Politecnico di Torino

Politecnico di Torino

M.Sc. in Computer Engineering

Integrating Solid Project into Native Android Development

Candidate: Erfan Gholami

Supervisors: Prof. Antonio Vetro'
Ing. Yashar Pourmohammad

Academic Year 2024/25

Abstract

With the rapid growth of digital platforms, user privacy, security, and data ownership have become major concerns. The Solid project, introduced by Tim Berners-Lee, tackles these issues by separating data from applications, giving users full control over their personal information through Personal Online Data Stores (Pods). While Solid has been widely adopted in web development, integrating it into native Android applications presents several challenges due to a lack of tools, and platform limitations. This thesis explores these challenges and provides a structured solution for making Solid work seamlessly within the Android ecosystem and providing libraries to Android developers to integrate their applications into Solid.

The study begins by analyzing Solid's architecture, and the lack of existing tools that properly support Solid integration. To address these issues, a new approach is introduced: a modular framework designed to bring Solid into the Android development workflow. The solution consists of three main components:

- Solid Android API, which handles authentication (via OpenID Connect), resource management, and Data Modules.
- Android Solid Services, a dedicated app which uses background service that acts as a central hub for authentication and data retrieval, reducing the complexity for third-party apps.
- Solid Android Client, a lightweight library that developers can use to easily integrate Solid functionality into their applications without having to deal with low-level implementation details.

To demonstrate the effectiveness of this solution, an address book application was developed as a practical use case. This app securely stores and manages address book information using Solid Pods, proving that Solid can be successfully integrated into native Android development while maintaining security, privacy, and interoperability. The evaluation of this approach highlights its benefits, such as improved user control over data and reduced reliance on centralized storage, but also reveals areas for improvement, including API optimizations, and performance enhancements for mobile environments.

This research contributes to the broader effort of bringing Solid technology to Android platform by providing a concrete, working solution in Android development. Future work includes refining the API for better usability, extending support for additional Solid features, and exploring new mobile applications that can benefit from Solid pods. By enabling seamless Solid integration in Android, this study lays the groundwork for a more privacy-focused and user-controlled mobile experience.

Acknowledgment

To my brother, who we live with his alive memories

Contents

1 Introduction

In the era of data exploration and explosion, data privacy and "who access what" is becoming increasingly important. Every day, billions of users share (produce) their data on social media and other applications on their computers, phones, and other smart gadgets. The collection of these data makes a complete info pack about any individual. The majority of big-tech companies that provide these services, such as Meta, which owns Facebook, Instagram, and WhatsApp, use these data for ads and sales which brings questions such as "How much users' data is important?", "Can we manage our data on the internet?" and many others. After the exploitation of World Wide Web (WWW), many services and platforms were shaped, and because of the lack of interoperability between those platforms, users are disempowered. They cannot for example easily share Facebook photos with LinkedIn colleagues. To achieve these things, users must choose between moving their activities to one platform or duplicating their data in different places. From developers' perspective, when a developer writes apps that involve personal data, things are more complicated than they should be[1]:

- Because there is no interoperability across data systems, developers must figure out how to get access to data, even if that data already exists in other places. This usually involves integration with dozens of APIs and using custom clients and protocols to access the data.
- Apps need to have a secure identity mechanism so people can log in, which today either means relying on incompatible identity solutions offered by large vendors or deploying your own.
- Custom data models, protocols, and authorization mechanisms must be devised and used to store and manage application data. This comes with its own unique set of challenges in the form of security and compliance with legal requirements.

To ease these burdens, developers require standard interfaces for identity and authorized data access, into which multiple providers can plug in. That allows developers to no longer worry about solving secure identity and storage themselves: instead, they can rely on spec-compliant third-party services. Rather than having to integrate with specific bespoke platforms, developers can build for the standard and can responsibly reuse data from existing storage servers rather than having to set up their own. For these matters, some projects have started, mostly among the open-source community, to provide solutions on data/service decoupling and give more freedom to users on what to do with their data or where to save it. One of these tries is the Solid project which was started by Sir Tim Berners-Lee, one of the creators of the Internet network. In this chapter, these technologies have been described and talked about their importance in itself and with other ecosystems, specifically Android phones, and in the end, the topics of upcoming chapters have been described.

1.1 What is Solid project?

Solid is an open standard for structuring data, digital identities, and applications on the Web. Solid aims to support the creation of the Web as Sir Tim Berners-Lee originally envisioned it when he invented the World Wide Web at CERN in 1989.

Tim sometimes refers to Solid as "the web - take 3"[2] — or Web3.0 — because Solid integrates a new layer of standards into the Web we already have. The goal of Solid is for people to have more agency over their data. Solid solves the mentioned problems by decoupling identity, data, and applications. For aligning interfaces between these three parties, we need standards for identity and authorized data access. As a result, any application can work with any identity provider and any data storage provider. This means that users can manage their data and identity separately from the platform, service, or application they use. Also, for developers, it means that they will develop their apps independently from the identity and data providers and they would not be worried about storing and collecting data. Solid protocol is a network between IDPs, Data Storages (servers), and Clients (Applications). These decentralized secure data storages are called Pods. Entities control access to the data in their Pods. Entities decide what data to share and with whom (be those individuals, organizations, applications, etc.), and can revoke access at any time. To store and access data in a Pod, Solid-enabled applications use standard, open, and interoperable data formats and protocols. Solid has the following properties which give users and developers great value:

- The app functionality is all in the client.
- Any App can work with any Pod. The user typically chooses which Pod.
- The server does not care which app is running so long as it is authenticated and authorized.
- Changes to app functionality require just the re-release of the client and no changes to the server.

Once the Solid Protocol specification is stable, then new interoperable apps can be created by rolling out new "client-client" specs for each domain such as contacts, medical data, financial data, and so on. Any kind of data can be stored in a Solid Pod — from structured data to files that one might store in a Google Drive or Dropbox folder. What makes Solid special is the ability to store data in a way that promotes interoperability. Specifically, Solid supports storing Linked Data. Structuring data as Linked Data means that different applications can work with the same data. All data in a Solid Pod is stored and accessed using standard, open, and interoperable data formats and protocols. Solid uses a common, shared way of describing things and their relationships to one another that different applications can understand. This gives Solid the unique ability to allow different applications to work with the same data. With Solid's Authentication and Authorization systems, one can determine which people and applications can access their data. Entities can grant or revoke access to any slice of their data as needed. Consequently, entities can do more with their data, because the applications they decide to use can be granted access to a wider and more diverse set of information. And just as one can share their data with others, others can also share their data in return. This creates rich and collaborative experiences across a combination of both personal and shared data.

1.2 Impact of Solid

Solid can be analyzed from different aspects:

1.2.1 Accessibility

The fact that the user can use multiple alternative apps means that differently, abled users will be able to use apps that do the same thing in very different ways, for example, conversational apps and graphics-based apps accessing the same data. Solid Apps are mostly web apps, and so all the knowledge and importance and techniques and guidelines which apply to Web Apps and PWAs also apply to Solid Apps.

1.2.2 Security

Protecting user data in the Solid system is of utmost importance. The Solid protocol requires encryption for all transmissions, ensuring data safety during transfer. Encryption at rest and within processing also plays a pivotal role in security development. However, it's essential to emphasize that these measures are related to, yet distinct from, the network protocol. Strong authentication using keys is naturally part of the access controls to establish a foundation for trustworthy personal data storage, based on principles of confidentiality, integrity, and availability achieved within the Solid system. The Solid protocol introduces an innovative market for Pods that serves a diverse range of organizations such as commercial enterprises, nonprofit entities, and government agencies. Within this market, these organizations provide a variety of Pod products, each tailored to specific needs and preferences. Notably, these products differ not only in their functionalities but also in the levels of security they offer, catering to a wide spectrum of users with varying safety requirements. Apps like social networks, which can use Solid to store their distinct sets of sensitive data in a Pod, make a very compelling case for improving online safety. While users must ensure the safety of their Pod data and will have many options and services available to do so, the Solid approach fundamentally minimizes their vulnerability to attacks on the data stores as each Pod is isolated from every other Pod. Consequently, users on Pods face less risk, benefiting directly by isolation from the usual news of large-scale indiscriminate breaches. Moreover, a Pod setup empowers users to easily migrate their data without losing app functionality (low or no exit barriers), taking it to a safer (more representative of their needs) environment should they deem their current provider to be compromised. Privacy value, which is increasing yet currently bottled up and depressed by steep exit barriers, will greatly expand the app development market as safer data storage options are enabled by the Solid protocol.

1.2.3 Privacy

The Solid protocol upends the traditional notions of user privacy compared to the prevalent online ad-based systems of the past. Solid's architectural design prioritizes data privacy by default, marking a paradigm shift in how user information is handled. Within a Solid Pod, data is inherently private, creating a secure digital enclave where users maintain control over their information, deciding whether to share it broadly or selectively with specific groups like family or healthcare providers. Acknowledging the pivotal role of user consent in meaningful data exchanges with services, Solid systems seamlessly integrate consent management into data-sharing flows. This involves clear and user-friendly processes for users to grant permission for the processing of specific data sets by designated entities for predefined purposes. Solid further

simplifies compliance with regulations like the General Data Protection Regulation (GDPR), streamlining the development of systems that adhere to stringent privacy standards. The introduction of the ability to understand the purpose of how data is being used underscores Solid's commitment to user-centric privacy and also provides for when regulators are going to make AI safe. Users not only control data access but also gain clarity regarding the context and purpose of its usage, controlling processing even for knowledge systems, through transparent consent mechanisms, achieving a delicate balance between privacy and utility. Solid offers a measured and user-oriented approach to privacy, demonstrating that effective privacy measures seamlessly coexist with and enhance digital services. It provides a perspective where privacy is an integral aspect of the user experience without the need for repetitive, tiresome, and explicit summaries.

1.2.4 Digital Sovereignty

The ability of a user to be a first-class participant in their digital identity management, their digital sovereignty, is crucial. This need has driven a lot of investment in systems like blockchain, where everyone stores everything, or IPFS, where everyone stores a randomized fraction of other people's data. The Solid Protocol better answers this need to re-empower people, by using and extending existing web protocols and efforts.

1.3 Importance of Solid in Android Ecosystem

With the widespread of smartphones, the number of internet users and the amount of data produced had a rapid increase. Being able to connect to the Internet wirelessly, increasing in computation power of mobile phones, and providing OSs with a variety of features and APIs and many other reasons made this growth. During this phase, a lot of big tech companies started using users' data in secret such as the scandal in 2012[3] that showed Facebook used the data of 700,000 users without their consent or attempts by Google and Amazon to patent devices that listen for emotional changes in human voices[4] are examples for this fact. According to BBC research[5], 77% of people feel vulnerable to having multiple providers holding their personal information, and given that there has been a massive increase (+68%) in major data breaches[6] since 2020 and 2021 alone, this is understandable. These facts show enough the importance of data privacy on mobile phones. Also, about 72% of mobile phone users use smartphones with Android OS[7]. So, Integrating Solid project into the Android ecosystem makes a big change for developers to develop native apps, making it more familiar to users and having more opportunities for application development with Solid.

In *chapter 2*, we will talk about data formats used in Solid, Android development in general, and the problem of integrating Solid technology in native Android development. Besides that, we will have a look at Android developers' needs for Solid and expected APIs. In the end, existing tools for development have been described and analyzed.
In *chapter* **??**, the experimental solution of integrating existing libraries into Android and trying to integrate with Solid with the help of these libraries has been discussed. From this trial, we understand some parts should be rewritten for the Android ecosystem.

In *chapter 3*, the final solution with the architecture and different modules of the project with implementation details has been shown.
Chapter 4 is dedicated to developing an Address Book Android app based on results as a real-world use case and how Data modules can help us make data management more abstract for developers who want to take advantage of Solid in Android apps.
For knowing the limitations of the work in the sense of practicality and integration, *chapter 5* is the place that can be referred to. In addition, there are a lot of opportunities for development in this field in the future and for other members of the open-source community to continue afterward.
In the end, we have an overall conclusion of the work in *chapter 6*.

2 Problem Definition

2.1 Solid Architecture

Before solving the problem, we must examine Solid architecture, components, interactions, data types, standards, and other necessary details to find the best Solid solution for mobile phones. The Figure 1 shows an overview of the Solid ecosystem.

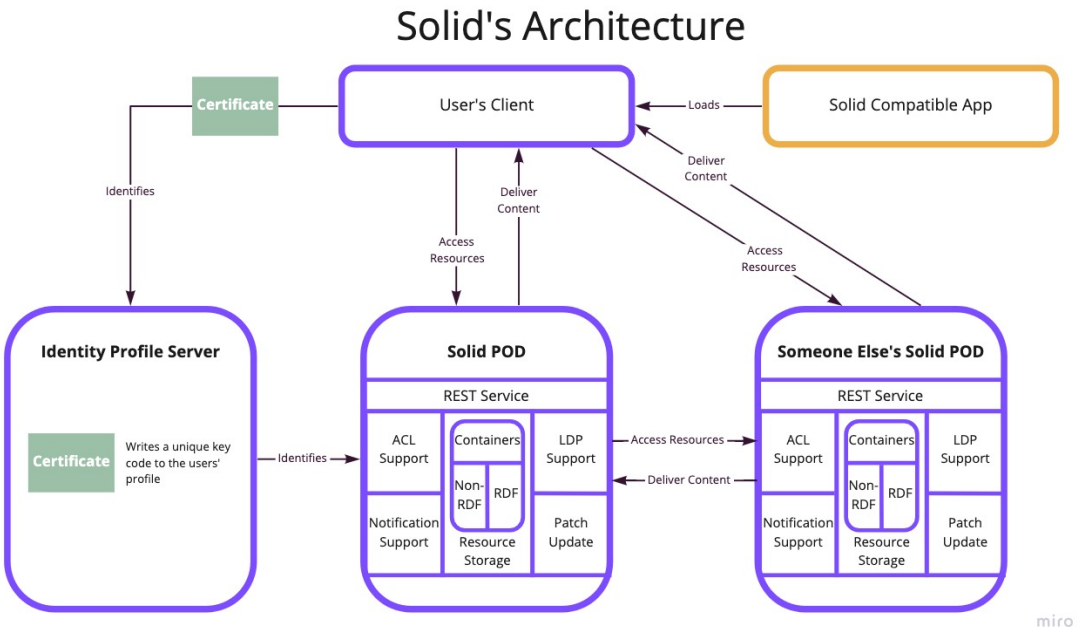


Figure 1: Simplified version of Solid Architecture

1. WebID is an HTTP URI representing an agent such as an organization, a user, or a software. When a WebID is dereferenced, the server provides a representation of the WebID Profile in an RDF document which uniquely describes an agent denoted by a WebID. WebIDs are an underpinning component in the Solid ecosystem and are used as the primary identifier for users and applications[8].

2. Authentication with the Solid server is done with Solid OpenID Connect (Solid-OIDC), which is performed by an OpenID provider.

3. Personal Online Data Stores (Pods) are the containers where data is saved/retrieved. Once data is saved, the owner of the Pod (agent) can control access to the resource. Pods can be hosted locally by user or using Pod providers. Here are details of Pod elements:

    3.1 Pod provider: The entities in charge of hosting Pods, like a cloud storage (Google Drive or Dropbox). Each provider follows the regulations depending on the country and region in the world for storing data and third-party access, as these storages are at a physical level and located in a geographical place. For example, refer to the Inrupt Pod Provider (https://start.inrupt.com/profile).

3.2 resource storage: Pod's storage allocated to users data. This data can be in two formats:

- Non-Resource Description Framework (Non-RDF) such as images, .pdf, and other files.
- Resource Description Framework (RDF) such as JSON-LD or Turtle which are data formats for preserving LD(core of solid data operation).

No matter which type of file the user saves, all would be stored as files similar to those on clouds and personal computers. The data on the pod comes from any solid compatible that the user uses.

3.3 Linked Data Platform (LDP container): Pods having the possibility of working with LDPs. Pods contains public and private LDP containers, each one with different access control regulations for defining what data is public and what has to stay private.

3.4 Access Control allows users to define who accesses what.

3.5 Access Control List (ACL) is a document containing a list of agents accessing what data and when. This helps Pods find out who can access data.

3.6 Representational State Transfer (REST) service: Allows other entities to interact with the pod with RESTful data management (based on HTTP).

The process is this way: the user starts by asking the IDP to be identified. After doing the process with Solid-OIDC, they would get the certificate (through a browser) and be used by a third-party application to present to the user's Pod provider and access the data with REST functions. Here, we can see the flow of the whole process[9] in Figure 2.

Because the data stored in Solid Pods are application-independent, they should be stored in a format that different apps can consistently read. For this reason, developers need to use a defined solid vocabulary or custom-defined vocabulary known as ontologies. These vocabularies can define the properties. In this way, each subject has a property and property value. All these subjects are IRIs in the system, like an address or a URL to a website; it shows the location of the stored file. For example, "user" is the subject, "eye color" is the property, and "green" is the property value. At least one or more triple of IRI, property, and property-value s can be linked together and form an RDF resource. An RDF resource then can be presented in different languages such as Turtle, N-Triples, N-Quads, RDF/XML, RDF/JSON, JSON-LD etc.[10]. Depending on the environment we are working in, we can choose a representation format.

2.2 Android App Development and Solid

With the birth of touch screens, the mobile phone industry had a boost and created a new market. Among all the players, Google and Apple could continue to dominate almost the entire mobile phone market. As we can see, almost everyone has at least one smartphone run by one of the mentioned vendors. Because of Android's open-source nature, it has been developed to be integrated with different vendors with more possibility of modifications and getting more popular. Google continues supporting Android, and almost every year, there is a new version of Android with

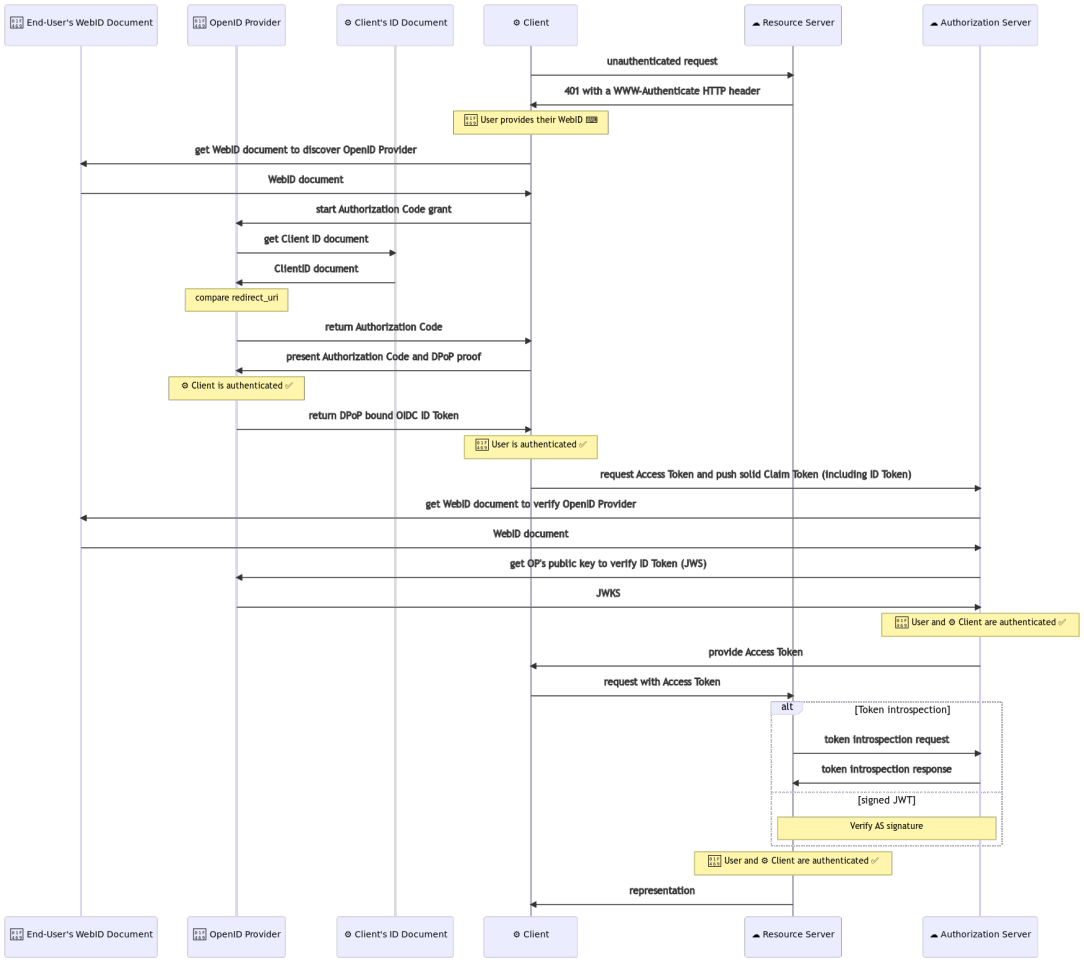


Figure 2: Solid Authentication Flow

new features that support the ecosystem's needs, such as foldable devices, wearable gadgets, cars, etc. A big part of these updates includes SDKs and APIs for developers to make new applications with needed features. In recent years, there has been a significant change in how an Android app develops; the new Jetpack collection has almost different parts of app development and is compatible with each other, making the creation of the Android app even more straightforward. Before this project, there wasn't any SDK or library to integrate Solid into Android development. In case a developer wanted to do so, he had to do the communication with basic REST APIs provided by the Pods in raw HTTP, handling authentication with IDP, creating sessions, translating Linked Data retrieved from Pods into readable data formats in Android such as JSON, and much more features that does not existed. So, the lack of any tool for developers could be sensed, which is the first point of app creation and introduction to 3.98 B users worldwide.

2.3 Existing Tools

Most tools are developed for web applications in JavaScript and some for Java clients such as Inrupt Java Client[11] library. As this library is written for the Java environment and Android supports Java, it can be imported into the project and used. There are some problems with using this library in Android, which will be

described in more detail in the next chapter because of Java usage in Java-based projects and the Android ecosystem. We do not have pure Java in Android, and some classes do not exist or have been changed in the implementation. For instance, the class ClassLoader has a different implementation. It has been used by some dependencies of Inrupt Java Client[11] library, such as Jena, which is responsible for working with Linked Data. So, some of the classes can be used, and others should be rewritten for Android.

2.4 Defining Desired Solution

By analyzing the structure of Solid, we find out that any Android application that wants to use Solid has to do the authentication, save the authentication details, and then use the credential to retrieve data. Here, it is understood that there is a need for a package that encapsulates the authentication. As each app will import this package and log in to solid (in the first use in the optimal conditions), there would be many logins that the user faces, which is not optimal in the sense of user experience. So, it came up with the idea of having a main service app, the same as Google Service, that the user logs in to once, and any third-party app that wants to connect to Solid would ask for this app. The app would need a dialog such as "App X wants to access your pod. Do you allow?" which is much better than multiple login processes and sessions.
After having a successful authentication process, there would be a Bearer or DPoP token (Depending on the way of handling OpenID), which can be used to call Pod REST APIs. For any request (GET, POST, PUT, DELETE), headers must be formed, the network body needs to be initialized, the function needs to be called, and the response and any possible error must be handled. So developers need another layer that helps them manage data on the Pod. Again, as apps are not responsible for authentication, the service app should perform the operations and retrieve the results for the third-party apps through Android Inter-Process Communication. Based on the Android solution, this can be done via 'Services' which is one of the four main components of Android (Activity, Service, Broadcast Receiver, Content Provider). In conclusion, the solution should contain an Android application responsible for authentication, saving credentials locally, and refreshing them, using the credentials for accessing data and providing at least one Android Service for other third-party apps to get their requested data. In addition, there should be a library that developers need to import to have the functionality of 'Connect to Solid' and resource management via Android Services; the functions should be predefined and dynamic to handle any data.

3 Final Solution

3.1 Integrate Existing Tools into the Android

For the authentication and resource management part, we started using Inrupt Java Client[11] and different modules in the library. For authentication, it came out that it is incompatible with Android as it does not contain Android elements such as Activity to send requests to a browser and return to the app. So instead of using the Authentication module, we imported OpenID Android AppAuth library[12], which is also compatible with Solid as Solid, also using OpenID configuration for the authentication process. The whole authentication process has been described in Authentication section.
In Resource Management, we can create a Client object to interact with a pod. However, the response should be in `ByteStream` or `String` because RDF-defined types use Jena or JSONB or Jackson as the parser of the data which all of them are not compatible with Android. We had a trial of using Desugaring in Android and trying to fork Jena and adapt it to Android, but it failed because of a couple of reasons:

- `ClassLoader` used in both libraries and changing them means having many dependencies that will not be updated as we will be responsible for changing them.
- In Inrupt Java Client[11] library also Java system calls have been used which is not presented in Android. So, it would need to change too.

So we decided to use `ByteStream` as the body of our HTTP requests and use another library such as Titanium JSON-LD[13], which converts `ByteStream` of data presented in Turtle into JSON-LD which can be used in Android.

3.2 Architecture

The project has four modules:

- `Shared`
- `SolidAndroidApi`
- `SolidAndroidClient`
- `app`

The internal module Shared is responsible for sharing standard definitions across the project, such as data classes and network response types. All other modules mentioned above will use data classes and tools existing in this module.

The module Solid Android API is another package responsible for authentication with Solid-OIDC server and communicating with the Solid server for accessing resources. Also, `SolidContactsDataModule` is a Solid DataModule for contacts present here. This data class is a higher-level resource that encapsulates how an RDF resource converts to a Contacts data class. This library is used in `app` module and exported for other developers to connect directly to Solid in their apps.

The package Android Solid Services contains an Android app in which users will log in to their pods by using Solid Android API library as a single point of login and third-party apps who want to connect to Solid without dealing with authentication and resource management will use Solid Android Client library in their apps and all the requests pass through this app using Android inter-process communication.

Finally, Solid Android Client module is another library that is exported, and developers can import it into their apps as a second and easier option, using defined interfaces to request a login and other resources and data modules already implemented into the project. All the functions presented here use `.aidl` definitions to ask Android Solid Services app to handle the requests.

The general scheme of the project architecture can be seen in Figure 3.

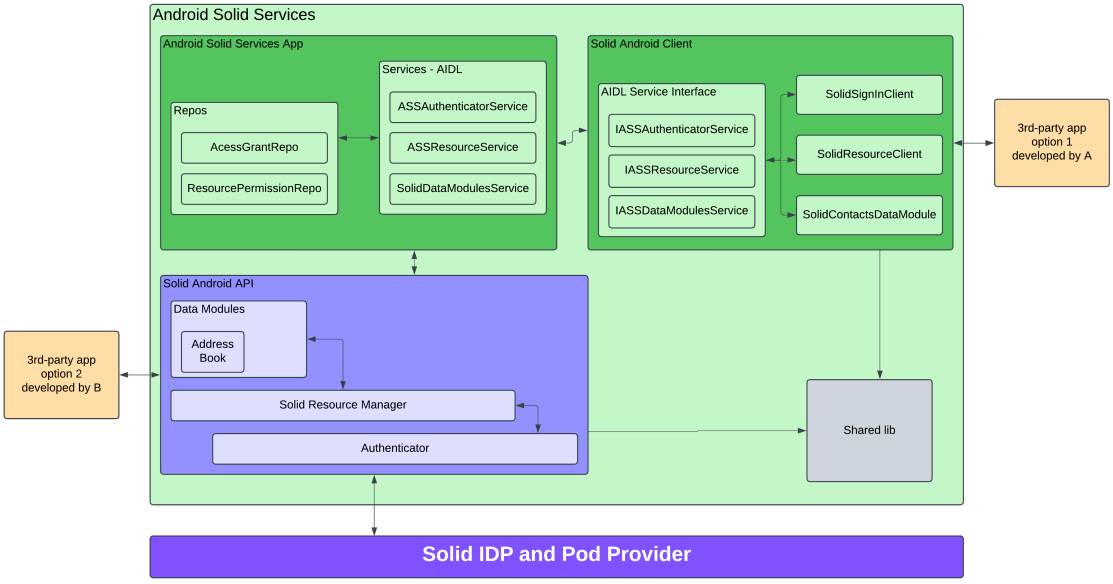


Figure 3: The overall architecture of Android Solid Services project with its modules and points of interacting with external parties.

3.3 Shared

This module contains shared classes and resources that are used across other modules in the project and has the following main base definitions:

3.3.1 `Resource`

`Resource` interface definition has been showed in the code 1. It inherits from `AutoCloseable` as the resources will be read and written as a `Stream`. It also inherits from `Parcelable`, which is a way of serialization in the Android ecosystem for reading, writing, or passing among different processes. The functions inside this interface will be used as:

- `getIdentifier():URI`: is responsible for returning the address of a resource in the form of URI.
- `getContentType():String`: this method will return the content type of the resource such as `application/ld+json`, `*/*`, and `text/turtle`.

- `getHeaders():Headers`: will return headers of resource in a API call. For now, there is no detailed implementation for this part.

- `getEntity():InputStream`: will give the content of the resource in the form of a `Stream`.

In general, we have three general-purpose resources that inherit from the `Resource` interface:

- `NonRDFSource` which is used for resources, not in the form of RDF.

- `RDFSource` for resources which are Linked Data.

- `SolidContainer` for containers, similar to folders, that contain other resources. This class inherits from `RDFSource`, so it indirectly inherits from `Resource`.

```
interface Resource : AutoCloseable, Parcelable {
    fun getIdentifier(): URI
    fun getContentType(): String
    fun getHeaders(): Headers
    @Throws(IOException::class)
    fun getEntity(): InputStream
}
```

Code Snippet 1: `Resource` interface definition

3.3.2 `NonRDFSource`

`NonRDFSource` is used for representing resources not in the form of RDF. It implements functions mentioned in `Resource` interface definition and the ones in `Parceable` as follows:

```
private constructor(inParcel: Parcel) {
    this.identifier = URI.create(inParcel.readString())
    this.contentType = inParcel.readString()!!
    this.headers = Gson().fromJson<Headers>(inParcel.readString(),
    ↪  object : TypeToken<Headers>() {}.type)
    this.entity = inParcel.readString()!!.byteInputStream()
}
...
override fun writeToParcel(dest: Parcel, flags: Int) {
    dest.writeString(identifier.toString())
    dest.writeString(contentType)
    dest.writeString(Gson().toJson(headers))
    dest.writeString(getEntity().bufferedReader().use {
    ↪  it.readText() })
}
```

Code Snippet 2: The way `NonRDFSource` writes to a `Parcel` object and reads from it to construct a new one

These implementations are important as they will pass objects between `AndroidSolidServices` app and other third-party apps.
The content of this resource is saved as an `InputStream`.

3.3.3 `RDFSource`

`RDFSource` is used to represent RDF resources. Same as `NonRDFSource`, it implements needed functions and creates a `dataset` to store the resource's content. Writing and reading from a `Parcel` object acts as shown in code 3.

```
private constructor(inParcel: Parcel) {
    this.identifier = URI.create(inParcel.readString())
    this.headers = Gson().fromJson<Headers>(inParcel.readString(),
    ↪  object : TypeToken<Headers>() {}.type)
    this.mediaType =
    ↪  Gson().fromJson<MediaType>(inParcel.readString(), object :
    ↪  TypeToken<MediaType>() {}.type)
    this.dataset = JsonLd.toRdf(JsonDocument.of(inParcel.readString
    ↪  ()!!.byteInputStream())).options(JsonLdOptions().apply {
        isRdfStar = true
    }).get()
    this.itselfSubject = inParcel.readString()!!
}
...
override fun writeToParcel(dest: Parcel, flags: Int) {
    dest.writeString(identifier.toString())
    dest.writeString(Gson().toJson(headers))
    dest.writeString(Gson().toJson(mediaType))
    dest.writeString(JsonLd.fromRdf(RdfDocument.of(dataset)).get().
    ↪  tString())
    dest.writeString(itselfSubject)
}
```

Code Snippet 3: The way `RDFSource` writes to a `Parcel` object and reads from it to construct a new one

The function `getEntity()` converts the stored dataset to an `InputStream` object shown in code 4. `JsonDocument` created out of the `dataset` then compacted it with or without a context document and asked the RDF processor to return its stream of bytes.

```
override fun getEntity(): InputStream {
    val toBeCompactDoc = JsonDocument.of(JsonLd.fromRdf(RdfDocument
    ↪  .of(dataset)).get().toString().byteInputStream())
    val compacted = JsonLd.compact(toBeCompactDoc,
    ↪  contextDocument).get()
    return compacted.toString().byteInputStream()
}
```

Code Snippet 4: The way `RDFSource` overrides `getEntity() function`

In `dataset:RdfDataset` property of the class, attributes have been stored in a triple of subject-predicate-object, which subject refers to the URI of the attribute, predicate to the key of the attribute and object to the real value of it. As an example for expressing the nationality of Alice, the subject would refer to "Alice," predicate to "Nationality," and object to "Italian."

For adding a new attribute, we can use `addTriple` function as shown 5. It gets a triple and the maximum number of recurrences of the attribute defined in the triplet. After checking the existing attribute, it is decided whether to add it. As the current `dataset` object is static, it needs to recreate the current dataset and add a new one. For completing the above example, Alice can have another nationality, so the value `maxNumber` would be the maximum number of nationalities she can have. For her birthday, as it is unique, this value must be 1.

```
fun addTriple(triple: RdfTriple, maxNumber: Int = 1) {
    val currentItemSize = dataset.defaultGraph.toList().filter {
        it.subject.equals(triple.subject) &&
        ↪ it.predicate.equals(triple.predicate)
    }.size
    if(currentItemSize < maxNumber) {
        dataset.add(triple)
        return
    } else {
        val all = dataset.defaultGraph.toList()
        val newDateset = rdf.createDataset()
        all.forEach {
            if (!it.subject.equals(triple.subject) ||
            ↪ !it.predicate.equals(triple.predicate)) {
                newDateset.add(it)
            }
        }
        newDateset.add(triple)
        dataset = newDateset
    }
}
```

Code Snippet 5: Add an attribute to a `RDFSource`.

In order to remove an attribute from the `dataset`, `clearProperties` function with the implementation in 6 can be called. It checks for all occurrences of the attribute and then creates a new `dataset` excluding the attribute.

```
fun clearProperties(
    predicate: String,
    subject: String = itselfSubject
) {
    val allWithProperties = dataset.defaultGraph.toList().filter {
        it.subject.value == subject && it.predicate.value ==
        ↪ predicate
    }
    if (allWithProperties.isNotEmpty()) {
        val all = dataset.defaultGraph.toList()
        all.removeAll(allWithProperties)
        val newDateset = rdf.createDataset()
        all.forEach {
            newDateset.add(it)
        }
        dataset = newDateset
    } else {
        //No need to clear as we already know it does not exist
    }
}
```

Code Snippet 6: Remove an attribute from a `RDFSource`.

Two functions `findAllProperties` and `findProperty` have been used to locate and return properties. The implementation can be seen in code 7.

```
fun findAllProperties(predicate: RdfResource): List<RdfValue> {
    return dataset.defaultGraph.toList()
        .filter{ it.predicate.value == predicate.value }
        .map { it.`object` }
}
...
fun findProperty(predicate: RdfResource): RdfValue? {
    return dataset.defaultGraph.toList()
        .find{ it.predicate.value == predicate.value }
        ?.`object`
}
```

Code Snippet 7: Find attributes inside a `RDFSource`.

3.3.4 `SolidContainer`

`SolidContainer` refers to resources in a Solid Pod which contains other resources. This class is a special case of a `RDFSource` resource, which constructs a list of resources inside it during its construction. The implementation can be seen in the code 8. Inside, it used `SolidSourceReference` data class for referencing a resource

with an identifier and a list of types. The types can be one of three base resource definitions described above in 3.3.1.

```
dataset?.defaultGraph?.toList()?.filter {
    it.predicate.equals(contains) && it.subject.value ==
    ↪ getIdentifier().toString()
}?.forEach {
    val identifier = it.`object`
    val types = dataset.defaultGraph.toList().filter {
        it.subject.equals(identifier) &&
        ↪ it.predicate.equals(rdfType)
    }.map { it.`object`.value }
    containerRes.add(SolidSourceReference(identifier.value, types))
}
```

Code Snippet 8: `SolidContainer` constructor

This class has some specific methods, such as `getContained():List<SolidSourceReference>` which returns a list of resources inside it and `getLabel():String?` that gives container's name.

3.3.5 `Profile`

`Profile` data class is responsible for holding the user's authentication state, user info, and his WebID. The class definition can be seen in code 9. `AuthState` is a part of `net.openid.appauth` library and contains, authentication state, `tokeId`, `accessToken`, `refreshToken` , and other necessary information during authentication processes. `WebId` data class inherits from `RDFSource` and has information about the logged-in user such as his `oidcissuer`, Pod storage addresses, `privateTypeIndex` , and `publicTypeIndex`.

```
data class Profile(
    var authState: AuthState = AuthState(),
    var userInfo: UserInfo? = null,
    var webId: WebId? = null
)
```

Code Snippet 9: `Profile` data class definition

3.3.6 `SettingTypeIndex`

`SettingTypeIndex` is a `RDFSource` which hols type index of a Pod, working like a mapping point when need to find something in the pod, as in instance, when a user needs to get Address Books in the pod, will refer to this index and find URIs of all the address books existing. It can be private or public, and external entities can access it. This document has two type keys, first `TypeIndex` and then for the private resource it is `UnlistedDocument`, and `ListedDocument` for the public index.

Any new index inside needs three triples: one for `TypeRegistration`, one for identifying the type of class it is referring to, and finally, the last one referring to the instance. As an example, it can be seen how it adds an `AdressBook` to its indexes in code 10.

These two implementations that inherit from `SettingTypeIndx` and can be used to locate indexes in a Pod, `PrivateTypeIndx` and `PublicTypeIndex`

```
1 fun addAddressBook(addressBook: String) {
2     val id = UUID.randomUUID().toString()
3     val subject = rdf.createIRI("${getIdentifier()}#${id}")
4     addTriple(rdf.createTriple(subject, typeKey, typeRegistration))
5     addTriple(rdf.createTriple(subject, forClass, AddressBook))
6     addTriple(rdf.createTriple(subject, instance,
      ↪  rdf.createIRI(addressBook)))
7 }
```

Code Snippet 10: `addAddressBook` function in `SettingTypeIndex`

3.3.7 `WebId`

This `RDFSource` resource contains general information about a user Pod such as the addresses of `PrivateTypeIndex`, `PublicTypeIndex`, storages, and `oidcissuer`. It has two static functions shown in code 11 for converting to a `JsonString` and vice versa; Used for saving and retrieving data stored in this class on local storage.

```
fun writeToString(webId: WebId?): String? {
    if (webId == null) {
        return null
    }
    return JsonObject().apply {
        addProperty(KEY_IDENTIFIER,
        ↪  webId.getIdentifier().toString())
        addProperty(KEY_TYPE, webId.getContentType())
        addProperty(KEY_DATASET, webId.getEntity().toPlainString())
    }.toString()
}

fun readFromString(objectString: String): WebId {
    val obj = JsonParser.parseString(objectString).asJsonObject
    return WebId(
        URI.create(obj.get(KEY_IDENTIFIER).asString),
        MediaType.of(obj.get(KEY_TYPE).asString),
        JsonLd.toRdf(JsonDocument.of(obj.get(KEY_DATASET).asString.
        ↪  byteInputStream())).get(),
    )
}
```

Code Snippet 11: `writeToString(...)` and `readFromString(...)` implementations

It also has other functions, as shown in the code 12.

```
fun getTypes(): List<URI> { ... }

fun getOidcIssuers(): List<URI> { ... }

fun getRelatedResources(): List<URI> { ... }

fun getStorages(): List<URI> { ... }

fun getPrivateTypeIndex(): String? { ... }

fun getPublicTypeIndex(): String? { ... }

fun getProfileUrl(): String { ... }
```

Code Snippet 12: `writeToString` and `readFromString` implementations

3.3.8 Contact Data Module

Data Modules are a higher level of a `RDFSource` for abstracting details of implementation and handling property management of a resource. As an example, the

Contact data module has been implemented in this project. It has four main data classes with respective `RDFSource` classes. There is another additional data class for holding two different lists of public and private address books. Each `AddressBook` has a title and contains a list of `Contacts` and `Groups`. Each `Contact` has a name and a list of phone numbers and emails. However, a group is defined by its title and a list of contacts that are a part of it. All these classes are `Parcelizable` for passing between Android processes.

```
@Parcelize
data class AddressBookList(
    val publicAddressBookUris: List<String>,
    val privateAddressBookUris: List<String>,
): Parcelable

@Parcelize
data class AddressBook(
    val uri: String,
    var title: String,
    var contacts: List<Contact>,
    var groups: List<Group>,
): Parcelable {
    companion object {
        fun createFromRdf(
            addressBookRdf: AddressBookRDF,
            nameEmailIndexRdf: NameEmailIndexRDF,
            groupsIndexRdf: GroupsIndexRDF
        ): AddressBook {
            return AddressBook(
                uri = addressBookRdf.getIdentifier().toString(),
                title = addressBookRdf.getTitle(),
                contacts = nameEmailIndexRdf.getContacts(
                    addressBookRdf.getIdentifier().toString()
                ),
                groups = groupsIndexRdf.getGroups(
                    addressBookRdf.getIdentifier().toString()
                )
            )
        }
    }
}
```

Code Snippet 13: `AddressBookList` and `AddressBook` definitions

```kotlin
@Parcelize
data class FullContact(
    val uri: String,
    val fullName: String,
    val emailAddresses: List<Email>,
    val phoneNumbers: List<PhoneNumber>,
): Parcelable {
    companion object {
        fun createFromRdf(contactRdf: ContactRDF): FullContact {
            return FullContact(
                uri = contactRdf.getIdentifier().toString(),
                fullName = contactRdf.getFullName(),
                emailAddresses = contactRdf.getEmails(),
                phoneNumbers = contactRdf.getPhoneNumbers()
            )
        }
    }
}
```

Code Snippet 14: `FullContact` definition

```kotlin
@Parcelize
data class FullGroup(
    val uri: String,
    val name: String,
    val contacts: List<Contact>,
): Parcelable {
    companion object {
        fun createFromRdf(groupRdf: GroupRDF): FullGroup {
            return FullGroup(
                uri = groupRdf.getIdentifier().toString(),
                name = groupRdf.getTitle(),
                contacts = groupRdf.getContacts()
            )
        }
    }
}
```

Code Snippet 15: `FullGroup` definition

3.4 Solid Android API

This module, which is also exported as an Android library, is responsible for authentication and resource management. It also provides some functions for handling the Contacts data module and an interface for storing `Profile` on Android local storage via `SharedPreferences`.

3.4.1 Authentication

An interface has been initialized named `Authenticator`, which has basic methods for doing authentication.
First, one of two functions `createAuthenticationIntentWithWebId()` or `createAuthenticationIntentWithOidcIssuer()` creating an Intent for starting the authentication flow that can be used by calling Android `startActivityForResult()` function. After completing the auth process in a browser, the user uses, the result can be passed to `submitAuthorizationResponse()` function. In case of a successful login, this function will get a token and `WebId` data class, which contains some information related to WebID issuer and address(es) of Pod location(s). For further uses, `getLastTokenResponse()` function can be used to get a valid token - in case of invalidity of the token, request a fresh token and return it. `getProfile()` is responsible for returning `Profile` data class, which has all the information related to the authentication state, WebID, and other information. In the case of a not-authenticated user, some parameters will be `null`.
For signing out and finishing an authenticated session, `getTerminationSessionIntent()` can be used with an Intent to be used in a browser. The class `AuthenticatorImplementation` implements `Authenticator` interface and has some private functions as well. Here are some important parts of the implementations that have been shown in codes 16, 17, 18, 19, 20, and 21:

```
1  private suspend fun getAuthorizationConf(
2      oidcIssuer: String
3  ): Pair<AuthorizationServiceConfiguration?,
   ↪  AuthorizationException?>{
4      return suspendCoroutine { cont ->
5          AuthorizationServiceConfiguration.fetchFromIssuer(Uri.parse
           ↪  (oidcIssuer)) { serviceConfiguration, exception ->
6              cont.resume(Pair(serviceConfiguration, exception))
7          }
8      }
9  }
```

Code Snippet 16: `getAuthorizationConf()` implementation that gets configuration of the WebID issuer

```kotlin
private suspend fun registerToOpenId(
    conf: AuthorizationServiceConfiguration,
    redirectUri: String,
) {
    val regReq = RegistrationRequest.Builder(
        conf,
        listOf(Uri.parse(redirectUri))
    ).setAdditionalParameters(mapOf(
        "client_name" to "Android Solid Services",
        "id_token_signed_response_alg" to conf.discoveryDoc!!.idTok
        ↪  enSigningAlgorithmValuesSupported[0],
    ))
        .setSubjectType("public")
        .setTokenEndpointAuthenticationMethod("client_secret_basic")
        .setGrantTypeValues(listOf("authorization_code",
        ↪  "refresh_token"))
        .build()

    val res = suspendCoroutine { cont ->
        authService.performRegistrationRequest(regReq) { response,
        ↪  ex ->
            cont.resume(response)
        }
    }
    updateRegistrationResponse(res)
}
```

Code Snippet 17: `registerToOpenId()` method implementation that requests to OpenID by passing oidc-issuer configuration and the app's redirect URI

```
private suspend fun requestToken()
    : Pair<TokenResponse?, AuthorizationException?> {

    val result : Pair<TokenResponse?, AuthorizationException?> =
    if (profile.authState.lastAuthorizationResponse != null) {
        suspendCoroutine { cont -> authService.performTokenRequest(
        ↪ profile.authState.lastAuthorizationResponse!!.createTok
        ↪ enExchangeRequest(),ClientSecretBasic(profile.authState
        ↪ .lastRegistrationResponse!!.clientSecret!!)) {
        ↪ tokenResponse, exception ->
                updateTokenResponse(tokenResponse, exception)
                cont.resume(Pair(tokenResponse, exception))
            }
        }
    } else {
        Pair(null, profile.authState.authorizationException)
    }
    return result
}
```

Code Snippet 18: Requesting a token for the first time after having a successful authentication

```kotlin
override suspend fun createAuthenticationIntentWithOidcIssuer(
    oidcIssuer: String,
    redirectUri: String,
) : Pair<Intent?, String?> {

    val conf = getAuthorizationConf(oidcIssuer)

    return if (conf.first != null) {

        registerToOpenId(conf.first!!, redirectUri)

        if (profile.authState.lastRegistrationResponse != null) {
            val builder = AuthorizationRequest.Builder(
                conf.first!!,
                profile.authState.lastRegistrationResponse!!.client
                ↪ Id,
                ResponseTypeValues.CODE,
                Uri.parse(redirectUri))

            val authRequest = builder
                .setScopes( "webid", "openid", "offline_access",)
                .setPrompt("consent")
                .setResponseMode(AuthorizationRequest.ResponseMode.
                ↪ QUERY)
                .build()

            val authIntent = authService.getAuthorizationRequestInt
            ↪ ent(authRequest)
            Pair(authIntent, null)
        }
        else {
            Pair(null, "can not register to OpenId")
        }
    } else {
        Pair(null, "can not get access to web-id issuer
        ↪ configurations.")
    }
}
```

Code Snippet 19: Creating an authentication Intent

```
override suspend fun submitAuthorizationResponse(
    authResponse: AuthorizationResponse?,
    authException: AuthorizationException?
) {
    updateAuthorizationResponse(authResponse, authException)
    if (authException == null && authResponse != null) {
        requestToken()
        profile.userInfo = getUserInfoFromIdToken()
        profile.webId = getWebIdProfile(profile.userInfo!!.webId)
        userRepository.writeProfile(profile)
    }
}
```

Code Snippet 20: Handling authorization response after completing the login in the browser

```
private fun parseIdToken(idToken: String, config: OpenIdConfig):
↪   JwtClaims {
    return try {
        val builder = JwtConsumerBuilder()

        // Required by OpenID Connect
        builder.setRequireExpirationTime()
        builder.setExpectedIssuers(true, *arrayOfNulls<String>(0))
        builder.setRequireSubject()
        builder.setRequireIssuedAt()

        // If a grace period is set, allow for some clock skew
        if (config.expGracePeriodSecs > 0) {
            builder.setAllowedClockSkewInSeconds(config.expGracePer
            ↪   iodSecs)
        } else {
            builder.setEvaluationTime(NumericDate.fromSeconds(Insta
            ↪   nt.now().epochSecond))
        }

        // If an expected audience is set, verify that we have the
        ↪   correct value
        if (config.expectedAudience != null) {
            builder.setExpectedAudience(true,
            ↪   config.expectedAudience)
        } else {
            builder.setSkipDefaultAudienceValidation()
        }

        // If a JWKS location is set, perform signature validation
        if (config.publicKeyLocation != null) {
            val jwks =
            ↪   HttpsJwks(config.publicKeyLocation.toString())
            val resolver = HttpsJwksVerificationKeyResolver(jwks)
            builder.setVerificationKeyResolver(resolver)
        } else {
            builder.setSkipSignatureVerification()
        }

        val consumer = builder.build()
        consumer.processToClaims(idToken)
    } catch (ex: InvalidJwtException) {
        throw OpenIdException("Unable to parse ID token", ex)
    }
}
```

Code Snippet 21: Getting `JwtClaims` object by `idToken` which contains user's WebID

3.4.2 Resource Management

An interface has been declared for interacting with Solid Pod, which can be seen in code 22.

```
interface SolidResourceManager {

    suspend fun <T: Resource> read(
        resource: URI,
        clazz: Class<T>,
    ): SolidNetworkResponse<T>

    suspend fun <T: Resource> create(
        resource: T
    ): SolidNetworkResponse<T>

    suspend fun <T: Resource> update(
        newResource: T
    ): SolidNetworkResponse<T>

    suspend fun <T: Resource> delete(
        resource: T,
    ): SolidNetworkResponse<T>

    suspend fun deleteContainer(
        containerUri: URI
    ): SolidNetworkResponse<Boolean>
}
```

Code Snippet 22: Definition of `SolidResourceManager` interface with its functions

In the code 22, the first four functions are the main functions for interacting with resources in a Pod and the function `deleteContainer()` defined for `SolidContainer` separately because the way Solid Pods delete containers is a recursive approach. The URI input parameters are the address of the working resource. The implementation of the output, `SolidNetworkResponse<T>`, is shown in code 23 which handles generic types. In case of a successful call, it returns `Success` with the `data` field containing the resource. If there is a problem in processing the request, an `Error` object with an error code and error message will be returned. This will happen, for example, when the user asks to read a resource that does not exist, and the user will get an error code `404` with a proper message. Sometimes, the call will face a failure because of an internet connection or similar issues, and an Exception will be returned to the caller.

```
1  sealed class SolidNetworkResponse<T> {
2      data class Success<T>(val data: T) : SolidNetworkResponse<T>()
3      data class Error<T>(val errorCode: Int, val errorMessage:
       ↪  String) : SolidNetworkResponse<T>()
4      data class Exception<T>(val exception: Throwable) :
       ↪  SolidNetworkResponse<T>()
5      ...
6  }
```

Code Snippet 23: `SolidNetworkResponse<T>` implementation

Class `SolidResourceManagerImplementation` implements the interface shown in code 22. As an example, function `read()` has been shown in code 24.
For the inputs, URI of the resource and the class type of the the resource have to be passed in lines 2 and 3. In line 7, we call `handleTokenRefreshAndReturn()` function to get the latest valid token of the authenticated user to be included in the `GET` call. Then, we construct the network request with the token and the `Accept` response type. If the resource is a subclass of `RDFSource`, the accept type will be JSON-LD, otherwise we get it as `ByteStream` (`*/*`). In line 16, we make the network call to the Solid server and return the result.

```
override suspend fun <T: Resource> read(
    resource: URI,
    clazz: Class<T>,
): SolidNetworkResponse<T> {

    try {
        val tokenResponse = handleTokenRefreshAndReturn()

        val request = Request.newBuilder()
            .uri(resource)
            .header(HTTPHeaderName.ACCEPT, if
            ↪  (RDFSource::class.java.isAssignableFrom(clazz))
            ↪  HTTPAcceptType.JSON_LD else HTTPAcceptType.ANY)
            .header(HTTPHeaderName.AUTHORIZATION,
            ↪  "${tokenResponse?.tokenType}
            ↪  ${tokenResponse?.idToken}")
            .GET()
            .build()

        val response: Response<InputStream> = client.send(
            request,
            Response.BodyHandlers.ofInputStream()
        )

        return if (response.isSuccessful()) {
            SolidNetworkResponse.Success(constructObject(response,
            ↪  clazz))
        } else {
            SolidNetworkResponse.Error(response.statusCode(),
            ↪  response.body().toPlainString())
        }
    } catch (e: Exception) {
        return SolidNetworkResponse.Exception(e)
    }
}
```

Code Snippet 24: Reading a resource from a Pod and convert it to type `T` which inherits from `Resource`

The implementation of the function `constructObject()` mentioned in code 24 has been shown in code 25. This function will construct the resource based on its type. First, we get the response type from the header and then convert the response to a `String` in line 7. In line 8, the resource type will be checked and, in the case of being a `SolidContainer`, enters the `if` body. Otherwise, it will be checked for `RDFSource` type; if not, behave as if it is a `NonRDFSource`. In line 10, the function `provideNormalizedDoc()` has been called to build a complete path of container URI. In case of being a `RDFSource`, the `Dataset` will be created in line

17. Finally, no matter the type, we get `clazz` constructor, make the object with its input parameters, and return the created object of the resource.

```
private fun <T> constructObject(
    response: Response<InputStream>,
    clazz: Class<T>
): T {
    val type =
     ↪  response.headers().firstValue(HTTPHeaderName.CONTENT_TYPE)
        .orElse(HTTPAcceptType.OCTET_STREAM)
    val string = response.body().toPlainString()
    if (SolidContainer::class.java.isAssignableFrom(clazz)) {
        val dataSet = JsonLd
            .toRdf(provideNormalizedDoc(string.byteInputStream()))
            .get()
        return clazz.getConstructor(URI::class.java,
         ↪  MediaType::class.java, RdfDataset::class.java,
                Headers::class.java)
            .newInstance(response.uri(), MediaType.of(type),
             ↪  dataSet, null)
    }
    else if (RDFSource::class.java.isAssignableFrom(clazz)) {
        val dataSet = JsonLd
            .toRdf(JsonDocument.of(string.byteInputStream()))
            .rdfDirection(JsonLdOptions.RdfDirection.I18N_DATATYPE)
            .mode(JsonLdVersion.V1_1)
            .produceGeneralizedRdf()
            .get()
        return clazz.getConstructor(URI::class.java,
         ↪  MediaType::class.java, RdfDataset::class.java,
         ↪  Headers::class.java
            ).newInstance(response.uri(), MediaType.of(type),
             ↪  dataSet, null)
    } else {
        return clazz.getConstructor(URI::class.java,
         ↪  String::class.java,
         ↪  InputStream::class.java).newInstance(response.uri(),
         ↪  type, string.byteInputStream())
    }
}
```

Code Snippet 25: Implementation of function `constructObject()`

3.4.3 User Repository

This simple interface has two functions, as shown in the code 26. A default implementation is also provided in the same package, and the way it reads/writes the user's profile is demonstrated in the code 27.

```kotlin
interface UserRepository {
    fun readProfile(): Profile
    fun writeProfile(profile: Profile)
}
```

Code Snippet 26: `UserRepository` definition

```kotlin
override fun readProfile(): Profile {
    val profile = Profile()
    val stateString =
    ↪ sharedPreferences.getString(PROFILE_STATE_KEY, null)
    val webIdString =
    ↪ sharedPreferences.getString(PROFILE_WEB_ID_DETAILS_KEY,
    ↪ null)
    if (!stateString.isNullOrEmpty()) {
        profile.authState =  AuthState.jsonDeserialize(stateString)
    }
    profile.userInfo = Gson().fromJson(sharedPreferences.getString(
    ↪ PROFILE_USER_INFO_KEY, null), UserInfo::class.java)
    if (!webIdString.isNullOrEmpty()) {
        profile.webId = readFromString(webIdString)
    }
    return profile
}

override fun writeProfile(profile: Profile) {
    sharedPreferences.edit().apply {
        putString(PROFILE_STATE_KEY,
        ↪ profile.authState.jsonSerializeString())
        putString(PROFILE_USER_INFO_KEY,
        ↪ Gson().toJson(profile.userInfo))
        putString(PROFILE_WEB_ID_DETAILS_KEY,
        ↪ writeToString(profile.webId))
        apply()
    }
}
```

Code Snippet 27: `UserRepository` main functions implementation

3.4.4 Solid Contacts Data Module

As mentioned in section 3.3.8, Data Modules are using built-in data classes of Kotlin instead of `RDFSource` classes. So this interface has been defined as a middleware to translate classes related to `AddressBook` from/to `RDFSource`. The abstract functions can be seen on the code 28, 29 and 30.
The class `SolidContactsDataModuleImplementation` has the implementation of

these functions, and it uses `SolidContactsDataModuleHelper` to collect data for each request by calling base functions in Resource Management and translate them.

```
//region AddressBooks
suspend fun getAddressBooks(
    webId: String
): DataModuleResult<AddressBookList>

suspend fun createAddressBook(
    title: String,
    isPrivate: Boolean = true,
    storage: String,
    ownerWebId: String,// TODO: we need default value
    container: String? = null,
) : DataModuleResult<AddressBook>

suspend fun getAddressBook(
    addressBookUri: String,
): DataModuleResult<AddressBook>

suspend fun renameAddressBook(
    addressBookUri: String,
    newName: String,
): DataModuleResult<AddressBook>

suspend fun deleteAddressBook(
    addressBookUri: String,
    ownerWebId: String,
): DataModuleResult<AddressBook>
//endregion
```

Code Snippet 28: `SolidContactsDataModule` definition - AddressBook-related function

```kotlin
//region Contacts
suspend fun createNewContact(
    addressBookString: String,
    newContact: NewContact,
    groupStrings: List<String> = emptyList(),
) : DataModuleResult<FullContact>

suspend fun getContact(
    contactString: String
): DataModuleResult<FullContact>

suspend fun renameContact(
    contactString: String,
    newName: String,
): DataModuleResult<FullContact>

suspend fun addNewPhoneNumber(
    contactString: String,
    newPhoneNumber: String,
): DataModuleResult<FullContact>

suspend fun addNewEmailAddress(
    contactString: String,
    newEmailAddress: String,
): DataModuleResult<FullContact>

suspend fun removePhoneNumber(
    contactString: String,
    phoneNumber: String,
): DataModuleResult<FullContact>

suspend fun removeEmailAddress(
    contactString: String,
    emailAddress: String,
): DataModuleResult<FullContact>

suspend fun deleteContact(
    addressBookUri: String,
    contactUri: String,
): DataModuleResult<FullContact>
//endregion
```

Code Snippet 29: `SolidContactsDataModule` definition - Contacts-related functions

```kotlin
//region Groups
suspend fun createNewGroup(
    addressBookString: String,
    title: String,
    contactUris: List<String> = emptyList(),
): DataModuleResult<FullGroup>

suspend fun getGroup(
    groupString: String,
): DataModuleResult<FullGroup>

suspend fun deleteGroup(
    addressBookString: String,
    groupString: String
): DataModuleResult<FullGroup>

suspend fun addContactToGroup(
    contactString: String,
    groupString: String,
): DataModuleResult<FullGroup>

suspend fun removeContactFromGroup(
    contactString: String,
    groupString: String,
): DataModuleResult<FullGroup>
//endregion
```

Code Snippet 30: `SolidContactsDataModule` definition - Groups-related function

3.4.5 How to use

This library has been published to Maven Central and can be imported. Developers must import it with the latest version (currently 0.2.0) into their app-level `build.gradle.kts` file as shown in code 31.

```kotlin
dependencies {
    ...
    implementation("com.pondersource.solidandroidapi:solidandroidap
    ↪ i:0.2.0")
}
```

Code Snippet 31: Importing Solid Android API library to a third-party application

Then, by doing step-by-step processes presented in 32, a single instance of `com.pondersource.solidandroidapi.Authenticator` can be obtained. With this instance, the developer can call the functions afterward one by one to pass the user

through the authentication process. After doing a successful login, the developer can do CRUD operations by getting an instance of `com.pondersource.solidandroidapi.SolidResourceManager` and calling methods respectively in code 33. For using the Contacts data module, the developer has to get an instance of `com.pondersource.solidandroidapi.SolidContactsDataModule` and call functions presented in Solid Contacts Data Module. A sample of how to use it is mentioned in code 34. Note that the function `extractResult()` can be used to extract the data module content from `DataModuleResult<T>` data class.

```
val auth: Authenticator =
    AuthenticatorImplementation.getInstance(context)
....
val intent = auth.createAuthenticationIntentWithWebId(
    USER_WEB_ID, APP_REDIRECT_URL)

//Use the code below or any other way to start an intent and get
    the result. This implementation is the old way; if you were
    using AndroidX or Compose, this way of calling
    startActivityForResult would be different.
startActivityForResult(intent, INTENT_CODE)
...
@override
fun onActivityResult(request code: Int, resultCode: Int, data:
    Intent) {
    if (requestCode == INTENT_CODE) {
        if (resultCode == RESULT_OK) {
            //Update authorization response
            auth.submitAuthorizationResponse(
                AuthorizationResponse.fromIntent(intent),
                AuthorizationException.fromIntent(intent)
            )

            //Check if user login was successful
            if(auth.isUserAuthorized()) {
                val userProfile = authenticator.getProfile()
            }
        }
    }
}

```

Code Snippet 32: Using Solid Android API library in a third-party application for doing the authentication with Solid-OIDC

```
val srm: SolidResourceManager =
 ↪  SolidResourceManagerImplementation.getInstance(context)

//Creating a resource in the user's pod
val createResult = srm.create(YOUR_RESOURCE_INSTANCE)

//Getting a resource from the user's pod. The second parameter
 ↪  refers to the class to which the resource will be cast. It can
 ↪  be NonRDFSource, RDFSource, SolidContainer, or any other data
 ↪  class that inherits from one of these classes.
val readResult = srm.read(URI.create(YOUR_RESOURCE_INSTANCE_URL),
 ↪  NonRDFSource::class.java)

//Updaing a resource in the user's pod
val updateResult = srm.update(YOUR_UPDSATE_RESOURCE_INSTANCE)

//Deleting a resource from the user's pod
val deleteResult = srm.delete(YOUR_RESOURCE_INSTANCE)
```

Code Snippet 33: Using Solid Android API library in a third-party application for accessing resources

```
val scdm: SolidContactsDataModule =
 ↪  SolidContactsDataModuleImplementation.getInstance(context)

//Get all Address Books
val addAddressBooks =
 ↪  scdm.getAddressBooks(auth.getProfile().userInfo!!.webId)

//Getting a contact
val contactDetails = scdm.getContact(contactUri).extractResult()

//Rename a contact with a new name
scdm.renameContact(contactUri, newName)

//Getting a group details
scdm.getGroup(groupUri)
```

Code Snippet 34: Using Solid Android API library in a third-party application for using Contacts data module

3.5 Android Solid Services

This module is dedicated to the Android app installed on the user's device. It is in charge of being the single source of truth for third-party apps that are connecting to

solid via Solid Android Client. The code architecture is MVVM and uses different libraries such as:

- Hilt for dependency injection within the app.
- Jetpack Compose that is used for creating the app's UI.
- Jetpack Datastore to save user's info and Access Grants locally.
- Navigation for navigating between different screens in the app.
- Coroutine to do operations and API calls asynchronously.
- Kotlin Serialization to serialize and deserialize working objects.

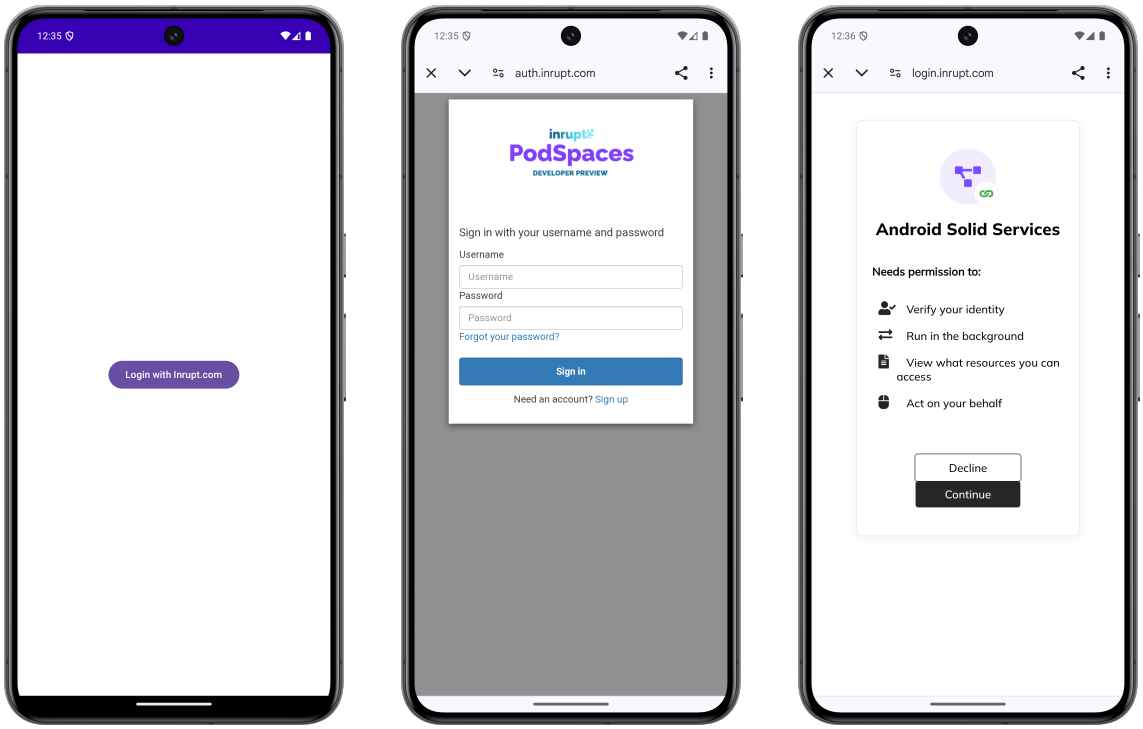
3.5.1 Screens

This app has four main screens as described below and a `Startup` page with the responsibility of checking the user's state at the application's opening stage and deciding to show Login or Main.

3.5.1.1 Login

`Login` page is shown when the user runs the app and has not logged into his Solid pod or logged out previously. It has a button element to log into Inrupt Solid pod space. It uses Solid Android API library for authentication. Same process as code 32 used, and in case of a successful login, transfer the user to the Main page; otherwise, stay on this page.
It uses `androidx.activity.compose.rememberLauncherForActivityResult` for starting an Intent. Two hardcoded variables `AUTH_APP_REDIRECT_URL` and `OIDC_ISSUER_INRUPT_COM` used for indicating the return URL from the browser page after finishing the authentication step and address of the Inrupt OIDC issuer, respectively.

In Figure 4a, a screenshot of the login page can be seen with `Login with Inrupt.com` button which by clicking, moves the user forward to Figure 4b. In this page that shows on a browser that the user has chosen, the login page of the pod provider will be shown. Entering the correct username and password logs into the pod and shows Figure 4c. The user has been asked to give permission to the Android Solid Servies app or deny it. In case of allowing, Main will pop up; otherwise, it will come back to Login page.



(a) Login page

(b) Transfering the user to a browser with the login page of the pod provider

(c) App permission request after user logged in his pod space

Figure 4: Android Solid Services - Login page

3.5.1.2 Main

The main page will be shown after a successful login or entering the app that has already been authenticated. Here, the WebID of the user has been shown with a drop-down menu of all pod spaces connected to that specific WebID. Figure 5 shows the status of this page.

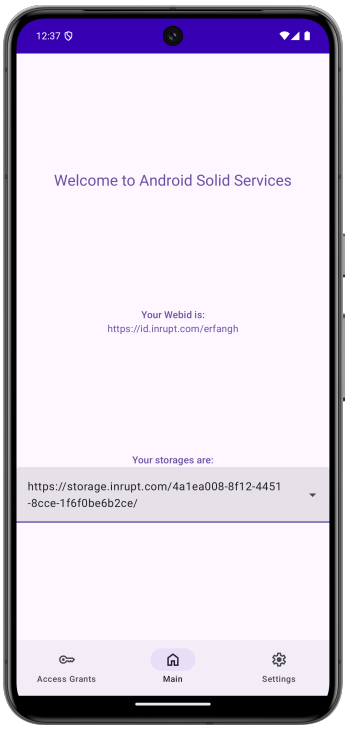


Figure 5: Android Solid Services - Main page

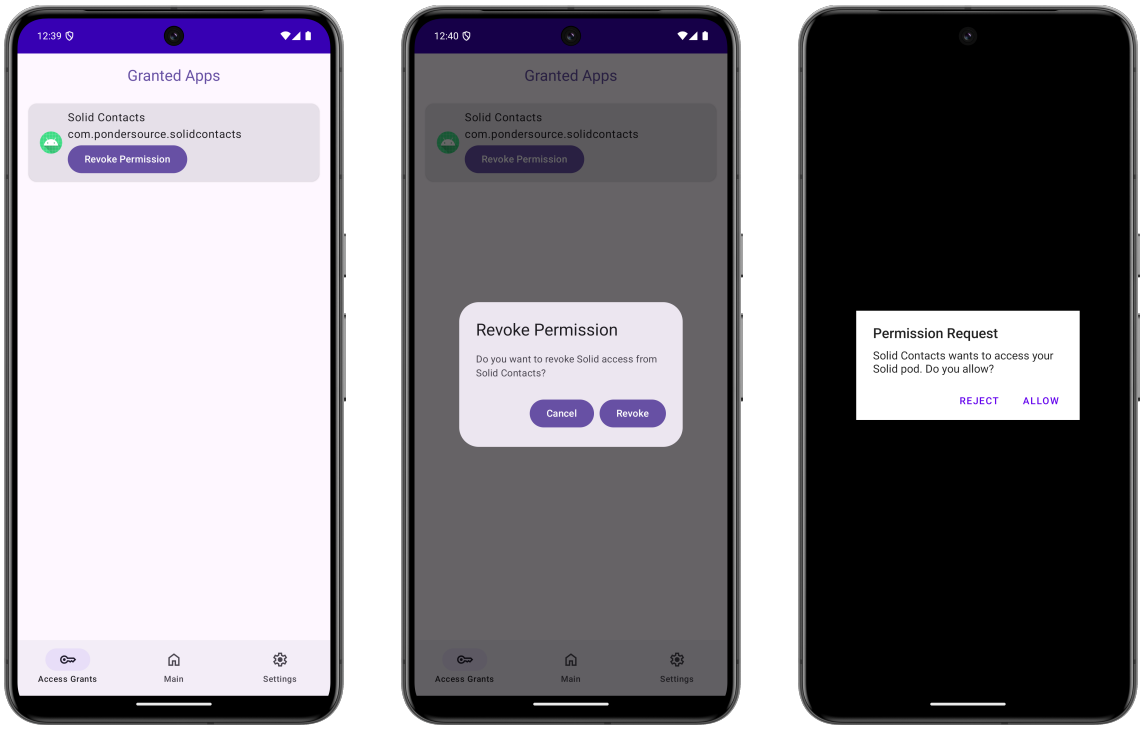
3.5.1.3 Setting

In the Settings page, the user can log out from the account by `Logout` button and relogin with the same or other different pods. A view of the page can be seen in Figure 6.

Figure 6: Android Solid Services - Settings page

3.5.1.4 Access Grants

In the Access Grants tab, the user can see and manage applications to which he has given permission. It contains a list of all applications with their icon, name, and package name. A dialog will pop up by clicking `Revoke Permission`, and the user can revoke the permission from the selected app. They can be seen in Figure 7a and Figure 7b. When a developer uses Solid Android Client library to integrate with Solid, while they ask for permission, a dialog same as Figure 7c will pop up by Android Solid Services app and allow the user to decide to give that third-party app permission or not.



(a) Access Grants page

(b) Revoking permission from an already-accessed app

(c) Request permission from a third-party app

Figure 7: Android Solid Services - Access Grants page

3.5.2 Repositories

3.5.2.1 Access Grant

This repository is created to manage access grants of third-party apps, and it provides the functions in code 35. It can check whether an app has permission or not, permit an app, revoke it, and return a list of all apps that have access granted. These functionalities are used in Access Grants screen. There is an implementation for this interface named `AccessGrantRepositoryImplementation`, which uses `AccessGrantLocalDataSource` interface that handles access grants locally with `SharedPreferences` in Android.

```kotlin
interface AccessGrantRepository {
    fun hasAccessGrant(appPackageName: String): Boolean
    fun addAccessGrant(appPackageName: String, appName: String)
    fun revokeAccessGrant(appPackageName: String)
    fun grantedApplications(): List<GrantedApp>
}
```

Code Snippet 35: `AccessGrantRepository` interface definition

3.5.2.2 Resource Permission

`ResourcePermissionRepository` is used to check each third-party app has access to which resources with which type of access grants such as `CREATE`, `READ`, `UPDATE` and `DELETE`. Only one function is shown in 36. For the current implementation, which is done by `ResourcePermissionRepositoryImplementation`, all apps that log in via Android Solid Services app have access to all resources with all permission types.

```kotlin
interface ResourcePermissionRepository {
    fun hasAccess(
        resourceClaimantPackageName: String,
        resourceUrl: String,
        permissionType: PermissionType
    ): Boolean
}
```

Code Snippet 36: `ResourcePermissionRepository` interface definition

3.5.3 Services

Services in Android Solid Services refers to the APIs that it provides to other apps to connect to it through Android Interprocess Communication. There are also instances of Service which run in the background without any user interface. Services here have been divided into three different parts for specific features. This feature has been used to communicate with this app in Solid Android Client library. In other words, a third-party app that uses Solid Android Client library will use these functions under the hood to ask Android Solid Services for authentication, resource management, and data modules.

3.5.3.1 Authenticator Service

This service provides an implementation for `IASSAuthenticatorService`. Solid SignIn Client starts this service and calls the functions presented here, and under the hood, it uses functions provided by Authentication and Access Grant. The system call `getCallingUid()` used in this call is for retrieving the caller's third-party app's user ID to get the name, package name, and icon. The implementations for methods `hasLoggedIn`, `isAppAuthorized` and `disconnectFromSolid` can be seen in code 37.

Also, how it handles a login request is demonstrated in 38. Line 10 checks if the user has logged into his pod before, and Line 11 checks if the calling app has been authorized before. If yes, it calls `onResult` function of the callback with value `true` which means granted. Show the dialog in 7b if no. In line 2, it checks if the app has permission to show the dialog; it calls system call `canDrawOverlays()` function by passing Android Solid Services app's `context` object.

```
1  override fun hasLoggedIn(): Boolean {
2      return authenticator.isUserAuthorized()
3  }
4
5  override fun isAppAuthorized(): Boolean {
6      return accessGrantRepository.hasAccessGrant(packageManager.getN
       ↪  ameForUid(getCallingUid())!!)
7  }
8  override fun disconnectFromSolid(callback: IASSLogoutCallback) {
9      val packageName =
       ↪  packageManager.getNameForUid(getCallingUid())!!
10     lifecycleScope.launch {
11         accessGrantRepository.revokeAccessGrant(packageName)
12         callback.onResult(true)
13     }
14 }
```

Code Snippet 37: `hasLoggedIn`, `isAppAuthorized` and `disconnectFromSolid` methods implementations

```kotlin
override fun requestLogin(callback: IASSLoginCallback) {
    if (Settings.canDrawOverlays(this@ASSAuthenticatorService)) {
        val packageName =
        ↪  packageManager.getNameForUid(getCallingUid())!!
        val name = packageManager.getApplicationLabel(
            packageManager.getApplicationInfo(
                packageName,
                0
            )
        ).toString()
        if (hasLoggedIn()) {
            if (isAppAuthorized()) {
                callback.onResult(true)
            } else {
                showLoginDialog(
                    packageManager.getNameForUid(getCallingUid())!!,
                    name,
                    callback
                )
            }
        } else {
            callback.onError(SOLID_NOT_LOGGED_IN, "User has not
            ↪  logged in.")
        }
    } else {
        callback.onError(DRAW_OVERLAY_NOT_PERMITTED, "Android Solid
        ↪  Services doesn't have permission to draw overlay. Please
        ↪  ask user to enable overlay drawing for Android Solid
        ↪  Services in app settings.")
    }
}
```

Code Snippet 38: `requestLogin` method implementation

3.5.3.2 Resource Service

This service provides an implementation for `IASSResourceService`, which runs by Solid Resource Client. Inside, it used Authentication, Resource Management and Resource Permission. It uses functions provided by Resource Management, catches the response or its error, and returns it to the caller using `IASSRdfResourceCallback` and `IASSNonRdfResourceCallback` callbacks. The implementation of the function `create` is shown in code 39 to show how it maps them. Method `handleBasicExceptions` controls the basic requirements such as LoggedIn user and having permission for the caller app. In case of not satisfying these basics, calls `onError` function of the callback with the error; otherwise, calls `create` function in Resource Management and returns the result with the callback.

```
override fun create(resource: NonRDFSource, callback:
↪  IASSNonRdfResourceCallback) {
    handleBasicExceptions(
        resource.getIdentifier().toString(),
        packageManager.getNameForUid(getCallingUid())!!,
        PermissionType.CREATE,
        { code, message ->
            callback.onError(code, message)
        }
    ) {
        lifecycleScope.launch(Dispatchers.IO) {
            val result =
                solidResourceManager.create(resource)
            when(result) {
                is SolidNetworkResponse.Success -> {
                    callback.onResult(resource)
                }
                is  SolidNetworkResponse.Error-> {
                    callback.onError(UNKNOWN, result.errorMessage)
                }
                is  SolidNetworkResponse.Exception-> {
                    callback.onError(UNKNOWN,
                     ↪  result.exception.message)
                }
            }
        }
    }
}
```

Code Snippet 39: `create` method implementation

3.5.3.3 Data Modules Service

This service is the implementation for `IASSDataModulesService` and `IASSContactsModuleInterface` which called by Solid Contacts Data Module. It is the same as the other two services in 3.5.3.1 and 3.5.3.2. It provides an instance of `IASSContactsModuleInterface` by calling `solidandroidclientaidlcontacts()` method of `IASSDataModulesService`. This instance's responsibility is to provide a one-to-one implementation of the interface method to return it to Solid Contacts Data Module.

3.6 Solid Android Client

This is one of the modules in this project and is also exported as an Android library in both Maven Central and Github. The responsibility of this library is to provide some APIs for third-party apps to connect to Android Solid Services and finally to the user's Solid pod. It provides functionalities of Solid Android API library with

the difference that Android Solid Services will do all the work and the APIs would be more straightforward. For instance, for authentication, the developer does not need to ask the user for authentication for every app and pass the user through entering username and password and doing Solid-OIDC. So it works this way: a user installs Android Solid Services app and signs into his pod, and this app becomes the single source of truth. All other third-party apps, including this library, will use provided APIs for requesting access grants with much simpler UX, resources, and contact data modules.

3.6.1 Solid SignIn Client

This singleton class provides authentication-related functionalities. In the time of construction, it starts running Authenticator Service using Intent demonstrated in code 40 and a callback to observe the connection of the service.

```
private val serviceConnection = object : ServiceConnection {
    override fun onServiceConnected(className: ComponentName,
    ↪  service: IBinder) {
        iASSAuthService =
        ↪  IASSAuthenticatorService.Stub.asInterface(service)
        connectionFlow.value = true
    }

    override fun onServiceDisconnected(className: ComponentName) {
        iASSAuthService = null
        connectionFlow.value = true
    }
}
...
val intent = Intent().apply {
    setClassName(
        ANDROID_SOLID_SERVICES_PACKAGE_NAME,
        ↪  //com.pondersource.androidsolidservices
        ANDROID_SOLID_SERVICES_AUTH_SERVICE //com.pondersource.and
        ↪  roidsolidservices.services.ASSAuthenticatorService
    )
}
context.bindService(intent, serviceConnection,
↪  Context.BIND_AUTO_CREATE)
```

Code Snippet 40: Starting Authenticator Service from Solid Android Client

It has a method named `authServiceConnectionState` that returns a `Flow` of the service connection state to observe the connection and, if the service has been connected, call the desired functions.

`checkConnectionWithASS` function checks if the Android Solid Services app has been installed, the service has been connected, and the user has already logged in

his pod, and the continuing procedure has been passed to in the input, otherwise throwing an Exceptions.

For getting access permission, `requestLogin` method can be called with a callback that has two parameters of `granted` and `exception` for returning the result.

For disconnecting from an already-connected app, `disconnectFromSolid` can be called, which inside calls the input callback with the result of disconnecting.

If a third-party app has already connected to Solid and the user relaunches the app, a developer can call `getAccount()` function, and if the result is not null, they can proceed to show functionalities. If it is null, the developer needs to make a login request. It also may throw an `Exception` from Exceptions in case something goes wrong.

The signature of the functions above can be seen in code 41.

```
fun authServiceConnectionState(): Flow<Boolean> {...}

fun checkConnectionWithASS(onContinue: () -> Unit) {...}

@Throws(Exception::class)
fun getAccount() : SolidSignInAccount? {...}

fun requestLogin(callBack: (Boolean?, SolidException?) -> Unit)
 ↪  {...}

fun disconnectFromSolid(callBack: (Boolean) -> Unit) {...}
```

Code Snippet 41: `SolidSignInClient` public functions signature

3.6.2 Solid Resource Client

Like other classes that connect to a service in Android Solid Services, this class needs to bind to the correspondence service via an Intent. The code in 40 has been repeated here because the service address differs with value `com.pondersource.androidsolidservices.servi` and the service type is `IASSResourceService`. It also has a function that returns `Flow<Boolean>` that observes the service connection state. Here, this method is called `resourceServiceConnectionState`.
`IASSRdfResourceCallback` and `IASSNonRdfResourceCallback` is also used here to get resource results from the service. The caller also uses `SolidResourceCallback<T>`, which is a generic callback and has two methods as other callbacks, and the result type is `T` that should be inherited from one of `RDFSource` or `NonRDFSource`.

This class has five main functions named `getWebId`, `read`, `create`, `update`, and `delete`. `getWebId` takes a callback of type `SolidResourceCallback<WebId>` and beneth, calls `getWebId` method from `IASSResourceService`; The result is a type of `RDFSource` and then construct `WebId`. The other four methods somehow work similarly internally but with different methods. First, they check that the input object is a type of `RDFSource` or `NonRDFSource` and then call the respective function for that type in the `IASSResourceService`. As a use case, the method `read` has been shown in 42.

```
fun <T: Resource> read(resourceUrl: String, clazz: Class<T>,
↪  callback: SolidResourceCallback<T>) {
    checkBasicConditions()
    if (RDFSource::class.java.isAssignableFrom(clazz)) {
        iASSAuthService!!.readRdf(resourceUrl, object:
        ↪  IASSRdfResourceCallback.Stub() {
            override fun onResult(result: RDFSource) {
                if (clazz.isInstance(result)) {
                    callback.onResult(result as T)
                } else {
                    val returnValue = //create an instance of class
                    ↪  clazz
                    callback.onResult(returnValue)
                }
            }
            override fun onError(errorCode: Int, errorMessage:
            ↪  String) {
                callback.onError(handleSolidResourceException(error
                ↪  Code, errorMessage))
            }
        })
    } else if (NonRDFSource::class.java.isAssignableFrom(clazz)) {
        iASSAuthService!!.read(resourceUrl, object:
        ↪  IASSNonRdfResourceCallback.Stub() {
            override fun onResult(result: NonRDFSource) {
                if (clazz.isInstance(result)) {
                    callback.onResult(result as T)
                } else {
                    val returnValue = //create an instance of class
                    ↪  clazz
                    callback.onResult(returnValue)
                }
            }
            override fun onError(errorCode: Int, errorMessage:
            ↪  String) {
                callback.onError(handleSolidResourceException(error
                ↪  Code, errorMessage))
            }
        })
    } else {
        throw SolidResourceException.NotSupportedClassException("Ob
        ↪  jects which are RDFSource or NonRDFSource or inherited
        ↪  from them can be read.")
    }
}
```

Code Snippet 42: `read()` function implementation in `SolidResourceClient`

3.6.3 Solid Contacts Data Module

Like the other two above, this class starts a service and observes its connection. All data modules are provided in one service with the name `IASSDataModulesService`, and instances of each data module interface can be taken from here. As only Contacts Data Modules has been implemented, this interface has only one method named `getContactsDataModuleInterface()`. So when it starts the service, via Intent, same as above, it takes the Contacts interface as soon as it has connected. As shown in `IASSContactsModuleInterface`, there are various methods to be called from this interface; each has a similar one in this class.

The functions working with this data module are `suspend` and should be called from the background thread or using a Coroutine. As all Services are sequential methods and need a callback to get the result, beneath the methods here, a `suspendCoroutine` has been initialized to convert sequential programming into asynchronous. For instance, `getAddressBook` function can be analyzed based on code shown in 43. The function input is a string that refers to the desired address book uri. Inside the function, it checks the service connectivity and throws proper Exceptions in case it has not been connected. The call `getAddressBook` function from `IASSContactsModuleInterface` with an on-call instantiated callback to get the result from the service and then return it to the coroutine. The inline function `suspendCoroutine` makes this call an asynchronous method.

```
1  suspend fun getAddressBook(
2      uri: String,
3  ): AddressBook? {
4      checkService()
5      return suspendCoroutine { continuation ->
6          iASSContactsModuleInterface!!.getAddressBook(uri, object :
           ↪ IASSContactModuleAddressBookCallback.Stub() {
7              override fun valueChanged(addressBook: AddressBook?) {
8                  continuation.resume(addressBook)
9              }
10         })
11     }
12 }
```

Code Snippet 43: `getAddressBook(...)` implementation in `com.pondersource.solidandroidclient.sdk.SolidContactsDataModule`

3.6.4 AIDL Definitions

In the Android environment, different applications can communicate with a standard language named Android Interface Definition Language (AIDL); before any communication, an app should define the functions and parameters to tell other applications how to call, and other apps should also know about these definitions. In this project, these definitions have been declared in `com.pondersource.solidandroidclient` package and in `aidl` subfolder. This is because Android Solid Services has the implementation of these interfaces, and third-party apps that include Solid Android

Client in their source code need to know about these definitions. The main interface definitions are as follows:

3.6.4.1 `IASSAuthenticatorService`

This interface is used to request an access grant from Android Solid Services app. The functions are used as follows, and the code can be found in 44:

- `hasLoggedIn():Boolean` checks if user has logged into Android Solid Services app.
- `isAppAuthorized():Boolean` checks whether the calling third-party app is authorized to access the logged-in Solid pod or not.
- `requestLogin(callback:IASSLoginCallback)` is used to ask Android Solid Services app from the calling third-party app to give access to the pod.
- `disconnectFromSolid(callback:IASSLogoutCallback)` is used to request from the calling third-party app to disconnect from the already-accessed Solid pod.

```
1 interface IASSAuthenticatorService {
2     boolean hasLoggedIn();
3     boolean isAppAuthorized();
4     void requestLogin(IASSLoginCallback callback);
5     void disconnectFromSolid(IASSLogoutCallback callback);
6 }
```

Code Snippet 44: `IASSAuthenticatorService` definition

3.6.4.2 `IASSResourceService`

This interface is designed for requesting a resource from Android Solid Services app. A function is also added for more straightforward use to get the user's `WebId`. Then, four functions for CRUD operations on `NonRDFSource` and another four for `RDFSource`. The interface definition is shown in code 45. `in` keyword is for input objects.

```
interface IASSResourceService {
    void getWebId(IASSRdfResourceCallback callback);
    void create(in NonRDFSource resource,
    ↪ IASSNonRdfResourceCallback callback);
    void createRdf(in RDFSource resource, IASSRdfResourceCallback
    ↪ callback);
    void read(String resourceUrl, IASSNonRdfResourceCallback
    ↪ callback);
    void readRdf(String resourceUrl, IASSRdfResourceCallback
    ↪ callback);
    void update(in NonRDFSource resource,
    ↪ IASSNonRdfResourceCallback callback);
    void updateRdf(in RDFSource resource, IASSRdfResourceCallback
    ↪ callback);
    void delete(in NonRDFSource resource,
    ↪ IASSNonRdfResourceCallback callback);
    void deleteRdf(in RDFSource resource, IASSRdfResourceCallback
    ↪ callback);
}
```

Code Snippet 45: `IASSResourceService` definition

3.6.4.3 `IASSContactsModuleInterface`

This interface has a map of functions in Solid Contacts Data Module with the difference that parameters and structure changed due to AIDL limitations. The source code of this interface can be seen in codes 46 and 47.

```
interface IASSContactsModuleInterface {

    void getAddressBooks(IASSContactModuleAddressBookListCallback
    ↪  callback);
    void createAddressBook(
            String title,
            boolean isPrivate,
            IASSContactModuleAddressBookCallback callback,
            @nullable String storage,
            @nullable String ownerWebId,
            @nullable String container
    );
    void getAddressBook(String uri,
    ↪  IASSContactModuleAddressBookCallback callback);
    void deleteAddressBood(
        String uri,
        @nullable String ownerWebId,
        IASSContactModuleAddressBookCallback callback
    );
    void createNewContact(
        String addressBookUri,
        in NewContact newContact,
        in List<String> groupUris,
        IASSContactModuleFullContactCallback callback
    );
    void getContact(
        String contactUri,
        IASSContactModuleFullContactCallback callback
    );
    void renameContact(
            String contactUri,
            String newName,
            IASSContactModuleFullContactCallback callback
    );
    void addNewPhoneNumber(
        String contactUri,
        String newPhoneNumber,
        IASSContactModuleFullContactCallback callback
    );
    void addNewEmailAddress(
        String contactUri,
        String newEmailAddress,
        IASSContactModuleFullContactCallback callback
    );
```

Code Snippet 46: `IASSContactsModuleInterface` definition - part 1

```
    void removePhoneNumber(
        String contactUri,
        String phoneNumber,
        IASSContactModuleFullContactCallback callback
    );
    void removeEmailAddress(
        String contactUri,
        String emailAddress,
        IASSContactModuleFullContactCallback callback
    );
    void deleteContact(
        String addressBookUri,
        String contactUri,
        IASSContactModuleFullContactCallback callback
    );
    void createNewGroup(
            String addressBookUri,
            String title,
            in List<String> contactUris,
            IASSContactModuleFullGroupCallback callback
    );
    void getGroup(
        String groupUri,
        IASSContactModuleFullGroupCallback callback
    );
    void deleteGroup(
        String addressBookUri,
        String groupUri,
        IASSContactModuleFullGroupCallback callback
    );
    void addContactToGroup(
        String contactUri,
        String groupUri,
        IASSContactModuleFullGroupCallback callback
    );
    void removeContactFromGroup(
        String contactUri,
        String groupUri,
        IASSContactModuleFullGroupCallback callback
    );
}
```

Code Snippet 47: `IASSContactsModuleInterface` definition - part 2

3.6.4.4 `IASSRdfResourceCallback` and `IASSNonRdfResourceCallback`

These two callbacks return results from Solid Resource Client inside Solid Android Client. Both of them have two methods named `onResult` and `onError` in which the second method is identical in both cases and the first method has different input types; `RDFSource` for `IASSRdfResourceCallback` and `NonRDFSource` for `IASSNonRdfResourceCallback`. As an example of how they look, the source of `IASSRdfResourceCallback` has been shown in 48.

```
interface IASSRdfResourceCallback  {
    void onResult(in RDFSource result);
    void onError(int errorCode, String errorMessage);
}
```

Code Snippet 48: `IASSRdfResourceCallback` interface definition

3.6.5 Exceptions

When a third-party app uses this library, apart from general `Exceptions`, one of these specific `Exceptions` can be thrown:

- `SolidAppNotFoundException`: If user has not installed Android Solid Services app on their phones.
- `SolidNotLoggedInException`: Thrown when the user has not logged in his pod in Android Solid Services.
- `SolidServiceConnectionException`: If the library can not connect to one of the services in Services.
- `SolidServicesDrawPermissionDeniedException` Shown when Android Solid Services App does not have permission to draw overlay; it can be asked the user to enable it from the app permissions section in settings.
- `NotSupportedClassException`: Thrown when a developer asks for a resource that is not inherited from either `NonRDFSource` or `RDFSource`.
- `NullWebIdException`: When WebId is null.
- `NotPermissionException`: Appear when the third-party app cannot access the requested resource.
- `UnknownException`: Thrown when the Exception is none of the above use cases.

3.6.6 How to use

This library, the same as Solid Android API, has been published to Maven Central and can be imported. Developers must import it with the latest version (currently 0.2.0) into their app-level `build.gradle.kts` file as shown in code 49.

```
dependencies {
    ...
    implementation("com.pondersource.solidandroidclient:solidandroidclient:0.2.0")
}
```

Code Snippet 49: Importing Solid Android Client library to a third-party application

After importing the library and syncing the project, before using Solid Resource Client or Solid Contacts Data Module, the developer needs to check the authentication state with Android Solid Services. A sample code in 50 has been provided.

```
val ssc: SolidSignInClient = Solid.getSignInClient(context)
CORROUTINES_SCOPE.launch {
    ssc.authServiceConnectionState().collect { hasConnected ->
        if(hasConnected) { //The Auth service is connected and functions can be called
            try {
                if(ssc.getAccount() != null) { //The app already has the permission and can work with pod resources
                } else {
                    //The app needs to ask permission
                    ssc.requestLogin { granted, exception ->
                        if (exception == null) {
                            if (granted == true) { //Have permission to get use pod resources
                            } else { //The user refused to permit this app
                            }
                        } else { //Exception handling
                        }
                    }
                }
            } catch (e: Exception) {
                //Exception handling
            }
        } else {
            //Can not connect to the Auth service
        }
    }
}
```

Code Snippet 50: How to use `SolidSignInClient` in a third-party application

After being sure the user gave the permission, a developer can do CRUD operations by using Solid Resource Client. To do so, we can check the service connectivity status and do the desired operation on a resource. For example, a developer can take inspiration from the code in 51.

```
val src: SolidResourceClient = Solid.getResourceClient(context)
src.resourceServiceConnectionState().collect { hasConnected ->
    if(hasConnected) {
        //The service has been connected
        src.getWebId(object: SolidResourceCallback<WebId>{
            override fun onResult(webid: WebId) {//Ready to use
            ↪  webid}
            override fun onError(exception:
            ↪  SolidException.SolidResourceException) {//Exception
            ↪  handling}
        })

        val myCallback = object: SolidResourceCallback<MyObject>{
        ↪  // class MyObject: RDFSource
            override fun onResult(myObject: MyObject) {/*Ready to
            ↪  use myObject*/}
            override fun onError(exception:
            ↪  SolidException.SolidResourceException) {/*Exception
            ↪  handling*/}
        }
        solidResourceClient.read(MY_OBKECT_URI_ON_POD,
        ↪  MyObject::class.java, myCallback)
        solidResourceClient.create(myObject, myCallback)
        solidResourceClient.update(myObject, myCallback)
        solidResourceClient.delete(myObject, myCallback)
    } else {
        //The service has not been connected yet
    }
}
```

Code Snippet 51: How to use `SolidResourceClient` in a third-party application

For using Solid Contacts Data Module, a developer can do the same procedure but with `SolidContactsDataModule` class. A sample of using it has been demonstrated in 52.

```
val scdm: SolidContactsDataModule =
    Solid.getContactsDataModule(context)
scdm.contactsDataModuleServiceConnectionState().collect {
    hasConnected ->
    if(hasConnected) {
        //The service has been connected
        CORROUTINE_SCOPE.launch {
            val allAddressBooks = scdm.getAddressBooks()
            val firstPrivateAddressBook = scdm.getAddressBook(allAd
                dressBooks.privateAddressBookUris[0])
            val contact = scdm.getContact(firstPrivateAddressBook.c
                ontacts[0].uri)
            val group =
                scdm.getGroup(firstPrivateAddressBook.groups[0].uri)
            //Other functions in the Contacts Data Module can be
                called the same as above
        }
    } else {
        //The service has not been connected yet
    }
}
```

Code Snippet 52: How to use `SolidContactsDataModule` in a third-party application

4 Usecase

This chapter is dedicated to developing a use case that shows how the results are used. The use case is an Android application for contacts named Solid Contacts. It uses Solid Android Client library to develop an application working with address books and contacts and storing them in the user's Solid pod. This project can be accessed through its Github repository. For developing this app, these libraries have been used:

- Hilt for dependency injection
- Navigation for creating navigation graphs for different screens
- Jetpack Compose for creating UI
- Solid Android Client for connecting to Solid pod and using Solid Contacts Data Module for working with Address books.

It has features such as:

- Connect to user's Solid pod
- Viewing public and private address books
- Adding a new address book
- Adding a contact or group to an address book
- Removing a contact or group from an address book
- Seeing a contact or group details

This application has been developed by referring to A solution in 3.

4.1 Screens

4.1.1 Login and Settings

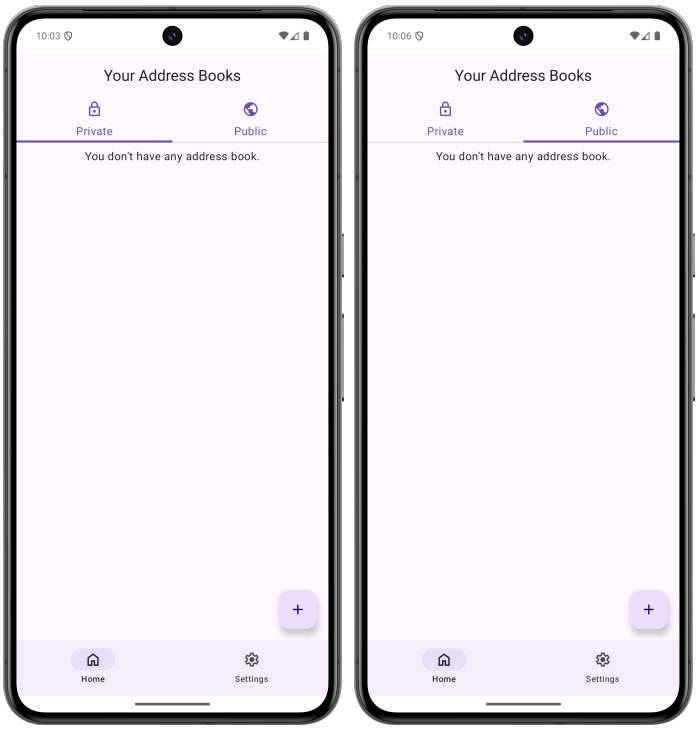
The Login page shows a button with "Connect to Solid" text and uses Solid SignIn Client to request a login. After logging in, it transfers the user to Adress Books page. Also, there is a button on the Settings page that again uses Solid SignIn Client for disconnecting from the pod. Two screenshots from these two pages can be seen in Figure 8.

(a) Login page (b) Settings page

Figure 8: Solid Contacts - Login and Settings pages

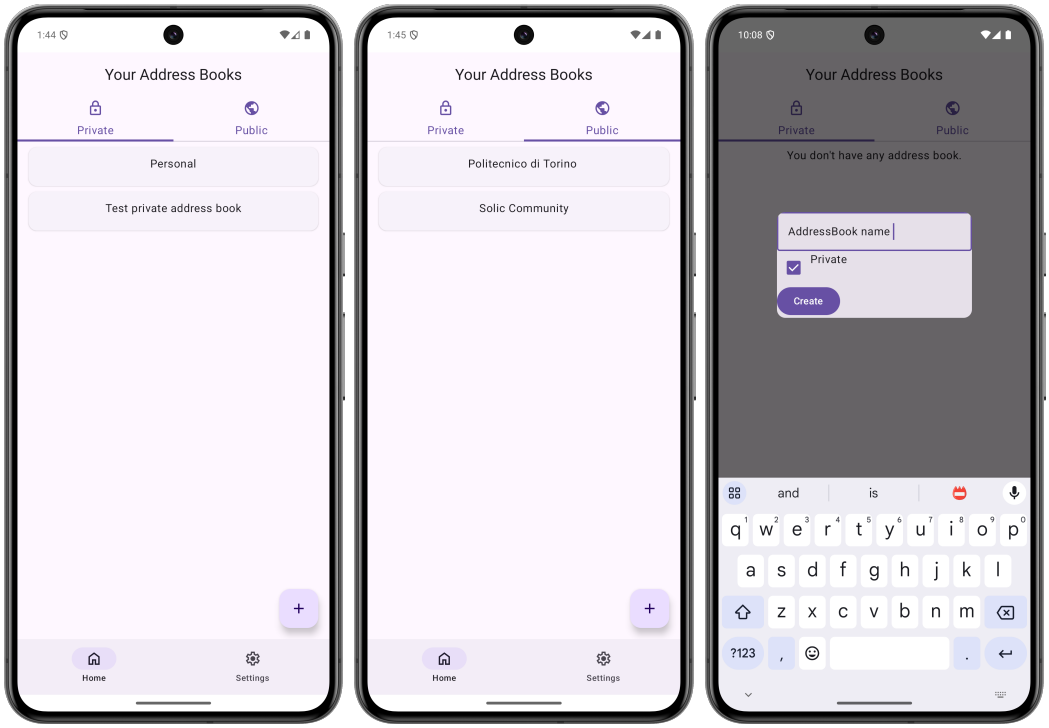
4.1.2 Adress Books

On this page, all the users' private and public address books on their pod will be shown. Figure 9a and Figure 9b show private and public address books when they are empty, while Figure 9c and Figure 9d demonstrate when there are some private and public address books. For creating a new address book, the plus button on the bottom right corner can be clicked, and a dialog such as Figure 9e will pop up with the address book's name field and a checkbox for making it private or public. Tapping on one of the address books transfers the user to the Address Book page.



(a) Empty private address books page

(b) Empty public address books page



(c) Private address books page

(d) Public address books page

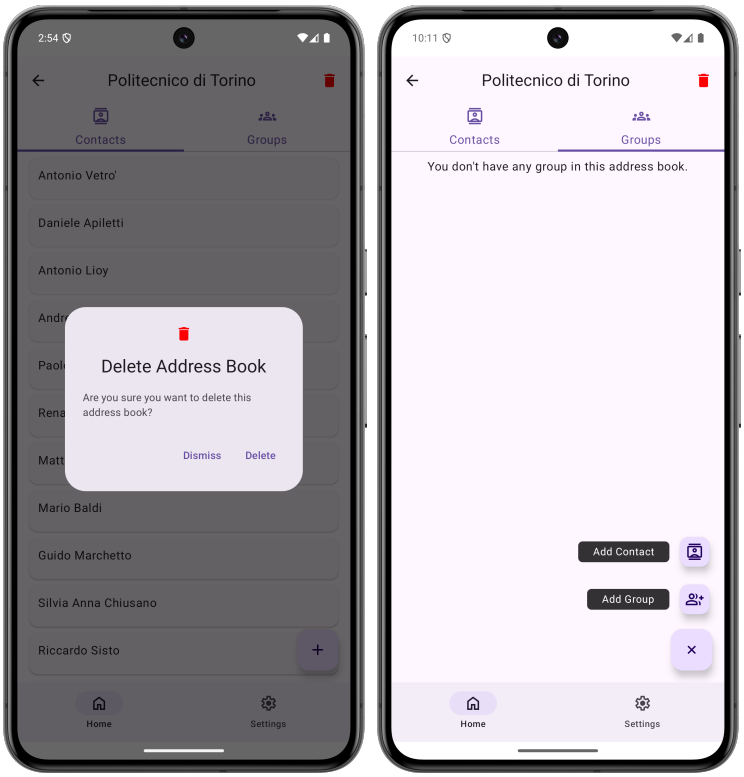
(e) Add a new AddressBook

Figure 9: Solid Contacts - AddressBooks pages

4.1.3 Address Book

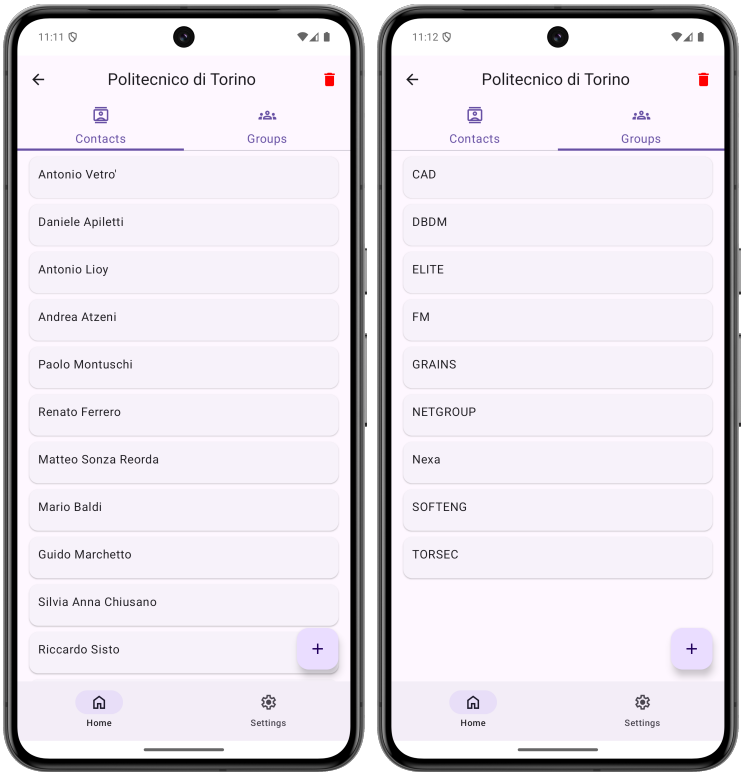
This page shows details about the current address book. On top, the name can be seen, and in the top right corner, there is a red bin icon which, by clicking, shows a

dialog to the user, same as Figure 10a and the user can decide to delete it or cancel the deletion. By clicking on the plus button in the bottom right corner, two other buttons will appear, same as Figure 10b, that the user can add a contact or group to his address book. "Add Contact" transfers the user to Figure 11a, and "Add Group" navigates him to Figure 12a. Two tabs show lists of contacts and groups already existing in the address book. They have been shown in Figure 10c and Figure 10d. By tapping a contact, it goes to the contact's details in Figure 11b, and tapping on a group item results in showing a group page in Figure 12b.



(a) Delete address book dialog

(b) Add a new contact or group button



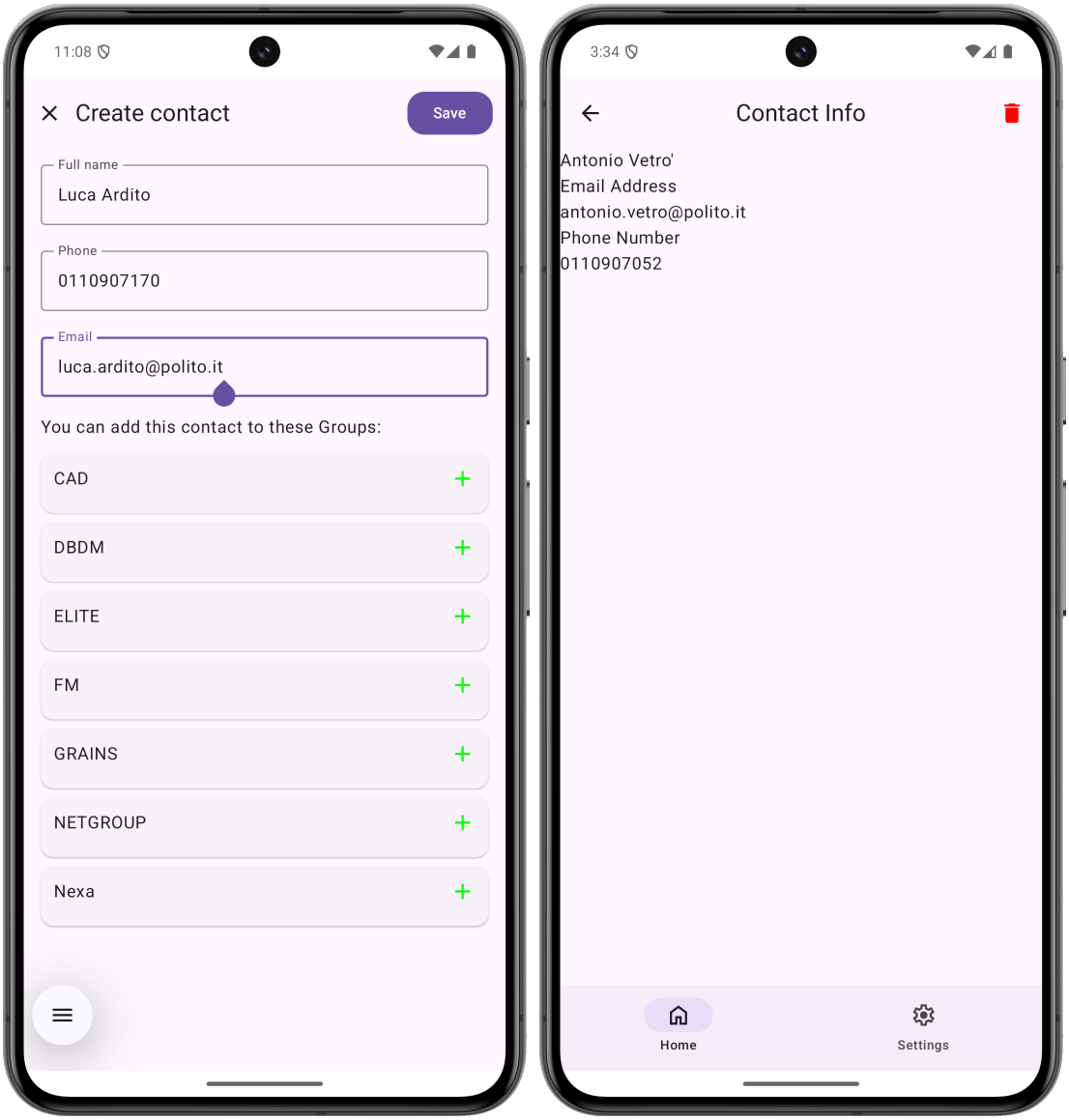
(c) Contacts list tab

(d) Groups list tab

Figure 10: Solid Contacts - AddressBook pages

4.1.4 Contact

Contact consists of two pages, one for creating a new contact and another for showing contact details. Figure 11a refers to the page where the user can put a name, a phone number, and an email for a new contact and include or exclude groups that are in the same address book by tapping the plus or cross icon on the right side of the group item. The cross button on the top left corner of the page results in discarding entered information and returning to the previous page, and "Save" creates the contact. All details of the selected contact are visible on the contact details page that can be seen in Figure 11b. Tapping the red bin button on the top right side of the screen results in contact deletion, and turning back to the previous page will allow the back button to navigate back in the navigation graph.

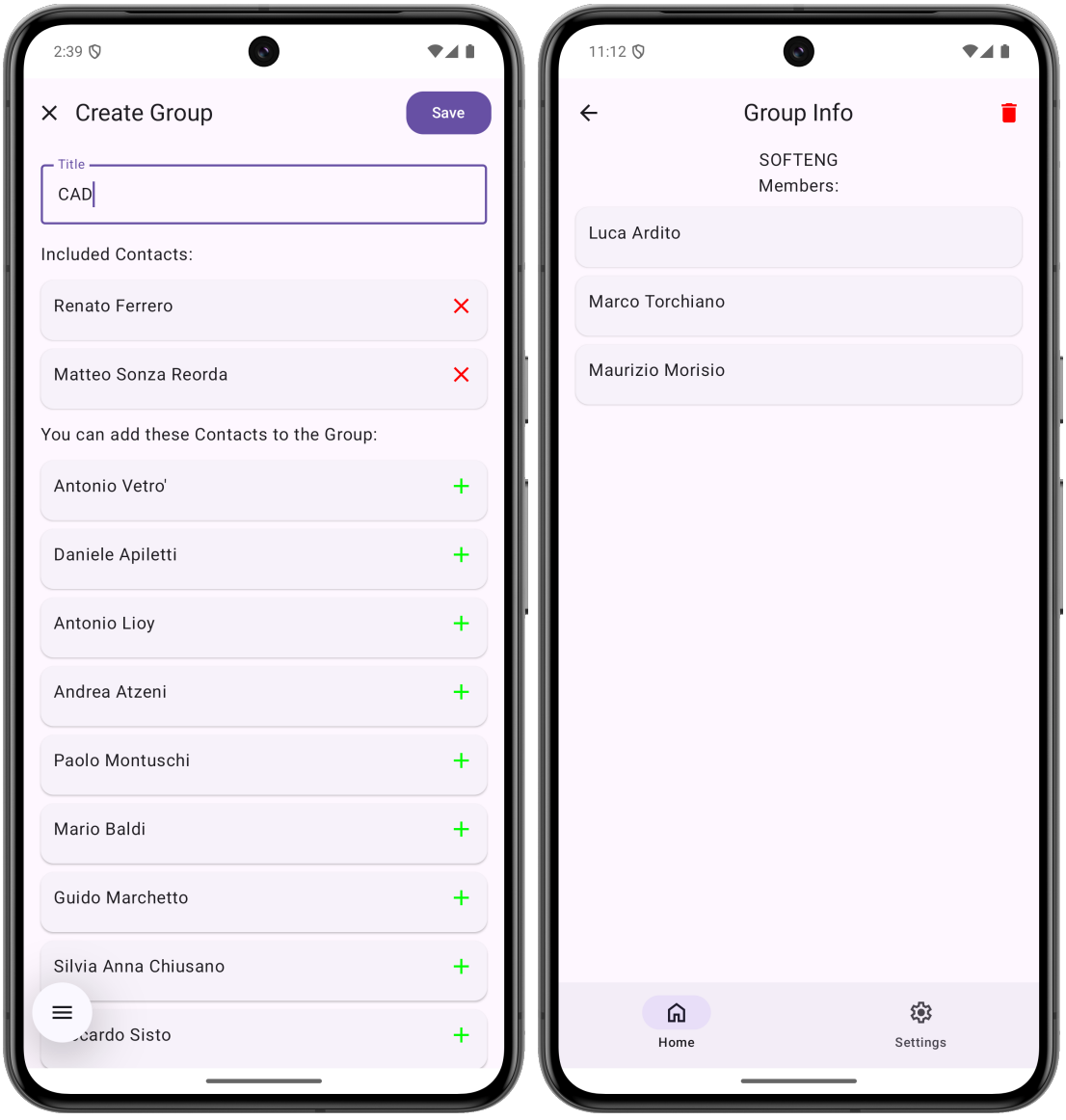


(a) Add a new contact page (b) Contact details page

Figure 11: Solid Contacts - Contact pages

4.1.5 Group

For groups in an address book, like Contacts, two pages exist: One for creating a new one and the other for showing group details. In the "Create Group" page showing in Figure 12a, the user can set a title and include or exclude contacts existing in the address book into the group, and by tapping "Save," create the group. In Figure 12, which shows the group name and its members, use has some abilities such as deleting it or clicking a member item and transferring to Figure 11b to see member details.



(a) Add a new group page (b) Group details page

Figure 12: Solid Contacts - Group pages

4.2 Analyzing Solution

This app shows a small use case for using Solid in the Android environment. The main feature is that all data will be saved in the user's pod, and he can edit or delete it at any time with this app or any other application that has been integrated into Solid. It has the highest level of privacy as a user can manage the data at any

given time, and no third party can store his data. If we compare it to other contact providers such as Google or Apple, the user can delete the data on these platforms, but he can not be sure whether it has been deleted from the platform's server. It is also a shift in the apps as it decouples the service and the data; Apps would be more service, and the data they need can be fed from the user's pod with the user's permission.

5 Limitations and Future Works

5.1 Limitations

Some limitations affected the speed of the work progress, the quality of the results, and the features provided.

The first main obstacle is the lack of any library that works with Linked Data. Most tools have been developed for web-based applications, and those written for Java environment have many conflicts with the Android development ecosystem. Also, the library used in this work, Titanium JSON-LD[13], lacks enough features, data conversion, and a variety of data format support. It made it difficult to work only with JSON-LD with an incomplete set of functions for creating a dataset from and Resource Description Framework (RDF) resources.

The second significant limitation is the features provided by Inrupt Java Client[11] library, which can not be used for handling authentication and authorization in the Android environment and can not save the state of the user based on `Client` classes provided. It made it more challenging to save the user state and added overhead of low-level token requests and token refreshing. Also, resource management is limited to working with `Streams` and not higher level classes such as RDF resources, which have been provided in this library due to using incompatible libraries such as Jena.

Another limitation caused by third-party libraries is that OpenID Android AppAuth library [12] handles authentications only based on Bearer tokens and not DPoP ones, which are a requirement in some Solid pod providers.

Due to the open source characteristics of Solid, different WebID and Pod providers use different versions of Solid server. These different versions use some features that do not exist in previous versions or vice versa. Data modules in the Solid community are still experimental and do not have an all-agreed standard. This work just started around some simple and small use cases such as Contacts, Calendar, Bookmarks, etc., and it needs more work to support complex data formats and be included in this work, too.

The Solid community is a small community mostly focused on web application solutions. During the work, there were many deadlocks that no one could help with, making the job much slower and taking more time than planned.

5.2 Future Works

There are many open fields regarding future work, such as overcoming some of the limitations or providing new features.

Developing a RDF-based processor library dedicated to Android or even mobile environment, developed by tools such as Kotlin Multiplatform, can help have a more straightforward way of working with Linked Data (LD) resources and having more functions for different uses.

Expanding OpenID open-source implementation in the Android environment to support more secure standards such as DPoP token.

The work in this project provides the basic functionalities. It has an open field for many features such as adding new data modules, adding offline-first, improving APIs provided to developers, improving the useability of Android Solid Services app, and others.

Adding support for Solid Access Control List (ACL) in Android and integrated into this project to manage sophisticated use cases of sharing resources with third-party applications or even entities outside the Solid ecosystem.

This groundwork can be used to develop user-centric applications that focus on providing some services based on users' data in their pods. At this moment, there is an initial effort to use this work and create an open-source digital wallet for users that totally depends on data provided by users in their pod spaces.

6 Conclusion

This thesis focused on integrating the Solid project into native Android development, tackling the challenges of integrating a data-based and privacy-first solution into the Android ecosystem. Solid gives users control over their data through Personal Online Data Stores (Pods), but its current ecosystem is primarily built around web technologies and web applications. Integrating Solid project with Android requires resolving several obstacles, such as authentication complexities, incompatibility of existing tools developed by Solid community and open-source projects, and the lack of direct support for Solid's protocols in Android's development environment.

To overcome these issues, a modular solution was developed with three key components: Solid Android API, Android Solid Services, and Solid Android Client. The Solid Android API enables authentication, resource management, and Contacts data module support, while the Android Solid Services Android app acts as a central authentication point on the user's phone, making it easier for third-party applications to interact with Solid Pods. The Solid Android Client provides a streamlined way for developers to integrate Solid into their Android applications without dealing with the complexities of authentication and data management.

A practical use case—a Solid-powered address book application—was implemented to test the solution. The results showed that it is possible to integrate Solid into Android while maintaining security, privacy, and interoperability. However, challenges remain, such as optimizing performance for mobile environments, adding more functionalities for the app and libraries, extending support for different LD formats, improving compatibility with existing Solid libraries, and making the development process more intuitive for Android developers.

Despite all these challenges, this project is the basis for bringing Solid project into Android development. Future work could involve refining the current APIs, producing documentation for developers, and adding other data modules. Work of Solid project adoption in the mobile ecosystem achieves the desired goal of giving users more control over their data management throughout the Internet and enhancing privacy.

Bibliography

[1] Tim Berners-Lee. *TAG Explainer for Solid*. URL: https://timbl.solidcommunity.net/2023/03%20EWADA/TAGExplainerforSolid.html.

[2] Tim Berners-Lee. *The Web - Take 3*. URL: https://solidproject.org/take3.

[3] Wikipedia. *Facebook–Cambridge Analytica data scandal*. URL: https://en.wikipedia.org/wiki/Facebook%E2%80%93Cambridge_Analytica_data_scandal.

[4] Katrina Brooker. *"I Was Devastated": Tim Berners-Lee, the Man Who Created the World Wide Web, Has Some Regrets*. URL: https://www.vanityfair.com/news/2018/07/the-man-who-created-the-world-wide-web-has-some-regrets.

[5] Eleni Sharp. *Personal data stores: building and trialling trusted data services*. URL: https://www.bbc.co.uk/rd/blog/2021-09-personal-data-store-research.

[6] Chris Morris. *Data Breaches Continue to Skyrocket in 2022*. URL: https://www.nasdaq.com/articles/data-breaches-continue-to-skyrocket-in-2022.

[7] Ahmed Sherif. *Market share of mobile operating systems worldwide 2009-2024, by quarter*. URL: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/#statisticContainer.

[8] Solid Community Group. *Solid Protocol*. URL: https://solidproject.org/ED/protocol#webid.

[9] Solid Community Group. *Solid-OIDC*. URL: https://solidproject.org/TR/oidc.

[10] Solid Community Group. *Solid Technical Reports*. URL: https://solidproject.org/TR.

[11] Inrupt. *Inrupt Java Client*. Version 1.2.0. URL: https://github.com/inrupt/solid-client-java.

[12] OpenID. *OpenID Android*. Version 0.11.1. URL: https://github.com/openid/AppAuth-Android.

[13] https://github.com/filip26. *Titanium JSON-LD 1.1 Processor & API*. Version 1.4.1. URL: https://github.com/filip26/titanium-json-ld.

[14] Wikipedia. *Intent (Android)*. URL: https://en.wikipedia.org/wiki/Intent_(Android).

List of Source Codes

List of Figures

Glossary

Activity is a fundamental Android component that manages the user interface and interactions of an application. 15, 16

Bearer token is a type of token used for authentication and authorization and is used in web applications and APIs to hold user credentials and indicate authorization for requests and access. 15, 72

Broadcast Receiver is one of the components in Android that enable apps to listen for and respond to broadcast messages from other apps or the system itself. 15

Content Provider is a way for different Android apps to share information with each other. 15

Coroutine A coroutine is a concurrency design pattern that you can use on Android to simplify code that executes asynchronously. 44, 55

Desugaring allows lower API levels to work with new Java libraries. 16

DPoP Demonstrating Proof of Possession (DPoP) is a relatively simple mechanism for sender-constraining access tokens. It gives a way to tie an access token to the client that originally receives it from the authorization server. This means that the access token is no longer a bearer token. 15, 72

Hilt Hilt is a dependency injection library for Android that reduces the boilerplate of doing manual dependency injection in Android projects. 44, 64

Intent in the Android operating system is a software mechanism that allows users to coordinate the functions of different activities to achieve a task[14]. 28, 31, 44, 52, 53, 55, 76

IPFS InterPlanetary File System (IPFS) is a modular suite of protocols for organizing and transferring data, designed from the ground up with the principles of content addressing and peer-to-peer networking. Because IPFS is open-source, there are multiple implementations of IPFS. While IPFS has more than one use case, its main use case is for publishing data (files, directories, websites, etc.) in a decentralised fashion. 10

Jackson is a high-performance JSON processor for Java. Its developers extol the combination of fast, correct, lightweight, and ergonomic attributes of the library. 16

Jena is a Java framework for building Semantic Web applications. It provides an extensive Java libraries for helping developers develop code that handles RDF, RDFS, RDFa, OWL and SPARQL in line with published W3C recommendations. Jena includes a rule-based inference engine to perform reasoning based on OWL and RDFS ontologies, and a variety of storage strategies to store RDF triples in memory or on disk. 15, 16, 72

Jetpack is a suite of libraries to help developers follow best practices, reduce boilerplate code, and write code that works consistently across Android versions and devices so that developers can focus on the code they care about. 14

Jetpack Datastore Jetpack DataStore is a data storage solution that allows you to store key-value pairs or typed objects with protocol buffers. DataStore uses Kotlin coroutines and Flow to store data asynchronously, consistently, and transactionally. 44

Jetpack Compose Jetpack Compose is Android's recommended modern toolkit for building native UI. It simplifies and accelerates UI development on Android. 44, 64

JSONB is a PostgreSQL data type used for holding semi-structured data in the Spanner PostgreSQL dialect. JSONB holds data in JavaScript Object Notation (JSON) format, which follows the specification described in RFC 7159. 16

Kotlin Multiplatform Kotlin Multiplatform is a technology that allows developers to create applications for various platforms and efficiently reuse code across them while retaining the benefits of native programming. The applications will run on iOS, Android, macOS, Windows, Linux, and more.. 72

Kotlin Serialization Kotlin Serialization is a cross-platform and multi-format framework for data serialization—converting trees of objects to strings, byte arrays, or other serial representations and back. 44

Maven Central Maven Central, as part of the build automation tool Apache Maven, is the primary software registry and repository for Java components, libraries, and frameworks, as well as Java Virtual Machine (JVM) languages. 41, 51, 60

Navigation Navigation is a framework for navigating between 'destinations' within an Android application that provides a consistent API whether destinations are implemented as Fragments, Activities, or other components. 44, 64

N-Quads is a line-based, plain text format for encoding an RDF dataset. RDF 1.2 N-Quads introduces triple terms as a fourth kind of RDF term which can be used as the subject or object of another triple, making it possible to make statements about other statements. 13

N-Triples is a line-based, plain text format for encoding an RDF graph. 13

OpenID Connect is an interoperable authentication protocol based on the OAuth 2.0 framework of specifications (IETF RFC 6749 and 6750). It simplifies the way to verify the identity of users based on the authentication performed by an Authorization Server and to obtain user profile information in an interoperable and REST-like manner. 12, 15, 16, 28, 29, 76

RDF/JSON represents a set of RDF triples as a series of nested data structures. 13

RDF/XML is a syntax defined by the W3C, to express (i.e. serialize) an RDF graph as an XML document. 13

Service in Android are a fundamental component of the Android operating system, designed to run long-running operations in the background without a user interface. 15, 48

Turtle Terse RDF Triple Language (Turtle) is a syntax and file format for expressing data in the Resource Description Framework (RDF) data model. 13, 16

Acronyms

AC Access Control. 13

ACL Access Control List. 13, 73

AI Artificial Intelligence. 10

AIDL Android Interface Definition Language. 55, 57

API Application Programming Interface. 7, 10, 14, 15, 18, 44, 48, 51, 52, 72

CRUD Create, Read, Update and Delete. 42, 56, 62

GDPR General Data Protection Regulation. 10

HTTP HyperText Transfer Protocol. 12, 13, 14, 16

IDP Identity Provider. 8, 13, 14

IRI Internationalized Resource Identifier. 13

JS JavaScript. 14, 80

JSON JavaScript Object Notation. 14, 80

JSON-LD JavaScript Object Notation for Linked Data. 13, 16, 35, 72

JVM Java Virtual Machine. 79

LD Linked Data. 8, 13, 14, 15, 18, 72, 74

LDP Linked Data Platform. 13

MVVM Model, View, ViewModel. 44

Non-RDF Non-Resource Description Framework. 13

OS Operation System. 10

Pod Personal Online Data store. 8, 9, 12, 13, 15, 22, 23, 24, 28, 34, 36, 72, 76

Pods Personal Online Data Stores. 3, 8, 9, 12, 13, 14, 34, 74

PWA Progressive Web App. 9

RDF Resource Description Framework. 12, 13, 16, 18, 19, 72

REST Representational State Transfer. 13, 14, 15

SDK Software Development Kit. 14

Solid-OIDC Solid OpenID Connect. 12, 13, 16, 42, 76

UI User Interface. 44

URI Uniform Resource Identifier. 12, 17, 20, 23, 29, 34, 35, 36, 76

URL Uniform Resource Locator. 13, 44

WebID Web Identity. 12, 23, 28, 33, 45, 72, 76

WWW World Wide Web. 7