2023

Relazione finale

CORSO UNIVERSITARIO REACT NATIVE ANNO 2022/2023

NICOLAS VACCARI VR436988 – SESSIONE FEBBRAIO 2023

Obbiettivo del progetto

Il progetto si pone come obbiettivo l'integrazione di controlli nativi in app per la riproduzione di video formativi, nascondendo i controlli inseriti all'interno della pagina di youtube.

L'introduzione di controlli nativi nell'app consente di agevolare un eventuale cambiamento di gestore del video (con una gestione privata dei video tramite self-hosting, oppure un gestore diverso come vimeo) senza stravolgere la visualizzazione grafica e la user experience, che in alcuni ambiti come quello ospedaliero dove spesso si ha a che fare con soggetti di età diversa e con poca praticità nell'uso di uno smartphone, anche un minimo cambiamento può provocare malumori e nel caso peggiore il completo abbandono dell'app da parte dell'utente finale.

Lo sviluppo di controlli personalizzati consente inoltre di amalgamare la grafica visiva dell'applicazione.

Una volta ottenuti l'elenco di video disponibili dal servizio backend, l'app deve:

- Presentare all'utente l'elenco dei video disponibili con la possibilità di selezionare il video da riprodurre
- Avviare o fermare la riproduzione del video selezionato
- Ascoltare o mutare il contenuto audio del video in riproduzione
- Sapere la durata totale del video e se in riproduzione, a che punto si trova
- La possibilità di scorrere il tempo del video avanti o indietro con un dito tramite uno slider
- La possibilità di scorrere il tempo del video avanti o indietro di dieci secondi tramite tasti
- Evidenziare all'interno della lista qual è il video in riproduzione mettendo l'icona "Play" in verde

Scelte progettuali effettuate

L'app viene fornita dal corso come parzialmente completa, con un sistema di autenticazione e di navigazione delle pagine, un interfacciamento verso un servizio backend e una libreria per potersi interfacciare con youtube *[2] e riprodurre i video.

Per non appesantire e stravolgere l'app, si è scelto di non aggiungere alcuna libreria esterna (come redux) e di utilizzare gli hooks già presenti in react per la gestione dello stato del video player.

E' stato effettuato un refactoring del codice, estraendo parti della vista in componenti più semplici contenuti all'interno di "views/components/MotivationalVideos", trasferendo con loro anche gli specifici CSS.

Il passaggio dei dati da componente padre a componente figlio viene effettuato tramite le props, e per la comunicazione da componente figlio a componente padre vengono utilizzate delle "props callback", ovvero delle funzioni richiamate dal componente figlio che vengono poi implementate e gestite dal componente padre (un esempio sono le righe 408-414 di MotivationalVideos.js).

```
/>
{/* Includi i controlli tramite bottoni interattivi */}
<VideoControls
  isPlaying={playing}
  isMuted={isMuted}
  onVideoMuteChange={newIsMuted ⇒ setIsMuted(newIsMuted)}
  onPlayingChange={newIsPlaying ⇒ setPlaying(newIsPlaying)}
  onVideoTimeChange={newTime ⇒
    changeVideoPlayerTimeIncrementally(newTime)
  }
/>
```

```
function VideoControls(props) {
  // Estrai le props per leggibilità
  const isPlaying = props.isPlaying;
  const isMuted = props.isMuted;

  return (
    <View style={styles.row}>
      <Pressable
        onPress={() ⇒ {
          // Comunica alla vista superiore di mutare / smutare il video.
          props.onVideoMuteChange(!isMuted);
        }}
```

Trattandosi di una vista che ha bisogno di rimanere sempre sincronizzata con lo stato del video player, si è optato per ridurre al massimo l'utilizzo di funzioni asincrone (dove possibile), così da garantire sincronia tra gli elementi di visualizzazione e gli elementi interni della libreria del video player.

Per le traduzioni è stata utilizzata la libreria già presente nell'app "react-i18next", a cui sono state aggiunte alcune voci in inglese e in italiano per la gestione degli errori del video player.

Si è scelto di mantenere i "console.log()" all'interno del progetto come forma di debug rapido per la correzione del progetto, consapevoli che prima della messa in produzione è consigliato rimuoverli *[3].

Sempre a scopo del progetto, si è optato per scrivere il codice in inglese (uniformandolo al resto dell'app) tenendo però dei chiari commenti in italiano.

Difficoltà incontrate durante lo sviluppo

All'inizio del progetto sono state riscontrate delle difficoltà nel configurare l'app a causa di alcune informazioni poco chiare, e a questo proposito ringrazio il Dott. Cristian Turetta per le celeri risposte via e-mail nonostante gli orari poco consoni. Lo scambio di messaggi ha poi portato alla creazione di una mia Pull Request sul github del progetto originale *[1] con l'obbiettivo di avvantaggiare l'esecuzione da parte di altri studenti e di ridurre i dubbi su come approcciare il progetto.

Una volta presa confidenza con il codice dell'applicazione e consultato la documentazione ufficiale della libreria youtube *[2], è parso chiaro che la libreria non fosse in grado di fornire informazioni riguardo al tempo corrente (in secondi) della riproduzione del video. Per questo motivo, è stato creato un timer a ripetizione con un intervallo di 1s, che viene avviato quando si sceglie un video, e rigenerato ad ogni cambio di video. Questo timer (definito nel codice come "videoCurrentTimeInterval") ha lo scopo di interrogare ripetutamente la libreria youtube per ottenere il tempo corrente di visualizzazione del video, e aggiornare gli elementi UI per sincronizzare sia lo slider temporale sia il tempo in formato testuale.

Purtroppo, non essendoci possibilità di comunicare direttamente con la libreria tramite degli eventi di cambio del tempo, il cambiamento del tempo tramite slider può provocare dei piccoli artefatti grafici, che non influiscono però sul funzionamento del video player, e vengono corretti quasi immediatamente grazie alla sincronizzazione offerta dal timer.

L'uso del timer in questo caso non provoca problemi di performance, in quanto ne viene attivato solamente uno per tutta la riproduzione del video.

La sincronizzazione di tutti i vari elementi ha richiesto parecchi test e diverse revisioni del codice prima di riuscire a trovare un flusso che possa coprire la quasi totalità della combinazione di eventi del video player e input da parte dell'utente.

Si è inoltre cercato di mettere dei paletti protettivi all'interno del codice dove possibile, andando ad inserire controlli specifici sui possibili valori passati alle funzioni di gestione del video player (ad esempio i controlli sul tempo massimo e minimo percorribili durante il cambiamento del tempo corrente).

Possibili estensioni future

- Conversione dell'applicazione in typescript, per migliorare la qualità dello sviluppo attraverso l'uso di interfacce per descrivere gli oggetti javascript e la tipizzazione degli elementi di codice
- Effettuare una ricerca per individuare librerie per youtube migliori, in quanto quella attuale dà l'impressione di essere abbandonata (nel progetto github sono presenti issues aperte da mesi senza risposta dall'auto originale, un fenomeno tipico nell'open source)
- Aggiungere la modalità di riproduzione del video a schermo intero

- Inserire uno storico dei video visualizzati tracciando il dato nel backend (per avere il dato sincronizzato su tutti i dispositivi dell'utente) memorizzando inoltre il punto a cui si è arrivati a guardarlo (nel caso si esca dal video prima di averlo terminato), fornendo così metriche per capire l'interesse di ogni video da parte degli utenti finali
- Inserire la possibilità avere dei video preferiti, salvati in una "playlist" dedicata all'utente, che potrà rivedere in qualsiasi momento
- Inserire dei video raccomandati in base ai video preferiti dell'utente

Referenze

*[1]: https://github.com/CristianTuretta/REACT-Native-Course/pull/1

*[2]: https://lonelycpp.github.io/react-native-youtube-iframe/

*[3]: https://reactnative.dev/docs/performance/#using-consolelog-statements