Progetto PAWM - matricola 101101

organizzazione dell'applicativo

scopo

L'applicazione ha lo scopo di reimplementare in PWA una vecchia applicazione intranet web-form .net con librerie obsolete, non piu' disponibili e con licenze modificate, attualmente pubblicata su un host da dismettere perche' fuori supporto.

funzionamento dell'applicativo

L'applicativo attualmente viene usato per censire i sistemi informativi (era nato con altro scopo nel 2011) e fa parte del workflow delle certificazioni ISO27000 di un ente pubblico.

- Tutti gli utenti (anche anonimi) devono poter vedere le informazioni disponibili.
- Solo gli utenti accreditati possono censire dei nuovi sistemi informativi.
- Una volta creato il sistema puo' esssere modificato solo dagli utenti abilitati.
- L'autore ed il il service owner sono automaticamente abilitati.
- L'autore puo' abiliare espressamente altri utenti accreditati.

L'autore e gli utenti abilitati:

- possono apportare modifiche al sistema informativo (e alla lista degli utenti abilitati)
- vengono notificati tramite il canale scelto (email, telegram,...) delle modifiche apportate (flusso realizzato da processi in ascolto non inclusi)

Alla modifica di un sistema informativo corrisponde poi dei flussi di approvazione e creazione di documenti che avvengono su altri sitemi informativi.

scelte effettuate

stack applicativo

L'applicazione viene scelta come banco di prova per alcune tecnologie che si vuole approfondire:

- **supabase** con installazione on-premise, per verificare l'usabilita' e la distanza con il servizio cloud ufficiale
- **postgrest**: forza lo sviluppo tramite API
- **postgrest + postgres**: come modo per forzare un profilo di sicurezza esteso a tutti i livelli
- **elm** : linguaggio funzionale puro che impedisce il verificarsi di errori di runtime e facilita la manutenzione del codice non avendo side-effects

autenticazione

Nel contesto in cui dovra' esistere l'applicazione il sistema di autenticazione e' quello ufficiale dell'ente, federato SAML e SPID.

Per l'esame di PAWM si sceglie di utilizzare l'autenticazione tramite OAuth basata sul Google IDS, in quanto l'ateneo e' registrato come organizzazione `unicam.it`, quindi tutti gli utenti hanno un account Google.

Usando quindi le credenziali `unicam.it` accedendo al portale di Google Cloud Platform si e' configurato un progetto a cui concedere il rilascio di credenziali

gestione dell'api-token Il token JWT ottenuto e' comprensivo di tutte le informazioni sull'utente autenticato e riporta le informazioni sul rilascio e la sua durata.

Per permettere la persistenza della sessione utente si sceglie di attivare una procedura di auto-rinnovo lanciato in prossimita' della scadenza, usando due strategie:

- configuando `supabase` in modo che proceda autonomamente al rinnovo del token prima della scadenza
- fallback: in background (tramite le `Background_Tasks_API`: https://developer.mozilla.org/en-US/docs/Web/API/Background_Tasks_API), per garantire la persistenza anche se l'applicativo e' in pausa

La background api e' disponibile in tutti i browser **tranne safari**, ma il rinnovo in background e' un'ottimizzazione del meccanismo di mantenimento della sessione, quindi non preclude il funzionamento su tutte le piattaforme. Si preferisce usare comunque la `Background Api` piuttosto che i `WebWorker` (api maggiormente diffusa) perche' il codice da eseguire e' talmente piccolo che e' sconveniente istanziare un thread per eseguirlo, come farebbero invece i `WebWorker` che risulterebbero troppo pesanti.

La chiamata alle `Background Api` rappresenta comunque una ridondanza perche' le funzionalita' di `supabase` prevedono gia' il rinnovo automatico del token alla scadenza (comportamento non di default ma attivato dalla configurazione corrente).

La verifica delle credenziali in corrispondenza di ogni richiesta di una pagina dietro autenticazione e' gestita in Elm da una unica funzione (in Main.elm) che esegue in fase di inizializzazione della pagina indicata dalla rotta corrente:

```
if Page.needAuth model.route &&

Session.viewer model.session.session == Nothing

then (HomePage, Api.login ())

else

case model.route of

[...]
```

autorizzazione - permessi

Si sceglie:

- di abilitare l'applicazione ai soli utenti interni all'organizzazione `unicam.it`
- di permettere l'accesso al solo ambito `userinfo.email`, mentre nell'applicazione reale verrebbero estratte dal token di autenticazione piu' informazioni (es. codice fiscale).

Ogni strato applicativo deve avere la garanzia che le informazioni vengono fornite e modificate in aderenza alle regole di cui sopra.

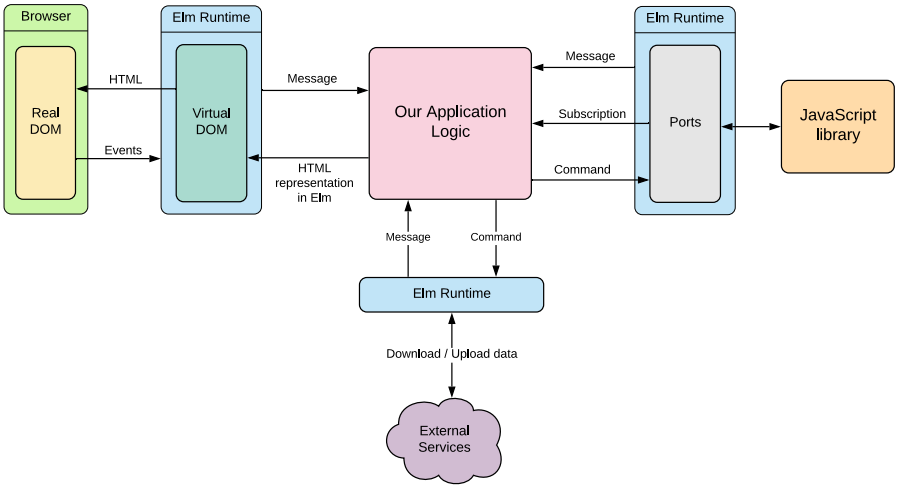
Si sceglie quindi di posizionare la verifica dei permessi/autorizzazioni nel livello piu' basso possibile dell'applicativo, sul database, in modo che sia impossibile un utilizzo improprio dei dati.

Questa scelta ha come conseguenze:

- maggiore velocita' di estrazione/elaborazione dei dati da parte del database:
- minor numero di dati estraibili dal db, in accordo ai permessi concessi
- il motore di ottimizzazione di Postgres tiene conto della visibilita' dei dati e del loro peso per valutare come eseguire le query complesse
- piu' facile intercettare errori durante lo sviluppo o modifica dell'applicativo
- unico punto in cui le regole vengono applicate e possono essere modificate

motivo della scelta di Elm

Elm e' un linguaggio funzionale puro che usa **comandi, abbonamenti e messaggi** per gestire correttamente gli effetti collaterali originati dall'interazione con il mondo esterno, portando ad una netta distinzione tra runtime Elm (dove gli effetti collaterali sono impossibili) ed il mondo esterno con cui scambia messaggi.



Elm adotta una strategia comune sia per la gestione dei servizi HTTP esterni che per la comunicazione con il runtime Js, fornendo un'interfaccia unica per tutto quello che e' esterno al runtime di Elm:

- Elm invia un comando
- riceve un messaggio come risposta

In Elm gli applicativi web vengno strutturati seguendo quella che e' definita l'Architettura delle applicazioni Elm, dove ogni pagina e' definita come una macchina a stati finiti in cui gli eventi sono **comandi, abbonamenti e messaggi** gestiti come dati in transito.

L'Arichitettura e' divisa nelle parti:

- Modello: lo stato della tua applicazione
- Visualizzazione: un modo per trasformare il tuo stato in HTML
- Aggiornamento: un modo per aggiornare il tuo stato in base ai messaggi

Dove il runtime Elm si fa carico "solo" di trasmettere i messaggi (tra codice Elm e resto del mondo) e invocare le funzioni di visualizzazione ad ogni modifica del modello.

Il resto deve essere definito tramite l'architettura descritta.

L'unione di:

- architettura semplice ed essenziale
- assenza di convenzioni e libreire auto-magiche
- netta separazione tra cio' che e' interno al runtime ed esterno
- linguaggio funzionale puro, con tipizzazione statica forte (dialetto di haskell)

producono applicazioni:

- esenti da errori a runtime non gestiti (sono sempre possibili errori ad esempio nell'invocazione di un'Api, ma il compilatore obbliga a gestirli)
- strutturalmente stabili: non ci saranno cambiamenti nell'architettura o nel linguaggio che porteranno a dover riscrivere o cambiare quanto gia' fatto (l'autore non prevede cambiamenti almeno per un decennio)
- esenti da effetti collaterali
- quindi facili da modificare, integrare, ristrutturare anche a distanza di anni
- prive di parti auto-magiche (possibile integrando alcune librerie ma la semplicita' dell'architettura non ne fa sentire la necessita')
- manutenibili negli anni
- verificabili
- facilmente debuggabili usanto la time-machine integrata (registrazione degli eventi/stati nel tempo), permettendo di comprendere il funzionamento interno di un'applicativo semplicemente usandolo
- particolarmente piccole e veloci

approcio di sviluppo con supabase

Supabase fornisce un client che permette

- l'orchestrazione di tutti i servizi di cui si compone
- startup dell'ambiente con caricamento di dati e configurazione da script versionati
- possibilita' di adottare dei workflow di sviluppo anche complessi (come git-workflow)
- completa integrazione con lo strumento git

L'applicazione sviluppata poi potra' essere pubblicata direttamente nel loro cloud (SaaS) o on-premis, permettendo in qualsiasi momento (es. esigenze di disaster recovery) di spostare l'applicativo on-premis in cloud.

I seed per il popolaento del database all'avvio sono stati esclusi dal repository per evitare problemi di privacy.

L'ambiente di sviluppo dell'applicativo e' stato creato strutturato usando Create Elm App che struttura un ambiente predisposto per applicazioni PWA.