Specifica Tecnica

Specifiche Protocollo WebApp Regione Veneto

Indice del documento

Scopo del documento

Il Committente è interessato alla creazione di un'infrastruttura innovativa per la creazione di portali web in grado di ospitare, all'interno delle sue pagine, applicativi già a sua disposizione.
L'obiettivo emerso è quello di permettere a tali applicativi di lavorare non solo separatamente, ma anche di collaborare tra di loro, di scambiarsi informazioni ed eventi, al fine di permettere all'utente una fruizione molto più efficace ed efficiente delle informazioni.
Nel caso specifico l'utente è un membro del personale sanitario di una (o più) aziende ospedaliere, il cui compito non è, ovviamente, quello di dover imparare le diverse logiche di funzionamento di ogni programma, ma bensì dovrebbe poter utilizzare la tecnologia per velocizzare la fase di diagnosi e di cura dei pazienti, in un contesto applicativo unificato.
Inoltre non è da sottovalutare l'impatto che avrebbe il riuso di componenti software (soprattutto interfacce grafiche) già valide in contesti anche diversi e nuovi rispetto a quelli per cui erano state pensate inizialmente.
Molte volte infatti software diversi sono costretti a sviluppare interfacce grafiche diverse per visualizzare o gestire gli stessi dati, solamente perchè le interfacce già esistenti risiedono isolate in un diverso programma.

Lo scopo di questo documento è quindi quello di fornire al Committente una proposta concreta di un protocollo di comunicazione che permetta ai vari software di dialogare tra loro, senza sconvolgerne la struttura.
I software che si adegueranno a tale protocollo saranno in grado di emettere o ricevere informazioni ed eventi da e per altri software (in seguito chiamati widget), di aprire su richiesta una nuova finestra contenente uno specifico widget e di ricevere informazioni di ritorno da essa.

Soluzione Proposta

In questo documento andiamo a descrivere una proposta di definizione di standard di comunicazione bidirezionale tra pagine html provenienti diversi fornitori software.
Definiamo due tipologie di elementi: Container e Widget.

1. Container: è una pagina html in grado di ospitare dei widget via *iframe*[1].
2. Widget: è una pagina html caricata da un container con cui è in grado di comunicare bidirezionalmente.

La comunicazione avviene usando l'API *window.postMessage*[2] che abilita la comunicazione *cross-origin* tramite messaggi.

Container

Un container è una pagina html che carica dinamicamente uno o più iframe.
Il caricamento e l'utilizzo di un widget segue il seguente schema:

1. Caricamento dell'iframe con sorgente il widget.
2. Registrazione del widget usando una procedura di handshaking descritta di seguito.
3. A questo punto il container può invocare i metodi pubblici del widget. Inoltre può stare in ascolto di eventuali eventi emessi dal widget.

Widget

Un widget è una pagina html di un applicativo web che è stata aggiornata per rispondere al protocollo proposto. Tale pagina può continuare a lavorare come in precedenza, nel caso sia utilizzata direttamente (in modalità stand-alone, senza interazioni con altri componenti) o può interagire in modo nuovo con altri widget, provenienti da altri applicativi eterogenei (che abbiano anch'essi implementato il protocollo).

Ad alto livello un widget si occupa di creare una sezione della pagina finale. Tale porzione è definita e delineata dal Container visto in precedenza. La pagina finale sarà il risultato del rendering di tutti i widget definiti dal Container. Ogni widget potrà comunicare direttamente con il Container, il quale potrà veicolare i messaggi anche agli altri widget (agendo come una sorta di "router" tra i vari widget).

[1] https://developer.mozilla.org/en/docs/Web/HTML/Element/iframe
[2] https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage

Un Widget esibisce 3 macro-stati.

Stato neutro

Il widget è caricato direttamente dalla sua pagina html, non quindi per mezzo di un iframe. Nessun evento viene emesso tramite chiamate postMessage. Il widget non è in ascolto di nessun evento (i.e. nessun listener di eventi *'MessageEvent'* con tipo *'message'* registrato).

Stato attivabile

Il widget è caricato con successo da un container per mezzo di un *iframe*, nessun evento viene emesso. Il widget è in ascolto, esclusivamente da parte del suo container, di un messaggio di handshaking (tipo 1.a, vedi sotto). Una volta terminato l'handshaking il widget passa allo stato attivo.

Stato attivo

Il widget emette tutti gli eventi in grado di emettere specificati nella sua business logic (tipo 2.a).
Il widget è in ascolto ed in attesa di messaggi esclusivamente da parte del suo container (tipo 2.b).
Il widget reagisce solo a messaggi noti al widget stesso.

Tipologie di messaggi e loro uso

La comunicazione è ristretta a quattro tipologie di messaggi divisi in due categorie:

1. Handshaking:
    a. *'handshake'*
    b. *'handshakeReply'*
2. Comunicazione bidirezionale:
    a. *'emit'*
    b. *'call'*

La struttura dei messaggi scambiati è la seguente. Per tutti il tipo MIME[3] corrisponde alla stringa 'application/x-spu-v1+json'.

Tipo di messaggio	Struttura
1.a *'handshake'*	`{ mimeType: 'application/x-spu-v1+json', type: 'handshake' }`
1.b *'handshakeReply'*	`{ mimeType: 'application/x-spu-v1+json', type: 'handshakeReply' }`
2.a *'emit'*	`{ mimeType: 'application/x-spu-v1+json', type: 'emit', event: ${eventName},`

[3] https://en.wikipedia.org/wiki/MIME

	data: ${eventData} }

2.b *'call'*	{
	mimeType: 'application/x-spu-v1+json',
	type: 'call',
	action: ${actionName},
	data: ${actionData}
	}

Handshaking

Per abilitare la comunicazione container/widget un processo di *handshaking* viene iniziato da parte del container usando un messaggio di tipo *'handshake'*. Il widget risponde tramite un messaggio *'handshakeReply'*.
Nota Bene: il widget può rifiutare l'attivazione (e.g. il container non ha una sorgente conosciuta).

Solamente i widget in stato attivabile sono in ascolto e rispondo al messaggio *'handshake'* (stato neutro e attivo ignorano il messaggio). Successivamente all'invio del messaggio *'handshakeReply'* il widget entra in stato attivo. Tutti gli eventi che il widget può emettere vengono ora emessi. Inoltre il widget reagisce a tutti gli eventi da lui conosciuti.

Comunicazione bidirezionale

Un widget in stato attivo interagisce con il suo container tramite i messaggi *'emit'* e *'call'*.
I messaggi di tipo *'emit'* sono usati per emettere eventi e possono essere accompagnati da un oggetto contenente dati contestuali.
I messaggi di tipo *'call'* sono usati dal container per invocare funzionalità del widget. Anch'essi possono essere accompagnati da un oggetto di dati contestuali. L'uso di questo messaggio è paragonabile all'invocazione di un metodo pubblico del widget senza valore di ritorno.

Autenticazione

Essendo i widget ospitati in pagine e domini diversi è necessario un meccanismo di autenticazione e autorizzazione condivisa tra widget. Come soluzione di single sign-on (SSO) si propone l'utilizzo dei JSON Web Token (JWT)[4] che permettono di firmare in maniera compatta un insieme di asserzioni (i.e. *claim*) in una struttura dati JSON.

I token saranno firmati tramite l'utilizzo di chiavi asimmetriche (pubblica/privata).
Tale meccanismo, supportato dal protocollo, permetterà a tutti i diversi widget di verificare l'autenticità della firma del token, semplicemente andando a verificare la firma con la chiave pubblica dell'*identity provider*. Per semplificare l'individuazione dell'identity provider si prevede di aggiungere un vincolo sulla struttura dei token: rendere obbligatorio il *claim* '*iss*' (*issuer*[5]).
L'*identity provider* che firma il token deve quindi obbligatoriamente inserire questa informazione che sarà successivamente utilizzata per reperire la chiave pubblica necessaria per verificare i token.

Nel prototipo implementato (descritto nel capitolo "Live Demo") prevediamo un widget specifico per l'autenticazione, che, una volta eseguita, verrà notificata al contenitore sotto forma di token JWT. Il contenitore ha la responsabilità di utilizzare questo token come dato contestuale durante l'invocazione dei metodi pubblici dei widget che richiedono autenticazione.

Ipotizzando l'applicazione della parte security del protocollo ad una struttura esistente realistica, si può evidenziare che l'uso del token JWT permette di fatto di comunicare alle applicazioni che ricevono il token una serie di informazioni molto ricca sull'identità e sui ruoli / permessi dell'utente.
La struttura del token è infatti estendibile a piacimento per includere tutte le informazioni che possono servire oggi o che potranno servire in futuro.
La firma del token garantirà, alle applicazioni che lo riceveranno, di poter validare i *claims* contenuti senza dover necessariamente collegarsi ad un backend per caricare i dati di profilazione.
Si potrà decidere di inserire nella struttura del token anche dati quali il reparto di appartenenza dell'utente, l'azienda di appartenenza o altro ancora.
L'applicazione che riceverà il token potrà ignorare questi dati, se non rilevanti, o usarli nel modo che preferisce.

Si può pensare al token JWT firmato come ad un certificato digitale rilasciato da uno o più issuer che l'organizzazione decide di ritenere affidabili.
L'aggiunta di un nuovo issuer è resa estremamente semplice dalla presenza di un repository centrale che contiene l'elenco degli issuers e delle loro chiavi pubbliche.

[4] https://en.wikipedia.org/wiki/JSON_Web_Token
[5] https://tools.ietf.org/html/rfc7519#section-4.1.1

Componenti Tecnologici

Di seguito la descrizione delle tecnologie e dei framework utilizzati.

Window.postMessage()

Web Message[6] è una *API* che abilita la comunicazione tra siti diversi in maniera sicura e diretta. Solitamente script in pagine diverse non possono comunicare tra di loro se l'origine non è la stessa.
Per definizione, due pagine hanno la stessa origine se e solo se protocollo, porta (se specificata) e host sono gli stessi per entrambe le pagine[7]

Questa API fornisce un meccanismo controllato e nativo per abilitare questo tipo di comunicazione che può essere considerato sicuro se usato correttamente. Il principio portante su cui si basa questa funzionalità si basa sul processare messaggi solo se attesi e con sorgente nota.
Avendo la possibilità di maneggiare questi vincoli in maniera nativa ci permette di specificare logiche complesse utilizzando come base l'*API postMessage()*.

In questo documento abbiamo descritto un protocollo di comunicazione tra container/widget costruibile interamente usando questa API. Una possibile implementazione di questo protocollo, con il suo utilizzo, è mostrato nella libreria di supporto sviluppata a completamento del prototipo.

Alcuni link utili:

- Living standard whatwg: https://html.spec.whatwg.org/multipage/comms.html#posting-messages
- W3C Recommendation: https://www.w3.org/TR/webmessaging

JWT

JWT (acronimo di JSON Web Token) è uno standard open basato su JSON per la creazione di access token che contengano un certo numero di asserzioni su chi possiede un dato token. Il token viene generato dal server e firmato, solitamente con una password. Viene poi inviato al client (tipicamente un Web Browser) che può usare il token per le successive richieste al server. Il server, in possesso della sua password, è in grado di verificare l'autenticità del token ricevuto e quindi di applicare gli opportuni filtri di sicurezza alle richieste provenienti dal client.
I token sono implementati in modo da essere compatti, URL-safe e ottimizzati per essere usati in ambito di single sign on web. Il token può (opzionalmente) essere anche criptato. In questo caso il client non sarà in grado di utilizzare le informazioni al suo interno, ma potrà solamente usarlo per confermare la propria identità.

L'uso dei token JWT è particolarmente adatto in architetture a microservizi, o nelle quali si debba interagire con più service provider differenti senza dover effettuare un login esplicito verso ognuno di essi. Questa tecnologia si sta affermando come standard de-facto nelle nuove architetture distribuite e session-less, in concomitanza con lo sviluppo sempre maggiore di backend che

[6] https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage
[7] Per dettagli:
https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy#Definition_of_an_origin

espongono interfacce REST (e la conseguente diffusione di framework di front-end basati su javascript).

Queste motivazioni ci hanno portato a sceglierlo come miglior soluzione di single sign on tra tutti i widget di una pagina.
Alcuni link utili:

- IETF RFC, Standard track. Status: proposed standard: http://www.rfc-editor.org/info/rfc7519
- Lista delle librerie supportate: https://jwt.io/#libraries
- Gentle introduction: https://jwt.io/introduction/
- Atlassian, Understanding JWT: https://developer.atlassian.com/static/connect/docs/latest/concepts/understanding-jwt.html
- Esempi di codice: https://github.com/alemhnan/SPU/tree/master/Docs/JWT

Iframe

L'iframe (*inline frame*) è un Elemento HTML. Si tratta infatti di un frame "ancorato" all'interno della pagina. Equivale ad un normale frame, ma con la differenza di essere un elemento inline (interno) della pagina, non esterno.
L'iframe viene generalmente utilizzato per mostrare il contenuto di una pagina web, o di una qualsivoglia risorsa, innestato all'interno di un riquadro in una seconda pagina principale.

Nel linguaggio HTML l'iframe viene rappresentato tramite il tag *<iframe></iframe>*.

Nel protocollo proposto l'uso dell'iframe serve per poter renderizzare all'interno di un'unica pagina contenuti diversi provenienti da applicazioni eterogenee.
La soluzione finale dovrà far credere all'utente finale di aver davanti un'unica pagina armonicamente composta, senza percepire la diversità dei vari componenti.
Ogni widget verrà di fatto caricato tramite un iframe presente nel container principale.
Purtroppo, a causa di limitazioni di sicurezza (correttamente) presenti in tutti i browser, non è possibile propagare la parte di stile (css) dal container verso i widget.
Sarà quindi necessario definire una strategia a livello implementativo del protocollo per fornire ai vari fornitori l'accesso a fogli di stile base condivisi, che ogni applicazione dovrà importare per poter dare alla pagina finale un aspetto unitario.

Live Demo

A dimostrazione della fattibilità della proposta di standard di cui sopra è stato sviluppato un prototipo funzionante che comprende tutti i casi di comunicazione prevista e che copre le criticità di sicurezza rilevate nella fase di analisi e definizione.

Nel prototipo sono presenti i seguenti componenti:

- Quattro Widget
- Due Container
- Due Server
- Una Libreria

Widget

I quattro widget comprendono:

- Login: https://loginspu.surge.sh: permette ad un utente già registrato di effettuare il login. Successivamente a questa operazione l'utente potrà avere accesso ai suoi dati (nel prototipo, un elenco di messaggi inviati) ed inviare nuovi dati;
- Signup: https://signupspu.surge.sh: permette ad un utente anonimo di registrarsi presso il sito e quindi di poter effettuare il login;
- Read: https://readspu.surge.sh: permette agli utenti loggati di recuperare i loro dati (come già, detto, un elenco di messaggi inviati)
- Write: https://writespu.surge.sh: permette agli utenti loggati di effettuare operazioni con side effect (nel nostro caso, l'invio di un messaggio)

Questi widget usano due server per svolgere le loro funzioni:

- https://spu.herokuapp.com: gestisce la registrazione di nuovi utenti e il rilascio di token di autenticazione una volta effettuato il login. Usato dai widget *Login* e *Signup*.
- https://readwritespu.herokuapp.com: gestire la lettura e scrittura di dati inerenti agli utenti usando come autenticazione il token di cui sopra. Usato dai widget *Read* and *Write*.

Container

Qui troviamo due istanze: una autorizzata ed una non autorizzata.
Si dimostra così un aspetto importante della sicurezza del protocollo. I widget, infatti, non sono tenuti a conoscere ex ante ogni possibile container che potrà invocarli. Pertanto è necessario che il protocollo impedisca ad un possibile attaccante di poter creare un container malevolo che includa a proprio piacimento uno o più widget. Se i widget accettassero la comunicazione con tale container si prospetterebbe una falla di sicurezza estremamente grave.

Il primo container, autorizzato, compone i quattro widget in due flussi principali (oltre alla registrazione utente): https://containerspu.surge.sh.

- Flusso 1.

- L’utente si autentica tramite il widget Login che emette un evento ‘LOGGED’ accompagnato dal token di autenticazione.
- Il container invoca due metodi pubblici sui widget Read and Write. Nel primo caso abilita la lettura dei dati specifici dell’utente (‘LOAD_USER_INFO’). Nel secondo caso abilita la scrittura di nuovi dati (‘ENABLE_WRITE’).

```
// Flow 1
loginWidgetHandler.on('LOGGED', (data) => {
    tokenDecoded = jwt_decode(data.token);
    token = data.token;
    readWidgetHandler.call('LOAD_USER_INFO', { token: data.token, userId: tokenDecoded.userId });
    writeWidgetHandler.call('ENABLE_WRITE', { token: data.token, userId: tokenDecoded.userId });
});
```

Riferimento:
https://github.com/alemhnan/SPU/blob/master/Widgets/Containers/MainContainer/container.js#L70-L75

- Flusso 2.
  - L’utente scrive nuovi dati tramite il widget Write che emette un evento ‘WROTE’.
  - Il container non appena notificato dell’evento ricarica i dati del widget Read.

```
// Flow 2
writeWidgetHandler.on('WROTE', () =>
    readWidgetHandler.call('LOAD_USER_INFO', { token, userId: tokenDecoded.userId }));
});
```

Riferimento:
https://github.com/alemhnan/SPU/blob/master/Widgets/Containers/MainContainer/container.js#L78-L79

Il secondo container è usa lo stesso codice ospitato in un dominio diverso non abilitato: https://badcontainerspu.surge.sh.
I widget singolarmente funzionano ma sono inerti (stato neutro) in quanto rifiutano l'inizializzazione.

Server

Finora tutti gli sviluppi e la maggior parte degli elementi del protocollo proposti hanno riguardato la componente frontend dei widget.
Immaginando uno scenario realistico di applicazioni web sono stati sviluppati quindi anche due componenti backend, per dimostrare che la loro presenza è assolutamente compatibile con la proposta. I backend sviluppati sono di tipo RESTfull, ma nulla vieta la presenza di backend tradizionali in grado di servire direttamente pagine html.
I due server sono:

- LoginSignup: https://spu.herokuapp.com
  - Ha la responsabilità di creare utenti e rilasciare token ad utenti che dimostrano le corrette credenziali (email/password). Inoltre fornisce un elenco di domini validi.
  - Le API rilevanti sono:
    - `POST /auth/login`
    - `POST /auth/signup`
    - `GET /auth/allowedDomains`
- ReadWrite: https://readwritespu.herokuapp.com

- Ha la responsabilità di leggere e scrivere informazioni contestuali all'utente se e solo se i token presentati sono validi.
- Le API rilevanti sono:
  - `GET /userinfo/:userid`
  - `POST /userinfo/:userid`

Il server LoginSignup rilascia JSON Web Token con la struttura descritta in questo documento e firmati con una sua chiave privata e algoritmo 'RSA256'. Il campo *issuer* (i.e. *iss*) dichiarato risulta essere uguale a: '`SPUWP`' (acronimo di Standard Portale Unico Working Prototype).

Il server ReadWrite verifica i token e decide di autorizzare o meno le chiamate utilizzando la chiave pubblica relativa all'*issuer* dichiarato nel token. L'elenco di chiavi pubbliche in questo esempio è poco rappresentativo in quanto limitato a una chiave. Il meccanismo tuttavia è espandibile ed utilizzabile con un numero a piacimento di chiavi. Nello scenario da noi immaginato, ogni ASL (come anche Regione Veneto) potranno avere una loro chiave privata per firmare i token jwt e quindi una corrispondente chiave pubblica che i widget potranno recuperare per verificare tali firme.

Libreria di supporto

A supporto del lavoro di sviluppo del prototipo è stata realizzata una libreria che implementa il protocollo descritto in questo documento. Il codice sorgente della libreria è liberamente utilizzabile e può considerarsi sufficientemente maturo e stabile. La libreria è fortemente ispirata a *Postmate*[8] e *Penpal* [9].
Tale libreria può essere utilizzata come base per l'implementazione di widget più complessi da parte dei fornitori delle applicazioni attuali.
Tuttavia al momento della scrittura del documento non è supportata e/o pubblicata ed è sprovvista di test di unità/funzionali.
Se richiesto potrà essere estesa in fase di implementazione del protocollo.

[8] https://github.com/dollarshaveclub/postmate
[9] https://github.com/Aaronius/penpal