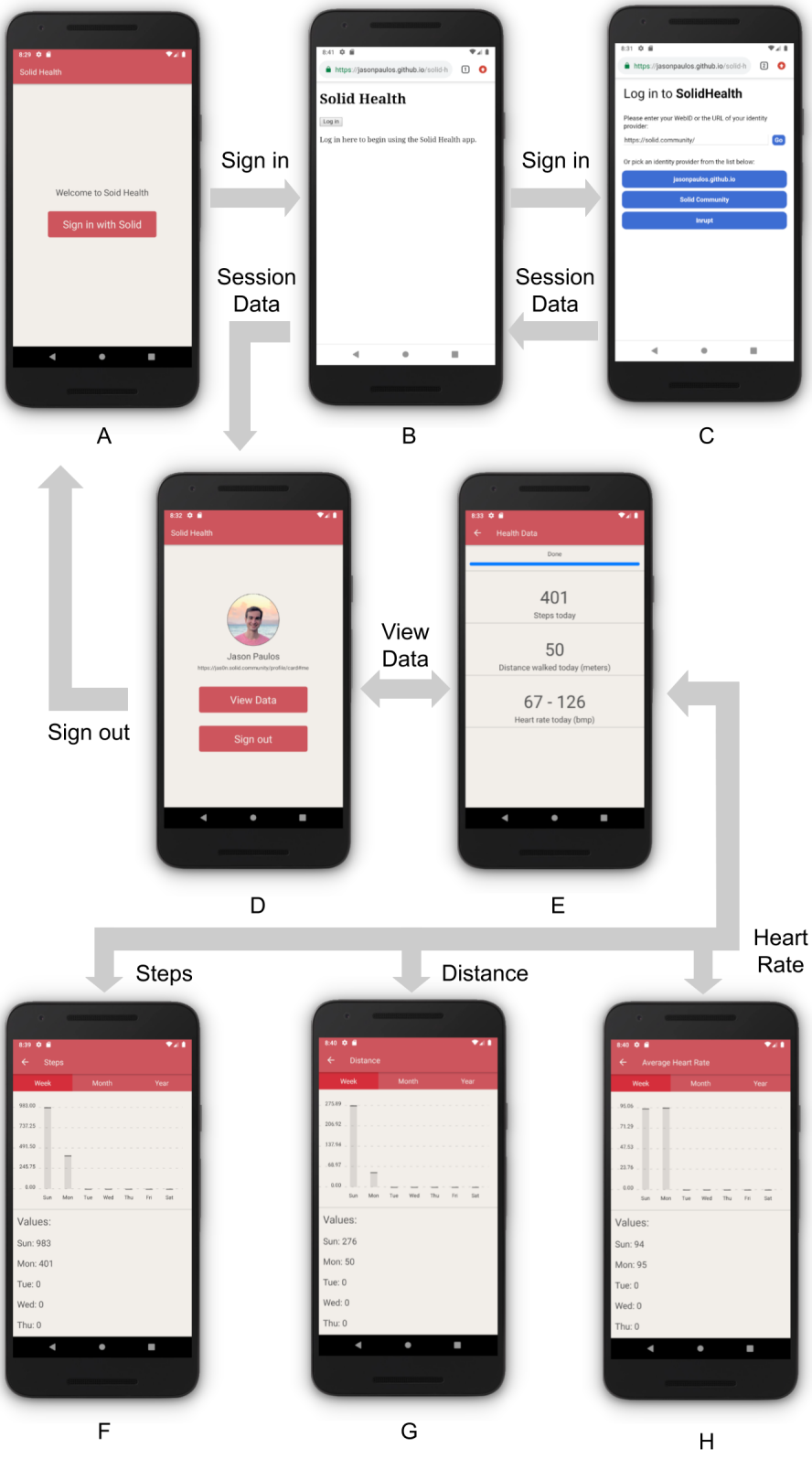


Figure 4-3: Solid Health application screens and features

the proxy website, `https://jasonpaulos.github.io/solid-health/`, which is screen B in the figure.

B. This is the proxy website opened in the device's default web browser. It runs an unmodified copy of the `solid-auth-client` JavaScript library. When the user presses log in, the Solid authentication popup opens, which is screen C in the figure. When the popup finishes authenticating the user successfully, it opens the Solid Health app again using a deep link which contains the Solid session data as a parameter. This session data includes the user's WebID and authentication tokens used to create subsequent requests to Solid pods.

C. This is the popup authentication screen provided by `solid-auth-client` [12]. It asks the user for the location of their pod provider, then authenticates the user with that provider. After authentication is successful, the popup screen sends the obtained Solid session data to screen B and closes.

D. This is the home page of the app when a user is logged in. The user's picture and name are fetched from their profile and displayed above their WebID. I chose to display the user's WebID as an additional security measure to make it clear if the wrong account is being used, either as an accident or as a data theft attempt. From here, the user can navigation to the data summary screen, E, or sign out of the application and return to screen A.

E. This is the data summary screen. It shows the user's steps, distance, and heart rate range for the current day, if available. From here, each of the three statistics can be pressed to open a data history screen, which is one of screens F, G, or H, depending on which record type is selected. The user can also press the back button in the app's header or press the physical Android back button to return to the home screen. Additionally, the top of this screen displays synchronization information about the user's data. In Figure 4-3, it displays a full loading bar and the message "Done" to indicate that all syncing has finished. When the app has received new data from Google Fit that it must sync with the user's

pod, the blue loading bar is updated in real time to display the percent of new data that has been uploaded so far, and the message will change to indicate that new data is being uploaded to a Solid pod.

F. This is the data history screen for steps. When it is first opened, it shows the user's step history for the current week. The top half of the screen shows a bar graph of the current week's history, and the bottom half shows the numerical values for each day. Additionally, the buttons labelled "Month" and "Year" can be pressed to change the period of data being shown. When changing to a monthly view, the number of total steps taken in each week of the current month are shown on this screen. When changing to a yearly view, the number of total steps taken in each month for the past 12 months are shown on this screen. The user can return to the data summary screen by pressing the back button.

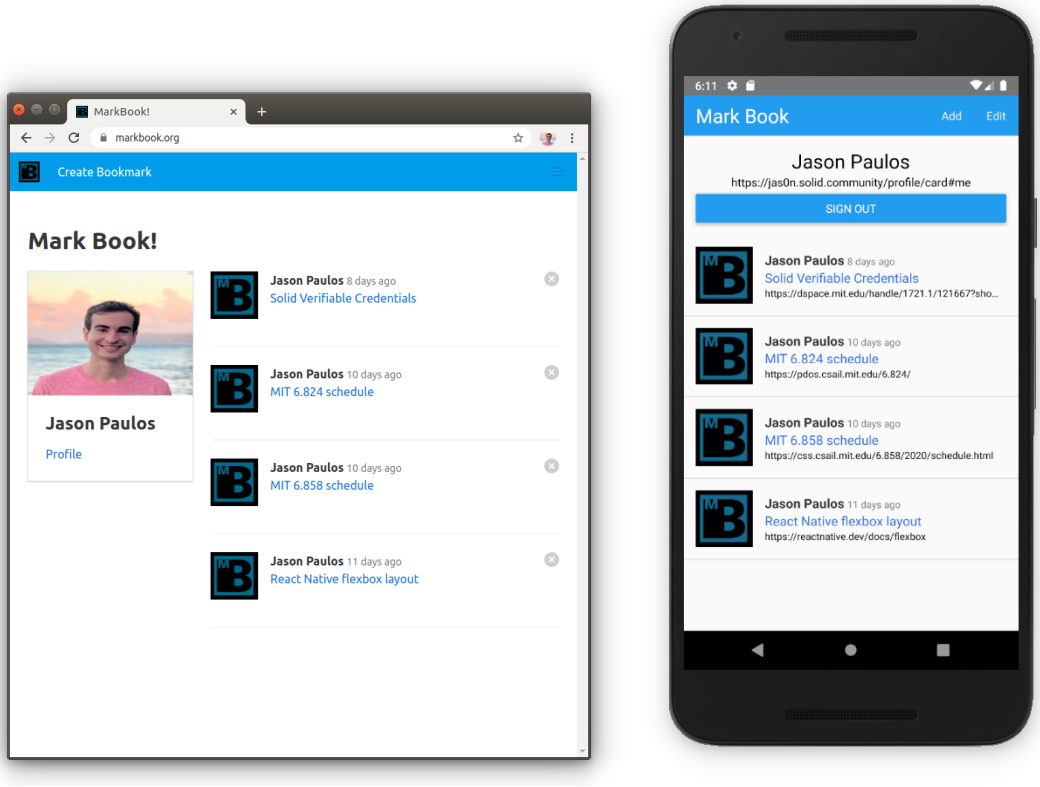
G. This is the data history screen for distance, where values are shown in meters. Like screen F, this screen shows a bar graph and numerical values for the current week, month, and year. The user can return to the data summary screen by pressing the back button.

H. This is the data history screen for heart rate. It is similar to screens F and G, except it displays the average heart rate for each period that can be viewed. The user can return to the data summary screen by pressing the back button.

From Figure 4-3, it is clear that the Solid Health app offers enough features to be a useful app. In addition to displaying the user's fitness history for three different metrics, this history is kept in sync with data on the user's pod and is updated every time the application is opened. As a result of syncing fitness records obtained from mobile sensors to the user's Solid pod, it is now possible for other Solid applications to read and understand this data since it is stored according to the FHIR RDF standard. This makes it possible to create Solid Health companion applications in the Solid ecosystem that consume this fitness data in order to offer additional benefits.

4.4.2 Porting an Existing App

In order for the Solid Health app to run on a mobile device, I had to extend some Solid JavaScript libraries to be compatible with React Native. This section evaluates this contribution by applying the steps I took to make Solid Health work in React Native on Android to an existing Solid web application, Mark Book.



(a) Browser application (b) Mobile application

Figure 4-4: Mark Book applications for browser and mobile

Mark Book is a Solid web application that manages a user's list of bookmarked websites [44]. The list of websites is stored in an RDF document on the user's pod. Mark Book is relatively simple, since it consists of only a single page of HTML with embedded JavaScript. When creating a mobile version of this app, I was able to reuse large portions of the JavaScript data fetching and uploading code, however I had to recreate the entire user interface for the app using React Native components. Figure 4-4 shows screenshots of the preexisting Mark Book browser application and

the mobile application I created for it. Since both application store their list of bookmarks in the same place in user's Solid pod, this list is always up to date in both applications.

The code for the mobile application is available in the repository `https://github.com/jasonpaulos/markbook-app`. An Android APK application bundle is also available to download from `https://github.com/jasonpaulos/markbook-app/releases` and install on a supported device.

It is important to note that while Mark Book was able to be converted to a mobile application without much hassle using the mechanisms I had already created for Solid Health, the same cannot be said for every Solid web app. This is because I have made modifications and created workaround for the `solid-auth-client` and `rdflib` libraries only [12, 45]. These two libraries are essential in many Solid applications, but applications are free to use other JavaScript libraries as well, many of which are not compatible with React Native. So while my methods work to create mobile applications using those two libraries, converting other existing Solid apps may require altering their dependencies to work with React Native as well.

Chapter 5

Future Work

5.1 Native Authentication

As I mentioned in Section 4.3.1, the current way authentication works in the mobile Solid applications I have created has some security issues. The OAuth best practices for native applications should be followed to create a login flow that is as secure as possible [40]. Unfortunately it is currently very difficult to follow these guidelines because Solid does not offer an authentication mechanism that can be used from React Native. Either through the use of Solid libraries for Android and iOS, or by changing the current authentication workflow of `solid-auth-client` to be more compatible with React Native, the Solid project should provide a solution so that authentication is as secure as possible.

5.2 Direct Data Transmission

As I mentioned in Section 4.3.2, when an Android wearable records fitness data, it sends this data to the paired Android phone, which then eventually stores this data in the user's Google Fit account on a remote server. Ideally the Solid Health application would be able to collect fitness data without the system uploading data to Google's servers, since there are additional privacy concerns if the data resides outside of a user's pod.

5.3 Offline Support

One major difference between mobile applications and Web applications are that mobile apps can be used without internet access. However, the Solid mobile applications I have created are not usable without an internet connection. According to the Solid project, the "long term vision includes local first and a flexibility of different topologies of patch-passing sync networks. However, there are no implementations yet" [46]. The applications I have created would benefit greatly from the ability to cache information from Solid pods for offline use and to make changes to RDF documents that are later relayed to the pod when network connectivity is available.

5.4 iOS Support

I have chosen to develop mobile applications for Android for this project. Since React Native applications can also run on iOS, it is possible to make the applications I have created work on iOS as well, probably without much difficulty. However, since the Google Fit APIs are only available on Android, Solid Health would need to be modified to support Apple's HealthKit API as the source of fitness activity [47].

5.5 Extending Solid Health

The Solid Health app brings together three components that have not been combined before: the Solid ecosystem, wearable devices and sensors, and the FHIR RDF ontology. I hope that more applications and systems will be built in this intersection as well, as there are many use cases for applications to build upon the work I have done with Solid Health. For instance, an application similar to Solid Health could be created that uses federated learning[1] to train and test models with health data from multiple users.

[1]Federated learning is a machine learning technique in which multiple parties create a collaborative model without revealing any of their data points. `https://federated.withgoogle.com/`

Chapter 6

Conclusion

Data ownership and privacy are extremely important, especially with respect to healthcare data. The Solid platform provides a general decentralized application framework capable of offering these features. In this thesis, I investigated the ability for Solid to act as a decentralized manager of health and fitness data. My contributions include extending Solid libraries to support the development of Solid applications in React Native, developing functionality to integrate sensor data from phones and wearable into Solid and model it in FHIR RDF, and creating Solid Health, a proof-of-concept decentralized mobile health application. I have evaluated these contributions by showcasing the features and screens of Solid Health, as well as porting an existing Solid application to a mobile app. Bringing healthcare data and devices with onboard sensors into the Solid ecosystem represents an exciting collision of opportunities that I hope will continue to be explored.

Bibliography

[1] J. Henry, Y. Pylypchuk, Searcy T., and Patel V. Adoption of Electronic Health Record Systems among U.S. Non-Federal Acute Care Hospitals: 2008-2015. ONC Data Brief 35, Office of the National Coordinator for Health Information Technology, May 2016.

[2] Healthcare data breach statistics. `https://www.hipaajournal.com/healthcare-data-breach-statistics/`. Accessed: 2020-05-09.

[3] A. Ferreira, R. Cruz-Correia, L. Antunes, and D. Chadwick. Access control: how can it improve patients' healthcare? *Stud Health Technol Inform*, 127:65–76, 2007.

[4] Solid. `https://solidproject.org/`. Accessed: 2019-12-06.

[5] Essam Mansour, Andrei Vlad Sambra, Sandro Hawke, Maged Zereba, Sarven Capadisli, Abdurrahman Ghanem, Ashraf Aboulnaga, and Tim Berners-Lee. A Demonstration of the Solid Platform for Social Web Applications. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 223–226, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee.

[6] Desktop vs Mobile vs Tablet Market Share Worldwide. `https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/#monthly-201101-202005`. Accessed: 2020-05-08.

[7] SNAPSHOT Study. `https://snapshot.media.mit.edu/info/`. Accessed: 2020-05-12.

[8] Andrei Sambra, Henry Story, and Tim Berners-Lee. Web Identity and Discovery. W3C recommendation, W3C, March 2014. `https://www.w3.org/2005/Incubator/webid/spec/identity/`.

[9] Solid. Solid server on top of the file-system in NodeJS. `https://github.com/solid/node-solid-server`. Accessed: 2020-05-03.

[10] Solid. Solid IDPs. `https://solid.github.io/solid-idps/`. Accessed: 2020-05-03.

[11] Solid. Web Access Control (WAC). https://solid.github.io/web-access-control-spec/. Accessed: 2020-05-03.

[12] Solid. A library for reading and writing to Solid pods. https://github.com/solid/solid-auth-client. Accessed: 2020-05-03.

[13] Solid. WebID-OIDC Authentication Spec. https://github.com/solid/webid-oidc-spec. Accessed: 2020-05-03.

[14] D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, RFC Editor, October 2012.

[15] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore. OpenID Connect Core 1.0 incorporating errata set 1. https://openid.net/specs/openid-connect-core-1_0.html, November 2014. Accessed: 2020-04-06.

[16] Nova Spivack. The Semantic Web, Collective Intelligence and Hyperdata. https://novaspivack.typepad.com/nova_spivacks_weblog/2007/09/hyperdata.html, September 2007. Accessed: 2020-05-07.

[17] Linked Data. https://www.w3.org/standards/semanticweb/data. Accessed: 2020-05-07.

[18] Markus Lanthaler, David Wood, and Richard Cyganiak. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, February 2014. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[19] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. W3C recommendation, W3C, January 2008. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[20] Ben Adida, Ivan Herman, Mark Birbeck, and Manu Sporny. RDFa 1.1 primer - third edition. W3C note, W3C, March 2015. http://www.w3.org/TR/2015/NOTE-rdfa-primer-20150317/.

[21] Markus Lanthaler, Manu Sporny, and Gregg Kellogg. JSON-ld 1.0. W3C recommendation, W3C, January 2014. http://www.w3.org/TR/2014/REC-json-ld-20140116/.

[22] Gavin Carothers and Eric Prud'hommeaux. RDF 1.1 turtle. W3C recommendation, W3C, February 2014. http://www.w3.org/TR/2014/REC-turtle-20140225/.

[23] Dan Brickley and Libby Miller. FOAF Vocabulary Specification 0.99. http://xmlns.com/foaf/spec/20140114.html, January 2014.

[24] Health Level Seven. FHIR v4.0.1. https://www.hl7.org/fhir/. Accessed: 2019-12-06.

[25] Health Level Seven and Semantic Web Health Care and Life Sciences Interest Group. Resource Description Framework (RDF) Representation - FHIR v4.0.1. https://www.hl7.org/fhir/rdf.html. Accessed: 2020-05-07.

[26] Alexandre Passant, Paula Gearon, and Axel Polleres. SPARQL 1.1 update. W3C recommendation, W3C, March 2013. http://www.w3.org/TR/2013/REC-sparql11-update-20130321/.

[27] React Native: A framework for building native apps using React. https://reactnative.dev/. Accessed: 2020-04-06.

[28] React Native iOS Native Modules. https://reactnative.dev/docs/native-modules-ios. Accessed: 2020-05-08.

[29] React Native Android Native Modules. https://reactnative.dev/docs/native-modules-android. Accessed: 2020-05-08.

[30] XMLHttpRequest Standard. https://xhr.spec.whatwg.org/, April 2020. Accessed: 2020-05-08.

[31] Fetch Standard. https://fetch.spec.whatwg.org/, April 2020. Accessed: 2020-05-08.

[32] I. Fette and A. Melnikov. The WebSocket Protocol. RFC 6455, RFC Editor, December 2011.

[33] DOM Standard. https://dom.spec.whatwg.org/, April 2020. Accessed: 2020-05-08.

[34] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman. MedRec: Using Blockchain for Medical Data Access and Permission Management. In *2016 2nd International Conference on Open and Big Data (OBD)*, pages 25–30, Aug 2016.

[35] Medicalchain Whitepaper 2.1. Technical report, Medicalchain SA, 2018. https://medicalchain.com/Medicalchain-Whitepaper-EN.pdf.

[36] Juan Benet. IPFS - Content Addressed, Versioned, P2P File System, 2014.

[37] Muneeb Ali, Jude Nelson, Aaron Blankstein, Ryan Shea, and Michael J Freedman. The Blockstack Decentralized Computing Network. Technical report, Blockstack PBC, 2019. https://blockstack.com/whitepaper.pdf.

[38] Christian Becker and Christian Bizer. Dbpedia mobile: A location-enabled linked data browser. In *LDOW*, 2008.

[39] Danh Phuoc, Josiane Parreira, Vinny Reynolds, and Manfred Hauswirth. RDF On the Go: RDF Storage and Query Processor for Mobile Devices. January 2010.

[40] W. Denniss and J. Bradley. OAuth 2.0 for Native Apps. RFC 8252, RFC Editor, October 2017.

[41] Google. Google Fit APIs for Android. https://developers.google.com/fit/android. Accessed: 2020-05-02.

[42] Stanislav Doskalenko. A React Native bridge module for interacting with Google Fit. https://github.com/StasDoskalenko/react-native-google-fit. Accessed: 2020-05-12.

[43] Solid. Solid application data discovery. https://github.com/solid/solid/blob/master/proposals/data-discovery.md. Accessed: 2020-05-03.

[44] Mark Book Mobile. https://github.com/jasonpaulos/markbook-app. Accessed: 2020-05-09.

[45] Javascript RDF library for browsers and Node.js. https://github.com/linkeddata/rdflib.js/. Accessed: 2020-05-12.

[46] Is it possible to use Solid offline (at least partially)? https://solidproject.org/faqs#is-it-possible-to-use-solid-offline-at-least-partially. Accessed: 2020-05-09.

[47] Apple HealthKit. https://developer.apple.com/health-fitness/. Accessed: 2020-05-09.