A.D. 1308
unipg
DIPARTIMENTO
DI INGEGNERIA

Tesi di Laurea in

INGEGNERIA INFORMATICA ED ELETTRONICA

Sviluppo di un blog decentralizzato tramite tecnologia Solid

Relatore
Luca Grilli

Candidato
Carlo Tosoni

Anno Accademico 2020/2021

Indice

Capitolo 1

Introduzione

Il 6 agosto 1991 venne inventata una delle tecnologie più importanti del secolo, destinata a rivoluzionare la società: questa invenzione è il `World Wide Web`. In quel giorno, l'informatico inglese Sir Tim Berners-Lee, fondatore del web, pubblicò presso il CERN di Ginevra il primo sito internet della storia che descriveva il progetto `WWW`. 17 giorni dopo il sito venne per la prima volta visitato da un utente esterno al centro di ricerca; da quel giorno il web si diffuse sempre di più, espandendosi ad una velocità impressionante. Oggi, nell'anno 2021, dei circa 7.83 miliardi di persone esistenti al mondo, ben 4.66 miliardi hanno accesso ad internet, per un numero complessivo di utenti che è pari a il 59.5% della popolazione mondiale totale [12]. Berners-Lee aveva originariamente immaginato il web come: "una piattaforma aperta che avrebbe permesso a tutti, ovunque, di condividere informazioni, di accedere ad opportunità e di collaborare attraverso differenti aree geografiche e confini culturali" [1]. Questa visione di Berners-Lee è tuttavia venuta a mancare con il passare degli anni, in quanto il web è diventato un luogo sempre più posto sotto il diretto controllo delle grandi multinazionali, le quali, attraverso il possesso dei dati degli utenti, monopolizzano la rete, impedendo agli utenti stessi di essere i reali proprietari del web. A seguito dello scoppio di alcuni scandali, come ad esempio quello relativo a Facebook-Cambridge Analytica [13], i quali hanno messo in evidenza le violazioni perpetuate da alcuni dei grandi colossi del web ai danni degli utenti di internet e della loro privacy, Berners-Lee ha deciso, in collaborazione col MIT di Boston, di fondare un nuovo progetto finalizzato a ridecentralizzare il web, esattamente come lo aveva immaginato il su fondatore nella sua visione originale. Tale progetto, che prende il nome di `Solid` (Social Linked Data), mira a dividere il piano delle applicazioni da quello dei dati, permettendo agli utenti di controllare in maniera diretta i propri dati personali, senza lasciarli sotto il controllo delle grandi multinazionali. `Solid` è una specifica che permette agli utenti di salvare i propri dati in sicurezza all'interno di archivi di dati decentralizzati chiamati `Pod`. I `Pod` sono come dei server web personali e sicuri per la memorizzazione dei dati dell'utente. Una volta salvati i dati all'interno del `Pod` di un utente, costui è in grado di controllare quali applicazioni e quali altri utenti possono

avere accesso a tali dati. `Solid` mira quindi a "cambiare il modo in cui funzionano le attuali applicazioni Web, per ottenere un reale possesso dei dati ed una maggiore privacy da parte degli utenti" [9]. Allo stato attuale, questa tecnologia risuta, tuttavia, essere ancora in fase di sviluppo.

Il presente progetto di tesi è finalizzato alla creazione di applicazioni decentralizzate utilizzando la tecnologia `Solid`, con il fine di evidenziare i vantaggi dovuti al loro utilizzo e di mostrare le differenze tra una normale applicazione ed una decentralizzata. Poiché le applicazioni decentralizzate non salvano localmente alcun dato relativo all'utente, quest'ultimo è portato a scegliere quali applicazioni utilizzare soltanto in base alla qualità del servizio che queste offrono e non per il possesso esclusivo dei suoi dati. A tal proposito ogni dato salvato all'interno del `Pod` può essere riutilizzato dagli utenti in base alle loro necessità, permettendo eventualmente ad altre applicazioni di accedervi o di revocarne i permessi, in lettura o in scrittura, precedemente concessi ad applicazioni o utenti.

Nello specifico, il presente elaborato di tesi è finalizzato allo sviluppo di due differenti applicazioni che formano il sistema denominato `SADeB` (Solid Authenticity Decentralized Blog). Il sistema `SADeB` ha lo scopo di mettere in evidenza il funzionamento di un'applicazione `Solid`, mostrando le modalità con cui un utente può permettere ad un'applicazione di accedere ai dati contenuti all'interno del proprio `Pod` e le modalità con cui tale applicazione `Solid`, autorizzata dall'utente, può andare a leggere o a scrivere i dati contenuti in esso. Nel dettaglio le due applicazioni che formano il sistema `SADeB` prendono il nome di `my-solid-blog` e `blog-validator`: la prima consiste in un social network, decentralizzato, che permette agli utenti `Solid` di creare e gestire blog personali; la seconda applicazione, svincolata da `my-solid-blog`, ha come scopo la validazione dei contenuti mostrati all'interno di `my-solid-blog`, con il fine di prevenire la diffusione di informazioni false su tale piattaforma. Differentemente da un altro social network non decentralizzato, `my-solid-blog` permette all'utente proprietario del `Pod` di visualizzare quali dati vengono effettivamente salvati dall'applicazione e di controllare in prima persona chi li sta utilizzando. Per implementare queste applicazioni sono state utilizzate la libreria `JavaScript` React e, in particolare, il comando create-react-app, per lo sviluppo di `my-solid-blog`, e la multipiattaforma orientata agli eventi `Node.js` per lo sviluppo dell'applicazione `blog-validator`.

vengono qui elencati i capitoli della tesi e i loro contenuti.

Capitolo 2: Concetti preliminari e tecnologie utilizzate: introduzione di alcuni concetti necessari per la comprensione del progetto di tesi, descrivendo le nozioni principali relative alla tecnologia `Solid`. Sono stati successivamente descritti ulteriori argomenti utilizzati per lo sviluppo del sistema `SADeB`, come `Inrupt`, `React`, `Bulma` e `Node.js`.

Capitolo 3: Motivazioni del progetto: In questo capitolo vengono illustrate le motivazioni che hanno portato il premio Turing Sir Tim Berners-Lee a fondare il progetto `Solid`, vengono poi elencate le finalità del sistema `SADeB`.

Capitolo 4: Il problema affrontato: Il capitolo contiene le specifiche descritte nel dettaglio del sistema `SADeB`, enunciando le modalità con cui le applicazioni di tale sistema gestiscono la procedura di autenticazione e la lettura/scrittura dei dati contenuti all'interno del `Pod`.

Capitolo 5: Il sistema SADeB: Vengono qui descritti nel dettaglio il funzionamento e la struttura delle due applicazione che formano il sistema `SADeB`, ovvero `my-solid-blog` e `blog-validator`.

Capitolo 2

Concetti preliminari e tecnologie utilizzate

Vengono di seguito chiariti nel dettaglio i concetti fondamentali necessari a comprendere appieno il progetto presentato in sede di discussione e le tecnologie utilizzate per svilupparlo.

2.1 Solid

`Solid`, (Social linked data) è una specifica che permette di salvare i propri dati in sicurezza in data stores decentralizzati chiamati `Pods`. Qualsiasi tipo di informazione può essere salvato in un `Pod` e l'utente che ne è proprietario può decidere con chi condividere i propri dati, concedendo permessi ad altre applicazioni o ad altri utenti, avendo comunque la possibilità di revocarli in un secondo momento. Ogni `Pod` è interamente controllabile dal proprio proprietario, che può quindi decidere autonomamente come gestirlo.

`Solid` è stato creato da Sir Tim Berners-Lee, fondatore del `World Wide Web`, in collaborazione con il MIT, con il fine di decentralizzare nuovamente il web; quest'ultimo era stato originariamente pensato come un luogo in cui tutti gli utenti avrebbero potuto collaborare per poter creare dati, permettendo loro quindi non solo di leggere i contenuti già esistenti, ma anche di cooperare per crearne dei nuovi. Oggi i dati relativi agli utenti vengono salvati dalle applicazioni, come i social network, in database locali; questo porta ad una serie di svantaggi per l'utente, al quale non è permesso di accedere direttamente ai propri dati.

Quando i dati vengono salvati lontano dagli utenti:

- Non si ha quasi nessuna visibilità su ciò che viene conservato.
- Si ha scarso controllo su come i dati vengano utlizzati, e su chi vi possa accedere.

- Non si può scegliere quali applicazioni usare per poter accedere a tali informazioni.
- Non si possono utilizzare i dati come unità coesiva, essendo questi sparsi per il web, sotto il controllo di proprietari differenti e salvati con diversi formati.

`Solid` mira a risolvere questo problema dividendo il piano dei dati da quello delle applicazioni; in questo modo l'utente è portato a scegliere quale applicazione usare soltanto in base alla qualità dei servizi offerti dall'applicazione stessa e non per il monopolio sul possesso dei dati dell'utente. `Solid` permette quindi ai singoli utenti di tornare ad essere i padroni del web, decentralizzandolo nuovamente, esattamente come era stato pensato in origine da Berners-Lee. Attualmente la tecnologia è ancora in via di sviluppo e i server `Solid` utilizzati, https://inrupt.net/ e https://solidcommunity.net/ non offrono garanzie riguardanti la stabilità e la sicurezza dei dati salvati.

2.2 Terminologia Solid

Per poter comprendere appieno la tecnologia `Solid`, è necessario spiegare il significato di alcuni termini connessi con essa.

- `Pod`: luogo in cui l'utente può scrivere i propri dati personali. Un utente può scegliere di utilizzare uno oppure più `Pod` per memorizzare le proprie informazioni. Le applicazioni possono eseguire diversi tipi di operazioni sui dati dell'utente in relazione ai tipi di permesso che sono stati loro concessi.
- `WebId`: un `Internationalised Resource Identifier (IRI)` necessario per identificare univocamente un utente.
  É qui di seguito mostrato un esempio di `webId`:
  $https://carlotosoni99.inrupt.net/profile/card\#me$.
- `Pod Provider`: compagnia o organizzazione che permette di ospitare i `Pod` degli utenti.
- `Identity Provider`: compagnia o organizzazione che mette a disposizione il servizio di autenticazione con il server `Solid`.
- `Solid Identity Provider`: compagnia o organizzazione che è contemporaneamente un `Pod Provider` e un `Identity Provider`.

Infine vengono dette applicazioni `Solid`, quelle applicazioni che accedono ai dati degli utenti, memorizzati all'interno dei rispettivi `Pod`, utilizzando il `Solid Protocol` [11], ovvero il protocollo utilizzato da `Solid` per permettere lo scambio di dati, il quale stabilisce che lo scambio di informazione, tra `Solid` app e `Pod`, debba avvenire tramite il protocollo `HTTP`.

2.3 Linked Data

Secondo la tecnologia Solid, qualisiasi applicazione, se autorizzata dall'utente, deve poter eseguire operazioni sui dati contenuti all'interno del Pod. Per questo motivo è necessario che i dati vengano rappresentati tramite un'unica modalità.

A tal proposito in Solid viene utilizzato il linguaggio RDF (Resource Description Framework) per rappresentare i dati contenuti all'interno del Pod. Qualsiasi dato in RDF viene detto risorsa; ogni risorsa è rappresentata tramite un IRI e pertanto è reperibile direttamente dal web. Oltre alle risorse, il modello di dati RDF è formato anche da proprietà e valori: le prime sono relazioni che permettono di collegare risorse e valori; un valore invece è la valenza della risorsa per tale proprietà; eventualmente un valore può essere un dato primitivo, come, ad esempio, una stringa. Le risorse vengono infine descritte in vocabolari, i quali a loro volta sono rappresentati tramite IRI. All'interno di un vocabolario vengono definite classi e proprietà, le classi sono utilizzate per indicare il tipo di dato che si vuole rappresentare, le proprietà sono semplicemente attributi relativi a istanze di una classe; visitando l'IRI relativo ad un vocabolario, si può trovare la documentazione necessaria per rappresentare un qualche tipo di informazione tramite le proprietà e le classi definite in tale vocabolario.

Spesso per la rappresentazione di dati si utilizzano dei vocabolari noti che ricorrono frequentemente in Solid, come ad esempio:

- RDF: vocabolario fondamentale per RDF, definisce un modello per rappresentare dati in RDF: per esempio rdfs:Class è la classe di risorse per rappresentare classi RDF, mentre rdf:Property è la classe delle proprietà RDF.
- FOAF (Friend of a Friend): usato per rappresentare persone e organizzazioni, definisce i termini per rappresentare il profilo di un utente e le interazioni tra gli utenti.
- ACL (Access Control List): vocabolario essenziale per Solid, definisce i livelli di accesso di un agente, stabilendo se esso è autorizzato a compiere una determinata operazione.
- Schema: vocabolario che definisce varie strutture di dati da poter utilizzare nel web.

In generale, se per rappresentare un qualche tipo di dato non dovesse esistere un adeguato vocabolario, è possibile crearne uno e pubblicarlo nel proprio Pod, cosicché anche altri utenti possano utilizzarlo. Per ulteriori informazioni relative ai vocabolari è possibile consultare il sito web di Solid dove è presente un tutorial per imparare a comprenderne il significato.

https://solidproject.org/developers/vocabularies

In RDF qualsiasi informazione deve essere rappresentata tramite uno `statement`, il quale è definito come una tripla composta da `soggetto`, `predicato` e `oggetto`. Gli `statement` permettono di rappresentare in maniera semplice qualsiasi tipo di informazione. Qui seguito è riportato un esempio di `statement`.

```
<https://carlotosoni99.inrupt.net/profile/card#me> (soggetto)
<http://xmlns.com/foaf/0.1/name> (predicato)
"Carlo" (oggetto)
```

Questa tripla sta a indicare che l'oggetto
https : *//carlotosoni99.inrupt.net/profile/card#me*, che rappresenta il profilo di un utente in `Solid`, ha una proprietà descritta nel `vocabolario FOAF` per rappresentare il nome di una generica persona o organizzazione, e che tale nome è pari a "*Carlo*", il quale è semplicemente una stringa.

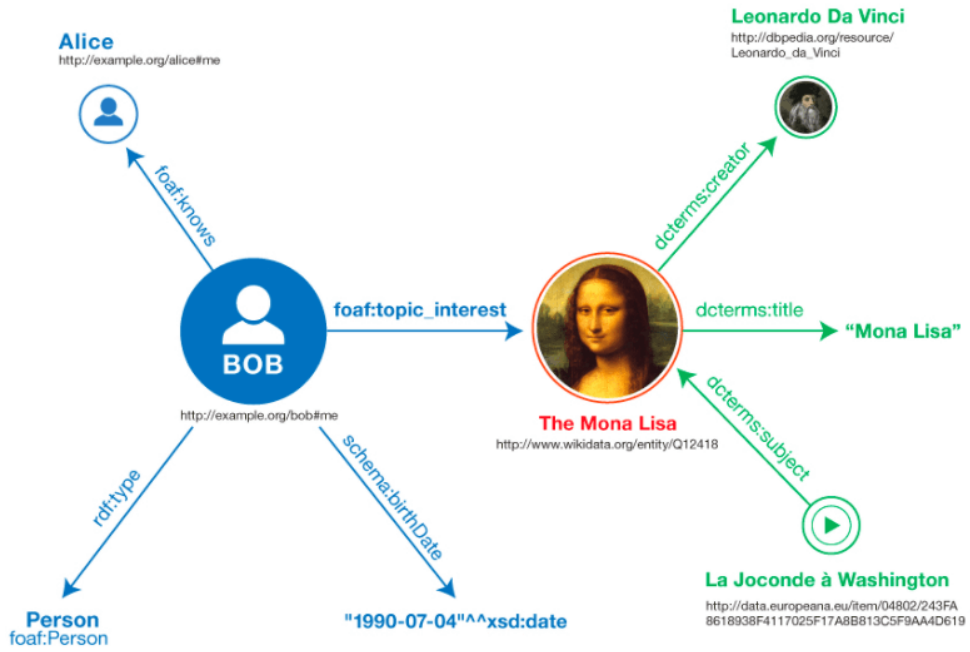

Figura 2.1: Esempio di dati rappresentati in RDF

Questa modalità di rappresentare i dati viene chiamata anche `Linked Data`, poiché `oggetti` o `classi` appartenti a `vocabolari` differenti, possono essere facilmente connessi tra loro con il fine di rappresentare strutture di dati più complesse.

I dati in RDF possono essere espressi tramite diverse sintassi; quella mostrata in questa tesi e utilizzata da `Solid`, prende il nome di `Turtle` ("Terse RDF Triple Language"); i file `Turtle` utilizzano il formato `.ttl`. Alcuni dei possibili tipi di dati salvabili in un `Pod` tramite `Turtle` sono: `Url, Boolean, Date, Decimal, Integer, String` e altri.

2.4 Struttura di un Pod

Per poter comprendere appieno il funzionamento dell'applicazione creata come progetto di tesi, è necessario conoscere come è strutturato un `Pod`. Innanzitutto un `Pod` è formato da `containers` e da `risorse`: i `containers` sono degli oggetti simili a delle cartelle che contengono le `risorse`; il `Pod` di un utente è infatti gestito in maniera simile ad un file system di un calcolatore. Le `risorse` invece vanno intese come l'insieme di oggetti e delle loro proprietà, espresse in linguaggio `RDF`, che servono a descrivere la informazioni contenute nel `Pod`.

Per esempio all'interno di ciascun `Pod` è presente il `container profile`, il quale contiene le informazioni relative al profilo dell'utente possessore del `Pod`.

$https://carlotosoni99.inrupt.net/profile/$

Visitando questo link è possibile vedere che all'interno di questo `container` si trova una risorsa chiamata `card`, nella quale sono presenti tutte le informazioni di interesse dell'utente. In particolare in questo `dataset` è collocato un oggetto avente `IRI` pari a:

$https://carlotosoni99.inrupt.net/profile/card\#me$

Questo oggetto serve a rappresentare tutte le informazioni più rilevanti riguardanti il `Pod` e il suo utente; da notare che l'`URL` di questo oggetto corrisponde con la `webId` dell'utente stesso. Ogni utente può liberamente leggere il `SolidDataset card` anche senza essersi autenticato con il `Solid Identity Provider`.

All'interno di un `Pod` esistono altri `containers` e altre `risorse`, oltre a quelli già elencati, che servono a memorizzare differenti tipi di informazioni. In generale, comunque, ogni utente avente diritto di scrittura è libero di aggiungere `containers` e `risorse` all'interno di un `Pod`, per poterlo gestire in base alle proprie necessità.

L'applicazione $https://podbrowser.inrupt.com/$ sviluppata da `Inrupt`, società fondata da Berners-Lee, permette di vedere il contenuto del proprio `Pod` in maniera user friendly, potendo visualizzare la struttura di `containers` e le rispettive `risorse` al loro interno.

2.5 Tipologie di permesso in Solid

In `Solid` non tutti gli utenti sono autorizzati a modificare le risorse contenute all'interno di un `Pod`; il `vocabolario ACL` (access control list), già enunciato, viene utilizzato per determinare se un utente è autorizzato o meno a compiere una determinata operazione. Il vocabolario `ACL` descrive quattro diversi tipologie di permessi:

- `Read`: permesso in lettura su una `risorsa`.

- `Append`: permesso di aggiungere ulteriori informazioni ad una `risorsa` già esistente, senza poterla eliminare.
- `Write`: permesso di eseguire operazioni in scrittura su una `risorsa`, potendo creare, modificare o eliminare una determinata `risorsa` nel `Pod`.
- `Control`: permesso di eseguire operazioni in lettura e scrittura su un file `ACL` correlato con una `risorsa`.

Nel momento in cui ci si autentica nel proprio `Pod` tramite un'applicazione `Solid` per la prima volta, viene data all'utente la possibilità di scegliere quali debbano essere i permessi di tale applicazione sulle `risorse` presenti all'interno del proprio `Pod`. Tali permessi possono essere eventualmente revocati in un secondo momento.

Inoltre modificando le `risorse ACL` presenti all'interno del proprio `Pod`, è possibile andare a personalizzare le tipologie di permesso dell'applicazione sulle singole `risorse` presenti.

Per ulteriori chiarimenti riguardanti il sistema di autorizzazione usato da `Solid` è possibile leggere il documento `Web Access Control` [5].

2.6 Autenticazione in Solid

Spesso le applicazioni `Solid` necessitano di autenticarsi con il/i `Solid Identity Provider` appartente/i all'/agli utente/i in questione; tale operazione è spesso necessaria per poter ottenere i permessi necessari all'applicazione per poter interagire con le `risorse` contenute all'interno dei `Pod`. A seguito dell'operazione di autenticazione, le applicazioni `Solid` vengono quindi autorizzate dal `Solid Identity Provider` ad eseguire alcune operazioni sui dati dell'utente in base alle tipologie di permesso concesse.

`Inrupt` mette a disposizione una serie di librerie che permettono di autenticarsi con il proprio `Solid Identity Provider` da Browser oppure tramite `Node.js`.

Utilizzando le librerie `Inrupt` la procedura di autenticazione è relativamente semplice da eseguire: è necessario innanzitutto indicare il proprio `Solid Identity Provider`, dopodiché l'utente viene indirizzato presso di questo per completare l'operazione di autenticazione con i propri dati (username e password); una volta terminata questa operazione l'utente è reindirizzato all'applicazione che stava utilizzando.

Per ulteriori informazioni riguardanti il processo di autenticazione in `Solid`, è possibile visitare la seguente pagina:
https://solid.github.io/authentication-panel/solid-oidc/.

2.7 Ulteriori chiarimenti riguardo la tecnologia Solid

Qui di seguito vengono ora espressi nel dettaglio ulteriori concetti relativi alla tecnologia `Solid`, i quali sono utili per comprendere in maniera più approfondita il progetto.

`Solid` è una tecnologia che non si pone l'obiettivo di reinventare il web, ma che mira ad aggiungere a questo alcune caratteristiche, come la presenza di un sistema di identificazione per evitare di dover creare un account su ogni piattaforma presente all'interno della rete, come Facebook, Twitter, Google e molte altre. Obiettivo di `Solid` è inoltre la possibilità di permettere ai vari social media, di condividere i dati relativi ai singoli utenti, indipendente dal tipo di social media utilizzato. Le applicazioni `Solid` possono funzionare correttamente anche senza salvare in database locali alcun dato relativo ai singoli utenti; infatti come già detto, tutti i dati vengono salvati all'interno dei `Pod` degli utenti. `Solid` inoltre mette a disposizione dell'utente la possibilità di ospitare il proprio `Pod` all'interno del suo hard disk o server, senza doversi affidare ad un `Solid Identity Provider`. É anche possibile, quindi, divenire un piccolo `Pod Provider` o `Identity Provider`, con il fine di ospitare i `Pod` appartenti, per esempio, alla propria famiglia, o ai membri dell'organizzazione di cui si fa parte. Un utente può inoltre possedere più di un `Pod` contemporaneamente e salvare su ognuno di essi differenti tipologie di dati in relazione alle proprie necessità; inoltre sussiste la possibilità di cambiare `Pod Provider` in un secondo momento, recuperando i dati presenti all'interno del `Pod`.

Riguardo al sistema di identificazione presente in `Solid`, una `webId` non deve necessariamente essere usata per identificare una persona fisica, bensì può riferirsi anche a organizzazioni, compagnie, famiglie, team o, più generalmente, a qualsiasi gruppo di persone.

Per quanto concerne la sicurezza dei dati salvati all'interno del proprio `Pod` è bene fare alcune precisazioni. Innanzitutto il livello di sicurezza dei propri dati dipende dal `Pod Provider` utilizzato; i `Pod Provider`, in generale, non hanno vincoli riguardanti la crittografia utilizzata per criptare i dati contenuti all'interno del `Pod`. L'utente ha il compito di scegliere un `Pod Provider` che cripti i propri dati adeguatamente se la sicurezza di questi è importante per lui. Infine, poiché `Solid` non prevede che i dati relativi ad un utente debbano essere necessariamente salvati all'interno di un singolo luogo, essendo possibile utilizzare più di un `Pod` per la memorizzazione delle proprie informazioni, `Solid` non rende gli utenti maggiormente vulnerabili da parte di attacchi informatici. A tal proposito è necessario considerare anche che attacchi informatici da parte di hackers sono maggiormente probabili se riferiti a una singola fonte di dati di molte persone, piuttosto che a livello individuale.

All'interno del sito web https://solidproject.org/ sono presenti ulteriori informazioni, per sviluppatori e non, riguardanti il progetto `Solid`.

2.8 Inrupt

`Inrupt` è una società fondata da Sir Tim Berners-Lee nel 2018 per fornire strumenti atti a favorire lo sviluppo di `Solid`. Secondo Berners-Lee, `Inrupt` mira a "fornire energia commerciale e un ecosistema per aiutare a proteggere l'integrità e la qualità del nuovo web costruito su Solid" [2].

In particolare `Inrupt` fornisce una serie di strumenti per potersi autenticare con il proprio `Solid Identity Provider` e per poter aggiungere, creare, modificare e cancellare le informazioni all'interno del proprio `Pod`.

Le librerie create da `Inrupt`, e usate anche nel progetto oggetto della tesi, sono:

- `solid-client-authn-browser`: tale libreria permette di autenticarsi con il proprio `Solid Identity Provider` tramite browser.
- `solid-client-authn-node`: questa libreria permette di autenticarsi con il proprio `Solid Identity Provider` tramite Node.js.
- `solid-ui-react`: mette a disposizione componenti per applicazioni in React per potersi autenticare con il proprio `Solid Identity Provider` e per poter leggere e scrivere le informazioni contenute all'interno del `Pod`.
- `solid-client`: fornisce funzioni atte a eseguire ogni possibile azione all'interno del proprio `Pod`, tra le quali: scaricare `SolidDataset` presenti all'interno del proprio `Pod`, modificare un `SolidDataset` già esistente o crearne uno all'interno del proprio `Pod`, caricare, modificare un `oggetto` appartenente ad un determinato `SolidDataset`, oppure crearne uno nuovo, leggere gli `statement` di un `oggetto` per poter ricavare informazioni presenti nel proprio `Pod`, oppure modificare `statement` già esistenti o crearne dei nuovi, leggere e scrivere `risorse ACL` e utilizzare altre funzionalità.

Le librerie messe a disposizioni da `Inrupt` sono attualmente le più utilizzate per programmare applicazioni in `Solid`. Oltre a questo, `Inrupt` gestisce uno dei due `Solid Identity Provider` più utilizzati al momento;
visitando il seguente URL si può accedere al sito https://inrupt.net/ per potersi registrare con il `Solid Identity Provider` e creare un proprio `Pod`.

Per leggere la documentazione relativa alle librerie sopra enunciate è possibile visitare i seguenti URL:

`solid-client-authn-browser` e `solid-client-authn-node`:
https://docs.inrupt.com/developer-tools/javascript/client-libraries/authentication/
`solid-ui-react`:
https://solid-ui-react.docs.inrupt.com/?path=/story/intro–page
`solid-client`:
https://docs.inrupt.com/developer-tools/javascript/client-libraries/tutorial/read-write-data/
e
https://docs.inrupt.com/developer-tools/api/javascript/solid-client/

2.9 React

`React` è una libreria `open-source`, `front-end`, `JavaScript`, finalizzata alla creazione di interfacce utente ed attualmente molto conosciuta e usata. Vengono qui elencate le caratteristiche principali di `React`:

- Permette di creare interfacce grafiche in maniera semplificata. Le interfacce sono progettate per ogni stato dell'applicazione; ad ogni cambio di stato `React` aggiorna solo le parti delle UI che dipendono da tali dati.
- Agevola la creazione di interfacce grafiche complesse, permettendo di creare componenti in isolamento e ricomponibili per formare UI complesse.
- `React` non è dipendente da altre tecnologie: in questo modo si possono sviluppare nuove funzionalità in `React` senza dover riscrivere qualsiasi codice già esistente.

`JSX` è la sintassi che `React` utilizza per descrivere l'aspetto delle interfacce grafiche. Comunque l'utilizzo di `JSX`, pur essendo raccomandato, risulta opzionale e non richiesto per programmare in `React`. `JSX` va inteso come un'estensione della sintassi di `Javascript`, riprendendo le proprie caratteristiche sia da `Javascript` stesso che da `HTML`.

L'applicazione `my-solid-blog` è stata sviluppata utilizzando `React` e, in particolare, il comando `create-react-app` di `npx`, il quale permette di creare applicazioni in `React` single-page. `create-react-app` crea autonomamente un ambiente di sviluppo che consente di utilizzare le caratteristiche più recenti di `JavaScript`, utilizzando, per esempio, `Babel` e `webpack`.

Le motivazioni che hanno portato a sviluppare `my-solid-blog` in `React` sono principalmente due: la prima è che la libreria `solid-ui-react` di `Inrupt` permette ad applicazioni in `React` di gestire alcune operazioni in `Solid`, come l'autenticazione con `Solid Identity Provider`, in maniera semplice e intuitiva; il secondo motivo riguarda la semplicità con cui `React` permette agli sviluppatori di creare e gestire interfacce grafiche.

Per renderizzare le proprie interfacce grafiche, l'applicazione `my-solid-blog` utilizza la sintassi `JSX`. Le applicazioni `React` create tramite comando `create-react-app` possono essere avviate in `development mode` utilizzando il comando `npm start`.

Per ulteriori informazioni relative alla libreria `React` è possibile visitare il sito web: https://it.reactjs.org/.

2.10 Bulma

`Bulma` è un framework `CSS open source` che si è particolarmente diffuso negli ultimi anni, raggiungendo 40.000 stelle su `GitHub`. È caratterizzato da una sintassi semplice che permette di importare nei propri progetti vari componenti predefiniti; `Bulma` presenta un design minimalista e unico nel suo genere, capace di rendere maggiormente accattivamente una pagina web.

Il framework `Bulma` è stato scelto per gestire il layout dell'applicazione `my-solid-blog` per la sua semplicità, per il suo design e per la compatibilità con la libreria `React`.

Visitando il seguente URL è possibile leggere la documentazione relativa a `Bulma`: https://bulma.io/

2.11 Node.js

`Node.js` è un `runtime system open source` multipiattaforma `orientato agli eventi`. Quando è stato introdotto, `Node.js` ha consentito l'esecuzione di codice `JavaScript` da `lato server`; precedentemente `JavaScript` era stato utilizzato principalmente da `lato Client`. Data la sua enorme potenzialità, `Node.js` è attualmente una delle tecnologie più richieste sul mercato del lavoro.

All'interno di questo progetto `Node.js` è stato utlizzato per sviluppare l'applicazione `blog-validator`, con lo scopo di effettuare alcuni controlli sui dati presenti nel `Pod`. Tale server si occupa infatti di verificare l'autenticità dei dati mostrati dall'applicazione `my-solid-blog` su richiesta dell'utente.

2.12 Altri strumenti utilizzati

Axios

`Axios` è una libreria che permette di inviare richieste `HTTP` da browser o da `Node.js`. Tale libreria è stata utilizzata per permettere la trasmissione di informazioni tra l'applicazione `my-solid-blog` e il server `Node.js`, con il fine di comunicare al server alcune informazioni relative ai contenuti mostrati all'interno dell'applicazione stessa.

https://github.com/axios/axios

vocab-common-rdf e rdf-namespaces

Queste due librerie di `JavaScript` contengono degli pseudonimi per i vocabolari `RDF` più utilizzati, permettendo di utilizzare `proprietà` e `classi` di alcuni vocabolari `RDF` in maniera semplificata e snellendo il codice dell'applicazione.

Ad esempio la libreria `rdf-namespaces` permette di utilizzare la proprietà `type` del vocabolario `http://www.w3.org/1999/02/22-rdf-syntax-ns` semplicemente scrivendo `rdf.type`, anziché
`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`.

Tale proprietà, fondamentale nel linguaggio `RDF`, è utilizzata per indicare che una `risorsa` è un'instanza di una `classe`.

Capitolo 3

Motivazioni del progetto

3.1 L'evoluzione del World Wide Web

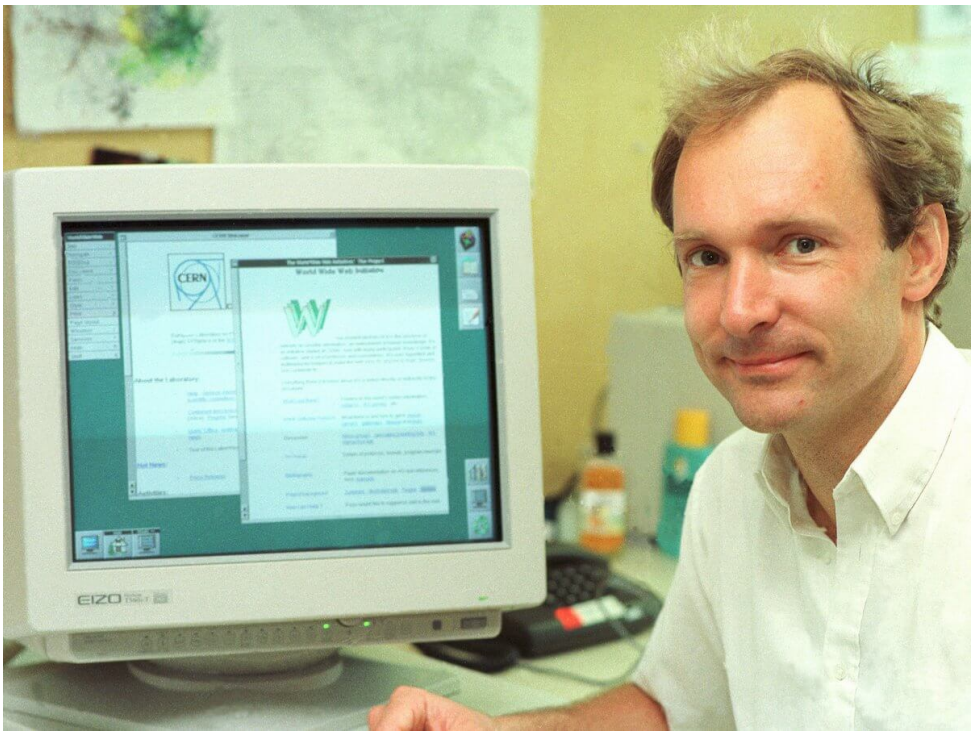

Figura 3.1: Sir Tim Berners-lee, fondatore del `World Wide Web`, davanti al primo prototipo di browser.

Sir Tim Berners-Lee, fondatore del `World Wide Web` e del progetto `Solid`, insignito del premio Turing nel 2016, con il passare degli anni fu mosso da una crescente preoccupazione relativa al modo con cui la sua creazione, il `WWW`, si era evoluta. Con il passare del tempo iniziò a dubitare del reale beneficio apportato da questa tecnologia alla società umana [7].

Nella visione di Berners-Lee, il `World Wide Web` sarebbe dovuto essere un luogo in cui tutti gli utenti, attraverso un meccanismo di autorizzazioni, avrebbero potuto collaborare per creare nuovi contenuti sulla rete. Infatti, il primo browser della storia, sviluppato nel `CERN` di Ginevra, era stato progettato per essere anche un editor, con il fine di permettere uno scambio di dati in maniera decentralizzata tra utenti. Tuttavia la prima versione di browser che rese il web popolare tagliò questa funzionalità, poiché ritenuta troppo difficile da implementare.

Berners-Lee, il 12 marzo 2017, in occasione del ventottesimo compleanno di internet, delineò i tre principali problemi principali relativi alla sua creazione [1]:

"We've lost control of our personal data". Gli utenti di Internet accettano di utilizzare servizi gratuitamente in cambio della cessione dei loro dati personali; questo ha portato le grandi multinazionali, come i più importanti social network, a collezionare i dati dei loro utenti per fini personali. Società e governi hanno iniziato a osservare i movimenti online degli tali utenti, portando, ad esempio, all'arresto e all'uccisione di alcuni cittadini in Paesi vittime di regimi dittatoriali. [14]

"It's too easy for misinformation to spread on the web". Tramite i dati raccolti dagli utenti, attraverso l'utilizzo di appositi algoritmi, i social network possono mostrare contenuti in base ai loro interessi, inviando loro link ad altri siti web. Tali siti spesso risultano essere fonti di disinformazione o di fake news e mostrano all'utente contenuti in base alle sue credenze e ai suoi pregiudizi in modo da condizionarne le idee [8].

"Political advertising online needs transparency and understanding". Berners-Lee ha infine evidenziato i problemi connessi con l'industria delle publicità politiche. Tramite i dati degli utenti, i quali vengono continuamente raccolti dai social network, vengono create pubblicità individuali rivolte ai singoli, in base ai loro dati. É stato stimato da una fonte [4] che, durante le elezioni presidenziali degli Stati Uniti d'America del 2016, siano state create ogni giorno circa 50.000 variazioni di pubblicità di natura politica con il fine di renderle più adatte ai singoli utenti. Berners-Lee ritenne che vi fossero molti indizi sulla non eticità di tali pubblicità.

Le preoccupazioni di Berners-Lee si sono ulteriormente acuite a seguito dello scandalo Facebook-Cambridge Analytica, avvenuto nel 2018. Tale scandalo mise in evidenza che il noto social network Facebook aveva raccolto dati di 87 milioni di utenti senza il loro consenso per scopi di propaganda politica [13].

3.2 Nascita del progetto Solid

Queste considerazioni hanno portato alla nascita del progetto Solid da parte di Berners-Lee. Egli mira a superare le problematiche elencate rendendo gli utenti i veri proprietari del web, ridecentralizzandolo, come quando lo aveva originariamente creato [10].

Il progetto nasce già nel 2015 in collaborazione con il MIT; allo sviluppo di questa tecnologia hanno collaborato anche l'Università di Oxford e l'Istituto di Ricerca Informatica del Qatar. Nel 2018 Berners-Lee si prende un anno sabbatico dal progetto per fondare la società Inrupt, già menzionata nel capitolo 2, con il fine di mettere a disposizione degli sviluppatori strumenti per poter interagire con Solid.

Come in parte già descritto, Solid mira a risolvere i problemi evidenziati da Berners-Lee, cambiando il modello attuale di consegna dei propri dati ai giganti informatici in cambio dell'accesso ad alcuni servizi, poiché tale modello penalizza gli utenti del web, privandoli della loro privacy. Solid mira ad evolvere il web riequilibrandolo, dando ad ognuno il completo controllo dei propri dati, personali e non, in modo innovativo. Berners-Lee è fermamente convinto riguardo la necessarietà di Solid per poter modificare il web, sostenendo che, tramite questa tecnologia le persone riusciranno a riprendersi il loro potere dalle grandi corporazioni. A tal proposito Berners-Lee ha dichiarato: "Non stiamo chiedendo a Facebook o a Google se introdurre o meno un completo cambiamento (sul web) che porterà a capovolgere completamente il loro modello di business, non stiamo chiedendo a loro permessi" [3].

3.3 Nascita del sistema SADeB

Il sistema SADeB (Solid Authenticity Decentralized Blog) nasce con l'intento di mettere in evidenza le potenzialità della tecnologia Solid, mostrando le differenze tra l'applicazione Solid my-solid-blog e un normale social network.

Come già descritto, le applicazioni Solid sono totalmente svincolate dai dati che mostrano all'utente con cui interagiscono; ciò permette all'utente stesso di controllare i propri dati in maniera diretta, di comprendere chi li sta utilizzando e di permettere che questi vengano riutilizzati da più applicazioni differenti in base alle sue necessità.

La decentralizzazione effettuata da Solid permette, inoltre, di tutelare l'utente che utilizza tali applicazioni, arginando in parte il problema relativo alla diffusione di disinformazione, evidenziato anche da Berners-Lee stesso [1]. Infatti, tramite Solid, è possibile implementare in maniera semplice ed efficiente dei controlli sull'autenticità dei dati mostrati all'utente. L'applicazione blog-validator, facente parte del sistema SADeB, si occupa, infatti, di controllare che l'applicazione my-solid-blog non abbia volontariamente modificato i dati contenuti all'interno del Pod dell'utente, con il

fine di diffondere informazioni false. `blog-validator` si occupa quindi di comparare i dati visualizzati da `my-solid-blog` con quelli realmente presenti all'interno del `Pod` dell'utente, indicando, eventualmente a quest'ultimo, se tali dati siano stati modificati. `blog-validator` è un'applicazione esterna e svincolata da `my-solid-blog`, creata per poter essere utilizzata, apportando alcune modifiche al codice in base alle necessità, non solo da `my-solid-blog`, ma anche da altre possibili applicazioni decentralizzate, per dare garanzie ai loro utenti riguardo la validità delle informazione loro mostrate.

Capitolo 4

Il problema affrontato

L'obiettivo del presente elaborato è mostrare il funzionamento di un blog decentralizzato sviluppato tramite tecnologia `Solid`, mettendo in evidenza i benefici che un'applicazione `Solid` può apportare all'utente.

Il progetto è composto da due applicazioni differenti e svincolate tra di loro. La prima è chiamata `my-solid-blog`; tale applicazione offre due differenti tipologie di servizi: la prima consiste nel permettere la creazione di un proprio blog, salvando tutte le informazioni di interesse all'interno del suo `Pod`. In questo caso, per poter ottenere il diritto in scrittura sui dati presenti all'interno del `Pod`, l'applicazione richiede preventivamente all'utente di autenticarsi con il proprio `Solid Idenitity Provider`; il secondo servizio implementato in `my-solid-blog` consiste nel permettere la lettura di blog di altre persone una volta conosciuta la loro `webId`. Per questa seconda funzionalità la procedura di autenticazione con il proprio `Solid Identity Provider` non è richiesta.

La seconda applicazione trattata in questo elaborato di tesi, chiamata `blog-validator`, permette, su richiesta dell'utente, di effettuare controlli riguardo l'autenticità dei dati mostrati da `my-solid-blog`. Tale applicazione ha lo scopo di evidenziare la semplicità con cui un'operazione di questo tipo può essere implementata in `Solid`.

4.1 Obiettivi del progetto

Il sistema `SADeB` consiste di due applicazioni differenti: `my-solid-blog` e `blog-validator`. Vengono ora elencate le specifiche relative all'applicazione `my-solid-blog`.

- L'applicazione `my-solid-blog` deve essere completamente decentralizzata, rispettando i principi della tecnologia `Solid`.
- Una volta avviata l'applicazione, l'utente deve poter scegliere la modalità con cui interagire con l'app, indicando se vuole autenticarsi con il proprio `Solid Identity`

Provider e modificare, creare ed eliminare i dati presenti all'interno del proprio blog, oppure se inserire la webId di un altro utente per visualizzare il blog di quest'ultimo.

- L'applicazione deve implementare un meccanismo di autenticazione con il server Solid, sfruttando le librerie messe a disposizione da Inrupt.
- Una volta che l'utente si è autenticato con il proprio Solid Identity Provider, l'applicazione my-solid-blog deve poter controllare se esiste all'interno del Pod dell'utente un SolidDataset relativo al blog. In questo caso, my-solid-blog deve caricare tale SolidDataset per poter leggere e modificare i dati contenuti in esso. Altrimenti l'applicazione deve creare tale dataset all'interno del Pod dell'utente. Il nome del SolidDataset utilizzato dall'applicazione è articlelist.ttl.
- I dati contenuti all'interno del SolidDataset articlelist.ttl utilizzato da my-solid-blog, devono rappresentare degli articoli da pubblicare sul blog relativo all'utente che è proprietario del Pod. Per poter rappresentare tali dati, l'applicazione deve far uso del vocabolario RDF Schema.
- Ogni articolo deve avere un titolo, un testo che ne rappresenti il contenuto, una data di creazione e un identificatore univoco. Ogni articolo è un oggetto e un'istanza della classe TextDigitalDocument del vocabolario Schema. L'applicazione deve permettere all'utente di visualizzare il titolo, il contenuto e la data di creazione di ciascun articolo presente.
- L'applicazione my-solid-blog deve permettere all'utente autenticato di leggere, modificare e cancellare i dati, cioé gli articoli, presenti all'interno del SolidDataset Articlelist.ttl. A tal proposito my-solid-blog deve utilizzare le librerie Inrupt per poter svolgere queste operazioni sui dati contenuti all'interno del Pod.
- L'utente autenticato deve essere, inoltre, in grado di scrivere nuovi articoli all'interno del proprio blog.
- Un utente può scegliere di accedere al contenuto di un blog di un secondo utente, inserendo la sua webId senza passare per alcun processo di autenticazione. In questo caso, se esiste il SolidDataset articlelist.ttl all'interno del Pod corrispondente alla webId inserita, l'applicazione deve permettere all'utente di visualizzare gli articoli relativi alla webId inserita.
- Se l'utente accede ad un blog inserendo la webId corrispondente al Pod ove questo è memorizzato, senza quindi passare per un processo di autenticazione, non deve essere in grado di modificare il contenuto degli articoli all'interno di tale Pod. Inoltre l'utente non autenticato non deve avere i permessi necessari per scrivere nuovi articoli all'interno del SolidDataset articlelist.ttl corrispondente alla webId inserita.

- L'applicazione `my-solid-blog` deve poter comunicare tramite il protocollo `HTTP` con il server `blog-validator`, per potergli inviare, su richiesta dell'utente, i dati necessari ad effettuare un controllo all'interno del `Pod` del proprietario di questi dati. Per implementare tale funzionalità, `my-solid-blog` deve renderizzare per ogni articolo un pulsante `Check` per innescare tale controllo sull'autenticità dei contenuti visualizzati. Il controllo è riferito al singolo articolo selezionato, e non all'intero `SolidDataset`.
- Ogni utente, indipendentemente dal fatto che abbia completato l'operazione di autenticazione o meno con il server `Solid`, deve essere in grado di richiedere un controllo sull'autenticità dei contenuti mostrati da `my-solid-blog`.
- L'applicazione `my-solid-blog` deve mettere a disposizione un link che rimandi a `blog-validator` per poter permettere a tutti gli utenti, autenticati e non, di verificare che eventuali controlli sull'autenticità dei dati effettuati siano stati eseguiti correttamente.
- Ogni utente deve poter visualizzare l'URL associato all'articolo che sta leggendo, ovvero l'URL ove tale risorsa è stata memorizzata.
- Le interfacce grafiche dell'applicazione, create tramite `React`, devono essere semplici e intuitive, in modo tale da rendere l'applicazione user friendly.

Vengono qui di seguito elencate le specifiche relative all'applicazione `blog-validator`.

- Tramite il protocollo `HTTP`, l'applicazione deve poter ricevere da `my-solid-blog` le informazioni necessarie per effettuare un controllo sull'autenticità dei dati mostrati. In particolare `blog-validator` si occupa di verificare che un singolo articolo, scelto dall'utente, sia effettivamente presente all'interno del `Pod` dell'utente proprietario del blog. Una volta terminato tale controllo, l'applicazione `blog-validator` deve inviare a `my-solid-blog` l'esito del controllo stesso.
- I possibili esiti di un controllo sono i seguenti:

  1) L'articolo in questione non esiste all'interno del `Pod` del proprietario.
  2) L'articolo esiste, ma il contenuto risulta essere differente.
  3) L'articolo esiste e presenta lo stesso contenuto, ma la data in cui è stato pubblicato è differente.
  4) Il controllo sull'autenticità del contenuto ha dato un risultato positivo: in questo caso il server non ha rilevato alcuna anomalia.

- `blog-validator` deve mettere a disposizione un URL per permettere all'utente di visualizzare i dati relativi ai controlli già effettuati sul `Pod` di un utente. Questa funzionalità di `blog-validator` serve ad assicurarsi che `my-solid-blog` abbia inviato correttamente i dati relativi ai controlli effettuati, senza manometterli.

- `blog-validator`, infine, deve poter permettere all'utente di effettuare controlli manualmente sulle risorse relative ai blog creati da `my-solid-blog`; insererendo l'url del articolo che si vuole controllare e del `SolidDataset` dove è memorizzato, `blog-validator` deve mostrare all'utente contenuto, titolo e data di creazione dell'articolo, nel caso in cui questo sia effettivamente presente all'interno del `Pod`.

4.2 Modalità di accesso ai dati contenuti nel Pod

Nel corso dello svolgimento del progetto si è rivelata fondamentale la comprensione delle modalità con cui un'applicazione `Solid` debba accedere ai dati contenuti all'interno del `Pod`. Innanzitutto entrambe le applicazioni, `my-solid-blog` e `blog-validator`, utilizzano la libreria `Inrupt solid-client` per poter interagire con tali dati; la libreria mette a disposizione una serie di funzioni per poter scaricare `SolidDataset` e per poter modificare gli oggetti al suo interno.

Vengono qui mostrate le funzioni più importanti contenute in tale libreria:

$getSolidDataset(url, options?) : Promise < SolidDataset \& WithServerResourceInfo >$

Funzione che restituisce una `promessa` che si risolve nel `SolidDataset` corrispondente all'URL indicato; l'utente deve possedere il diritto in lettura su tale risorsa per poter caricare tale `SolidDataset`. A tal scopo, se necessario, può passare alla funzione i dati relativi alla sessione di autenticazione con il `Solid Identity Provider`, per poter acquisire tale diritto in lettura.

$saveSolidDatasetAt < Dataset > (url, solidDataset, options?) : Promise < Dataset \& WithServerResourceInfo \& WithChangeLog >$

Tale funzione permette di salvare un determinato `SolidDataset` all'interno dell'URL specificato; si possono passare a questa funzione anche i dati relativi alla sessione di autenticazione con il `Solid Identity Provider` per acquisire diritto in scrittura sulle risorse del `Pod`. Il possesso del diritto in scrittura è una condizione necessaria per poter utilizzare tale funzione. Restituisce una `promessa` che si risolve nel nuovo `SolidDataset` presente a seguito dell'esecuzione di questa funzione. Tale funzione, inoltre, può essere utilizzata per tenere traccia dei cambiamenti apportati all'interno di tale `SolidDataset`.

$createSolidDataset() : SolidDataset$

Permette di creare un nuovo `SolidDataset`.

$getThing(solidDataset, thingUrl, options?) : ThingPersisted|null$
questo esiste

Ritorna un oggetto presente all'interno di un `SolidDataset`, è necessario passare a tale funzione il `SolidDataset` che contiene tale oggetto e l'URL relativo all'oggetto che si vuole ottenere. La funzione ritorna il valore `null` se l'oggetto specificato non esiste.

$getThingAll(solidDataset, options?) : Thing[]$

Ritorna un array contenente tutti gli oggetti presenti all'interno di un `SolidDataset`.

$createThing(options) : ThingPersisted$

Crea un nuovo oggetto.

$setThing < Dataset > (solidDataset, thing) : Dataset\&WithChangeLog$

Permette di aggiungere un nuovo oggetto all'interno di un `SolidDataset`, rimpiazzando evenutali precedenti instanze di tale oggetto.

$removeThing < Dataset > (solidDataset, thing) : Dataset\&WithChangeLog$

Rimuove un oggetto contenuto all'interno di un `SolidDataset`.

$asUrl(thing, baseUrl) : UrlString$
questo esiste
Restituisce l'URL dell'oggetto passato come parametro.

$getUrl(thing, property) : UrlString|null$

Permette di restituire l'oggetto di uno `statement` contenuto all'interno di un oggetto. È necessario passare a tale funzione l'oggetto e il predicato di tale `risorsa`, cioè il soggetto e il predicato dello `statement RDF`. Restituisce il valore `null` se tale `risorsa` non esiste o se il tipo della `risorsa` non è URL. Se esistono più `risorse` di tipo URL corrispondenti a tale predicato, la funzione restituisce un URL fra quelli presenti.

$getUrlAll(thing, property) : UrlString[]$

Restituisce tutti i valori URL relativi alla proprietà di una `risorsa`.

Esistono inoltre funzioni simili per poter ottenere differenti tipologie di dati, come ad esempio: `string, boolean, decimal, integer` e altre ancora.

$addUrl < T > (thing, property, value) : T$

Crea un nuovo oggetto con un URL aggiunto come proprietà; eventuali altri valori relativi a tale proprietà non vengono modificati.

Esistono funzioni simili per poter aggiungere proprietà relative ad altre tipologie di dati, come per esempio: `string, boolean, decimal, integer` e altre.

$setUrl < T > (thing, property, value) : T$

Crea un nuovo oggetto con gli esistenti valori relativi ad una proprietà rimpiazzati con l'URL passato come parametro.

Esistono funzioni simili per poter sostituire proprietà relative ad altre tipologie di dati, come ad esempio: `string, boolean, decimal, integer` e altre.

Per accedere ad alcuni dati relativi alle informazioni sul profilo dell'utente sono stati utilizzati anche i componenti della libreria `Inrupt solid-ui-react` $< CombinedDataProvider >$ e $< Text >$.

4.3 Modalità di autenticazione con il Solid Identity Provider

Come già accennato riguardo alle specifiche relative all'applicazione `my-solid-blog`, l'applicazione in questione deve poter permettere ai propri utenti di autenticarsi con il proprio `Solid Indentity Provider` per modificare il proprio blog. Per implementare questa funzionalità è stata utilizzata la libreria `Inrupt solid-ui-react`, la quale mette a disposizione componenti `React` per autenticarsi e per leggere/scrivere dati.

Vengono ora elencati i componenti appartenenti a questa libreria usati per implementare tale funzionalità:

$< SessionProvider >$

Tale componente è usato per avvolgere la propria applicazione; è necessario passare al compomente una `sessionId` universalmente unica. La `sessionId` utilizzata nel caso dell'applicazione `my-solid-blog` è `my-solid-blog`. Tale componente permette di utilizzare all'interno del progetto il componente $< LoginButton >$.

$< LoginButton >$

Permette di autenticarsi con il proprio `Solid Identity Provider`, avviando quindi il procedimento di autenticazione descritto all'interno del capitolo 2. È necessario passare a tale componente le seguenti informazioni:

- `oidcIssuer`: in questo campo va indicato l'URL del `Solid Identity Provider` con il quale ci si vuole autenticare, come per esempio `https://inrupt.net/`.
- `redirectUrl`: indica l'URL sul quale l'utente viene reindirizzato a seguito del processo di autenticazione. Poiché l'applicazione `my-solid-blog` è di tipo single-page e utilizza la porta 3000, il `redirectUrl` usato è pari a `http://localhost:3000/`.
- `authOptions`: opzionale, serve ad indicare informazioni aggiuntive relative alla sessione di autenticazione. `my-solid-blog` utilizza tale attributo per indicare il nome dell'applicazione che si sta autenticando con il server `Solid`, che è semplicemente `my-solid-blog`.

useSession()

Funzione contenuta all'interno della libreria `solid-ui-react`, restituisce dati relativi alla sessione di autenticazione; il codice sotto indicato consente di raccogliere tali dati.

```
const  session  = useSession();
```

L'oggetto `session` contiene quindi dati relativi alla propria sessione di autenticazione, come ad esempio:

- `session.info.webId`: contiene la `webId` dell'utente autenticato.
- `session.info.isLoggedIn`: proprietà pari a `true` se l'utente ha completato il processo di autenticazione ed è attualmente loggato con il proprio `Solid Identity Provider`, altrimenti tale proprietà è pari a `false`.
- `session.fetch`: metodo usato per interagire con i dati contenuti all'interno del `Pod`, usando le informazioni relative al login dell'utente. Tale funzione può essere passata come metodo alle funzioni `getSolidDatasetAt` e `saveSolidDatasetAt` per ottenere i permessi necessari a completare tali operazioni sui dati all'interno del `Pod`.

Capitolo 5

Il sistema SADeB

Come già precedentemente indicato, il sistema `SADeB` (Solid Authenticity Decentralized Blog), è formato dalle due applicazioni `my-solid-blog` e `blog-validator`.

Vengono ora descritti nel dettaglio l'implementazione e le modalità di funzionamento dell'applicazione my-solid-blog.

5.1 Directory e file presenti in my-solid-blog

Come già accennato, `my-solid-blog` è un'applicazione creata tramite il comando `create-react-app` di `npx`, il quale permette di sviluppare applicazioni in `React` single-page. All'interno della directory principale sono presenti le seguenti cartelle:

- `node-modules`: cartella contenente tutti i package scaricati tramite `npm` per il progetto `JavaScript` in questione. Alcuni di questi package sono stati scaricati nel momento in cui l'applicazione è stata creata tramite il comando `create-react-app`, ad esempio quelli relativi a `babel` o `webpack`. Altri package, invece, sono stati successivamente aggiunti tramite il comando `npm install`, come ad esempio: `solid-client` e `rdf-namespaces`.
- `public`: La cartella `public` contiene il file `HTML` in modo da poterlo modificare, per esempio per impostare il titolo del progetto e alcuni metadati relativi all'applicazione. Le cartella `public`, secondo la documentazione di `React`, può essere utilizzata per uno dei seguenti casi:

  1) Se si hai bisogno di un file con un nome specifico nell'output di compilazione.
  2) Se sono presenti nel progetto migliaia di immagini e si ha la necessità di referenziare dinamicamente il loro percorso.
  3) Se si vuole includere in questa cartella un piccolo file `JavaScript` lontano dagli altri file `.js`.

4) Se alcune librerie sono incompatibili con `webpack` e vanno pertanto necessariamente incluse all'interno del file `HTML`, tramite il tag `<script>`.

Nel caso specifico del progetto `my-solid-blog`, la cartella `public` è stata usata, oltre che per indicare il titolo e alcuni metadati relativi all'applicazione, per memorizzare l'icona di `my-solid-blog`.

- `src`: la cartella `src` contiene tutti i principali file `JavaScript` del progetto. Il file `index.js`, creato direttamente dal comando `create-react-app`, è l'entry point di `JavaScript`.

  Oltre ai file formato `.js`, la cartella `src` contiene anche il file `CSS` per la formattazione del documento `HTML` generato da `React`.

  `React` permette agli sviluppatori di riorganizzare la cartella `src` in diverse sottocartelle per poter raggruppare alcuni file `JavaScript` aventi caratteristiche in comune. Nel caso particolare di `my-solid-blog`, sono presenti due sottocartelle chiamate `components` e `utils`.

All'interno della directory principale del progetto, oltre a queste tre cartelle, sono presenti anche due file con le seguenti funzioni:

- `package-lock.json`: file che viene modificato a seguito di ogni operazione di `npm` che vada a cambiare il contenuto della cartella `node-modules` o il file `package.json`. Descrive l'esatta struttura della cartella `node-modules`.
- `package.json`: file contenente importanti informazioni riguardo al progetto, come il nome del progetto stesso e una lista di dipendenze richieste da esso.

5.2 Directory src

La cartella `src`, come già detto, contiene tutti i file `JavaScript` e `CSS` per la creazione del documento `HTML` da parte di `React`: gestisce quindi l'impaginazione, la formattazione e stabilisce il comportamento dell'applicazione.

All'interno di tale directory sono presenti i seguenti file `JavaScript`:

- `index.js`: entry point di `JavaScript`
- `App.js`: file `JavaScript` solitamente presente in progetti `React`, all'interno di `my-solid-blog` viene utilizzata per capire quale interfaccia grafica deve essere mostrata all'utente: `App.js` si occupa di renderizzare un'interfaccia grafica diversa all'utente, una volta che quest'ultimo ha effettuato l'accesso all'interno dell'applicazione.

All'interno della cartella `components`, sottocartella di `src`, sono contenuti tutti i file `JavaScript` che definiscono le varie interfacce utente utilizzate da `my-solid-blog`. Nel dettaglio i file contenuti in `components` sono:

- `ChoiceMenu.js`: Permette all'utente di scegliere se autenticarsi o inserire la `webId` di un altro utente.
- `EntryArea.js`: si occupa di renderizzare l'interfaccia grafica dell'applicazione quando l'utente non ha ancora effettuato l'accesso. In base alla scelta effettuata dall'utente, renderizza i componenti necessari per loggarsi con il proprio `Solid Identity Provider`, oppure per poter inserire una `webId` di un secondo utente.
- `MainPage.js`: una volta effettuato l'accesso, si occupa di renderizzare le interfacce grafiche fornite da `Header.js` e `Articlelist.js`.
- `Header.js`: renderizza nella parte superiore della pagina web alcune informazioni riguardanti l'utente proprietario del `Pod`, come il nome, l'organizzazione a cui appartiene e il ruolo in tale organizzazione.
- `Articleslist.js`: importando `Article.js` renderizza l'intera lista di articoli presenti all'interno del `Pod`. Nel caso in cui l'utente sia autenticato, renderizza anche l'interfaccia grafica fornita da `WriteArticles.js`.
- `Article.js`: Crea un'interfaccia grafica per mostrare il contenuto di un articolo, indicando titolo, testo e data. Inoltre, qualora l'utente sia autenticato, permette di modificare o cancellare un articolo.
- `WriteArticles.js`: Permette di scrivere nuovi articoli all'interno del proprio `Pod`.

Oltre a `components` è presente una seconda sottocartella chiamata `utils`. Tale directory contiene i seguenti file:

- `GetOrCreateDataset`: tale file esporta una funzione necessaria per poter scaricare il dataset contenente gli articoli salvati all'interno del `Pod`, chiamata `getOrCreateDataset()`. Tale funzione controlla prima di tutto se esiste già un `SolidDataset` all'interno del `Pod`, in cui sono memorizzati gli articoli relativi all'utente; se questo esiste, allora viene correttamente scaricato, altrimenti tale `SolidDataset` viene inizializzato dall'applicazione stessa.
- `fontawesome.js`: file di secondaria importanza, permette di scaricare alcune icone dalle librerie `fontawesome` per poterle utilizzare all'interno del progetto.

Oltre a queste cartelle e file, la directory `src`, come già affermato, contiene anche i file necessari per la formattazione del documento `HTML` generato. A tal proposito l'applicazione `my-solid-blog` utilizza il framework `CSS Bulma`.

Il file `index.scss`, il quale si occupa della formattazione del documento `HTML` creato da `my-solid-blog`, in accordo con la documentazione di `Bulma`, importa solo i componenti del framework `Bulma` utilizzati all'interno del progetto per la formattazione del documento `HTML`. Inoltre modifica alcune variabili relative a tale framework e scarica i font utilizzati dall'applicazione.

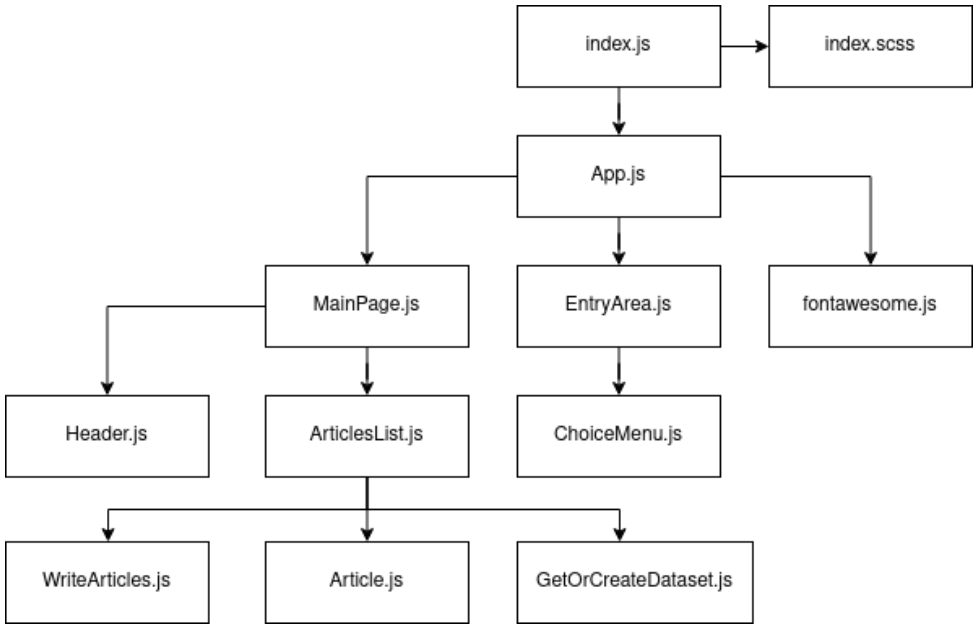

Figura 5.1: L'albero delle dipendenze dei moduli di `my-solid-blog` contenuti all'interno della directory `src`.

5.3 Gestione delle interfacce grafiche

Importante argomento all'interno di `my-solid-blog` è quello riguardante la renderizzazione delle interfacce grafiche presenti all'interno del progetto. Infatti l'utente deve poter visualizzare differenti interfacce grafiche ad esempio in relazione al fatto che abbia già effettuato l'accesso o meno.

`React` effettua un rerender dei propri componenti ogni volta che lo stato di un componente cambia. A tale scopo è stato utilizzato l'`hook state` di `React`, per poter andare a modificare lo stato dei vari componenti in base alle decisioni prese dall'utente all'interno dell'applicazione.

Innanzitutto va precisato che gli `hooks` sono una feature di `React` aggiunta recentemente; questa permette di utilizzare `state` ed altre features caratteristiche di `React` senza dover necessariamente scrivere una classe, come precedentemente richiesto. Al contrario utilizzando l'`hook state`, è stato possibile realizzare l'applicazione `my-solid-blog` senza dover scrivere alcuna classe.

Viene qui di seguito riportato un esempio di codice per l'utilizzo dell'hook `State`.

```
import React, { useState } from 'react';
 ...
const [state, setState] = useState("")
 ...
```

La variabile `state`, in questo esempio, rappresenta il valore attuale dello stato, mentre la funzione `setState()` permette di modificare lo stato `state`; il valore contenuto all'interno di `useState()`, in questo esempio `""`, rappresenta il valore dello stato nel momento in cui tale componente viene caricato per la prima volta.

All'interno del progetto `my-solid-blog`, tale hook è stato, per esempio, utilizzato in `App.js`, per determinare quale interfaccia grafica mostrare all'utente in relazione al fatto che quest'ultimo abbia già completato la procedura di accesso o meno. Viene ora mostrato parte del codice contenuto nel modulo `App.js`.

```
import React, { useState } from 'react';
...
function App () {
  const [entered, setEntered] = useState(false);
  const [webId, setWebId] = useState("");
  return (
    <div>
    { !session.info.isLoggedIn && !entered ? (
      <EntryArea
        entering={() => setEntered(true)}
        setWebId={(id) => setWebId(id)}
      />
    ) : (
      <MainPage
        webId={webId}
      />
    )}
    </div>
  );
}
```

Il codice sopra mostrato è un perfetto esempio di come il problema di rerenderizzazione delle interfacce grafiche è stato affrontato all'interno dell'applicazione `my-solid-blog`. In questo caso lo stato `entered` è una variabile booleana pari a `false` nel caso in cui

l'utente debba ancora accedere all'applicazione e pari a `true` altrimenti. Lo stato `webId` serve invece per poter salvare la `webId` del proprietario del blog da visualizzare, nel caso in cui non si voglia accedere al proprio blog autenticandosi. Se l'utente non ha ancora effettuato l'accesso all'applicazione, il modulo `App.js` renderizza il componente $< EntryArea >$ che gestisce l'interfaccia grafica dell'applicazione prima di aver effettuato l'accesso, altrimenti viene renderizzato il componente $< MainPage >$ il quale contiene anche l'interfaccia grafica relativa ad ogni singolo articolo. La funzione $() => setEntered(true)$ viene passata al componente $< EntryArea >$ per permettergli di modificare lo stato `entered` di `App.js`. In questo modo `EntryArea.js` può utilizzare tale funzione nel momento in cui l'utente ha effettuato l'accesso al blog appartenente ad una determinata `webId` selezionata. Modificando lo stato `entered`, `EntryArea.js` forza `App.js` a rerenderizzare la propria interfaccia grafica, in modo tale da visualizzare la lista di articoli presenti nel blog selezionato, renderizzando il componente $< MainPage >$.

5.4 Container e risorse utilizzati per la gestione del blog

Dalle specifiche dell'applicazione `my-solid-blog` risulta che i dati relativi agli articoli devono essere necessariamente pubblici, in quanto devono essere accessibili anche da parte utenti non autenticati. Infatti, nota la `webId` di un utente, chiunque deve essere in grado di leggere il suo blog tramite l'applicazione `my-solid-blog`.

Da tali considerazioni si evince che il `container` ideale per salvare le informazioni inerenti al blog è il `container public`. Tale `container` è normalmente presente di default in ogni `Pod`. Le risorse `ACL` di tale `container` consentono di avere diritto in lettura sulle risorse all'interno di tale `container`; questo significa che le risorse interne a tale `container` possono essere scaricate anche senza essersi precedentemente autenticati con il server `Solid`. Inoltre all'interno del `container setting`, anch'esso di norma presente di default all'interno del `Pod`, è contenuto un `SolidDataset` chiamato `publicTypeIndex.ttl` avente lo scopo di tenere traccia di tutti i `SolidDataset` accessibili pubblicamente all'interno del `Pod`; anche la risorsa `publicTypeIndex.ttl` è accessibile pubblicamente in lettura agli utenti non autenticati. L'applicazione `my-solid-blog` salva il proprio `SolidDataset`, chiamato `articlelist.ttl`, all'interno di un `container` chiamato
`my-solid-blog`, creato dall'applicazione stessa, e situato a sua volta all'interno del `container public` per le considerazioni esposte in precedenza. Poiché `articlelist.ttl` è una risorsa pubblicamente accessibile, `my-solid-blog` riporta all'interno del `SolidDataset` `publicTypeIndex.ttl` alcune informazioni riguardanti `articlelist.ttl`, tra le quali l'URL della risorsa per poterla scaricare tramite `publicTypeIndex.ttl`. Infine l'applicazione legge anche alcuni dati contenuti direttamente all'interno del `SolidDataset` relativo al profilo dell'utente. Il profilo di un utente è anch'esso una risorsa pubblicamente

accessibile. Le informazioni lette dal profilo sono, ad esempio, il nome, l'organizzazione di cui fa parte l'utente e il ruolo che ha in tale organizzazione; queste informazioni vengono lette dall'applicazione per poterle mostrare all'utente insieme agli articoli contenuti all'interno del blog.

5.5 Scaricamento del SolidDataset articlelist.ttl

Nel momento in cui l'applicazione viene avviata, essa necessita di scaricare il `SolidDataset` `articlelist.ttl` dal `Pod` dell'utente. I moduli che si occupano di eseguire queste operazioni sono `ArticlesList.js` e `GetOrCreateDataset.js`. Vengono qui di seguito elencate le operazioni eseguite da questi moduli per poter scaricare tale risorsa.

- Per prima cosa è necessario conoscere la `webId` dell'utente in questione. La `webId` rappresenta l'URL dell'oggetto chiamato `me` all'interno del profilo dell'utente sul quale vengono memorizzate le informazioni relative all'utente proprietario del `Pod` e alla struttura del `Pod` stesso. Se un utente si è autenticato con il `Solid Identity Provider` allora l'applicazione può leggere la sua `webId` utilizzando l'oggetto `session` e in particolare la proprietà `session.info.webId`. Se l'utente ha effettuato l'accesso senza autenticarsi, allora ha dovuto indicare la `webId` dell'utente proprietario del blog che vuole leggere. Questo significa che, anche in questo caso, l'applicazione conosce la `webId` da utilizzare.

- All'interno dell'oggetto `me` è presente l'URL della radice del `Pod`, necessario per poter conoscere l'ubicazione dei `container` `settings` e `public`. Tale informazione è salvata utlizzando il vocabolario `RDF` `http://www.w3.org/ns/pim/space#`, e in particolare, la proprietà `storage` di tale vocabolario. Come descritto nel precedente capitolo, tale valore può essere letto utilizzando la funzione `getUrl()`.

- Il vocabolario `<http://www.w3.org/ns/solid/terms#>`, ovvero il vocabolario di `Solid`, viene utilizzato all'interno dell'oggetto `me` per salvare l'URL del `SolidDataset` `publicTypeIndex.ttl`; in particolare tale URL viene salvato utilizzando la proprietà `publicTypeIndex` del vocabolario `solid`.

- Viene scaricato il `SolidDataset` `publicTypeIndex.ttl`. Tale `SolidDataset`, come in parte già accennato, contiene una lista di oggetti che si riferscono alle risorse accessibili pubblicamente all'interno del `Pod`. L'applicazione usa `publicTypeIndex.ttl` per controllare se esite un oggetto al suo interno che si riferisce ad `articlelist.ttl`.

- Se esiste tale oggetto riferito ad `articlelist.ttl`, questo significa che tale `SolidDataset` esiste già all'interno del `Pod`, cioè è stato creato in precedenza. In questo caso l'applicazione deve solamente leggere il valore sotto la proprietà `instance` del vocabolario `solid` per conoscere l'URL sotto il quale `articlelist.ttl` è stato

salvato. Una volta noto l'URL di `articlelist.ttl`, l'applicazione può procedere scaricando tale `SolidDataset`.

- Nel caso in cui tale oggetto non esista, questo significa che `articlelist.ttl` non è stato inizializzato. L'applicazione deve quindi procedere creando tale risorsa.
- Viene quindi composto l'URL della risorsa `articlelist.ttl` da creare, aggiungendo all'URL della radice del `Pod`, precedentemente ottenuto utilizzando l'oggetto `me`, la stringa "public/my-solid-blog/articlelist.ttl".
- Il `SolidDataset` in questione viene creato sotto tale URL; tale operazione può essere compiuta solo se l'utente ha effettuato l'accesso all'applicazione completando l'operazione di logging. In caso contrario `my-solid-blog` non possiede il diritto in scrittura necessario per effettuare tale operazione, ed è costretto ad arrestarsi, senza permettere all'utente di visualizzare alcun articolo.
- Come ultima operazione viene creato un oggetto all'interno di `publicTypeIndex.ttl`, per poterci salvare l'URL scelto per `articlelist.ttl`. In questo modo l'applicazione potrà utilizzare tale informazione la volta successiva in cui dovrà leggere tale blog per poter scaricare tale `SolidDataset`.

5.6 Struttura di articlelist.ttl

`articlelist.ttl` contiene al suo interno vari oggetti, ognuno dei quali si riferisce ad un articolo differente pubblicato all'interno del `Pod`. Nel momento in cui si accede all'interno di `my-solid-blog`, l'applicazione permette all'utente di visualizzare la lista di articoli presenti all'interno di tale `SolidDataset`.

I singoli articoli sono stati definiti utilizzando il vocabolario `RDF` `http://schema.org/`, in particolare sono state utilizzate le seguenti proprietà e classi di tale vocabolario.

- Ogni oggetto-articolo è un'istanza della classe `TextDigitalDocument`, classe definita, appunto, nel vocabolario `schema`; tale classe ha lo scopo di rappresentare file composti prevalentemente da testo.
- La proprietà `dataCreated`, viene utizzata per indicare la data di creazione dell'articolo.
- La proprietà `headline` indica il titolo dell'articolo.
- La proprietà `text` è utilizzato per poter salvare il contenuto di un articolo.
- Infine la proprietà `identifier` viene utilizzata per poter identificare univocamente ogni articolo presente in `articlelist.ttl`.

Come indicato all'interno della lista delle specifiche, l'applicazione mostra all'utente il titolo, la data e il contenuto di ogni articolo. L'identifier viene invece utilizzato come chiave di `React`, per poter permettere a `React` di renderizzare correttamente la lista di articoli presenti. Infine, su richiesta dell'utente, `my-solid-blog` visualizza all'utente l'URL in cui la risorsa relativa al singolo articolo è stata memorizzata. Tale funzione, come verrà spiegato in seguito, è necessaria per permettere all'utente di interagire correttamente con il server `blog-validator`.

5.7 Gestione degli articoli

Come già in parte accennato, i moduli che si occuppano di gestire e renderizzare la lista di articoli presenti all'interno del blog sono: `ArticlesList.js` e `Article.js`.

Nel dettaglio `Articleslist.js` si occupa di scaricare il `SolidDataset` `Articlelist.ttl`, di renderizzare e gestire la lista di articoli presenti importando `Article.js`, di aggiungere nuovi articoli alla lista per gli utenti autenticati, di definire le funzioni per poter modificare o eliminare un articolo e di passare alcune informazioni come parametri necessari alla gestione dei singoli articoli ad `Article.js`.

Il modulo `Article.js` invece ha i seguenti compiti: renderizzare i singoli articoli contenuti all'interno del blog, utilizzare le funzioni definite in `ArticlesList.js` per permettere agli utenti autenticati di modificare o cancellare un articolo, consentire a `my-solid-blog` e `blog-validator` di comunicare tra di loro per verificare l'autenticità dei singoli articoli mostrati da `my-solid-blog`.

Una volta caricato il `SolidDataset` `Articlelist.ttl`, il modulo `ArticlesList.js` utilizza la funzione `getThingAll()` della libreria `solid-client` per poter ottenere un array contenente tutti gli articoli del blog. Tale array viene quindi ordinato in base alla data di creazione degli articoli stessi, utilizzando la funzione `byDate()`, anch'essa definita all'interno di `ArticlesList.js`. Una volta che tale operazione è terminata, viene eseguito il metodo `map()` sull'array precedentemente ottenuto, in modo tale da renderizzare il contenuto dei singoli articoli utilizzando il modulo `Article.js`; tale funzione `map()` ritorna il singolo componente $< Article >$ per ogni elemento presente all'interno dell'array. Al componente $< Article >$ vengono passati i seguenti argomenti:

- `thing`: oggetto rappresentante il singolo articolo all'interno del blog; tramite esso è possibile leggere e modificare le informazioni relative al singolo articolo.
- `remove`: funzione che permette all'utente autenticato di eliminare un articolo all'interno del blog. Questa funzione asincrona riceve in ingresso l'oggetto, ovvero l'articolo, da eliminare da `articlelist.ttl`; una volta eliminato la funzione salva all'interno del `Pod` il `SolidDataset` aggiornato.

- `change`: funzione asincrona avente lo scopo di modificare il contenuto di un articolo: in particolare permette all'utente autenticato di modificare titolo e contenuto dell'articolo. Riceve in ingresso l'oggetto da modificare, il nuovo titolo e il nuovo contenuto. Una volta che l'articolo viene aggiornato la funzione salva all'interno del `Pod` il `SolidDataset` aggiornato.
- `webId`: variabile contenente la `webId` del proprietario del blog. Tale variabile viene utilizzata solamente se l'utente ha effettuato l'accesso senza autenticarsi; in caso contrario la `webId` può essere letta direttamente da `session.info.webId`.

`Article.js` mette quindi a disposizione due componenti $<$ *button* $>$ per permettere all'utente di modificare ed eliminare l'articolo renderizzato; tali componenti vengono visualizzati dall'utente solo se la variabile `session.info.isLoggedIn` è pari a `true`, cioè solamente se l'utente si è correttamente autenticato con il proprio `Solid Identity Provider`.

5.8 Comunicazione con l'applicazione blog-validator

Per effettuare un controllo sull'autenticità dei contenuti mostrati all'interno del blog dell'utente, l'applicazione `my-solid-blog` comunica attraverso il protocollo `HTTP` con il server `blog-validator`.

A tale proposito viene utilizzato il metodo `POST` per effettuare una richiesta a `blog-validator`, dopodichè `my-solid-blog` aspetta la risposta del server per poter mostrare all'utente l'esito di tale controllo.

Per inviare tale richiesta `POST` al server, come già esposto all'interno del capitolo 2, l'applicazione `my-solid-blog` fa uso della libreria `axios`; in particolare utilizza il metodo `axios.post()`; tale metodo ritorna all'applicazione una promessa. L'applicazione `my-solid-blog` rimane in attesa che tale promessa si risolva o venga rifiutata. Nel caso in cui si risolva, allora questo significa che la comunicazione con il server è andata a buon fine. In questo caso l'applicazione può quindi mostrare all'utente l'esito del controllo effettuto da `blog-validator`; in caso contrario, qualora la promessa venga rifiutata, si è verificato un errore durante tale comunicazione.

Per poter comunicare l'applicazione `my-solid-blog` utilizza la porta `3000`, mentre `blog-validator` utilizza la porta `8081`. Gli argomenti passati alla funzione `axios.post()` sono i seguenti: l'URL su cui il server è in ascolto, che è pari a `http://localhost:8081/auth` e tutte le informazioni necessarie al server per effettuare tale controllo attraverso un file formato `JSON`. In particolare le proprietà presenti all'interno di tale file JSON sono le seguenti:

- `webId`: viene passata a `blog-validator` la `webId` del proprietario del `Pod`. La `webId` del proprietario è necessaria a comprendere se il server ha cercato le informazioni all'interno del `Pod` corretto.
- `urlDataset`: URL relativo al `SolidDataset` da scaricare; `my-solid-blog` invia, infatti, a `blog-validator` l'URL relativo a `articlelist.ttl`, per poterlo scaricare e poter leggere i dati al suo interno.
- `urlThing`: URL in cui la risorsa relativa all'articolo è stata memorizzata, permette al server di trovare l'articolo in questione all'interno del `SolidDataset`, nel caso in cui questo esista effettivamente all'interno di `articlelist.ttl`.
- `title`: titolo dell'articolo, necessario per controllare che questo non sia stato alterato da `my-solid-blog`.
- `content`: contenuto dell'articolo, inviato per le stesse motivazioni della proprietà `title`.
- `date`: data di creazione dell'articolo; è necessario inviare tale informazione per controllare che anche questo dato non sia stato manomesso in nessun modo dall'applicazione.

L'applicazione invia tale richiesta `POST` a `blog-validator` ogni volta che si preme il pulsante chiamato `Check`, situato al di sotto di ogni articolo renderizzato da `my-solid-blog`. L'esito del controllo viene anch'esso stampato al di sotto dell'articolo controllato.

5.9 Utilizzo dell'hook Effect

Oltre all'`hook State`, l'aplicazione `my-solid-blog` utilizza un altro `hook` di `React`, chiamato `Effect`, anch'esso aggiunto recentemente. Tale `hook` è utilizzato per implementare side effects all'interno dei componenti delle funzioni. Esempi di side effects all'interno dei componenti `React` sono: il download di dati, l'impostazione di una iscrizione o il cambiamento manuale del `DOM`.

Talvolta, quando si utilizza `React` è necessario eseguire del codice aggiuntivo, dopo che `React` ha aggiornato il `DOM`. `useEffect` serve ad informare i vari componenti `React`, su cosa questi sono tenuti a fare una volta terminata l'operazione di renderizzazione delle interfacce grafiche. `React` memorizza la funzione passata a questo `hook` e la esegue in un secondo momento, ovvero dopo che il `DOM` è stato aggiornato. Il codice all'interno `useEffect` viene di default eseguito ogni volta che `React` termina l'operazione di render.

Questo hook è utilizzato all'interno di `my-solid-blog` per scaricare il `SolidDataset` `articlelist.ttl`; infatti scaricare dati dalla rete è un tipico side effect da eseguire tramite `Effect`. Per chi ha familiarità con le classi `React`, l'`hook` `useEffect` sostituisce i metodi `componentDidMount`, `componentDidUpdate` e `componentWillUnmount` delle classi `React`.

Nel caso di `my-solid-blog`, viene passata all'`hook` `useEffect` non solo la funzione da eseguire, ma anche l'array `[session]`. Questo permette di ottimizzare l'applicazione, in quanto, passando questo argomento, la funzione all'interno di `useEffect` non viene più eseguita al termine di ogni renderizzazione da parte di `React`, ma soltanto quando uno o più elementi dell'array `[session]`, fra un render e l'altro, sono cambiati.

5.10 Link a blog-validator

All'interno dell'applicazione, come già precedentemente detto, è possibile inviare, tramite procollo `HTTP`, una richiesta `POST` al server `blog-validator`, con il fine di evitare la diffusione di informazioni false all'interno dell'applicazione `my-solid-blog`. Per effettuare tale controllo `my-solid-blog` invia a `blog-validator` alcune informazioni riguardanti i dati relativi all'articolo in questione. Una volta terminata la comunicazione con il server, l'applicazione `my-solid-blog`, infine mostra all'utente l'esito del controllo effettuato da `blog-validator`. Tuttavia l'utente, non avendo modo di vedere in prima persona i dati scambiati tra le due applicazioni, non può avere certezze riguardo i seguenti aspetti:

- L'utente non può verificare se l'applicazione `my-solid-blog` abbia effettivamente inviato correttamente i dati relativi all'articolo in questione. `my-solid-blog` potrebbe inviare intenzionalmente dati relativi ad un altro articolo, per poter andare a modificare l'esito del controllo effettuato da `blog-validator`.
- L'applicazione `my-solid-blog` potrebbe far credere all'utente che `blog-validator` non abbia rilevato anomalie durante il controllo, visualizzando a questo un esito differente da quello effettivamente rilevato dal server.

Per evitare tale problema all'interno dell'applicazione `my-solid-blog` è possibile trovare un link per potersi connettere direttamente con `blog-validator` tramite browser. Visitando tale URL, l'utente può vedere quali sono stati realmente i dati inviati da `my-solid-blog` a `blog-validator` insieme al reale esito di tale controllo. In questo modo l'utente può verificare in prima persona se il controllo sull'autenticità degli articoli mostrati da `my-solid-blog` può ritenersi valido o meno.

Vengono ora descritte le modalità di funzionamento e l'implementazione dell'applicazione blog-validator.

`blog-validator` è un server implementato tramite `Node.js`, utilizzato per controllare l'autenticità dei dati mostrati dall'applicazione `my-solid-blog`; `blog-validator` utilizza la porta 8081.

5.11 Directory e file presenti in blog-validator

Sono qui di seguito elencati tutti i file e le directory presenti all'interno del progetto `blog-validator`.

La cartella `node-modules` e i file `package-lock.json` e `package.json`, presenti all'interno dell'applicazione `blog-validator`, sono già stati trattati in quanto presenti anche all'interno del progetto `my-solid-blog`; per questo motivo non verranno ripetute le loro funzioni all'interno di questa sezione.

- `index.js`: file entry point `JavaScript` del progetto `blog-validator`. Definisce alcuni URL per effettuare richieste `GET` e `POST` da parte di terze applicazioni.
- `solid.db`: Database utilizzato da `blog-validator` per salvare alcuni dati necessari per il funzionamento dell'applicazione.
- `utils`: Directory contenente i moduli `solidRequest.js` e `requestsData.js`.
- `solidRequest.js`: File utilizzato per leggere alcuni dati all'interno del `Pod` dell'utente. La lettura di tale dati è necessaria per poter effettuare controlli riguardanti l'autenticità dei dati visualizzati da `my-solid-blog`.
- `requestsData.js`: Modulo per la gestione del database `solid.db`, si occupa della creazione di tabelle, dell'inserimento di tuple, e della lettura di alcuni dati all'interno di tale database.

5.12 Tecnologie utilizzate per il funzionamento di blog-validator

Vengono ora brevemente descritti alcuni moduli fondamentali per il funzionamento del server `blog-validator`.

Express

`Express` è un framework per applicazioni web Node.js che fornisce una serie di funzioni avanzate per le applicazioni web e per dispositivi mobili; è ormai definito come il server framework standard de facto per `Node.js`. Tale framework permette la creazione di `API` in maniera semplice e veloce.

cookie-session

I `Session cookies` sono i cookie temporanei che vengono generati principalmente sul lato server, il cui uso principale è quello di tenere traccia di tutte le informazioni della richiesta che è stata fatta dal cliente in una particolare sessione. La sessione viene memorizzata solo temporaneamente, quando l'utente chiude la sessione del browser costui la distrugge in automatico.

Il modulo `express-session` chiamato `cookie-session` è usato per la gestione delle sessioni in `Node.js`.

body-parser

`body-parser` è una libreria `npm` usata per processare i dati inviati al server attraverso un metodo di richiesta `HTTP`.

sqlite3

Libreria `npm` utilizzata per la creazione e la gestione del database utilizzato da `blog-validator`.

5.13 API methods di blog-validator

Vengono ora elencati gli API methods messi a disposizione da `blog-validator`.

- `POST`, `http://localhost:8081/auth`: Questo metodo è utilizzato dall'applicazione `my-solid-blog` per inviare al server i dati necessari a controllare i dati richiesti dall'utente. I dati ricevuti da `my-solid-blog` vengono poi passati al modulo `solidRequest.js`, il quale si occupa di confrontare le informazioni ricevute con quelle presenti all'interno del `Pod`. I dati inviati da `my-solid-blog` vengono salvati nel database `solid.db`.
- `GET`, `http://localhost:8081/check`: Tale metodo è utilizzato per permettere all'utente di interagire con l'applicazione. Consente all'utente di controllare un articolo, una volta noti gli URL del suo `SolidDataset` e dell'articolo stesso, o di controllare il corretto svolgimento di un controllo sull'autenticità dei dati effettuato da `blog-validator` attraverso l'API method `POST`, `http://localhost:8081/auth`, con il fine di verificare che `my-solid-blog` non abbia inviato al server dati scorretti.

- `POST`, `http:localhost:8081/control`: Metodo utilizzato per inviare all'utente i dati relativi al controllo sul quale è stata richiesta una verifica. Attraverso il modulo `requestsData.js`, utilizza il database `solid.db` per caricare tali dati richiesti.
- `POST`, `http:localhost:8081/read`: Metodo utilizzato per inviare all'utente i dati relativi ad un articolo, ovvero titolo, contenuto e data di pubblicazione, salvato all'interno del `Pod`. Tale metodo può essere utilizzato per verificare che tali dati coincidano con quelli visualizzati da `my-solid-blog`.

5.14 Controllo sull'autenticità dei dati

Viene ora descritto nel dettaglio il procedimento, effettuato da `blog-validator`, per controllare l'autenticità dei dati riguardanti un articolo mostrato dall'applicazione `my-solid-blog`. L'articolo sul quale effettuare il controllo viene scelto direttamente dall'utente che sta utilizzando `my-solid-blog`.

- Come già indicato precedentemente, quando un utente che sta utilizzando l'applicazione `my-solid-blog` preme il pulsante "Check", `my-solid-blog` invia una richiesta `POST` utilizzando il metodo `http://localhost:8081/auth`. `blog-validator` riceve quindi tutti i dati necessari per verificare l'autenticità dell'articolo, selezionato dall'utente, mostrato da `my-solid-blog`.
- I dati ricevuti vengono passati alla funzione `readSolidData()` esportata dal modulo `solidRequest.js`.
- Poiché l'applicazione `my-solid-blog` ha passato al server la proprietà `urlDataset`, il quale indica l'URL ove è situato il `SolidDataset` contente l'articolo da controllare, `readSolidData()` può scaricare tale `SolidDataset` utilizzando le funzioni messe a disposizione dalla libreria di `Inrupt solid-client`.
- Una volta scaricato il `SolidDataset` in questione, attraverso la proprietà `urlThing`, anch'essa ricevuta da `my-solid-blog`, la quale rappresenta l'URL corrispondente all'articolo in questione, `readSolidData()` può andare a cercare l'oggetto all'interno del `SolidDataset` scaricato che si riferisce a tale articolo.
- Se l'articolo in questione non esiste `blog-validator` restituisce a `my-solid-blog`:

  ```
  "Article not found:  " + req.body.webId + " is not the author"
  ```

- Se, invece, l'articolo esiste all'interno del `Pod`, `readSolidData()` legge il titolo e il contenuto di tale articolo e li confronta con quelli che ha ricevuto da `my-solid-blog`.

- Se il titolo e/o il contenuto dell'articolo non corrispondono a quelli inviati da `my-solid-blog`, `blog-validator` restituisce a `my-solid-blog` la seguente risposta:

  `"Article of " + req.body.webId + " found, but the title or the content are different"`

- Se il titolo e il contenuto dell'articolo corrispondono a quelli ricevuti da `my-solid-blog`, `readSolidData()` confronta la data di pubblicazione dell'articolo con quella ricevuta da `my-solid-blog`.

- Se queste non corrispondono, `blog-validator` invia a `my-solid-bog` la risposta:

  `"Article of " + req.body.webId + " found, but the date is different, the real date is: " + date`

- Se c'è corrispondenza anche sulla data di pubblicazione dell'articolo allora `blog-validator` invia a `my-solid-blog` come risposta:

  `"Article of " + req.body.webId + " found, all in order"`

  Arrivati a questo punto `blog-validator` ha quindi terminato di controllare l'autenticità dell'articolo in questione.

Il corretto funzionamento della procedura sopra descritta è stata testata tramite il programma `Postman`.

5.15 Gestione del database solid.bd

`blog-validator`, a seguito delle operazioni sopra descritte, utilizza la libreria `sqlite3`, per salvare all'interno del database `solid.db` i dati relativi alle richieste che sono state effettuate da `my-solid-blog`. Tale database è molto semplice e presenta un'unica tabella chiamata `userAuth`; all'interno di questa vengono salvate le informazione necessarie per permettere all'utente di verificare in prima persona che la comunicazione tra `blog-validator` e `my-solid-blog` sia avvenuta correttamente, senza che i dati relativi a tale comunicazione siano stati volontariamente manipolati da quest'ultima applicazione.

Ogni tupla della tabella `userAuth` rappresenta un singolo controllo richiesto dall'applicazione `my-solid-blog` attraverso il metodo `POST` : `http://localhost:8081/auth`. All'interno della tabella `userAuth` sono presenti i seguenti attributi:

- `webId`: la `webId` del proprietario del blog.

- `urlDataset`: URL del `SolidDataset` dove è memorizzato il blog. Tale `SolidDataset` corrisponde con il Dataset `articlelist.ttl` precedentemente trattato.

- `urlThing`: URL corrispondente all'articolo sul quale è stato richiesto il controllo.
- `title`: titolo dell'articolo.
- `content`: contenuto dell'articolo.
- `date`: data di pubblicazione dell'articolo.
- `checkDate`: data nella quale è stato richiesto il controllo dall'utente.
- `result`: esito del controllo.

Tutti i seguenti attributi elencati sono di tipo `TEXT`.

5.16 Il metodo GET, http://localhost:8081/check

Utilizzando l'API method `GET, http://localhost:8081/check` l'utente ha modo di visualizzare i dati contenuti in tale database. Tale link, come già precedentemente indicato, è presente anche all'interno dell'applicazione `my-solid-blog`. Visitandolo un utente può inserire l'URL relativo ad un articolo per poter visualizzare i dati, salvati all'interno del database, relativi al controllo più recente che è stato richiesto su tale articolo. In alternativa, come già indicato, l'utente può digitare l'URL di un articolo e del `SolidDatatset` in cui è salvato, per controllare manualmente il contenuto di tale articolo.

Il motivo per cui `my-solid-blog` mette a disposizione dell'utente il link `http://localhost:8081/check` è quello di provare a quest'ultimo che l'applicazione sta rispettando i criteri relativi alla tecnologia `Solid`: l'applicazione, infatti, non salva localmente nessun dato relativo all'utente, ma utilizza unicamente i dati contenuti all'interno del `Pod` dell'utente proprietario del blog.

5.17 Possibili futuri miglioramenti del sistema SADeB

In questa sezione sono elencate alcune ulteriori funzionalità del sistema `SADeB` che, per alcune motivazioni, non sono state implementate.

- Quando si effettua un controllo su un articolo, `my-solid-blog` potrebbe inviare a `blog-validator` soltanto l'hash relativo al titolo e al contenuto di un articolo. Utilizzando, per esempio, l'algoritmo `SHA-256`, il quale permette di generare `digest` di lunghezza 256 bit per comunicare con il server `blog-validator`, la comunicazione attraverso il protocollo `HTTP` risulterebbe essere più veloce. In questo caso al server basterebbe soltanto confrontare l'hash inviatogli da `my-solid-blog`

con quello del titolo o del contenuto dell'articolo, per stabilire l'autenticità o meno dei contenuti postati all'interno del blog.

- Poiché `Solid` permette ai propri utenti di memorizzare anche file all'interno del proprio `Pod`, l'applicazione `my-solid-blog` potrebbe in futuro permettere agli utenti di caricare un'immagine per il proprio blog, salvandola nel `Pod`, e di consentire di visualizzarla ogni volta che un qualsiasi altro utente visiti il profilo. Attualmente `Inrupt` mette a disposizione alcune funzioni all'interno della libreria `solid-client` per poter leggere e scrivere file contenuti nel proprio `Pod`.

- L'applicazione `blog-validator` potrebbe memorizzare i dati relativi ai controlli richiesti dal client tramite la tecnologia `Solid`, anziché utilizzare un `database` locale. In questo caso sarebbe necessario trovare uno o più vocabolari `RDF` per poter rappresentare tale tipologia di dati. Nel caso in cui non esistessero vocabolari adatti a tale scopo, sarebbe necessario scriverne uno e pubblicarlo all'interno del `container public` presente all'interno del `Pod`. In questo modo il vocabolario risulterebbe accessibile ed utilizzabile anche da altri utenti che hanno la necessità di rappresentare la stessa tipologia di dati descritta all'interno di tale ipotetico vocabolario.

- L'applicazione `my-solid-blog` potrebbe permettere di creare blog appartenenti a gruppi di utenti, anziché ad uno singolo; in questo caso il blog sarebbe formato dai vari articoli appartenenti agli utenti. Attraverso il vocabolario `ACL`, che `Solid` utilizza per determinare i livelli di permesso concessi ad un utente o ad un'applicazione all'interno di un `Pod`, ogni utente potrebbe decidere quali altre persone oltre a lui possano modificare gli articoli presenti nel proprio `SolidDataset` riferito al `Pod`. Attualmente la documentazione relativa alla gestione delle `Access Control List` in `Solid` è limitata e soggetta a frequenti cambiamenti.

Capitolo 6

Conclusioni e sviluppi futuri

Solid è una tecnologia che nasce a seguito della presa visione di un grave problema riguardante il web, ovvero quello relativo alla perdita di privacy da parte degli utenti di internet. La tecnologia Solid mira a risolvere tale problema in maniera molto innovativa, attraverso un nuovo modo di concepire lo stesso web. I vantaggi, da parte dell'utente, nell'utilizzo di applicazioni decentralizzate risultano essere molteplici: in primo luogo attraverso l'utilizzo di Solid, viene superato il problema relativo alla perdita di privacy e al controllo dei dati dell'utente da parte delle grandi multinazionali di internet. Inoltre, rappresentando i dati contenuti all'interno dei Pod attraverso un unico linguaggio, ovvero RDF, è possibile permettere a qualsiasi tipo di applicazione di collaborare nella creazione di un'unica struttura di dati condivisa, in quanto tutti i dati relativi alle applicazioni decentralizzate vengono salvati utilizzando il medesimo formato. La separazione del piano delle applicazioni da quello dei dati, auspicata da Berners-Lee, comporterebbe inoltre per l'utente la possibilità di poter scegliere quali applicazioni web utilizzare, basandosi unicamente sulla qualità del servizio offerto dall'applicazione stessa; infatti, non avendo quest'ultima il controllo dei dati relativi all'utente, non può in alcun modo reclamare i dati che sono già stati salvati all'interno del Pod. Da queste considerazioni si può concludere che è possibile, se non auspicabile, che in futuro Solid, o un'altra tecnologia simile a questa, possa diventare di uso comune, rimpiazzando l'attuale modello utilizzato per l'interazione con le applicazioni web, il quale prevede la consegna dei propri dati personali in cambio dell'utilizzo gratuito del servizio offerto.

L'apprendimento della programmazione di applicazioni decentralizzate in accordo con questa tecnologia, non è stato per me banale: per poter imparare a programmare applicazioni Solid è stato infatti necessario, prima di tutto, imprare a conoscere alcuni linguaggi di programmazione come HTML, CSS e JavaScript, fondamentali per lo sviluppo di applicazioni web, e le tecnologie React e Node.js. In secondo luogo è stato, ovviamente, necessario imparare a padroneggiare il linguaggio RDF, e ad utilizzare le librerie messe a disposizione da Inrupt. Oltre a queste considerazioni, è bene anche precisare che, essendo una tecnologia in via di sviluppo, non è presente ancora

una documentazione particolarmente dettagliata che consenta di imparare agevolmente a programmare applicazioni decentralizzate tramite `Solid`. Inoltre attualmente non esiste ancora una grande comunità di sviluppatori dietro ad essa; pertanto spesso risulta difficile trovare un supporto in caso di difficoltà. È bene anche menzionare il fatto che, essendo una tecnologia in pieno sviluppo, alcuni aspetti chiave relativi ad essa, come il `Solid Protocol` o il `Web Access Control` citati nei capitoli precedenti, sono ancora soggetti a frequenti modifiche, non avendo attualmente raggiunto uno stadio definitivo. Ovviamente questo comporta allo stato attuale alcune difficoltà allo sviluppatore nel programmare applicazioni in accordo con questa tecnologia. A titolo di esempio all'interno dell'applicazione `my-solid-blog` non è stato per me possibile andare a creare e modificare le risorse `ACL` relative al blog per stabilire livelli di permesso intermedi ad altri utenti; questa funzionalità non è stata, infatti, implementata a causa dello scarso materiale messo a disposizione dagli sviluppatori per imparare a gestire questa particolare tipologia di risorse. Attualmente infatti non esiste una guida sufficientemente dettagliata per risolvere tale problema; per esempio, all'interno della documentazione relativa alla libreria `solid-client` di `Inrupt`, è possibile leggere le seguenti frasi nella descrizione di ogni funzione finalizzata alla gestione delle risorse `ACL`: "La specifica Web Access Control non è ancora stata finalizzata. Come tale, questa funzione è ancora sperimentale e soggetta a cambiamenti, anche in una non-major release" [6]. Le problematiche sopra enunciate riguardo alla gestione delle risorse `ACL` hanno fatto sì che l'applicazione `my-solid-blog` abbia potuto usufruire soltanto delle risorse `ACL` di default, contenute all'interno del `Pod`, non potendo andarne a creare delle nuove.

Nonostante le problematiche sopra elencate, `Solid` rimane una tecnologia dal potenziale enorme, in grado di risolvere uno dei problemi più importanti della società di oggi, ovvero quello relativo alla perdita di privacy e alla diffusione di disinformazione online. `Solid` ha acquisito, a partire dalla sua nascita, uno sviluppo e una popolarità sempre maggiori e probabilmente questo andamento continuerà a proseguire nei prossimi anni. In considerazione di quanto esposto all'interno di questo capitolo e di quelli precedenti, mi sento di poter affermare che una tecnologia come `Solid` è di vitale importanza per migliorare una delle più grandi invenzioni del secolo: il `World Wide Web`, e che nell'utilizzare tale tecnologia gli utenti internet possono trarne grandi benefici.

Bibliografia

[1] Sir Tim Berners-Lee. World wide web foundation, three challenges for the web, according to its inventor. https://webfoundation.org/2017/03/web-turns-28-letter/, marzo 2017.

[2] Sir Tim Berners-Lee. Inrupt, one small step for the web. https://inrupt.com/one-small-step-for-the-web, ottobre 2018.

[3] Katrina Brooker. Fast company, exclusive: Tim berners-lee tells us his radical new plan to upend the world wide web. https://www.fastcompany.com/90243936/exclusive-tim-berners-lee-tells-us-his-radical\-new-plan-to-upend-the-world-wide-web, settembre 2018.

[4] Carole Cadwalladr. The guardian, google, democracy and the truth about internet search. https://www.theguardian.com/technology/2016/dec/04/google-democracy-truth-internet-search-facebook, dicembre 2016.

[5] Web Access Control. https://solid.github.io/web-access-control-spec/.

[6] Inrupt. solid-client documentation. https://docs.inrupt.com/developer-tools/api/javascript/solid-client/modules/acl_agent.html.

[7] Thomas Ling. Science focus, interview: Web inventor tim berners-lee thinks his creation is out of control. here's his plan to save it. https://www.sciencefocus.com/future-technology/, giugno 2021.

[8] Sapna Maheshwari. The new york times, how fake news goes viral: A case study. https://www.nytimes.com/2016/11/20/business/media/how-fake-news-spreads.html, novembre 2016.

[9] MIT. What is solid? https://web.archive.org/web/20180629163803/https://solid.mit.edu/.

[10] Sanjay Kumar Monu. Techchahiye, tim berners lee solid project decentralize the whole web soon. https://techchahiye.com/tim-berners-lee-solid-project/, settembre 2018.

[11] Solid Protocol. https://solidproject.org/TR/protocol.

[12] Matteo Starri. We are social, digital 2021: i dati globali. `https://wearesocial.com/it/blog/2021/01/digital-2021-i-dati-globali`, gennaio 2021.

[13] Wikipedia. Scandalo facebook-cambridge analytica. `https://it.wikipedia.org/wiki/Scandalo_Facebook-Cambridge_Analytica`, febbraio 2021.

[14] Wikipedia. Zone 9 bloggers. `https://en.wikipedia.org/wiki/Zone_9_bloggers`, luglio 2021.