ITS Running

Documentazione del progetto

Giorgio Garasto

Tecnologie Utilizzate

Interfaccia Web

L'interfaccia web è stata sviluppata utilizzando la metodologia PWA (Progressive Web App), ossia costruendo una Single Page Application che lavora esclusivamente lato client, che può funzionare (per quanto possibile) anche offline e che si comporta come un'applicazione nativa sui dispositivi su cui viene utilizzata. La web app utilizza il pattern PRPL (Push, Render, Pre-cache, Lazy-load), che permette di servire rapidamente i contenuti strettamente necessari al client per effettuare il rendering della route iniziale, per poi richiedere i file necessari alle altre route dopo aver effettuato il primo rendering, in modo da rendere il più rapido possibile sia il rendering della route iniziale, sia il passaggio alle altre route.
Non è stato utilizzato alcun framework per lo sviluppo della web app, ma sono stati utilizzati i componenti web, in modo da mantenere l'applicativo il più leggero possibile. Per effettuare il rendering delle parti dinamiche dell'applicativo, è stata utilizzata la libreria *lit-html* (o meglio, un wrapper di essa all'interno di un componente web chiamato *lit-element*). Lo stato dell'applicazione viene gestito per mezzo di Redux, che permette di separare lo stato dell'applicazione dalla sua presentazione.
Per comodità, è stato scelto di utilizzare una versione di JavaScript molto moderna anche lato client (ES6, con alcune feature di ES7/8). Dato che molte delle feature utilizzate non sono compatibili con browser più vecchi (const/let, moduli, operatori rest/spread, variabili di default, classi…), è stato deciso di utilizzare la CLI di Polymer, che, per mezzo di Babel, permette di creare build differenti in base ai browser target (bundle transpilato ad ES5, bundle transpilato ad ES6 e versione minimizzata che utilizza i moduli, senza quindi fare il bundle dei file JavaScript). Per poter sfruttare al massimo le potenzialità del JavaScript moderno, l'applicazione viene servita utilizzando PRPL Server, un server HTTP per Node che permette di effettuare un serving differenziale in base alle capacità del client. Per le ultime versioni dei browser viene servita la versione con i moduli, per i browser che ancora non lo supportano viene servito il bundle ES6, per i browser più vecchi viene servito il bundle ES5, ed infine per i crawler viene servita una versione dell'applicazione renderizzata lato server.

API

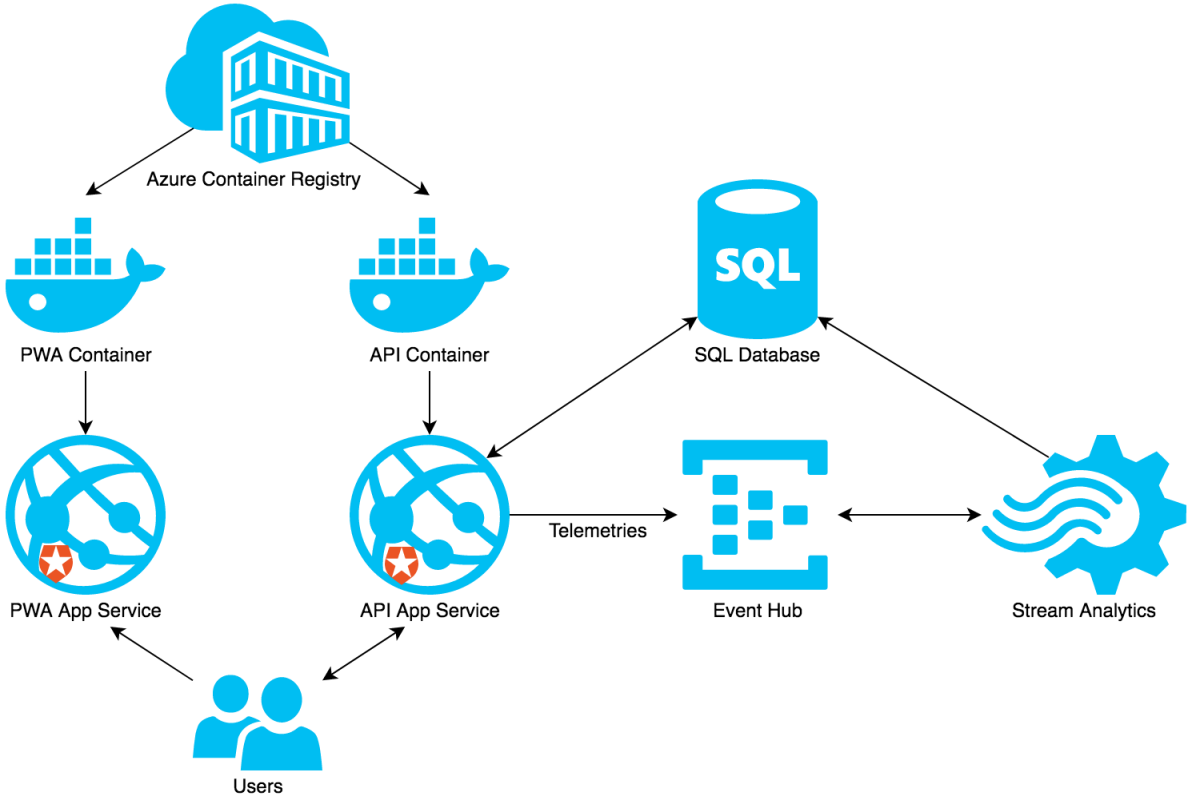
È stato scelto di creare delle API anziché di effettuare il rendering dell'applicativo lato server in modo da poter tenere la logica che gestisce i dati e quella che li presenta completamente separate. Le API sono state sviluppate secondo l'architettura REST (REpresentational State Transfer).

Deployment

Entrambi gli applicativi sono stati pensati in modo da poter essere containerizzati utilizzando Docker. Uno script npm si occuperà di effettuare il deploy dei container Docker su un registry privato su Azure. Ogni volta che viene effettuato il push di una nuova versione di un container Docker, la CI (Continuous Integration) di Azure effettuerà in automatico il deployment di esso sul relativo App Service. è stato scelto di tenere web app ed API su due applicativi separati per poterli scalare in modo indipendente. Gli App Service su Azure sono configurati in modo da accettare soltanto connessioni HTTPS che utilizzino almeno TLS 1.2. Questo è stato deciso per assicurarsi che l'utente abbia il massimo livello di sicurezza possibile sulla nostra app. Per i client che lo supportano, l'App Service è anche configurato per servire i file utilizzando il protocollo HTTP/2.

Funzionamento

Schema



Descrizione

AAA (Authentication Authorization Accounting)

Tutto ciò che riguarda l'autenticazione e le autorizzazioni dell'utente è gestito da Auth0, una soluzione AaaS (Authentication as a Service) che permette di implementare registrazione e login degli utenti con molta facilità. Quando l'utente entra nel sito, viene controllato se è già loggato, se non lo è o se la sessione è scaduta utilizzando il token che gli viene assegnato al momento del login. Se la sessione è scaduta o l'utente non è loggato, viene effettuato immediatamente il redirect alla pagina di login su Auth0, che permette di entrare tramite Social Login o con Email e Password. Quando l'utente effettua il login, Auth0 effettua il redirect alla home del sito, passando nell'hash dell'URL l'ID del client, permettendoci così di salvarlo nel Local Storage per poterlo usare nelle chiamate alle API. Le chiamate delle API necessiteranno quindi del token dell'utente per poter funzionare correttamente. Se le API non ricevono il token tramite l'header Authorization, o se il token ricevuto non è valido, esse risponderanno con un errore 401 Unauthorized.

Allenamenti

Quando l'utente vuole iniziare un allenamento o una gara, prima di tutto viene richiesto l'accesso alla posizione e alla fotocamera dell'utente. Il consenso alla posizione sarà obbligatorio, mentre quello della fotocamera no, in quanto è necessaria solo se si vogliono scattare dei selfie durante l'allenamento. Il nome dell'allenamento e la sua località vengono impostati in automatico dall'applicativo in base a data, ora e posizione del dispositivo, ma possono essere modificati a piacimento dall'utente. Quando l'utente preme il pulsante per iniziare l'allenamento, viene inviata una richiesta POST alle API per la creazione di un nuovo allenamento. Se questa fallisce (utente o API offline), l'allenamento viene comunque iniziato. La richiesta viene memorizzata su IndexedDB per essere ritentata non appena possibile (quando l'utente torna online o durante il sync del Service Worker). Dopo aver creato l'allenamento, viene aperta una connessione tramite WebSocket tra client e API, sulla quale ogni secondo il client invierà la telemetria dell'utente (posizione, timestamp ed eventuale selfie). Anche in questo caso, se per qualche motivo il WebSocket non è disponibile, i dati vengono memorizzati su IndexedDB per poter poi essere reinviati. Lato API, quando l'utente invia una richiesta POST per la creazione di un allenamento, dopo alcuni controlli viene effettuata una query INSERT sul Database SQL che, appunto, creerà il nuovo allenamento con i dati inseriti dall'utente, per poi restituire all'utente il nuovo allenamento appena creato (ID compreso). Quando arrivano le telemetrie sul WebSocket, anziché inserirle direttamente nel DB (sarebbe un carico di lavoro decisamente troppo pesante), le API le mandano ad un Event Hub su Azure. Un job di Stream Analytics si occuperà poi di prendere i dati inseriti nell'Event Hub ed inserirli nel database. È stato scelto di utilizzare Stream Analytics in modo da potere, in futuro, elaborare statistiche in tempo reale sui dati dei corridori. La scelta di un database SQL è dovuta alla necessità di mettere in relazione tra loro i corridori, le attività, le gare e le telemetrie, oltre che al fatto che è meno costoso di un DB NoSQL.

Profilo Utente

Tutti i dati dell'utente (nome, cognome, username, foto, ecc…) sono salvati sul data store di Auth0. Sul nostro database abbiamo solo l'ID degli utenti, che ci permette di ricavare tutte le altre informazioni da Auth0 quando necessario. Le richieste di visualizzazione, modifica e cancellazione dei dati dell'utente alle nostra API vengono passate alle API di Auth0. Non è necessario passare l'ID dell'utente nelle varie chiamate alle API perché essendo l'utente autenticato e autorizzato, possiamo già ricavare questa informazione dal suo token d'accesso. Questo ci permette anche di evitare problemi di modifiche non autorizzate da parte di altri utenti.

Possibili Miglioramenti

- **Sostituire Auth0 con il servizio di Autenticazione/Autorizzazione di Azure**
  Anche se dal punto di vista pratico non cambia nulla, questo ci permetterebbe di avere la gestione della Web App completamente centralizzata
- **Sostituire Stream Analytics con una Function App**
  Nel caso in cui si decida che non è necessario fare l'analisi dei dati degli utenti in tempo reale, sarebbe meglio migrare ad una Function App per decrementare i costi ed avere un po' più di versatilità