
Solid Specification Draft



Latest version: `v.0.7.0` (see CHANGELOG.md)

Publication status: Unofficial Draft

Current development version: `v.0.7.0-next` (evolving)

This document contains an informal description of implementation guidelines for Solid servers and clients. A normative specification is in the making at <https://github.com/solid/specification/>. For the time being, the present document contains the best approximation of expected server and client behavior.

Table of Contents

1. Overview
2. Identity
3. Profiles
 - WebID Profile Documents
4. Authentication
 - Primary Authentication
 - WebID-TLS
 - Alternative Authentication Mechanisms
 - Secondary Authentication: Account Recovery
5. Authorization and Access Control
 - Web Access Control
6. Content Representation
7. Reading and Writing Resources
 - HTTPS REST API
 - WebSockets API
8. Social Web App Protocols
 - Notifications
 - Friends Lists, Followers and Following

-
9. Recommendations for Server Implementation
 10. Recommendations for Client App Implementation
 11. Examples
 12. Current Implementations

Overview

Solid is a proposed set of conventions and tools for building *decentralized applications* based on Linked Data principles. Solid is modular and extensible. It relies as much as possible on existing W3C standards and protocols.

See Also:

- About Solid
- Contributing to Solid
 - Pre-Requisites
 - Solid Project Workflow
- Standards Used
- Platform Notes
- Solid Project directory

Identity

Solid uses WebID URIs as universal usernames or actor identifiers. Frequently referred to simply as *WebIDs*, these URIs form the basis of most other Solid-related technologies, such as authentication, authorization, access control, user profiles, discovery of user preferences and server capabilities, and more.

WebIDs provide globally unique decentralized identifiers, enable cross-service federated signin, prevent service provider lock-in, and give users control over their own identity. *The WebID URI's primary function is to point to the location of a public WebID Profile document (see below).*

Example WebIDs: <https://alice.databox.com/profile/card#me> or <http://somepersonalsite.com/#webid>

Profiles

Solid uses WebID Profile Documents for management of user identity and security credentials (such as public keys), and user preferences discovery.

Although here we mostly refer to them in the context of user profiles, other types of actors use these profiles as well, such as groups, organizations, devices, and software applications.

WebID Profile Documents

A WebID URI, when dereferenced, yields a WebID Profile Document in a Linked Data format (Turtle by default, but often available as JSON-LD or HTML+RDFa). Parsing this document provides a client application with useful information, such as the user's name and profile image, links to user preferences and related documents, and lists of public key certificates or other relevant identity credentials.

See component spec: Solid WebID Profiles Specification

Authentication

Authentication is the process of determining a user's identity, of asking the question "How do I know you are who you say?"

How do web applications typically authenticate users (that is, how do they verify identity)? The most common method is usernames and passwords. A *username* uniquely identifies a user (and ties them to a user profile), and a *password* verifies that the user is who they say they are. Many applications or services also have a *secondary authentication mechanism* (usually an external email address) that they use for account recovery (in case the user forgets or loses their primary authentication tokens, username and password).

Solid currently uses WebID-TLS as its primary authentication mechanism. Alternative complementary mechanisms are also being actively investigated. In addition, Solid recommends that server implementations also offer secondary authentication available for users for Account Recovery (via email or some other out-of-band mechanism).

Primary Authentication

Solid, being a decentralized web application platform, has a set of requirements for its authentication mechanisms that are not commonly encountered by most platforms and ecosystems. Specifically, it requires *cross-domain*, de-centralized authentication mechanisms not tied to any particular identity provider or certificate authority.

WebID-TLS Note: Several browser vendors (Chrome, Firefox) have removed support for the [KEYGEN](#) element, on which WebID-TLS relied for in-browser certificate generation.

Solid uses the WebID-TLS protocol as one of its primary authentication mechanism. Instead of usernames, it uses WebIDs as unique identifiers, as previously mentioned. And instead of using passwords as bearer tokens, it uses cryptographic certificates (stored and managed by the user's web browser) to prove a user's identity.

When accessing a Solid server using WebID-TLS, a user is presented by their web browsers with a popup asking them to select an appropriate security certificate for that site. After a user makes their selection, the server securely matches the private key stored by the browser with the public key stored in that user's WebID Profile Document, and authenticates them.

See component spec: Solid WebID-TLS Specification

WebID-OIDC The Solid team is currently implementing support for WebID-OIDC as another primary authentication mechanism. It is based on the OAuth2/OpenID Connect protocols, adapted for WebID based decentralized use cases.

See component spec: WebID-OIDC Specification

Alternative Authentication Mechanisms There are several other authentication mechanisms that are currently being investigated, such as combinations of traditional username-and-password authentication and WebID-TLS Delegation).

Secondary Authentication: Account Recovery

Regardless of the primary authentication mechanism, bearer tokens and other proofs of identity tend to get lost by users. Passwords can be forgotten, browser certificates can be lost to hardware failure, and so on. Solid recommends that secondary Account Recovery mechanisms are provided by server implementers, to aid in these scenarios.

Authorization and Access Control

Authorization is the process of deciding whether a user has access to a particular resource. If authentication asks "who is the user?", authorization is concerned with "what is the user allowed to do?".

Solid currently uses the Web Access Control (WAC) mechanism for cross-domain authorization for all its resources.

Web Access Control

Web Access Control (WAC) is a decentralized system that allows different users and groups various forms of access to resources where users and groups are identified by HTTP URIs. The system is similar to the access control system used within many file systems except that the documents controlled, the users, and the groups, are all identified by URIs. Users are identified by WebIDs. Groups of users are identified by the URI of a class of users which, if you look it up, returns a list of users in the class. This means a WebID hosted by any server can be a member of a group hosted some other server.

Users do not need to have an account (i.e. WebID) on a given server to have access to documents on it.

See component spec: Solid WAC Specification

Content Representation

Solid deals with reading and writing two kinds of resources:

1. Linked Data resources (RDF in the form of JSON-LD, Turtle, HTML+RDFa, etc)
2. Everything else (binary data and non-linked-data structured text)

While you can build Solid applications with non-linked data resources, using actual RDF-based Linked Data provides you with considerable benefits in terms of interoperability with the rest of the Solid app ecosystem.

Resources are grouped in directory-like **Containers** (currently conforming to the LDP Basic Container spec).

See component spec: Solid Content Representation

Reading and Writing Resources

HTTPS REST API

Solid extends the Linked Data Platform spec to provide a simple REST API for CRUD operations on resources and containers.

See component spec: HTTPS REST API

WebSockets API

Solid also provides a WebSockets based API for a PubSub (Publish/Subscribe) mechanism, through which clients can be notified in real time of changes affecting a give resource.

See component spec: WebSockets API

Social Web App Protocols

In addition to read/write operations on resources, Solid provides a number of specs and recommendations to help developers achieve interoperability between various social web applications that are part of the ecosystem.

Notifications

See component spec: Linked Data Notifications

Friends Lists, Followers and Following

API recommendations for managing subscriptions and friends lists are still being discussed. TBD.

Recommendations for Server Implementations

See component spec: Recommendations for Server Implementations

Recommendations for Client App Implementations

See component spec: Recommendations for Client Implementations

Examples

- User Posts a Note

Current Implementations

Server Implementations: See [solid/solid-platform](#) for a list of Solid servers and developer tools. Note: The Solid team uses [node-solid-server](#) as its main server implementation.

Client App Implementations: See [solid-client](#) for the main client library, and [solid/solid-apps](#) for an example list of Apps built using Solid.