

VCardManager Solid

DOCUMENTO DI TRACKING DEL PROGETTO

ANDREA VITTI

Sommario

Configurazione iniziale	3
Installazione di un server Solid in locale e configurazione di un POD	3
Costruzione della prima applicazione Solid sul server locale	3
Analisi e Prime Modifiche del codice originale.....	4
Considerazioni sul codice	4
Specificare altri campi della card.....	5
Riprogettazione della UI	7
Analisi	7
Progettazione	7
Post-Sviluppo	7
Ampliamento delle funzionalità	8
Analisi	8
Progettazione	9
Post-Sviluppo	9
Configurazione repository github.....	12
Caricamento delle informazioni del profilo nei campi di Input.....	13
Analisi	13
Progettazione	13
Post – Sviluppo	13
Ottimizzazione del salvataggio di un avatar nel POD	15
Analisi	15
Progettazione	15
Post-Sviluppo	15
Lettura automatica delle informazioni del profilo all’accesso o all’autenticazione.....	18
Analisi	18
Progettazione	18
Post-sviluppo	18
Aggiunta di feedback visivi nella box di modifica del profilo	19
Analisi	19
Progettazione	19
Post-Sviluppo	19
Mantenimento della sessione e funzionalità di logout	24
Analisi	24

Progettazione	24
Post-Sviluppo	24
Ottimizzazione aggiornamento e lettura proprietà della <i>vCard</i>	26
Analisi	26
Progettazione	27
Post-sviluppo	27
Gestione lista amici	32
Analisi	32
Progettazione	32
Post-Sviluppo	32

Configurazione iniziale

In data: 04/12/2022

Installazione di un server Solid in locale e configurazione di un POD

Passaggi

- Installazione di [Node.js 16.13.0](#) (richiesta al massimo questa versione per motivi di compatibilità con *solid-community-server*)
- Installazione di *npm*
- Installazione di *solid-community-server*
- Creazione dell'account *unittest* su [solidcommunity.net](#)

Link al profilo dell'account: <https://unittest.solidcommunity.net/profile/card#me>

Costruzione della prima applicazione Solid sul server locale

Passaggi

- Creazione prima app seguendo il tutorial [Getting Started - Solid](#)
- Installazione delle [Inrupt Solid JavaScript Client Libraries](#)
- Test dell'app appena costruita e delle sue interazioni col POD indicato nella sezione della configurazione iniziale:
 - E' necessario lanciare l'applicazione sul server, mandando in esecuzione il comando di terminale
`npm parcel index.html`, lanciato dalla directory dell'app
 - Collegandosi a [localhost:1234](#), si visualizza la seguente pagina dove è possibile interagire con il Profilo a cui si effettua il login. E' possibile aggiornare il nome del profilo o visualizzare il nome registrato dando in input il **WebID** del profilo.

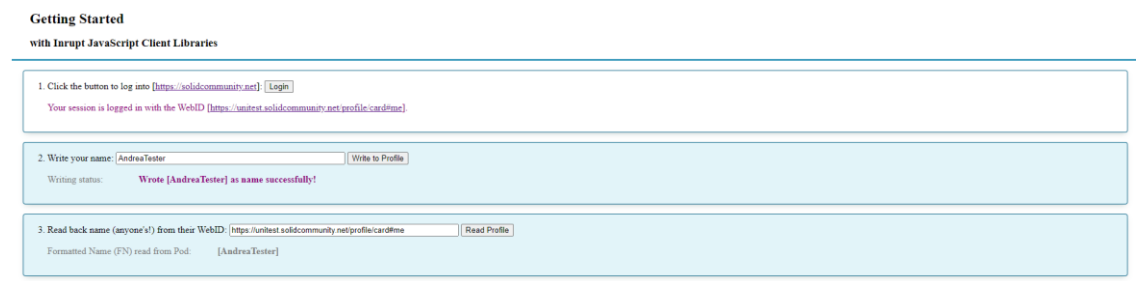


Figura 1 - Applicativo di partenza

- Aprendo l'indirizzo del [Profilo](#), si visualizzano correttamente gli aggiornamenti effettuati dalla applicazione web:

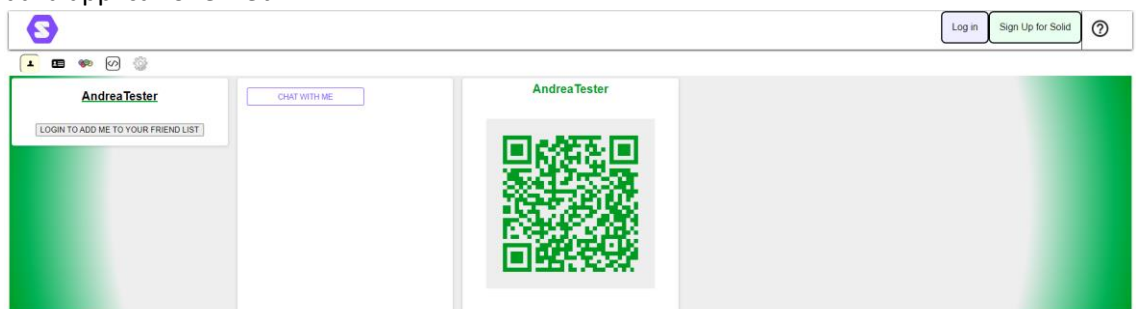


Figura 2- Visualizzatore del profilo di [solidcommunity.net](#)

[Documentazione sulle API di Inrupt](#)

Analisi e Prime Modifiche del codice originale

Data: 04/12/2022

Considerazioni sul codice

Il codice prende in input dei campi da dei form e aggiorna le proprietà della [VCARD](#). Nell'esempio dell'aggiornamento del nome, viene presa in considerazione la proprietà `.fn_`

Da: [VCARD Format Specification](#)

7.2. Identification Properties

TOC

These types are used to capture information associated with the identification and naming of the person or resource associated with the vCard.

7.2.1. FN

TOC

Purpose:

To specify the formatted text corresponding to the name of the object the vCard represents.

Value type:

A single text value.

Special notes:

This property is based on the semantics of the X.520 Common Name attribute. The property **MUST** be present in the vCard object.

Example:

```
FN:Mr. John Q. Public\, Esq.
```

Figura 3 – Ontologia della VCARD, proprietà `fn`

Dal proprio Pod, sarà possibile aprire i dati della propria **VCARD** seguendo i passaggi:

- Cliccare sull'icona



- Cliccare su card

Document



card

La schermata sarà simile alla seguente :

card	type	Personal Profile Document
	maker	Andrea Tester
	primary Topic	Andrea Tester
Andrea Tester	type	Person
		Person
	email	tester@test.com
	fn	Andrea Tester
	Email	id1670184799172
	trusted App	mode
		Append
		Control
		Read
		Write
		localhost:1234
	inbox	origin
	preferences File	inbox
	storage	prefs.ttl
	account	unittest.solidcommunity.net
	oidc Issuer	unittest.solidcommunity.net
	private Type Index	solidcommunity.net
	public Type Index	private Type Index.ttl
	name	public Type Index.ttl
		Andrea

Figura 4 - Visualizzazione vCard di solidcommunity.net

Specificare altri campi della card

Modifiche index.html

Ho inserito un nuovo campo di inserimento per l'email

Getting Started
with Inrupt JavaScript Client Libraries

1. Click the button to log into [\[https://solidcommunity.net\]](https://solidcommunity.net):

Your session is logged in with the WebID [\[https://unittest.solidcommunity.net/profile/card#me\]](https://unittest.solidcommunity.net/profile/card#me).

2. Write your name:

Writing status: **Wrote [AndreaTest] as name successfully!**

2. Write your email:

Writing status: **Wrote [ciao@test.it] as email successfully!**

3. Read back name (anyone's!) from their WebID: [\[https://unittest.solidcommunity.net/profile/card#me\]](https://unittest.solidcommunity.net/profile/card#me)

Formatted Name (FN) read from Pod: ...click the 'Read Profile' button to to see what the birthday email be now...?!

Figura 5 - Aggiunta di un campo per modificare l'email

Modifiche index.js

Ho modificato la funzione `writeProfile()` acquisendo i dati dell'input email e inserendoli nella giusta proprietà della `VCARD`.

- Acquisisco i campi dall'input email:

```
async function writeProfile() {  
  const name = document.getElementById("input_name").value;  
  const email = document.getElementById("input_email").value; //modificato Andrea
```

- Aggiorno la proprietà `.email` della `VCARD`:

```
// Using the name provided in text field, update the name in your profile.  
// VCARD.fn object is a convenience object that includes the identifier string "http://www.w3.org/2006/vcard/ns#fn".  
// As an alternative, you can pass in the "http://www.w3.org/2006/vcard/ns#fn" string instead of VCARD.fn.  
profile = setStringNoLocale(profile, VCARD.fn, name);  
profile = setStringNoLocale(profile, VCARD.email, email); //modificato da Andrea
```

Definizione proprietà email:

7.4.2. EMAIL

TOC

- Purpose:
To specify the electronic mail address for communication with the object the vCard represents.
- Value type:
A single text value.
- Special notes:
The type can include the type parameter "TYPE" to specify the format or preference of the electronic mail address. The TYPE parameter values can include: "internet" to indicate an Internet addressing type, "x400" to indicate a X.400 addressing type, "uri" to indicate a URI useable for electronic communication, "home" to indicate an address associated with a residence, "work" to indicate an address associated with a place of work, or "pref" to indicate a preferred-use email address when more than one is specified. Another IANA registered address type can also be specified. The default email type is "internet". A non-standard value can also be specified.

Type example:

```
EMAIL;TYPE=internet:jpublic@xyz.example.com  
  
EMAIL;TYPE=internet,pref:jane_doe@example.com  
  
EMAIL;TYPE=uri,work:http://example.com/contact.php
```

Figura 6 - Ontologia della vCard, proprietà email

Risultato nella VCARD

card	type	Personal Profile Document
	maker	AndreaTest
	primary Topic	AndreaTest
AndreaTest	type	Person
		Person
	email	ciao@test.it
	fn	AndreaTest

Figura 7 - Proprietà email aggiornata

Riprogettazione della UI

Inizio: 12/12/2022

Fine: 15/12/2022

Analisi

Le applicazioni Solid fino a ora intraviste sfruttano una interfaccia utente un po' datata e meno responsive. Ho pensato di riprogettare gli elementi dell'interfaccia in modo da renderla più accattivante, facilitare la fruizione delle informazioni della vCard e la loro modifica.

Interfaccia per la modifica:

Getting Started
with Inrupt JavaScript Client Libraries

1. Click the button to log into [https://solidcommunity.net]: [Login](#)
Your session is logged in with the WebID [https://unittest.solidcommunity.net/profile/card#me].

2. Write your name: [Write to Profile](#)
Writing status: **Wrote [AndreaTest] as name successfully!**

2. Write your email: [Write to Profile](#)
Writing status: **Wrote [ciao@test.it] as email successfully!**

3. Read back name (anyone's!) from their WebID: [Read Profile](#)
Formatted Name (FN) read from Pod: ...click the 'Read Profile' button to see what the birthday email be now...?!

Figura 8 - Interfaccia di partenza

Progettazione

Intervenendo sul file *index.html* e sul relativo foglio di stile, creerò un'interfaccia grafica dal layout scalabile in funzione del display su cui verrà caricata. Inserirò anche dei tasti per il *login* e la presenza di un elemento grafico, richiamabile da un pulsante, per visualizzare le informazioni della vCard formattate.

Post-Sviluppo

L'interfaccia presenta un design più user friendly e responsive

Interfaccia Riprogettata, visualizzazione da PC:



vCard Manager
read vCards from Existing Pods and update your vCard's info

[Login](#) [vCard](#)

Quick Start
Click the button <#> to log into
[https://solidcommunity.net](#)
Update your vCard info

Update Infos
Write your name:

Write your email:

Write your birthday:

Select your gender:

Select your country:

Select your country:

[Write to Profile](#)

Writing status:
...not written yet...

Read Profile Data
Read back profile (anyone's!) data from their WebID:
[Read Profile](#)

Data read from Pod:
Provided WebID [] is not a valid URL - please try again

© Best University "Abdo Mena"

Figura 9 - Nuova interfaccia, vista Desktop

Visualizzazione mobile:

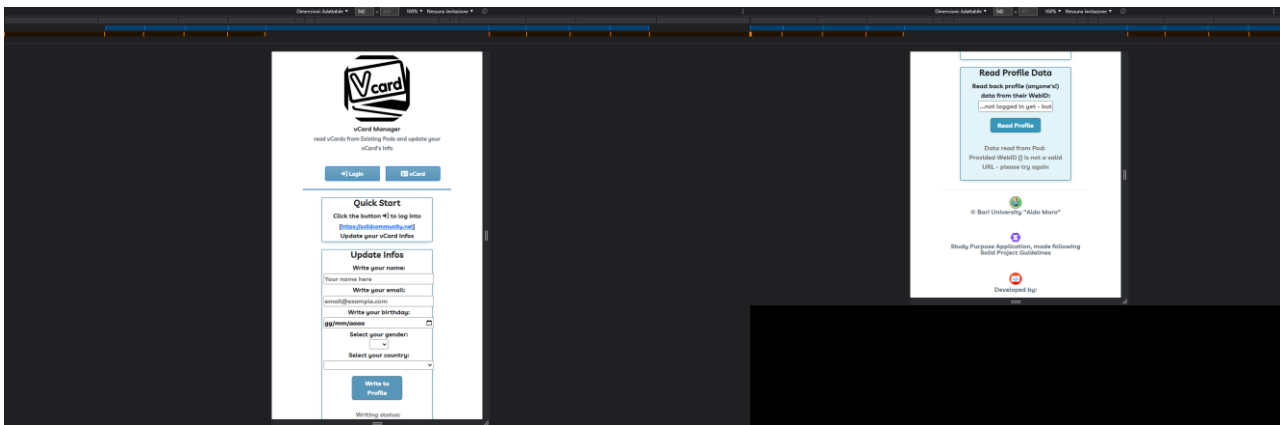


Figura 10 - Nuova interfaccia, vista Mobile

Ampliamento delle funzionalità

Inizio: 16/12/2022

Fine: 19/12/2022

Analisi

All'interno della applicazione si possono leggere i dati di una *vCard* esterna o modificare quelli della propria. Dal POD è possibile inoltre visualizzare le informazioni della *vCard* in una formattazione standard. L'obiettivo è rendere entrambe le funzionalità fruibili dalla stessa interfaccia, senza bisogno di spostarsi su un altro indirizzo web per passare da lettura a modifica. Prevedo inoltre un ampliamento del pacchetto di informazioni specificabili per personalizzare le proprietà della *vCard* e la possibilità di caricare istantaneamente le informazioni di una *vCard* non solo all'inserimento del link da cui leggere i dati ma anche, in automatico, al login dell'utente o all'aggiornamento dei suoi dati.

Interfaccia base del POD ([POD Unitest](#)) per la visualizzazione della *vCard*:




Figura 11 - Visualizzatore vCard di solidcommunity.net



Log in

Sign Up for Solid

?



Andrea

Gender

bday

country name

email

fn

trusted App

Male

1995-12-18

Italy

andrea@unitest.com

Andrea

mode

Append

Read

Write

localhost:1234

origin

preferences File

preferred Object Pronoun

preferred Relative Pronoun

preferred Subject Pronoun

public Type Index

Preferences.ttl

him

his

he

public Type Index.ttl

Progettazione

Ho pensato di integrare un bottone per aprire una formattazione compatta della *vCard* letta attraverso i dati caricati dalla funzione *readProfile()* nel file *index.js*.

La funzione caricherà una anteprima dei dati contenuti nella *vCard* all'interno di un messaggio e inserirà tali dati in un elemento grafico richiamabile da un bottone.

Per migliorare l'affidabilità delle informazioni, rendendole sempre disponibili in tempo reale, ho integrato una chiamata della funzione *readProfile()* dopo l'esecuzione delle funzioni *login()* e *writeProfile()*.

Post-Sviluppo

E' stato inserito un elemento nella vista principale che permetterà di richiamare la vista della *vCard*. Tale vista offrirà un messaggio standard nel caso in cui nessun profilo sia ancora stato caricato. Fornirà, invece, una visualizzazione formattata dei dati della *vCard* all'interno di un contenitore grafico.



Figura 12 – Elemento grafico, bottone vCard

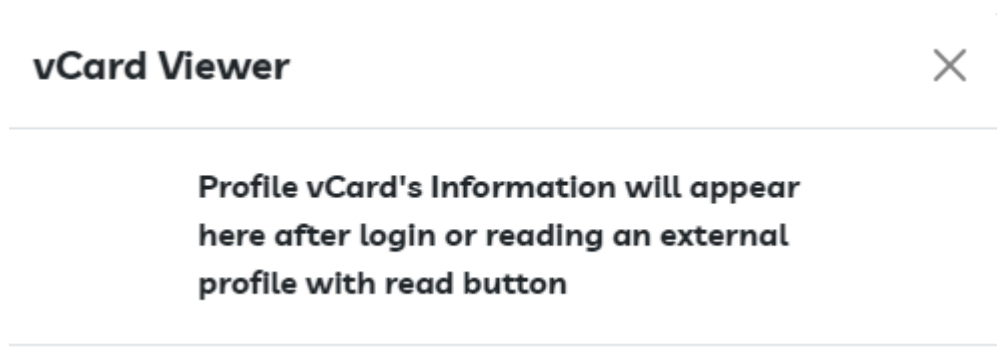


Figura 13 - Contenitore dati vCard, profilo non caricato



Figura 14 - Contenitore dati vCard, profilo caricato

Modifiche a `index.js`:

Ampliamento funzione `writeProfile()`:

- Inserimento di nuovi campi aggiornabili nella vCard

```
const birthday = document.getElementById("input_birth").value;
const gender = document.getElementById("input_gender").value;
const country = document.getElementById("input_country").value;
```

Figura 15 – Aggiunta dei campi birthday, gender, country

-

```
profile = setStringNoLocale(profile, VCARD.bday, birthday);
profile = setStringNoLocale(profile, VCARD.Gender, gender);
profile = setStringNoLocale(profile, VCARD.country_name, country);
```

Figura 16 - Aggiornamento dell'oggetto profile con le nuove proprietà della vCard aggiunte

- Inserimento di un array *storage* che salvi tutti i campi acquisiti o inseriti per l'aggiornamento delle informazioni. Richiamandone il contenuto, il sistema sarà in grado di fornire un feedback su quali dati siano stati scritti o letti in relazione alla *vCard*

```
storage.push(name, email, birthday, gender, country);
for (let i = 0; i < storage.length; i++) {
  if (storage[i] === "") {
    storage.splice(i, 1);
  }
}
```

Figura 17 - Aggiornamento dell'array per il tracciamento delle informazioni scritte o lette

```
// Update the page with the retrieved values.
const labelWriteStatus = document.getElementById("labelWriteStatus");
labelWriteStatus.textContent = `Wrote [${storage}] in your vcard successfully`;
labelWriteStatus.setAttribute("role", "alert");
labelWriteStatus.classList.add("longurl");
```

Figura 18 - Mostra a video un feedback alla scrittura delle informazioni

- Chiamata della funzione *readProfile()* per caricare le informazioni aggiornate dopo l'esecuzione della funzione di aggiornamento dati (la stessa cosa è stata inserita alla fine della funzione *login()*)

```
// Update the page with the retrieved values.
const labelWriteStatus = document.getElementById("labelWriteStatus");
labelWriteStatus.textContent = `Wrote [${storage}] in your vcard successfully`;
labelWriteStatus.setAttribute("role", "alert");
labelWriteStatus.classList.add("longurl");
readProfile();
```

Figura 19 - Chiamata della funzione *readProfile()* per caricare gli aggiornamenti del profilo nell'applicazione

- Inserimento di un campo per specificare una foto nella *vCard* :
 - Acquisizione input

```
const file = document.querySelector("input[type=file]")["files"][0];
let photo;
```

Figura 20 - Inserimento di un input file

- Inserimento di una funzione per convertire il file in *base64* :

```
if (file) {
  // Encode the file using the FileReader API
  const reader = new FileReader();
  reader.onloadend = () => {
    photo = reader.result; //store the base64 code for the picture
  };
  reader.readAsDataURL(file);
  storage.push("New Avatar");
}

for (let i = 0; i < storage.length; i++) {
  if (storage[i] === "") {
    storage.splice(i, 1);
  }
}
```

Figura 21- Conversione del file in b64 e aggiornamento dello storage

- Inserimento nella proprietà della *vCard* :

```
profile = setStringNoLocale(profile, VCARD.photo, photo);
```

Figura 22 - Aggiornamento oggetto con aggiunta della proprietà *vCard.photo*

Configurazione repository github

Data: 20/12/2022

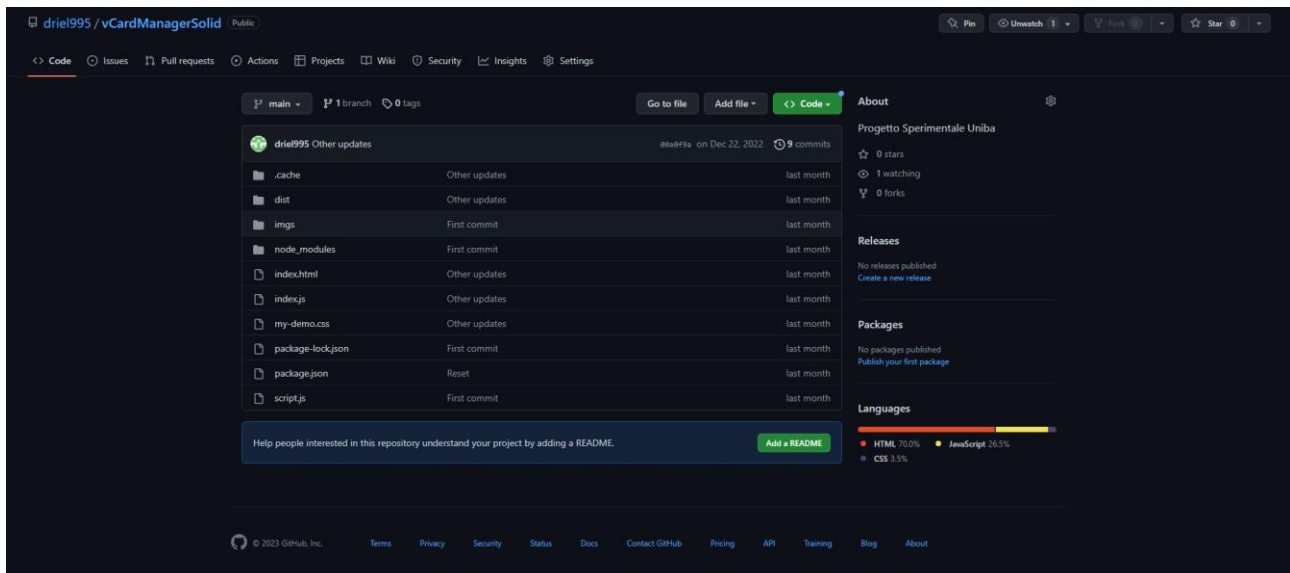


Figura 23 - *vCardManagerSolid* repository

[vCardManagerSolid on gitHub](#)

Caricamento delle informazioni del profilo nei campi di Input

Data: 27/12/2022

Analisi

Il requisito nasce dalla necessità di poter visualizzare, nella sezione di modifica del profilo, le informazioni già contenute all'interno del proprio profilo. L'utente dovrà poter visualizzare, all'interno dei relativi campi, dei valori, qualora essi siano già associati alle relative proprietà della vCard nel POD.

Progettazione

Per rendere possibile la visione dei dati della vCard nei relativi campi di input, sarà sufficiente aggiornare il loro valore interno al caricamento della pagina del proprio profilo.

Per far ciò, si interverrà sulla funzione di lettura presente già all'interno dell'applicativo.

Post – Sviluppo

Modifiche a index.js

```
//Get inputFields elements
const inputName = document.querySelector("#input_name");
const inputEmail = document.querySelector("#input_email");
const inputBirth = document.querySelector("#input_birth");
const inputGender = document.querySelector("#input_gender");
const inputCountry = document.querySelector("#input_country");
const inputPhoto = document.querySelector("#input_img");
```

Figura 24 - Acquisizione degli elementi HTML dei campi di input

```
//Updates the input fields with the retrieved values
inputName.value = formattedName;
inputEmail.value = formattedEmail;
inputBirth.value = formattedBirth;
inputGender.value = formattedGender;
inputCountry.value = formattedCountry;
```

Figura 25 - Aggiornamento dei valori degli input

Le variabili *formatted* contengono i valori delle proprietà della vCard formattati in stringa, per poterli stampare a video nella sezione di visualizzazione della vCard. Nell'esempio della figura 26:

```
//Get vCard Info
const formattedName = getStringNoLocale(profile, VCARD.fn);
```

Figura 26- Variabile *formatted* per la proprietà vCard.fn

Interfaccia

Si denota il caricamento delle informazioni della vCard all'interno delle relative box di input quando si caricano i dati del proprio profilo.

Update Infos

Write your name:

Write your email:

Write your birthday:

Select your gender:

Select your country:

Upload your profile pic:

Write to Profile

Figura 27 - Valori dei campi di input importati dalla vCard

Ottimizzazione del salvataggio di un avatar nel POD

Inizio: 28/12/2022

Fine: 29/12/2022

Analisi

L'obiettivo di questa task è, in accordo col funzionamento del POD, di abilitare il salvataggio del file in una directory del proprio POD e specificare tale file come avatar del proprio profilo.

Progettazione

La precedente implementazione dell'aggiornamento immagine del profilo (immagine salvata come una stringa in base64), verrà sostituita da una che sfrutti il salvataggio del file in un container del POD.

Si è notato come, cambiando l'immagine del profilo dal POD Browser di Solid, un file venga salvato nella cartella *profile* ed un *URL* a questo file venga indicato come valore della proprietà *hasPhoto* della *vCard*.

Rimuoverò il vecchio metodo per svilupparne uno che sfrutti questa proprietà, aspettandomi inoltre di garantire lo stesso funzionamento del POD Browser all'interno della mia applicazione.

Per far ciò, sfrutterò i metodi delle librerie di Inrupt per la gestione dei POD.

Post-Sviluppo

Metodi importati

Dalla libreria di Inrupt, ho importato i seguenti metodi per gestire le proprietà della *vCard* che accolgono come valori dei collegamenti:

```
getUrl,  
setUrl,  
getUrlAll,  
addUrl,  
removeUrl,  
} from "@inrupt/solid-client";
```

Figura 28 - Importazione metodi di gestione URL

Modifiche a index.js

Nella funzione `writeProfile()`:

```
//Avatar update  
//Instead of "c" flags, avatar modifications can be spotted by checking if a file has been uploaded  
if (file) {  
  document.querySelector("#writeimg").innerHTML =  
    'Upload your profile pic: <i class="fa-solid fa-check"></i>';  
  
  let avatarUrl = podUrl + "profile/avatar.png"; //Define a URL for the new avatar  
  placeFileInContainer(file, avatarUrl);  
  profile = setUrl(profile, VCARD.hasPhoto, avatarUrl); //specify the avatar URL in the relative property  
}  
//End avatar update
```

Figura 29 - Aggiornamento dell'oggetto *profile* con la proprietà *vCard.hasPhoto*


```

//Upload file into the targetContainer.
async function placeFileInContainer(file, targetContainerURL) {
  try {
    const savedFile = await overwriteFile(
      targetContainerURL, // Container URL
      file, // File
      {
        slug: file.name,
        contentType: file.type,
        fetch: session.fetch, //this consent the operation to be made using the current user logged in session, otherwise it produces an error of illegal invocation or 401
      }
    );
    console.log(`File saved at ${getSourceUrl(savedFile)}`);
  } catch (error) {
    console.error(error);
  }
}

```

Figura 30- Specifica funzione `placeFileInContainer()`

```

/**
 * Create a new Thing with existing values replaced by the given URL for the given Property.
 *
 * To preserve existing values, see [[addUrl]].
 *
 * The original `thing` is not modified; this function returns a cloned Thing with updated values.
 *
 * @param thing Thing to set a URL value on.
 * @param property Property for which to set the given URL value.
 * @param url URL to set on `thing` for the given `property`.
 * @returns A new Thing equal to the input Thing with existing values replaced by the given value for the given Property.
 */
export declare const setUrl: SetOfType<Url | UrlString | Thing>;

```

Figura 31 - Prototipo metodo `setUrl()` dalla libreria di `Inrupt`

Risultato

Si può notare come , dopo aver eseguito l'upload di una immagine del profilo dalla sezione di modifica del proprio profilo, il nuovo avatar comparirà all'interno del POD , nella directory *profile* .

La proprietà della *vCard* relativa alla descrizione dell'immagine del profilo, conterrà un URL all'immagine non appena caricata.

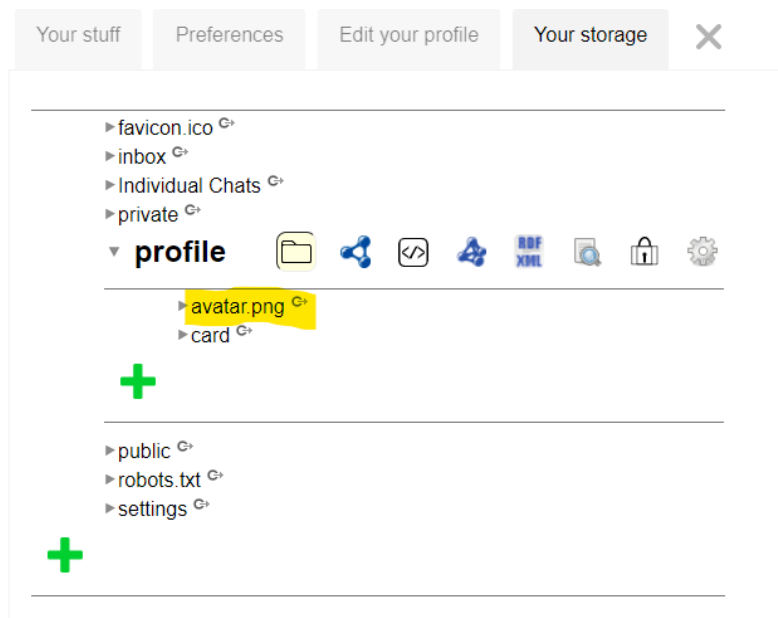


Figura 32 - Percorso avatar POD

card	type	Personal Profile Document
	maker	Andrea
	primary Topic	Andrea
Andrea	type	Person
		Person
	Gender	Male
	bday	2023-01-28
	fn	Andrea
	Address	16742606421434548364699473164
	Email	16747477578797465651743066135
	Photo	avatar.png

Figura 33 - URL avatar nella proprietà vCard.hasPhoto

Lettura automatica delle informazioni del profilo all'accesso o all'autenticazione

Data: 30/12/2022

Analisi

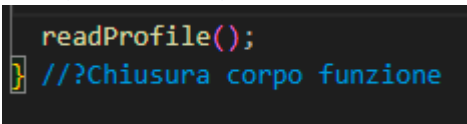
Si vuole ottimizzare l'usabilità dell'applicativo, automatizzando il caricamento delle informazioni dell'utente all'autenticazione e aggiornando a video le informazioni qualora venissero modificate.

Progettazione

La funzione di lettura va richiamata a seguito delle istruzioni per eseguire il login o per aggiornare le informazioni del profilo utente. Per far ciò, si inserisce un richiamo della funzione `readProfile()` alla fine delle funzioni `login()` e `writeProfile()`.

Post-sviluppo

Modifiche a `index.js`:



```
readProfile();  
} //?Chiusura corpo funzione
```

Figura 34 - Sintassi per richiamare la funzione `readProfile()` alla fine delle funzioni in oggetto

Risultato

La nuova logica rende l'esperienza d'uso dell'applicativo più fluida e gradevole, eliminando la macchinosità del dover copiare e incollare il proprio `webID` per leggere le informazioni del profilo appena autenticato o aggiornato.

Aggiunta di feedback visivi nella box di modifica del profilo

Data: 02/01/2023

Analisi

Update Infos

Write your name:

Write your email:

Write your birthday:

Select your gender:

Select your country:

Upload your profile pic:
 Nessun ...ezionato

Writing status:
...not written yet...

Figura 35 - Interfaccia modifica profilo

Durante l'utilizzo del prototipo, sono emersi due requisiti di usabilità :

1. l'utente deve poter vedere quali campi ha modificato, prima di confermare l'aggiornamento delle informazioni del proprio profilo;
2. il sistema deve poter notificare all'utente l'avvenuto salvataggio dei campi modificati.

Progettazione

Ho pensato di far comparire dei simboli vicino ai campi da aggiornare, per metterli in evidenza rispetto a quelli non modificati. A salvataggio avvenuto sul proprio POD, comparirà un ulteriore simbolo di conferma, a sostituzione del precedente, per indicare la corretta scrittura di quei campi nel proprio profilo.

Post-Sviluppo

Modifica dei campi

Per ogni campo di input presente nella box di modifica del profilo, viene decretato un *event listener* che controlli quando quel campo viene modificato. Nell'etichetta di quel campo, verrà visualizzato un simbolo che ne indichi delle modifiche a quel campo da dover ancora applicare.

Vengono anche definiti dei flag che l'*event listener* provvederà ad aggiornare quando il valore in un campo viene cambiati.

Modifiche a index.js

```
//Get inputFields elements
const inputName = document.querySelector("#input_name");
const inputEmail = document.querySelector("#input_email");
const inputBirth = document.querySelector("#input_birth");
const inputGender = document.querySelector("#input_gender");
const inputCountry = document.querySelector("#input_country");
const inputPhoto = document.querySelector("#input_img");
```

Figura 36 - Acquisizione campi di input

```
//flags for any vcard data to apply changes on - not required for the avatar
let cname, cemail, cbirth, cgender, ccountry;
```

Figura 37 - Definizione flag sulla modifica dei campi

```
inputName.addEventListener("change", (event) => {
  document.querySelector("#writename").innerHTML =
    'Write your name: <i class="fa-solid fa-spinner"></i>';
  cname = true;
});

inputEmail.addEventListener("change", (event) => {
  document.querySelector("#writeemail").innerHTML =
    'Write your email: <i class="fa-solid fa-spinner"></i>';
  cemail = true;
});

inputBirth.addEventListener("change", (event) => {
  document.querySelector("#writebirth").innerHTML =
    'Write your birthday: <i class="fa-solid fa-spinner"></i>';
  cbirth = true;
});

inputGender.addEventListener("change", (event) => {
  document.querySelector("#writegender").innerHTML =
    'Select your gender: <i class="fa-solid fa-spinner"></i>';
  cgender = true;
});

inputCountry.addEventListener("change", (event) => {
  document.querySelector("#writecountry").innerHTML =
    'Select your country: <i class="fa-solid fa-spinner"></i>';
  ccountry = true;
});

inputPhoto.addEventListener("change", (event) => {
  document.querySelector("#writeimg").innerHTML =
    'Upload your profile pic: <i class="fa-solid fa-spinner"></i>';
});
```

Figura 38 - Event listeners alla modifica dei campi

Interfaccia

Update Infos

Write your name: 📄

Write your email: 📧

Write your birthday:

Select your gender:

Select your country:

Upload your profile pic: 📷

512x512.png

Write to Profile

Writing status:
...not written yet...

Figura 39 - Interfaccia con modifiche in sospeso

Salvataggio delle modifiche

Al corretto salvataggio dei valori da aggiornare nel profilo, un simbolo di conferma sostituirà quello nelle etichette dei campi in sospeso.

Alla fine della funzione di scrittura del profilo, viene inserito un controllo che aggiunga un simbolo di conferma del salvataggio nell'etichetta di un campo se e solo se esso sia stato modificato dall'utente prima di salvarlo nel POD. Ciò avviene tramite il controllo dei flag definiti in precedenza nel codice.

Modifiche a index.js

```
//Show on which data modifies were applied
if (cname) {
  document.querySelector("#writename").innerHTML =
    'Write your name: <i class="fa-solid fa-check"></i>';
  cname = false;
}
if (cemail) {
  document.querySelector("#writeemail").innerHTML =
    'Write your email: <i class="fa-solid fa-check"></i>';
  cemail = false;
}
if (cbirth) {
  document.querySelector("#writebirth").innerHTML =
    'Write your birthday: <i class="fa-solid fa-check"></i>';
  cbirth = false;
}
if (cgender) {
  document.querySelector("#writegender").innerHTML =
    'Select your gender: <i class="fa-solid fa-check"></i>';
  cgender = false;
}
if (ccountry) {
  document.querySelector("#writecountry").innerHTML =
    'Select your country: <i class="fa-solid fa-check"></i>';
  ccountry = false;
}
```

Figura 40 - Conferma dei campi modificati

```
//Instead of "c" flags, avatar modifications can be spotted by checking if a file has been uploaded
if (file) {
  document.querySelector("#writeimg").innerHTML =
    'Upload your profile pic: <i class="fa-solid fa-check"></i>';
}
```

Figura 41 - Conferma della modifica avatar

Interfaccia

Update Infos

Write your name: ✓

Write your email: ✓

Write your birthday:

Select your gender:

Select your country:

Upload your profile pic: ✓

Scegli file 512x512.png

Write to Profile

Writing status:

Wrote [Andrea,andrea@unitest2.com,1995-12-18,Male,Italy,New Avatar] in your vcard successfully!

Figura 42- Interfaccia con modifiche salvate nel POD

Mantenimento della sessione e funzionalità di logout

Data: 03/01/2023

Analisi

Si vuole migliorare l'esperienza di utilizzo dell'applicativo cercando di rendere più immediato il caricamento del profilo utente nel caso in cui sia stato effettuata già l'autenticazione e la pagina venga ricaricata.

Si vuole, inoltre, aggiungere la possibilità di terminare la sessione corrente e non visualizzare più l'interfaccia dell'applicativo nella modalità di utente autenticato.

Progettazione

Si utilizzeranno dei metodi per conservare la sessione al ricaricarsi della pagina o alla sua chiusura, in modo da rendere disponibile all'utente autenticato la sua vista.

Sarà aggiunto un elemento cliccabile che eseguirà un metodo per terminare la sessione utente. Al termine, l'utente dovrà eseguire nuovamente il *login* per accedere alla vista autenticata.

Post-Sviluppo

Conservazione della sessione

Modifiche a index.js:

```
import { Session } from "@inrupt/solid-client-authn-browser";
```

Figura 43- Importazione metodi per la gestione della sessione dalla libreria di Inrupt

```
// 1b. Login Redirect. Call session.handleIncomingRedirect() function.  
// When redirected after login, finish the process by retrieving session information.  
async function handleRedirectAfterLogin() {  
    await session.handleIncomingRedirect({  
        restorePreviousSession: true /*this option allows to store already authenticated user.  
        |                         | Default would be window.location.href*/,  
    });  
}
```

Figura 44- Funzione di redirect dopo l'autenticazione, aggiunta dell'opzione per riprendere la sessione precedente

Risultato

La sessione viene correttamente conservata, aggiungendo all'applicativo un comportamento più funzionale per l'esperienza d'uso dell'utente autenticato.

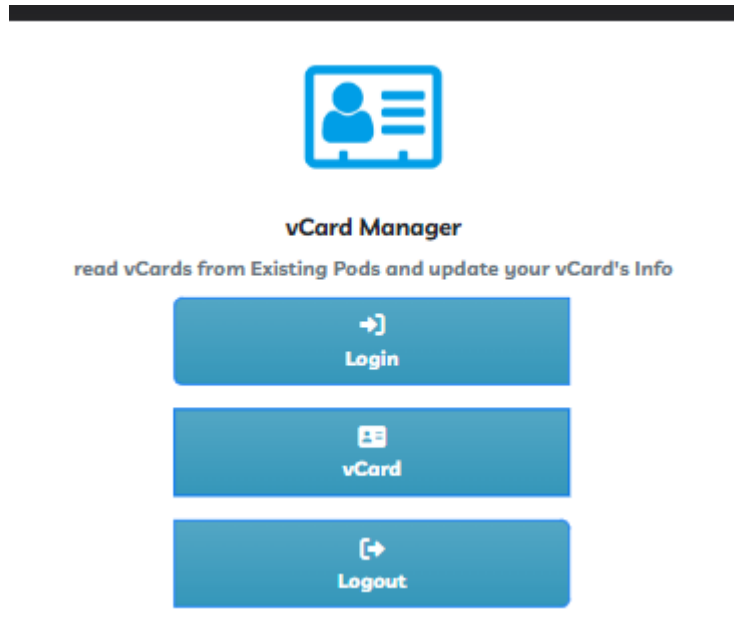


Figura 45 - Aggiunta elemento grafico per richiamare la funzione di logout

Modifiche a index.js

```
//Logout
async function exit() {
  if (session.info.isLoggedIn) {
    await session.logout();
  }
}
```

Figura 46 - Funzione di logout

```
buttonLogout.onclick = function () {
  exit();
  window.location.reload(true);
};
```

Figura 47 - Event listener per richiamare la funzione al click dell'elemento grafico

Ottimizzazione aggiornamento e lettura proprietà della vCard

Inizio: 09/01/2023

Fine: 13/01/2023

Analisi

Si vogliono rendere le informazioni specificate all'interno della vCard in un formato che sia conforme allo standard, ossia descrivendo correttamente qualora una informazione sia descritta da una semplice stringa, un URL o un dato strutturato.

Studiando le [proprietà generali](#) nell'Ontologia della vCard, sono stati individuati alcune informazioni del profilo trattate dall'applicativo che andrebbero specificate in un formato più strutturato.

Data di nascita

La data di nascita è una proprietà che ammette una data formattata con la sintassi *aaaa-mm-gg*

BDAY	Birth date of the object. Should only apply to Individual.	bday
------	--	------

Figura 48- Ontologia vCard, proprietà bday

bday 2023-01-28

Figura 49 – Visualizzatore vCard del POD, proprietà vCard.bday.

Indirizzo

L'indirizzo è una proprietà identificata da un collegamento a un oggetto Indirizzo. Tale oggetto avrà diverse proprietà per dettagliare le informazioni di un indirizzo fisico dell'utente.

Nel caso in oggetto, si lavorerà con la proprietà *country name* per la specifica della nazione associata al relativo indirizzo.

RFC Property	Note	Ontology Property	N-Ary Property
ADR	The address of the object represented in structured parts	hasAddress (range of class Address) street-address locality region country-name postal-code	hasStreetAddress hasLocality hasRegion hasCountryName hasPostalCode

Figura 50 - Ontologia vCard, proprietà hasAddress

Address 16742606421434548364699473164

Figura 51 - Visualizzatore vCard del POD, proprietà vCard.hasAddress

16742606421434548364699473164 country name Italy

Figura 52- Visualizzatore vCard del POD, dettaglio oggetto Address

Email

Il caso della proprietà per l'email è analogo a quello per l'indirizzo.

Nel caso di studio, al momento, verrà presa in considerazione la proprietà *value* per specificare un collegamento *mailto*, utilizzato per gestire l'invio di email all'indirizzo specificato.

EMAIL	The email address as a mailto URI	hasEmail
-------	-----------------------------------	----------

Figura 53 - Ontologia vCard, proprietà hasEmail

Email 16747477578797465651743066135

Figura 54 - Visualizzatore vCard del POD, proprietà vCard.hasEmail

16747477578797465651743066135 value mailto:esempio@unittest2.com

Figura 55- Visualizzatore vCard del POD, dettaglio oggetto Email

Progettazione

La libreria di inrupt offre dei [metodi per gestione dei dati strutturati](#) e delle loro proprietà. Una entità nel POD è definita dalle librerie come *Thing*. Per ognuna di esse (profilo, indirizzo, email, ecc..), possono essere specificate diverse proprietà in diversi formati di dati.

Post-sviluppo

Modifiche a `index.js`

```
getThing,  
createThing,  
setThing,  
getDate,  
getUri,  
setUri,  
getUriAll,  
addUri,  
removeUri,  
from "@inrupt/solid-client";
```

Figura 56 - Importazione metodi per gestione dati strutturati

```

//Birthday update
const birthdate = new Date(birthday)
    .toISOString()
    .split("T", 1)[0]; /*Convert the date in a string in format yyyy-mm-dd*/

profile = setStringNoLocale(profile, VCARD.bday, birthdate);

//? Not used since, in function setDate(), there's a bug that appends a Z char to the date string
/*const birthdate = new Date(birthday);
profile = setDate(profile, VCARD.bday, birthdate);*/

//End Birthday update

```

Figura 57 - Aggiornamento della data di nascita, funzione writeProfile()

```

//Get birthday
//The application saves the date as a string in the vCard, while the POD uses a date obj.
let birth = getStringNoLocale(profile, VCARD.bday); //1. Get the information as a string
let formattedBirth;
//2. If it's not a valid string, retrieve the information as a date obj and format it
if (birth == null) {
    birth = getDate(profile, VCARD.bday);
    formattedBirth = Intl.DateTimeFormat("fr-CA", {
        year: "numeric",
        month: "2-digit",
        day: "2-digit",
    }).format(birth);
} //If it's a valid string, use it
else {
    formattedBirth = birth;
}

//End Get birthday

```

Figura 58- Recupero della proprietà bday, funzione readProfile()

```

//Address.country update
let addressThing;
let addressUrl = getUrl(profile, VCARD.hasAddress);

// If there's already a saved address
if (addressUrl != null) {
    // Take the dataset for the hasAddressproperty

    let addressDataset = await getSolidDataset(addressUrl, {
        fetch: session.fetch,
    });

    addressThing = getThing(addressDataset, addressUrl);
    addressThing = setStringNoLocale(addressThing, VCARD.country_name, country);
    addressDataset = setThing(addressDataset, addressThing); //Update the dataset with the new address

    await saveSolidDatasetAt(addressUrl, addressDataset, {
        fetch: session.fetch,
    });
} else {
    addressThing = createThing();
    addressThing = setStringNoLocale(addressThing, VCARD.country_name, country);
    myProfileDataset = setThing(myProfileDataset, addressThing);
    addressUrl = addressThing.url.slice(46);
    addressUrl = profileDocumentUrl.href + "#" + addressUrl;
    profile = setUrl(profile, VCARD.hasAddress, addressUrl);
}
// End Address update

```

Figura 59 - Aggiornamento della nazione dell'indirizzo, funzione writeProfile()

```

//Get address
let formattedCountry = "";
let addressThing;
const addressUrl = getUrl(profile, VCARD.hasAddress);
if (addressUrl != null) {
    addressThing = getThing(userDataset, addressUrl);
}
if (addressThing != null) {
    formattedCountry =
        addressThing.predicates["http://www.w3.org/2006/vcard/ns#country-name"]
        .literals["http://www.w3.org/2001/XMLSchema#string"][0];
}
//End Get address

```

Figura 60 - Recupero della proprietà countryname, funzione readProfile()

```

// Email update
let mailThing;
let mailUrl = getUrl(profile, VCARD.hasEmail);
const mailLink = "mailto:" + email;

// If there's already a saved email
if (mailUrl != null) {
  // Take the dataset for the hasEmail property

  let mailDataset = await getSolidDataset(mailUrl, {
    fetch: session.fetch,
  });

  mailThing = getThing(mailDataset, mailUrl);
  mailThing = setUrl(mailThing, VCARD.value, mailLink);
  mailDataset = setThing(mailDataset, mailThing); //Update the dataset with the new email

  await saveSolidDatasetAt(mailUrl, mailDataset, {
    fetch: session.fetch,
  });
} else {
  mailThing = createThing();
  mailThing = setUrl(mailThing, VCARD.value, mailLink);
  myProfileDataset = setThing(myProfileDataset, mailThing);
  mailUrl = mailThing.url.slice(46);
  mailUrl = profileDocumentUrl.href + "#" + mailUrl;
  profile = setUrl(profile, VCARD.hasEmail, mailUrl);
}
//End Email update

```

Figura 61 - Aggiornamento dell'email, funzione writeProfile()

```

// Get email
let formattedEmail = "";
let mailLink = "";
const mailUrl = getUrl(profile, VCARD.hasEmail);
if (mailUrl != null) {
  const mailThing = getThing(userDataset, mailUrl);
  if (mailThing != null) {
    mailLink =
      mailThing.predicates["http://www.w3.org/2006/vcard/ns#value"]
        .namedNodes[0];
  }
  formattedEmail = mailLink.replace("mailto:", "");
}
//End Get email

```

Figura 62 - Recupero della proprietà hasEmail, funzione readProfile()

Risultato

I formati dei dati rispecchiano lo standard per la loro specifica, pertanto i tipi di dato usati vengono riconosciuti dal lettore profilo del POD.

Street address	<input type="text"/>
Locality	<input type="text"/>
Postal code	<input type="text"/>
Region	<input type="text"/>
Country name	<input type="text" value="Italy"/>

 Add address

* Select type *	<input type="text"/>
email	<input type="text" value="esempio@unitest2.com"/>

 Add email

 Add another phone

Born



Figura 63- Editor profilo del POD di solidcommunity

Gestione lista amici

Inizio: 20/01/2023

Fine: 24/01/2023

Analisi

La lista amici è una delle *features* più importanti per un applicativo che abbia lo scopo caratteristiche da *social network*. L'utente deve poter aggiungere amici alla sua lista, visualizzare gli amici già aggiunti e rimuovere utenti dalla lista amici.

Progettazione

Si renderanno disponibili tre elementi grafici per attivare la funzionalità di aggiunta o rimozione e per aprire una vista della lista amici dell'utente autenticato.

Ogni funzionalità interagirà con il POD utente, aggiornando o leggendo la lista degli utenti i cui *webID* siano salvati tra gli amici.

Per far ciò esiste una proprietà, denominata *knows*, la quale descrive una lista di URL corrispondenti ai *webID* di altre persone: essa fa parte della specifica *foaf* ([friend of a friend](#)).

Come per la *vCard*, le [librerie di inrupt](#) offrono dei metodi per gestire le il vocabolario *foaf*.

Post-Sviluppo

Modifiche a *index.html*

```
<button class="btn btn-primary bg-success d-none" id="addfriend">
  Add Friend
</button>
<button class="btn btn-primary bg-danger d-none" id="removefriend">
  Remove Friend
</button>
```

Figura 64 - Aggiunta di due elementi grafici cliccabili all'interno della vista profilo di un utente

```

<!--FRIENDS Modal-->
<div
  class="modal fade mx-auto w-100"
  id="friendsModalToggle"
  aria-hidden="true"
  aria-labelledby="vcardModalToggleLabel"
  tabindex="-1"
>
  <div class="modal-dialog modal-dialog-centered">
    <div class="modal-content">
      <div class="modal-header">
        <h1 class="modal-title fs-5" id="mh-2">Your friends list</h1>
        <button
          type="button"
          class="btn-close"
          data-bs-dismiss="modal"
          aria-label="Close"
        ></button>
      </div>
      <div class="modal-body" id="mb-2">
        <p>
          Log in your account to see your friends list or other users
          friends list
        </p>
      </div>...
    </div>
  </div>

```

Figura 65 - Aggiunta di un elemento grafico per ospitare la vista lista amici

Modifiche a index.js

```
import { VCARD, FOAF } from "@inrupt/vocab-common-rdf";
```

Figura 66 - Importazione metodi foaf

```

buttonFriend.onclick = async function () {
  buttonFriend.innerHTML = "<small>Friend added</small>";
  buttonRemove.innerHTML = "Remove Friend";

  let promise = new Promise((resolve, reject) => {
    addFriend();
  });

  promise.then(() => {
    buttonFriend.classList.remove("bg-success");
    buttonFriend.classList.add("d-none");

    buttonRemove.classList.remove("d-none");
  });
  readProfile();
};

```

Figura 67 - Vista profilo utente. Event listener per il click del tasto di aggiunta amico.

```

buttonRemove.onclick = async function () {
  buttonRemove.innerHTML = "<small>Friend removed</small>";
  buttonFriend.innerHTML = "Add Friend";

  let promise = new Promise((resolve, reject) => {
    removeFriend();
  });

  promise.then(() => {
    buttonRemove.classList.remove("bg-danger");
    buttonRemove.classList.add("d-none");

    buttonFriend.classList.remove("d-none");
  });
  readProfile();
};

```

Figura 68 - Vista profilo utente. Event listener per il click del tasto di rimozione amico.

```

//Show vcard modal footer for functions' buttons
if (!session.info.isLoggedIn) {
  document.getElementById("vcardfooter").innerHTML =
    "<p class='text-center text-muted'>You must be authenticated to add this person to your friends list or sending a message</p>";
}

```

Figura 69 - Vista profilo utente. Controllo per mostrare una vista ospite se l'utente non è autenticato.

```

/**Add a new friend to your friends' list
async function addFriend() {
  let webID = session.info.webId;
  // The WebID can contain a hash fragment (e.g. `#me`) to refer to profile data
  // in the profile dataset. If we strip the hash, we get the URL of the full
  // dataset.
  let myProfileDocumentUrl = new URL(webID);
  myProfileDocumentUrl.hash = "";

  // To write to a profile, you must be authenticated. That is the role of the fetch
  // parameter in the following call.
  let myProfileDataset = await getSolidDataset(myProfileDocumentUrl.href, {
    fetch: session.fetch,
  });

  // The profile data is a "Thing" in the profile dataset.
  let profile = getThing(myProfileDataset, webID);
  webID = document.getElementById("webID").value;

  if (webID === NOT_ENTERED_WEBID) {
    document.getElementById(
      "labelFN"
    ).textContent = `Login first, or enter a WebID (any WebID!) to read from its profile`;
    return false;
  }
  try {
    new URL(webID);
  } catch (_) {
    document.getElementById(
      "labelFN"
    ).textContent = `Provided WebID [${webID}] is not a valid URL - please try again`;
    return false;
  }

  let profileDocumentUrl = new URL(webID);
  profileDocumentUrl.hash = "";
  profile = addUrl(profile, FOAF.knows, webID);

  // Write back the profile to the dataset.
  myProfileDataset = setThing(myProfileDataset, profile);
  // Write back the dataset to your Pod.
  await saveSolidDatasetAt(myProfileDocumentUrl.href, myProfileDataset, {
    fetch: session.fetch,
  });
}

```

Figura 70 - Implementazione della funzione di aggiunta di un utente alla lista amici

```

/**Remove a friend from friends' list
async function removeFriend(id = null) {
  let webID = session.info.webID;
  let friendWebID;
  if (id) {
    friendWebID = id;
  } else {
    friendWebID = document.getElementById("webID").value;
  }
  // The WebID can contain a hash fragment (e.g. `#me`) to refer to profile data
  // in the profile dataset. If we strip the hash, we get the URL of the full
  // dataset.
  let myProfileDocumentUrl = new URL(webID);
  myProfileDocumentUrl.hash = "";

  // To write to a profile, you must be authenticated. That is the role of the fetch
  // parameter in the following call.
  let myProfileDataset = await getSolidDataset(myProfileDocumentUrl.href, {
    fetch: session.fetch,
  });

  // The profile data is a "Thing" in the profile dataset.
  let profile = getThing(myProfileDataset, webID);

  // friendWebID = document.getElementById("webID").value;

  profile = removeUrl(profile, FOAF.knows, friendWebID);

  // Write back the profile to the dataset.
  myProfileDataset = setThing(myProfileDataset, profile);
  // Write back the dataset to your Pod.
  await saveSolidDatasetAt(myProfileDocumentUrl.href, myProfileDataset, {
    fetch: session.fetch,
  });
}

```

Figura 71 - Implementazione della funzione di rimozione di un utente dalla lista amici

```

let myWebID = session.info.webId;

// The WebID can contain a hash fragment (e.g. `#me`) to refer to profile data
// in the profile dataset. If we strip the hash, we get the URL of the full
// dataset.
let myProfileDocumentUrl = new URL(myWebID);
myProfileDocumentUrl.hash = "";

// To write to a profile, you must be authenticated. That is the role of the fetch
// parameter in the following call.
let myProfileDataset = await getSolidDataset(myProfileDocumentUrl.href, {
  fetch: session.fetch,
});

// The profile data is a "Thing" in the profile dataset.
profile = getThing(myProfileDataset, myWebID);

let friendsUrl = getUrlAll(profile, FOAF.knows); //get url of all friends
let friendsList = [];

if (friendsUrl.length != 0) {
  for (let i = 0; i < friendsUrl.length; i++) {
    const personUrl = friendsUrl[i];
    const myFriendDataset = await getSolidDataset(personUrl, {
      fetch: session.fetch,
    });

    const personProfile = getThing(myFriendDataset, personUrl);
    const personName = getStringNoLocaleAll(personProfile, VCARD.fn);
    const personAvatar = getUrl(personProfile, VCARD.hasPhoto);

    const friend = {
      name: personName[0],
      url: personUrl,
      avatar: personAvatar,
    };

    friendsList.push(friend);
  }
}

```

Figura 72 - Funzione readProfile(), caricamento della lista amici in una variabile locale

```

if (friendsUrl.includes(webID)) {
  document.getElementById("addfriend").classList.add("d-none");
  document.getElementById("removefriend").classList.remove("d-none");
} else {
  document.getElementById("removefriend").classList.add("d-none");
  document.getElementById("addfriend").classList.remove("d-none");
}

} else {
  document.getElementById("removefriend").classList.add("d-none");
  document.getElementById("addfriend").classList.remove("d-none");
}
}

```

Figura 73 - Funzione readProfile(), scelta elemento grafico da visualizzare, in base alla presenza o meno dell'utente visualizzato nella lista amici

```
//Update the friends modal with the user friend list
document.getElementById("mh-2").innerHTML = `${formattedName}'s Friend List`;
if (friendsList.length != 0) {
  document.getElementById("mb-2").innerHTML = `<ul id="friendlist"></ul>`;

  for (let i = 0; i < friendsList.length; i++) {
    if (friendsList[i]["avatar"] != null) {
      document.getElementById("friendlist").innerHTML += `
<li class="friendelem" id="${friendsList[i]["url"]}">
<a href="${friendsList[i]["url"]}">${friendsList[i]["name"]}</a>

<button type="button" class="btn btn-primary w-25 auxbtnread" data-user="${friendsList[i]["url"]}">
<i class="fa-solid fa-glasses" ></i> Read </button>
<button type="button" class="btn btn-danger w-25 auxbtnremove" data-user="${friendsList[i]["url"]}" data-fn="${friendsList[i]["name"]}">
<i class="fa-solid fa-xmark" ></i> Remove </button>
</li>`;
    } else {
      document.getElementById("friendlist").innerHTML += `
<li class="friendelem" id="${friendsList[i]["url"]}"><a href="${friendsList[i]["url"]}">${friendsList[i]["name"]}</a>
<button type="button" class="btn btn-primary w-25 auxbtnread" data-user="${friendsList[i]["url"]}"><i class="fa-solid fa-glasses" ></i>
Read </button>
<button type="button" class="btn btn-danger w-25 auxbtnremove" data-user="${friendsList[i]["url"]}" data-fn="${friendsList[i]["name"]}">
<i class="fa-solid fa-xmark" ></i> Remove </button>
</li>`;
    }
  }
}
}
```

Figura 74 - Aggiunta di elementi HTML per visualizzare, nel modale, gli utenti appartenenti alla lista amici e i tasti per leggere il profilo o rimuovere l'amico

```
let list = document.querySelector("#friendlist");

list.addEventListener("click", (e) => {
  if (e.target.classList.contains("auxbtnremove")) {
    buttonFriend.innerHTML = "Add Friend";

    if (
      confirm(
        "Are you sure you want to remove " +
        e.target.getAttribute("data-fn") +
        " from your friends list?"
      ) == true
    ) {
      removeFriend(e.target.getAttribute("data-user"));

      for (let i = 0; i < friendsList.length; i++) {
        if (friendsList[i]["url"] == e.target.getAttribute("data-user")) {
          document
            .getElementById(friendsList[i]["url"])
            .classList.add("d-none");
          friendsList.splice(i, 1);
        }
      }

      if (friendsList.length == 0) {
        document.getElementById(
          "mb-2"
        ).innerHTML = `<p>No friends in your list. </p>`;
      }
      readProfile();
    }
  }
});
```

Figura 75 - Funzione readProfile(), event listener per gli elementi grafici per leggere o rimuovere un amico nella lista, parte 1

```
        if (e.target.classList.contains("auxbtnread")) {  
            readProfile(e.target.getAttribute("data-user"));  
        }  
    });  
} else {  
    document.getElementById(  
        "mb-2"  
    ).innerHTML = `

No friends in your list. </p>`;  
}  
}


```

Figura 76 - Funzione readProfile(), event listener per gli elementi grafici per leggere o rimuovere un amico nella lista, parte 2

Interfaccia

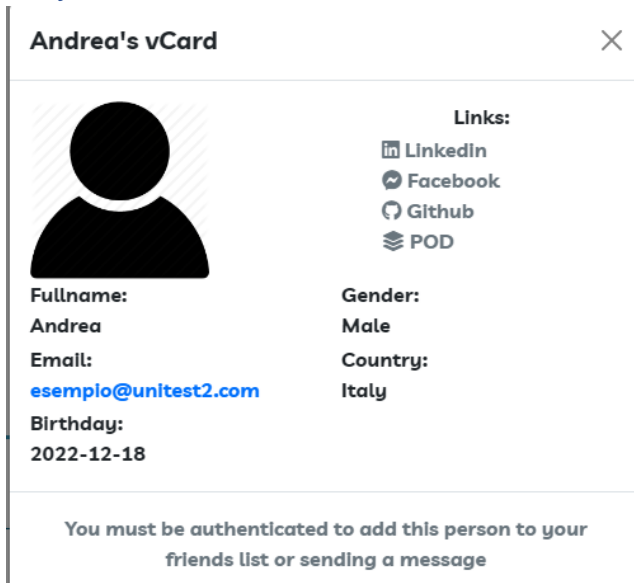


Figura 77 - Vista profilo esterno come utente non autenticato

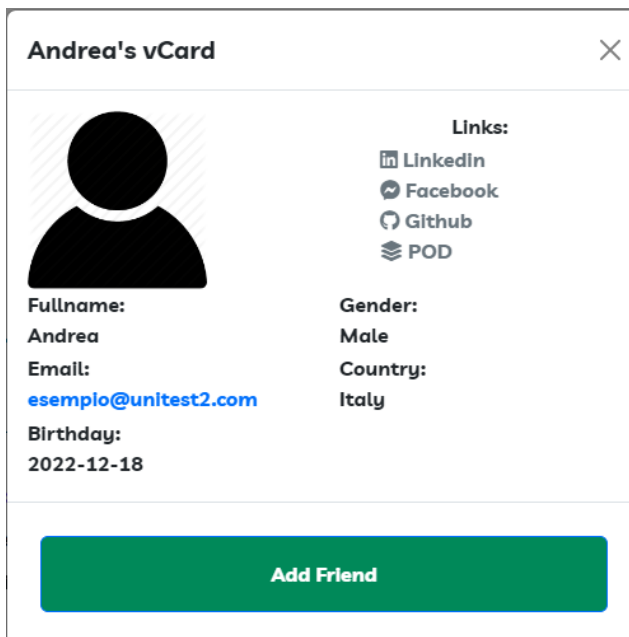


Figura 78 - Vista profilo esterno come utente autenticato (utente non in lista amici)

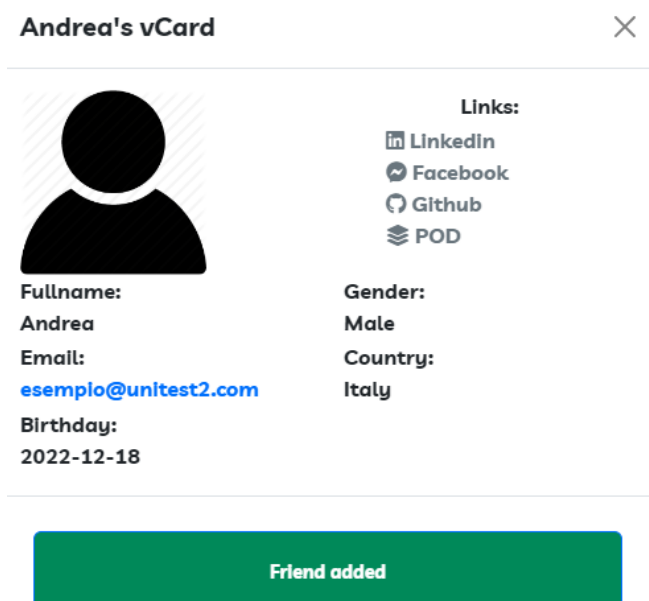


Figura 79 - Feedback visivo al click del tasto di aggiunta amico

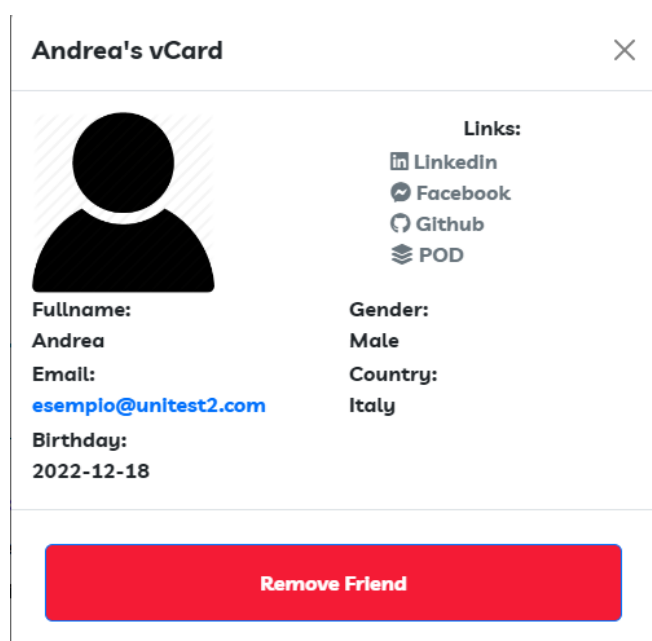


Figura 80 - Vista profilo esterno come utente autenticato (utente già in lista amici)

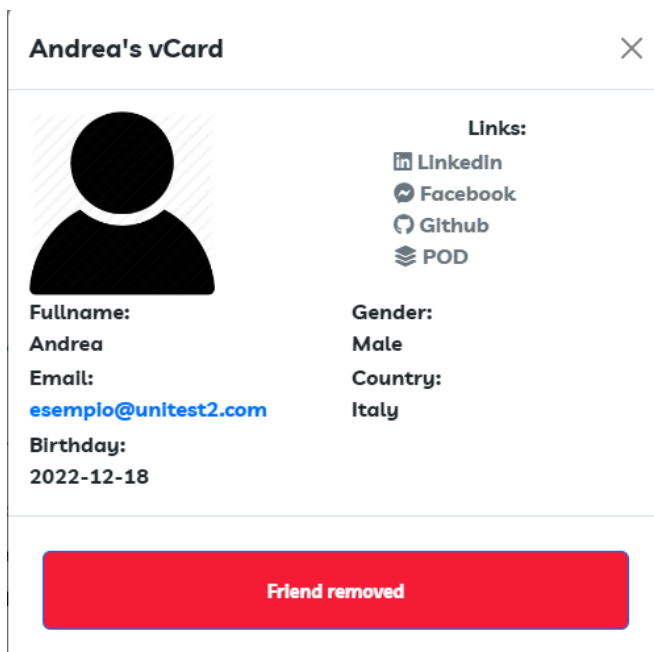


Figura 81 - Feedback visivo al click del tasto rimozione amico

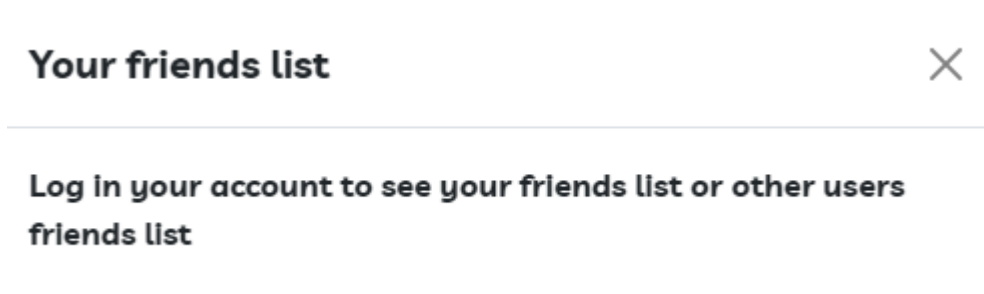


Figura 82 -Vista lista amici, utente non autenticato

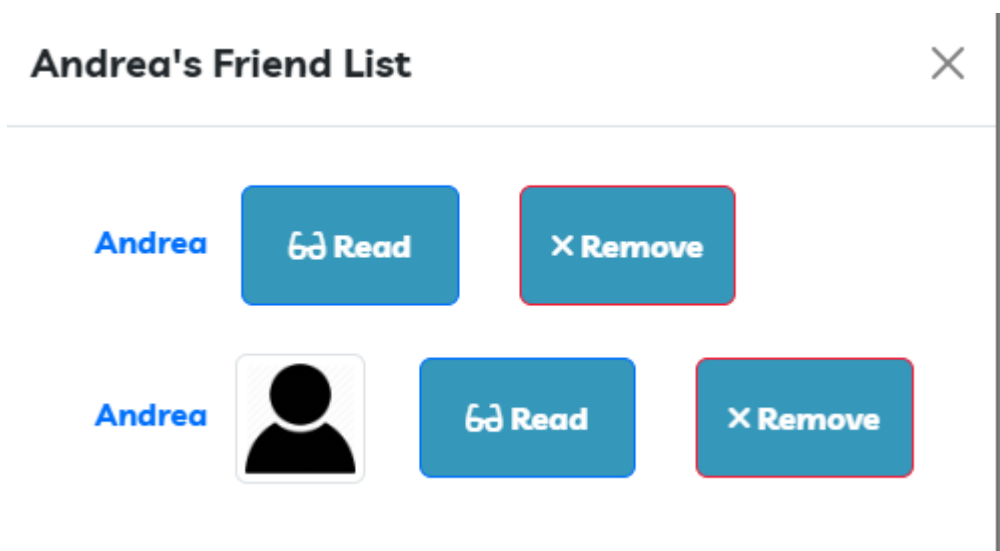


Figura 83 - Vista lista amici, utente autenticato

Risultato

Verificando lo stato della lista amici dal visualizzatore del POD , si denota che le operazioni di aggiornamento o lettura della lista dei propri amici funzionano correttamente col cambiamento di stato delle informazioni del POD.

