UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di laurea triennale in Ingegneria Informatica e dell'Automazione

Progettazione e sviluppo di una web app per la sonorizzazione di itinerari geografici

Engineering and development of a web app for the soundtracking of geographical itineraries

Relatore:

Prof. **Emanuele Storti**

Tesi di laurea di:

Matteo Sonaglioni

Anno Accademico 2023/2024

INDICE

INDICE DELLE FIGURE

1. INTRODUZIONE

La presente tesi è basata su quanto svolto durante il tirocinio presso l'Università Politecnica delle Marche per un progetto di ricerca sull'approfondimento della gestione dei suoni a partire da luoghi specifici e dal loro contesto attraverso la realizzazione di una web app **[1]**. L'obiettivo principale dell'applicazione è la sonorizzazione di itinerari geografici, con funzionalità di generazione di suoni a partire dalla posizione e dalle caratteristiche del tragitto.

I fini ultimi dell'applicazione potrebbero includere molte più realtà di quelle che si sono scelte di rappresentare ed espandersi in altrettanti ambiti, poiché il trattamento dei dati recuperati dalle coordinate geografiche può essere ampliato ad una moltitudine di situazioni, sia quotidiane che no. L'idea è nata da un artista che vuole utilizzare un'applicazione per eseguire delle performance live, trasformando i dati processati dalle coordinate in suoni.

Di fronte alle varietà di possibili casi, si è deciso di partire con la sperimentazione dell'applicazione facendo riferimento a qualche esempio, che possano riassumere chiaramente come possono essere interpretate le caratteristiche geografiche di un tragitto.

Nel primo capitolo, dunque, oltre a delineare l'ambito e l'obiettivo dell'applicazione attraverso un'analisi, verrà presentato uno spunto iniziale intuitivo da cui partire che sarà poi elaborato nella fase di progettazione. Saranno quindi identificati e discussi i requisiti che la web app dovrà supportare per garantire una base solida su cui lavorare in futuro.

Nel secondo, si discuterà della fase di progettazione per la realizzazione di un'applicazione robusta e funzionale. Verranno esaminati nel dettaglio i programmi ed i servizi utilizzati per lo sviluppo dell'applicazione, così come le librerie ed i framework selezionati per implementare le varie funzionalità. Inoltre, verranno fatte delle considerazioni relative al caricamento e al recupero dei dati online, specialmente considerando le limitazioni delle API utilizzate in fase di sviluppo.

Poi, il terzo capitolo si concentrerà su un'analisi dettagliata delle varie funzionalità che l'applicazione dovrà supportare e sul percorso scelto per la loro implementazione. Saranno esaminati esempi di codice per illustrare come diverse parti del progetto collaborino tra loro. Ogni funzionalità sarà esplorata nel contesto del suo contributo alla visione complessiva dell'applicazione.

Infine, verranno tratte delle conclusioni basate sull'esperienza dello sviluppo di questo progetto. Saranno discussi i successi raggiunti, le sfide incontrate e le opportunità future per l'applicazione. Saranno inoltre presentati screenshot della web app durante il suo funzionamento, per mostrare agli utenti il risultato finale del lavoro.

2. ANALISI

2.1. Punto di partenza

Inizialmente, ci si è concentrati su alcuni esempi chiave che potessero illustrare in modo chiaro come le caratteristiche del percorso potessero essere interpretate e tradotte in suoni, ponendo particolare attenzione alla versatilità dell'applicazione, poiché i dati geografici possono essere sfruttati in molteplici contesti. Tutto ciò ha guidato nella definizione delle funzionalità iniziali e dei requisiti di base del progetto. Dapprima, la funzionalità primaria era di generare dei suoni specifici per particolari tratti di itinerari: ad esempio un percorso per lo più marittimo potrebbe creare dei suoni inerenti alle onde del mare o, in generale, rilassanti; al contrario, un percorso in città avrebbe potuto creare dei suoni più caotici e veloci. Quindi, l'approccio principale è quello di associare un suono ad un luogo in base al suo contesto, analizzandone i vari elementi caratteristici ricavati dall'ambiente circostante.

Il progetto parte con lo sviluppo del back-end della web app, al fine di creare una solida infrastruttura di base in grado di offrire servizi fondamentali. Questo approccio consentirebbe agli utenti e agli studenti futuri di approfondire ulteriormente il progetto, sfruttando i servizi esistenti come punto di partenza per la creazione di nuove funzionalità o per l'estensione dell'applicazione. Alcune di queste possono includere l'elaborazione dei dati geografici, la gestione degli utenti, l'accesso ai dati archiviati e così via. L'obiettivo di questa strategia è fornire una piattaforma stabile e scalabile su cui gli utenti potessero lavorare per costruire ed ampliarla facilmente, evitando così di dover partire da zero ogni volta che si desidera aggiungere nuove funzionalità o espandere l'applicazione. Di conseguenza, l'architettura dovrà essere il più possibile modulare ed estensibile, in modo da facilitare l'integrazione di aggiornamenti futuri senza dover ristrutturare l'intero sistema, promuovendo una maggiore flessibilità e adattabilità nel tempo. In questo modo, si favorirebbe anche la collaborazione e lo sviluppo continuo da parte di una comunità più ampia di utenti e sviluppatori.

Partendo da questo primo obiettivo, si è pensato di sviluppare delle funzionalità che permettono di creare un itinerario attraverso l'utilizzo di una mappa, utilizzando le coordinate geografiche come primo dato su cui lavorare per ricavarne degli altri. Questo contesto fornisce una base solida per la progettazione e lo sviluppo dell'applicazione come una web app, avendo la necessità di utilizzare delle interfacce grafiche per sfruttare appieno le funzionalità ed analizzare efficacemente i dati durante la progettazione. Poiché l'applicazione mira ad offrire una esperienza interattiva agli utenti, è stato ritenuto fondamentale integrare un front-end semplice che consentisse di interfacciarsi facilmente con i dati geografici e le funzionalità dell'applicazione stessa, oltre che per rendere l'applicazione disponibile da qualsiasi dispositivo con accesso ad Internet e capacità GPS. Inoltre, un altro vantaggio potrebbe essere quello che gli utenti potranno utilizzare l'applicazione sia privatamente che in maniera condivisa.

In sintesi, lo sviluppo di una web app che integra sia il back-end che il front-end è stato motivato dalla necessità di fornire un'esperienza interattiva agli utenti, semplificare l'analisi dei dati durante lo sviluppo e garantire una maggiore accessibilità e portata dell'applicazione stessa.

2.2. Requisiti

I requisiti dell'applicazione si suddividono in diverse categorie che coprono le funzionalità principali ed i vincoli di sistema. In primo luogo, è necessario che l'applicazione consenta agli utenti di creare e visualizzare itinerari geografici attraverso l'utilizzo di una mappa interattiva. Questo richiede una prima interfaccia utente intuitiva che supporti funzionalità come la creazione di percorsi, l'aggiunta di punti di interesse ed alcune caratteristiche del tragitto o di coordinate specifiche. A questi scopi, si andranno ad integrare le seguenti operazioni:

- la possibilità di poter navigare liberamente in una mappa;
- la creazione di nuovi percorsi all'interno della mappa;
- l'aggiunta di punti di interesse, come ad esempio dei marker che, a mano a mano che vengano aggiunti, creeranno un itinerario;
- il salvataggio dell'itinerario creato per poter analizzare i relativi dati;
- la visualizzazione di una lista dei marker inseriti all'interno della mappa;
- la ricerca di luoghi o città specifici all'interno della mappa;
- la possibilità di trovare la posizione corrente del dispositivo utente;
- la possibilità di mostrare all'interno della mappa tutti gli itinerari che sono stati salvati, con le relative indicazioni stradali, la lunghezza del percorso ed il tempo che si impiega per effettuarlo;
- la possibilità di ricavare l'altitudine di una coordinata specifica;
- la possibilità di calcolare la distanza (in linea d'aria) tra due coordinate;
- la possibilità di ricercare la città più vicina partendo da una coordinata specifica.

In secondo luogo, l'applicazione deve supportare la gestione dei file audio, consentendo agli utenti di caricare ed organizzare le proprie tracce audio da utilizzare nella sonorizzazione degli itinerari. Questo richiede funzionalità per il caricamento dei file audio, la creazione di playlist personalizzate e la possibilità di associare specifiche tracce audio a determinati itinerari. Questa seconda interfaccia, quindi, dovrà includere le seguenti funzionalità:

- la possibilità di poter caricare e salvare un file audio direttamente dal dispositivo, scegliendo il genere più adatto ad esso all'interno di una lista predefinita;
- la visualizzazione della lista degli audio salvati con relativi attributi e metodi;

- la possibilità di poter ricercare e salvare un luogo o città specifico;

- la possibilità di poter associare un audio ad uno di questi luoghi o città.

Inoltre, l'interpretazione dei dati geografici a partire da un itinerario potrà essere visualizzata in una terza interfaccia, che consente di scegliere un percorso tra quelli salvati ed analizzare delle statistiche relative alla propria conformazione geografica e geofisica, ricavandone poi un profilo che verrà utilizzato per l'interpretazione e generazione di suoni. Quindi, selezionando uno degli itinerari, verranno mostrati a schermo le seguenti caratteristiche e funzionalità:

- un grafico che mostri l'andamento dell'altitudine del percorso;

- un grafico che mostri, in percentuali, le caratteristiche geofisiche del percorso analizzando un intorno predefinito, come la copertura del suolo, ovvero il materiale fisico sulla superficie della Terra, e l'utilizzo del suolo, ovvero la gestione e la modifica dell'ambiente naturale in ambiente edificato dall'uomo;

- un grafico che mostri una lista di luoghi o città vicine all'itinerario entro un certo range (in km);

- la generazione di note musicali attraverso l'interpretazione dell'altitudine del percorso e le relative caratteristiche principali;

- la generazione di una playlist di audio casuale attraverso l'interpretazione del profilo del terreno circostante all'itinerario;

- una lista degli audio associati ai luoghi o città più vicine all'itinerario.

Infine, un'ultima interfaccia dovrà utilizzare tutti i dati ricavati ed interpretati dall'itinerario per effettuare una vera e propria simulazione di un ipotetico utente che viaggia all'interno del percorso, partendo dalla posizione iniziale ed arrivando a quella finale, mostrando a schermo le peculiarità e caratteristiche a mano a mano che si procede lungo il tragitto.

3. PROGETTAZIONE

3.1.Strumenti utilizzati

Per implementare il progetto, è stato necessario integrare diversi strumenti e tecnologie di natura diversa in maniera graduale rispetto allo sviluppo. Questo è comune durante la realizzazione di progetti complessi, in quanto ogni fase della progettazione potrebbe richiedere l'utilizzo di strumenti specifici per soddisfare i requisiti dell'applicazione.

3.1.1. Visual Studio Code

Per lo sviluppo della web app è stato scelto Visual Studio Code **[2]**, un editor di codice sorgente sviluppato da Microsoft per Windows, Linux e macOS. Esso include:

- il supporto per *debugging*, per l'individuazione e correzione da parte del programmatore di uno o più errori rivelati nel software, direttamente in fase di programmazione oppure a seguito della fase di testing;
- un controllo per *GitHub* integrato;
- *syntax highlighting*, una colorazione della sintassi;
- *IntelliSense*, una forma di completamento automatico per visualizzare la descrizione delle funzioni, variabili e dei metodi usando metadati e reflection;
- *snippet*, un frammento o esempi di codice sorgente;
- *refactoring del codice*, una tecnica strutturata per modificare la struttura interna di porzioni di codice senza modificarne il comportamento esterno;

È un software libero e gratuito per uso personale e commerciale. Visual Studio Code è basato su Electron, un framework con cui è possibile sviluppare applicazione Node.js. Può essere utilizzato con vari linguaggi di programmazione, tra cui la famiglia di linguaggi C, HTML, PHP, Java e molti altri. La finestra di comando è un'interfaccia a riga di comando ed è un software che può essere ampliato attraverso dei plugin, disponibili da una repository centrale. Essi includono ampliamenti all'editor e supporto ai linguaggi. **[3]**

I vantaggi nell'utilizzare questo editor per creare una web app sono: la semplicità con cui si possono integrare servizi cloud, come ad esempio Firebase o Github; il debugger integrato per ricercare eventuali bug; il supporto a più linguaggi di programmazione; una community attiva e una vasta documentazione online che aiuta agli sviluppatori a risolvere problemi e ad imparare nuove tecniche di sviluppo.

3.1.2. React.js

React.js (React) **[4]** è una libreria open-source, front-end, JavaScript per la creazione di interfacce utente. È mantenuto da Meta e da una comunità di singoli sviluppatori ed aziende. React può essere utilizzato come base nello sviluppo di applicazioni a pagina singola, ma è utilizzabile anche su mobile tramite React Native, una libreria che tramuta i componenti React in componenti nativi (iOS e Android). Tuttavia, React si occupa solo del rendering dei dati sul DOM, pertanto la creazione di applicazioni React richiede

generalmente l'uso di librerie aggiuntive per lo state management ed il routing. Infatti, il codice di React è costituito da entità denominate componenti, che possono essere sottoposti a rendering utilizzando la libreria React DOM. Quando si esegue il rendering, si possono passare valori noti come "props". Questi componenti possono essere funzionali, ovvero dichiarati con una funzione che quindi restituisce alcuni JSX, o basati su classi, anche chiamati "stateful", perché il loro stato può contenere valori in tutto il componente e può essere passato ai componenti figlio tramite props.

Un'altra caratteristica notevole è l'uso di un Document Object Model virtuale: React crea una cache della struttura dati in-memory, calcola le differenze risultanti ed aggiorna in modo efficiente il DOM visualizzato dal browser. Ciò consente al programmatore di scrivere codice come se l'intera pagina fosse renderizzata su ogni modifica. Per quanto riguarda JSX, o Javascript XML, è un'estensione della sintassi simile nell'aspetto all'HTML, fornendo un modo per strutturare il rendering dei componenti usando una sintassi familiare a molti sviluppatori. **[5]** Tutte queste caratteristiche hanno portato a scegliere React.js per la progettazione dell'applicazione poiché rende molto efficiente ed efficace la creazione della struttura dell'app e la correzione del codice durante il processo, rendendola più modulare, performante, prevedibile e facile da mantenere nel tempo.

3.1.3. HTML, CSS, JavaScript

L'HyperText Markup Language (HTML) **[6]** è il linguaggio di marcatura più usato per i documenti web ed è nato per la formattazione ed impaginazione di documenti ipertestuali e per il disaccoppiamento della struttura logica di una pagina web e la sua rappresentazione, gestita tramite gli stili CSS. È un linguaggio di pubblico dominio, la cui sintassi è stabilita dal World Wide Web Consortium. Quindi, descrive la visualizzazione grafica, o layout, del contenuto testuale e non, di una pagina web attraverso tag di formattazione. Non è un linguaggio di programmazione, non prevedendo alcuna definizione di variabili, strutture dati, funzioni o strutture di controllo: il suo codice è in grado soltanto di strutturare e decorare dati testuali. I documenti HTML vengono immagazzinati sui dischi rigidi di macchine elaboratrici, ovvero computer-server, costantemente collegate e connesse alla rete Internet. Il componente principale della sintassi è l'elemento, ognuno racchiuso all'interno di marcature dette tag. Quando il tag deve essere applicato ad una sezione di testo o di codice, l'ambito di applicazione deve essere delimitato fra un tag di apertura ed uno di chiusura. **[7]**

Per quanto riguarda il Cascading Style Sheet **[8]**, è un linguaggio usato per definire la formattazione di documenti HTML, come appunto siti e pagine web. L'introduzione del CSS si è resa necessaria per separare i contenuti delle pagine HTML dalla loro formattazione o layout, permettendo una programmazione più chiara e facile da utilizzare, sia per gli autori delle pagine stesse sia per gli utenti, garantendo contemporaneamente il riutilizzo del codice ed una sua più facile manutenzione.

Infine, per la parte logica dell'applicazione si è scelto di utilizzare JavaScript, un linguaggio di programmazione multi paradigma orientato agli eventi, utilizzandolo sia nel lato client web che lato server (Node.js **[9]**) per la creazione di RESTful API, applicazioni desktop e embedded, siti web ed applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi, innescati a loro volta in vari modi dall'utente sulla pagina web in uso, ad esempio attraverso mouse, tastiera, caricamenti della pagina, etc.

È stato formalizzato con una sintassi più vicina al linguaggio Java e le funzioni di script possono essere opportunamente inserite in file HTML o in appositi file separati con estensione ".js", poi richiamati nella logica di business, ovvero il nucleo di elaborazione che rende operativa l'applicazione. **[10]**

Le caratteristiche principali di JavaScript sono:

- essere un linguaggio interpretato: il codice non viene compilato, ma eseguito direttamente;

- la sintassi è relativamente simile a quella di altri linguaggi come C, C++ e Java;

- è un linguaggio debolmente tipizzato;

- è un linguaggio debolmente orientato agli oggetti;

- può usare caratteri Unicode;

- definisce le funzionalità tipiche dei linguaggi di programmazione ad alto livello, come strutture di controllo e cicli.

JavaScript, quindi, viene utilizzato soprattutto come linguaggio di scripting integrato, ovvero all'interno di altro codice. L'idea di base è che il programma ospite fornisca allo script un'API ben definita, che consente l'accesso ad operazioni specifiche, la cui implementazione è a carico del programma ospite spesso. Lo script quindi, quando eseguito, utilizza riferimenti a questa API per richiedere l'esecuzione di operazioni specifiche, non previsti dai costrutti del linguaggio in sé.

L'integrazione di questi tre linguaggi è un aspetto fondamentale nello sviluppo di un'applicazione web e, grazie alla loro complementarità, rendono il processo di sviluppo più semplice, grazie ad una sintassi familiare ed intuitiva, separando le responsabilità nelle diverse strutture apposite, ed essendo facili da integrare l'una con l'altra. Inoltre, le numerose librerie, framework e la vasta documentazione di eventuali problemi risolti da altri utenti online migliorano di gran lunga la produttività e l'efficienza nella progettazione.

3.1.4. OpenStreetMap

OpenStreetMap (OSM) **[11]** è un progetto collaborativo finalizzato a creare mappe del mondo a contenuto libero. Il progetto punta ad una raccolta mondiale di dati geografici, con scopo principale la creazione di mappe e cartografie. I dati geografici presenti in OSM vengono distribuiti con una licenza libera, utilizzandoli liberamente per qualsiasi scopo, anche commerciale, con il solo vincolo di citare la fonte. Tutti possono contribuire arricchendo o raccogliendo i dati. Le mappe sono create usando come riferimento i dati registrati da dispositivi GPS portatili, fotografie aeree ed altre fonti libere. I rilevi sul territorio vengono effettuati da volontari a piedi, in bici, auto, treno o in qualsiasi altro mezzo di trasporto, usando un'unità GPS. Inoltre, alcune agenzie di governo hanno fornito i propri dati ufficiali con licenze appropriate per l'importazione in OpenStreetMap. Questi dati vengono "posizionati" in modo da evidenziare punti di interesse, strade, edifici, attività commerciali, corsi d'acqua, caratteristiche geografiche e così via: ogni collaboratore può etichettare ciò che è presente sulla cartina, descrivendo

con precisione qualunque elemento che caratterizza un determinato luogo. Trattandosi di un'esperienza collaborativa, inoltre, questa soluzione ha permesso agli utenti di mappare anche aree remote, come le zone rurali, dove i classici sistemi di mappatura potrebbero fare fatica ad arrivare o semplicemente potrebbero non avere interesse nella definizione di tali luoghi.

3.1.5. Firebase

Firebase **[12]** è una piattaforma per la creazione di applicazioni per dispositivi mobili e web sviluppata da Google. Permette di salvare e sincronizzare i dati elaborati da applicazioni in un database NoSQL, ad alta disponibilità ed integrabile in tempi rapidissimi in altri progetti software, semplicemente sottoscrivendo un account al servizio. Rientra nella categoria di back-end che tramite API, servizi quali autenticazione, storage dei dati, push notification, comunicazione tra utenti, permette ai dispositivi di interagire con servizi remoti creando reti "social". **[13]**

Le sue caratteristiche peculiari sono:

- la capacità di sincronizzazione dei dati, essendo in grado di aggiornare i dati istantaneamente, sia se integrato in app web che mobile;
- la disponibilità di librerie client per integrare Firebase in ogni app;
- le API REST rendono disponibili le funzionalità di Firebase per ogni tecnologia per cui non esistano librerie apposite o in caso di operazioni non contemplate in esse;
- la sicurezza, poiché i dati immagazzinati in Firebase sono replicati e sottoposti a backup continuamente.

Firebase fornisce un database in tempo reale, chiamato Realtime Database, che permette di memorizzare e sincronizzare i dati in tempo reale tra i client ed il server. È quindi possibile salvare i dati ottenuti da OSM in una struttura dati ben organizzata e personalizzata e modellarli in seguito in base alle esigenze specifiche dell'applicazione. Inoltre, Firebase Storage offre uno spazio di archiviazione per gestire file di grandi dimensioni, come i poligoni delle città, riducendo al minimo il carico su Realtime Database e garantendo tempi di risposta più veloci. Date le limitazioni dello storage in termini di quantità massima di chiamate o dimensioni giornaliere, è possibile che i file di grandi dimensioni, come gli audio, non siano stati salvati direttamente nello storage di Firebase. Questo perché i file audio possono facilmente superare le dimensioni consentite dalle quote giornaliere.

3.1.6. Google Drive

Google Drive **[14]** è un servizio web, in ambiente cloud computing, di memorizzazione e sincronizzazione online introdotto da Google. È basato su un software open source, comprendendo file hosting, file sharing e la modifica collaborativa di documenti. Alcune delle sue peculiarità più vantaggiose sono:

- ampio spazio di archiviazione, gratuito fino a 15 GB;
- accesso da qualsiasi dispositivo;

- sincronizzazione automatica;
- collaborazione in tempo reale;
- facile condivisione di file;
- backup automatico.

Supporta la visualizzazione in anteprima di molti formati, come immagini, video, file di testo, audio, Word, Excel, e così via. Utilizzando la console cloud di Google, è possibile utilizzare le API di Google Drive collegando direttamente un progetto utente all'applicazione tramite delle autorizzazioni. Così facendo, è possibile accedere ai dati di un Google Drive personale o condiviso, per creare nuovi file, effettuare il download, ricercare file o cartelle e così via. Avendo uno spazio di memorizzazione più grande e dei limiti giornalieri di chiamate API abbastanza elevati, si è preferito salvare i file audio utilizzati dall'applicazione qui, in modo da non avere un sovraccarico durante la fase di sviluppo e di testing. **[15]**

In sintesi, Google Drive è un servizio di archiviazione cloud versatile ed affidabile, una soluzione ideale per memorizzare, gestire e condividere i file di una web app in maniera efficace e sicura.

3.1.7. Express & Node.js

Express **[16]** è un framework open source per applicazioni web per Node.js. È stato progettato per creare web app e API ed ormai definito il server framework standard per Node.js. È il framework back-end più popolare e fa parte dell'ecosistema JavaScript. Offre un sistema di routing e funzioni semplificate, fornendo richieste e risposte HTTP attraverso uno strumento di interfaccia a riga di comando chiamato Node Package Manager, da cui gli sviluppatori possono reperire i pacchetti sviluppati. Utilizza quindi un modello client server per accettare le richieste degli utenti ed inviare le risposte al client, simile ad altri framework popolari come Laravel. Quando un utente invia una richiesta dal proprio browser web, esso invia una richiesta HTTP all'applicazione/server. Il server riceve la richiesta attraverso uno dei suoi percorsi e la elabora utilizzando il controller, che corrisponde al percorso richiesto. Dopo l'elaborazione, il server invierà una risposta al client utilizzando il protocollo HTTP, poiché si tratta di un protocollo di comunicazione "back-and-forth". La risposta restituita al client può essere un testo standard, una pagina HTML dinamica, oppure dati JSON che gli sviluppatori del front-end gestiranno per visualizzare le informazioni sulla pagina web. Un semplice server Express.js ascolta le richieste, ad esempio, da uno specifico URL in arrivo su uno specifico numero di porta, restituendo una risposta adattata poi alle esigenze dell'utente. **[17]**

Node.js, invece, è un runtime system open source multipiattaforma orientato agli eventi, per l'esecuzione di codice JavaScript. Consente di utilizzare il codice anche per eseguirlo lato server, ad esempio per la produzione del contenuto delle pagine web dinamiche, prima che la pagina venga inviata al browser dell'utente. Quindi, rende possibile l'I/O asincrono, puntando ad ottimizzare scalabilità ed il throughput in applicazioni web a sistema real-time. Il modello di networking è quello dell'I/O event-drive: Node.js richiede al sistema operativo di ricevere notifiche al verificarsi di determinati eventi, e rimane

quindi in sleep fino alla notifica stessa; solo in tale momento torna attivo per eseguire le istruzioni previste nella funzione di callback, così chiamata perché da eseguire una volta ricevuta la notifica che il risultato dell'elaborazione del sistema operativo è disponibile. Questo sistema è ritenuto più efficiente nelle situazioni critiche in cui si verifica un elevato traffico di rete. In sintesi, un server Express potrà eseguire delle richieste direttamente al Google Drive collegato all'applicazione, permettendo la gestione dei file audio in maniera rapida ed esaustiva. **[18]**

3.1.8. GitHub

GitHub **[19]** è una piattaforma di sviluppo collaborativo basata su Git, un sistema di controllo versione distribuito. Consente agli sviluppatori di lavorare insieme su progetti software, tenere traccia delle modifiche apportate al codice sorgente e gestire il processo di sviluppo del software in modo efficiente. Su GitHub, i progetti sono organizzati in repository, dove gli sviluppatori possono caricare, condividere e collaborare su codice sorgente, documentazione, problemi e richieste di pull. È ampiamente utilizzato nell'industria del software per la collaborazione open-source, lo sviluppo di progetti aziendali e la condivisione di codice con la comunità globale degli sviluppatori. Consente inoltre di tornare a versioni precedenti se necessario, assumendo così anche il compito di effettuare eventuali backup.

3.2. Progettazione software

L'applicazione è basata su un modello client-server, dove il client si riferisce alla parte di front-end ed il server alla parte di back-end. Nel caso specifico, il back-end è implementato utilizzando Node.js con il framework Express ed implementando richieste a servizi online. Si occupa delle operazioni di elaborazione dei dati e dell'accesso alle risorse esterne di Google Drive, utilizzando un server Express. In questo modo l'applicazione può effettuare chiamate verso il cloud per accedere ai file audio memorizzati: il back-end esegue le richieste del client, recupera i dati necessari e li elabora, per poi restituire le risposte al client. D'altra parte, il front-end è realizzato utilizzando le tecnologie HTML, CSS, JavaScript e React.js. Queste tecnologie permettono agli utenti di interagire con l'applicazione tramite interfacce intuitive. Gli utenti possono gestire, analizzare, creare, eliminare e modificare i dati all'interno dell'applicazione e dei database in modo semplice ed efficiente.

3.2.1. Schema iniziale

Uno schema iniziale è essenziale poiché fornisce una visione generale della struttura e dell'organizzazione dell'applicazione prima che venga effettivamente implementata. Un primo impatto permette anche di studiare l'organizzazione dei componenti, il flusso di dati, le interazioni utente e come dovrà essere sviluppata la navigazione all'interno della web app. In questo caso, è stata applicata una delle strategie classiche per lo sviluppo dello scheletro di uno schema concettuale, ovvero la strategia TOP-DOWN:

- fase 1: si considerano le specifiche globalmente, producendo uno schema iniziale completo ma con pochi concetti molto astratti;

- fase 2: si raffinano i concetti astratti fino ad arrivare allo schema concettuale completo in ogni dettaglio.

Questo processo permetterà di acquisire una visione chiara e completa del progetto, consentendo di pianificare efficacemente la struttura dell'applicazione in termini di organizzazione. Sarà fondamentale scegliere con attenzione la disposizione delle varie sottocartelle logiche, in modo da garantire una gestione efficiente ed intuitiva delle diverse parti del progetto. Infine, le varie componenti verranno collegate tramite la strategia BOTTOM-UP per fornire uno schema finale.

Lo schema iniziale è suddiviso tra front-end e back-end che, una volta inizializzata l'applicazione, comunicheranno tra loro:

- il primo utilizza delle interfacce come punto di ingresso dell'applicazione ed è quindi responsabile per l'interazione diretta con l'utente attraverso quattro entry-point principali:
    - Map, per interagire direttamente con la mappa e gli itinerari;
    - Statistics, per visualizzare le caratteristiche di uno specifico percorso ed analizzare i dati elaborati relativi agli audio;
    - Music, per la gestione dei file audio e dei poligoni di città o luoghi specifici;
    - Emulator, per osservare la dinamicità di alcune statistiche durante l'evoluzione del percorso.
- il back-end comunica con il client attraverso richieste HTTP ed API per:
    - inviare e ricevere dati da Google Drive utilizzando un server Express;
    - leggere e scrivere dati in Firebase storage e realtime database;
    - inviare e ricevere dati da servizi online.

Lo schema iniziale sarà quindi il seguente *(figura 1)*:

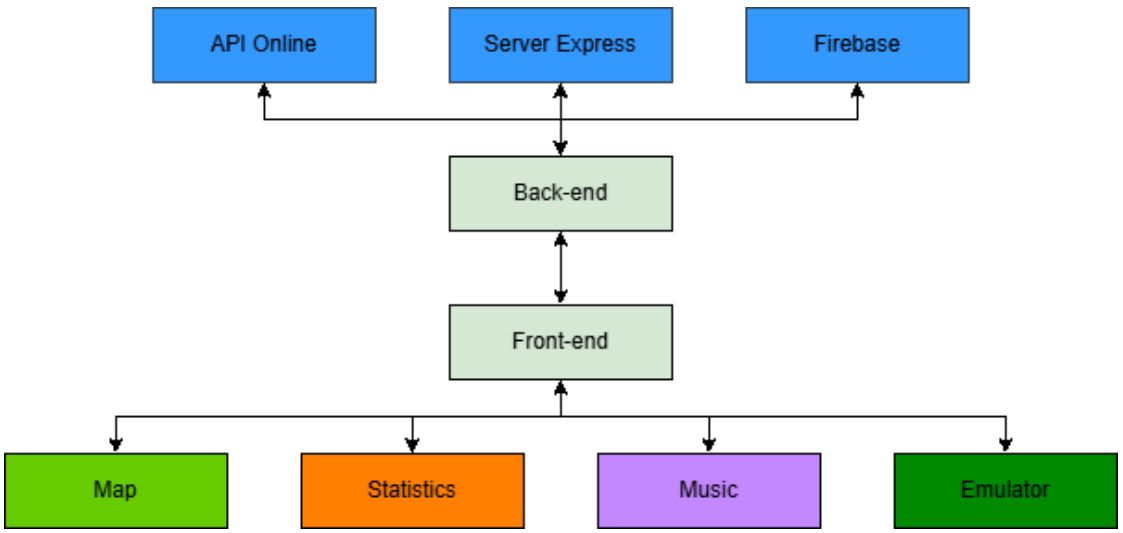


Figura 1: Schema a blocchi dell'architettura

3.2.2. Diagramma Map

Per quanto riguarda il diagramma del componente Map *(figura 2)*, è necessario partire dall'inserimento di una mappa online con cui l'utente potrà interagire per soddisfare i requisiti richiesti dall'applicazione. La struttura verrà suddivisa in quattro sottocategorie:

- Maps: per l'implementazione della mappa e poterci navigare liberamente, per l'aggiunta di punti di interesse e per la creazione del routing tra due o più luoghi;
- Searchbox: per la ricerca, attraverso l'utilizzo di API online, di luoghi o città specifici all'interno della mappa;
- Controller Action: per permettere all'utente di utilizzare tutte le funzionalità implementate in Lista Action:
    - Aggiungi un marker: aggiungere un punto di interesse nella mappa;
    - Salva dati itinerario: salvare nel database le caratteristiche del tragitto;
    - Lista itinerari: visualizzare tutti gli itinerari salvati nel database;
    - Ricevere l'altitudine: ricavare l'altitudine da una coordinata specifica;
    - Calcolare la distanza: calcolare la distanza tra due luoghi specifici;
    - Poligono più vicino: ricercare la città più vicina da una coordinata specifica.
- Controller Marker: per permettere all'utente di gestire i luoghi di interesse salvati provvisoriamente all'interno della mappa.

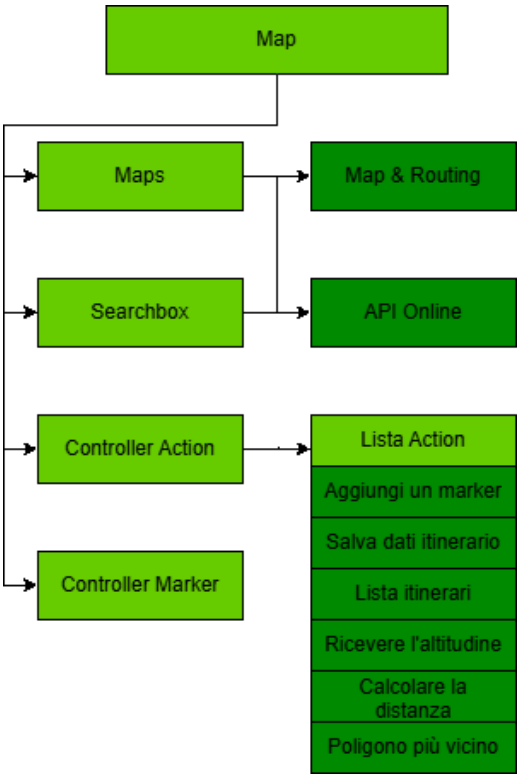


Figura 2: Diagramma del componente Map

3.2.3. Diagramma Statistics

Per il diagramma del componente Statistics *(figura 3)*, è necessario implementare i requisiti per la visualizzazione e per la creazione di statistiche relative alle caratteristiche geofisiche di un itinerario specifico. La struttura è basata sulla suddivisione delle statistiche in tre principali macrocategorie:

- Itineraries: per la visualizzazione degli itinerari ed il download delle statistiche attraverso il richiamo di API online;
- Stats: per la visualizzazione di grafici che descrivono le caratteristiche del tragitto, dopo aver effettuato il download, che sono suddivisi in:
    - Grafico altimetrico: per la visualizzazione dell'andamento dell'altitudine del percorso;
    - Grafico del terreno: per la visualizzazione delle caratteristiche geofisiche del percorso;
    - Grafico dei poligoni: per la visualizzazione degli audio associati a luoghi e/o città vicine all'itinerario.
- Sounds: per la creazione di suoni a partire dai dati ricavati dopo aver effettuato il download delle caratteristiche del tragitto, ognuna per ciascuna delle tipologie dei dati trattati:
    - Suoni dell'altitudine: si occupa della generazione dei suoni partendo dal profilo altimetrico dell'itinerario, attraverso l'utilizzo di:
        - Lista delle note: una struttura opportuna per la conversione tra altezza e note musicali;
        - Funzioni: funzionalità che permettono di trattare le note musicali per la creazione di una melodia e per la visualizzazione delle caratteristiche principali di quest'ultima;
    - Suoni del terreno: partendo dalle caratteristiche del terreno circostante, entro un certo raggio, verranno associati dei generi musicali a delle categorie specifiche, attraverso una opportuna struttura (Generi);
    - Audio dei poligoni: permette di visualizzare, entro una certa distanza dall'itinerario, una lista dei poligoni più vicini ad esso ed i relativi file audio associati.

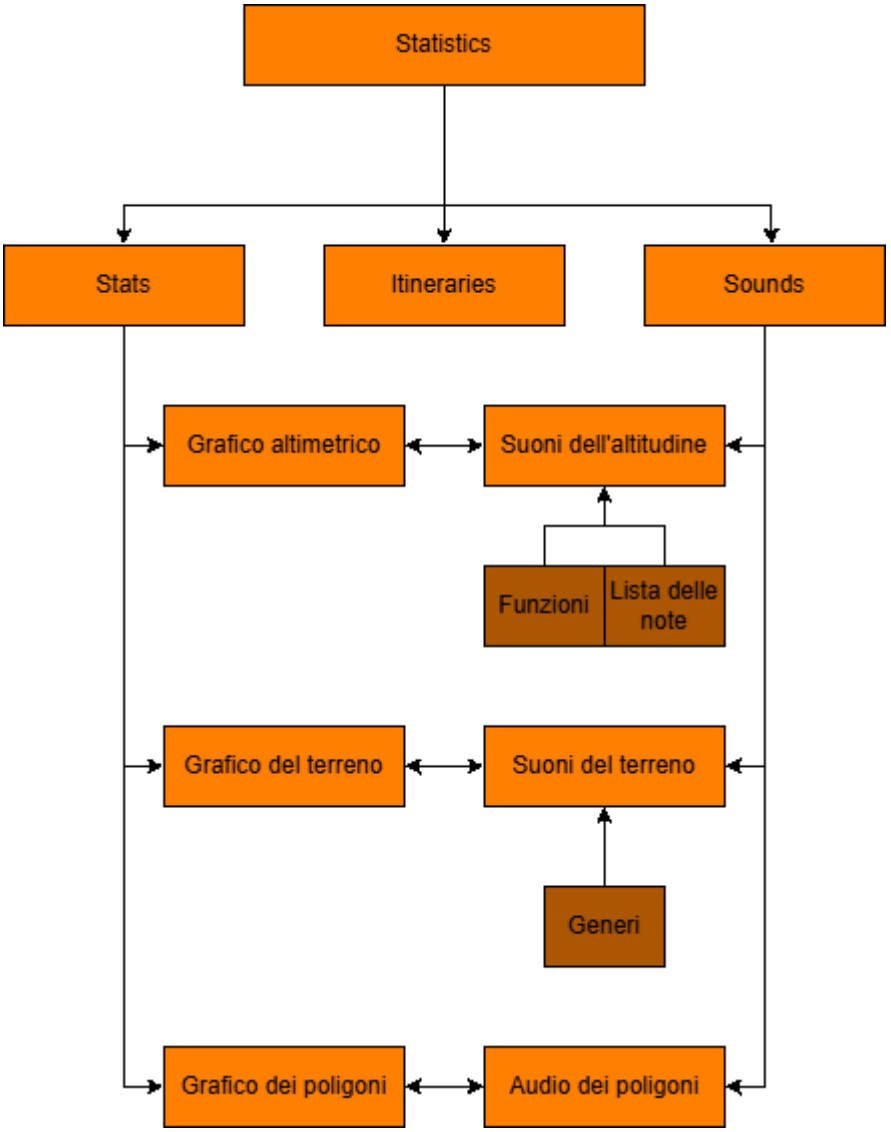


Figura 3: Diagramma del componente Statistics

3.2.4. Diagramma Music

Il diagramma del componente Music *(figura 4)*, descrive come l'utente potrà interagire con l'applicazione per la gestione dei file audio e dei poligoni. La struttura è suddivisa in due:

- Songs: permette all'utente di gestire i file audio e le relative informazioni salvati nel database:
    - Salva audio: attraverso il caricamento di un file audio con estensione appropriata direttamente dal proprio computer ed assegnandogli un genere, l'utente potrà caricare il tutto nel database;
    - Tabella audio: visualizza la lista dei file audio e le relative informazioni in una tabella.
- Polygons: permette all'utente la gestione dei poligoni associati a città o paesi salvati

nel database:

- Salva poligono: attraverso la ricerca e l'utilizzo di API online, l'utente potrà salvare le caratteristiche di un poligono nel database;
- Audio dei poligoni: permette di visualizzare la lista dei poligoni salvati ed associare a questi ultimi uno o più file audio.

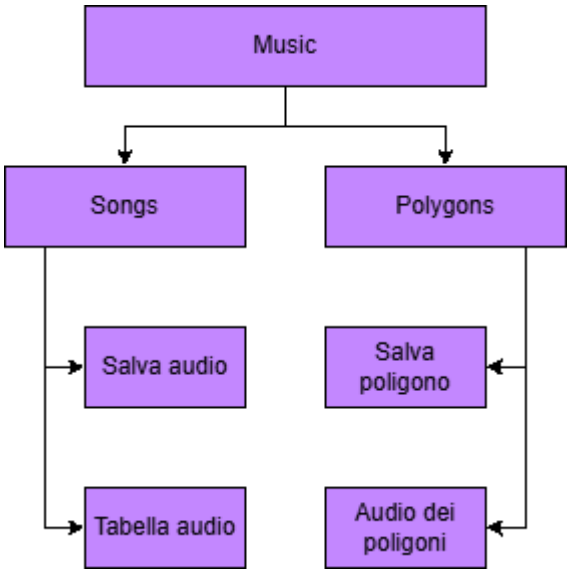


Figura 4: Diagramma del componente Music

3.2.5. Diagramma Emulator

Per quanto riguarda il diagramma del componente Emulator *(figura 5)*, bisognerà usufruire delle funzionalità già create e dei dati raccolti per permettere all'utente di effettuare una vera e propria simulazione del percorso.

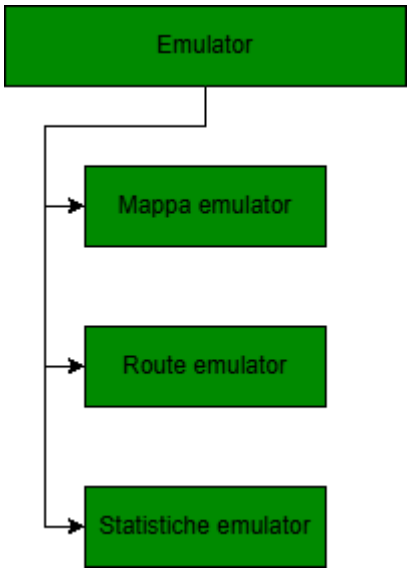


Figura 5: Diagramma del componente Emulator

3.2.6. Diagrammi back-end

Per quanto riguarda la struttura del back-end, il diagramma sarà articolato in quattro blocchi principali. Il back-end descriverà come l'utente, attraverso le interfacce dell'applicazione, interagirà con i servizi online e con il database per la manipolazione dei dati. Viene suddiviso in:

- Firebase *(figura 6)*: effettua la connessione al database online ed offre all'utente delle funzionalità per il trattamento e la gestione dei dati riguardanti itinerari, file audio e poligoni:
    - Salva itinerario: salva i dati inerenti ad un itinerario creato dall'utente (come la lista di coordinate, i luoghi di interesse, la lunghezza del tragitto, etc.);
    - Ricevere la lista di itinerari: permette di accedere alla lista degli itinerari salvati nel database;
    - Eliminare un itinerario: elimina un itinerario a scelta tra quelli salvati sul database;
    - Salva poligono: salva un nuovo file che contiene le caratteristiche del poligono inerente alla città o paese;
    - Ricevere la lista dei poligoni: permette di accedere alla lista dei poligoni salvati nel database;
    - Eliminare un poligono: elimina un poligono a scelta tra quelli salvati sul database;
    - Salva statistiche: effettua il salvataggio delle statistiche di un itinerario (come il profilo altimetrico e del terreno);
    - Ricevere le statistiche: permette di accedere alle statistiche di un itinerario specifico;
    - Salvare i dati di un audio: salva i dati di un file audio (come il nome e il genere) nel database;
    - Ricevere i dati degli audio: permette di accedere alla lista dei file audio e ai relativi dati;
    - Eliminare un audio: elimina un audio a scelta tra quelli salvati sul database;
    - Aggiungere audio ad un poligono: permette di associare un audio ad un poligono specifico;
    - Ricevere audio da un poligono: permette di accedere alla lista dei file audio associati ad un poligono.

Figura 6: Diagramma delle funzionalità di Firebase

- Express *(figura 7)*: esegue delle richieste al server Express per la connessione con la cartella di Google Drive dove andranno salvati e gestiti i file audio:
    - Salvare i file audio: salva un file audio in Google Drive con estensione ".mp3" o ".wav";
    - Ricevere la lista degli audio: permette di accedere alla lista dei file audio salvati in Google Drive;
    - Download di un file audio: permette di effettuare il download di un singolo file audio, in modo da poterlo scaricare nel proprio computer o per poterlo utilizzare all'interno dell'applicazione;
    - Eliminare un file audio: elimina un file audio a scelta tra quelli salvati in Google Drive.

Figura 7: Diagramma del server Express

- Actions *(figura 8)*: effettua delle operazioni di calcolo e di chiamate a servizi online per ottenere dati ed informazioni relativi a coordinate specifiche, luoghi, e poligoni. In particolare, le funzionalità sono:
    - Ricavare la posizione corrente: permette di ottenere la posizione corrente del dispositivo in uso durante l'utilizzo dell'applicazione;

- Creare un marker: permette di creare un marker provvisorio attraverso latitudine e longitudine;
- Eliminare un marker: permette di eliminare un marker all'interno della lista provvisoria;
- Ricavare latitudine & longitudine: permette di ottenere latitudine e longitudine di un luogo all'interno della mappa;
- Ricerca di un luogo dal nome: permette di trovare un luogo di interesse fornendo in input il nome;
- Calcolare la distanza tra due punti: permette di ricavare la distanza, in linea d'aria, tra due coordinate;
- Ricavare l'altitudine: permette di ottenere l'altitudine di una coordinata;
- Ricerca di un poligono da un ID: permette di trovare un poligono fornendo in input il suo identificatore univoco;
- Ricerca dei poligoni più vicini: permette di calcolare il poligono più vicino ad una coordinata specifica, prendendo come riferimento il suo centro;
- Calcolare il centro dei poligoni: permette di calcolare la coordinata che indica il centro del poligono nello spazio.

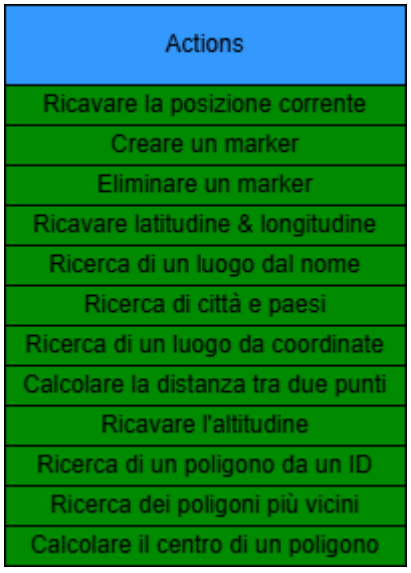


Figura 8: Diagramma delle funzionalità in Actions

- Utils *(figura 9)*: è un gruppo di funzionalità connesse a dei servizi online che permettono di ricavare le statistiche del profilo di un itinerario, e comprende:
  - API Online per il terreno: consente di ricavare dati inerenti ad un intorno specifico di una o più coordinate, in modo tale da rilevare le caratteristiche del terreno intorno ad un itinerario;
  - API Online per l'altitudine: consente di ricavare dati inerenti all'altitudine di una lista di coordinate, in modo tale da creare un profilo altimetrico associato

ad un itinerario specifico;

- Controllo del profilo dell'itinerario: gestisce la creazione dei due profili, richiamando, quando necessario, i servizi online.

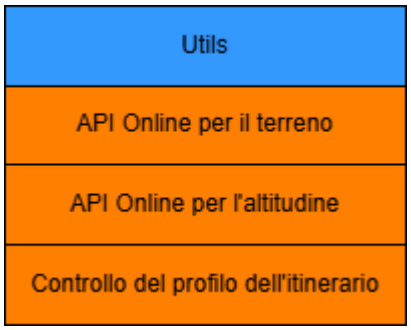


Figura 9: Diagramma delle funzionalità in Utils

3.3. Progettazione dei dati

Al fine di comprendere appieno le operazioni richieste dall'applicazione, è essenziale definire quali dati saranno trattati e manipolati, e quali metodi saranno impiegati per farlo. Sono state definite delle strutture iniziali di supporto su cui lavorare, per osservare quali attributi ci si aspetta di trovare quando si visualizzano le varie tipologie di dati.

Analizzando i requisiti dell'applicazione, i dati principali su cui lavorare ed ottenere informazioni saranno i seguenti:

- Coordinate singole *(figura 10)*: sono fondamentali per la creazione di itinerari e poligoni, in quanto rappresentano punti specifici sulla mappa, che definiscono le caratteristiche di tali entità geografiche. Inoltre, sono essenziali per operazioni basilari come il calcolo dell'altitudine, la determinazione di distanze e altre misurazioni geografiche di base. Questi dati sono utilizzati anche per ricavare informazioni dettagliate sulle caratteristiche del terreno circostante. I suoi attributi principali saranno: latitudine, longitudine e altitudine.

Coordinata	
Latitudine	Float
Longitudine	Float
Altitudine	Float

Figura 10: Struttura di una coordinata

- Liste di coordinate *(figura 11)*: sono impiegate per creare percorsi, route e statistiche più complesse. Utilizzando una serie di coordinate in sequenza, è possibile tracciare percorsi specifici lungo la mappa e generare route ottimali per gli itinerari desiderati. Le liste di coordinate, inoltre, permettono di calcolare delle statistiche dettagliate del percorso, come il proprio profilo di altitudine ed il profilo del terreno circostante al percorso. Gli attributi di questa tipologia di lista saranno proprio un insieme di coordinate, numerate ed ordinate, ognuna contenente le proprie informazioni.

Lista di coordinate	
Coordinata	Coord
Coordinata 2	Coord
...	Coord
Coordinata N	Coord

Figura 11: Struttura di una lista di coordinate

- Itinerari *(figura 12)*: sono costituiti da un insieme di coordinate, statistiche e le eventuali informazioni relative, come nomi dei luoghi di interesse dei marker o descrizioni aggiuntive riguardanti le tappe e le indicazioni dell'itinerario. Questi dati sono essenziali per la visualizzazione e la gestione degli itinerari all'interno dell'applicazione, consentendo agli utenti di esplorare e pianificare percorsi personalizzati. I suoi attributi, quindi, comprendono: una lista di coordinate, una lista di indicazioni stradali associati alle coordinate, delle informazioni generali e le statistiche del percorso inerenti ad altitudine e terreno.

Itinerario	
Lista di coordinate	List
Indicazioni	String
Informazioni	String
Statistiche	String

Figura 12: Struttura di un itinerario

- Poligoni *(figura 13)*: sono definiti da un insieme di coordinate che delineano un confine geografico, come quello di un paese, città o di uno stato. Questi dati sono utilizzati per creare aree geografiche definite e per identificare i confini amministrativi e territoriali. Sono inoltre utili per visualizzare ed analizzare specifiche aree geografiche all'interno dell'applicazione associandoli ad uno specifico itinerario. Gli attributi di un poligono sono: il nome, il tipo di indirizzo (città o paese) ed una lista di coordinate che lo delimitano.

Poligono	
Nome	String
Tipo di indirizzo	String
Coordinate	List

Figura 13: Struttura di un poligono

- File audio *(figura 14)*: rappresentano risorse aggiuntive che possono essere associate a specifici itinerari, coordinate o poligoni all'interno dell'applicazione. Questi file possono essere utilizzati per fornire feedback audio agli utenti durante l'esplorazione dei percorsi o per aggiungere elementi sonori alle varie aree geografiche visualizzate sulla mappa. Un file audio deve comprendere di: un nome univoco (titolo + nome artista), un genere musicale, il formato del file e il suo URL.

Audio	
Nome	String
Genere	String
Formato	String
URL	String

Figura 14: Struttura di un audio

4. IMPLEMENTAZIONE

Partendo dallo schema iniziale e dai diagrammi dei componenti, è stata inizializzata una prima struttura del progetto in Visual Studio Code, creando una nuova applicazione React attraverso la linea di comando "npx create-react-app *nomeapp*" nel terminale. Successivamente, la struttura del progetto è stata suddivisa in sottocartelle logiche, ad esempio per i componenti, le pagine, gli stili, i servizi, etc., al fine di organizzare in modo efficiente i file e facilitare la manutenzione. Questo approccio ha permesso una gestione più ordinata ed intuitiva del codice fin dall'inizio, facilitando l'aggiunta di nuove funzionalità e la risoluzione di problemi.

4.1. Components e props

Per la renderizzazione grafica del progetto, sono stati utilizzati i Components di React, noti per la loro capacità di organizzare l'interfaccia utente in modo modulare e dinamico. Questi sono annidabili ed interattivi, consentendo di sfruttare l'applicazione in modo efficiente e garantendo una gestione ottimale del codice, oltre che una maggiore riutilizzabilità delle parti dell'interfaccia utente. Nel contesto specifico, il progetto è stato strutturato attraverso una gerarchia di componenti, ognuno dei quali svolge un ruolo specifico nell'ambito dell'applicazione. Grazie alla loro natura dinamica, questi possono reagire in tempo reale agli input dell'utente ed agli aggiornamenti dei dati, garantendo un'esperienza utente fluida ed interattiva. Inoltre, per trasferire i dati tra i diversi livelli di annidamento, sono state utilizzate le props di React: fungono da parametri che vengono passati da un component padre ad uno figlio, consentendo una comunicazione efficace e trasparente tra i diversi componenti. Questo approccio consente una gestione efficiente dei dati all'interno dell'app, facilitando la collaborazione e lo scambio di informazioni tra i vari livelli del progetto. Infine, i componenti di React possono essere facilmente personalizzati attraverso l'uso di Material-UI (MUI), una libreria di componenti UI React basata sul design system di Google Material Design **[20]**. Offre una vasta gamma di components predefiniti e stili, consentendo agli sviluppatori di personalizzare l'aspetto ed il comportamento dei componenti in modo rapido ed intuitivo. In alternativa, sono stati integrati anche HTML e CSS, offrendo una maggiore flessibilità nel design e nell'estetica dell'interfaccia utente.

4.2. Specifiche dei dati

I dati raccolti dalle chiamate API non sono gestiti da nessun modello all'interno dell'applicazione e verranno salvate direttamente nei database sotto forma di stringhe annidate o file JSON **[21]** convertiti. Questa scelta è definita dal fatto che le varie chiamate API non sempre restituiscono le stesse strutture, essendo che alcune coordinate, ad esempio, potrebbero avere una varietà di attributi molto ampia o troppo ristretta, che potrebbe comportare una scarsa efficienza nell'occupare la memoria nel database. Nello specifico, si analizzerà quello che ci si aspetta di osservare una volta recuperati i dati:

1- Coordinate singole *(figura 15)*: ognuna è rappresentata come un array di float (salvate su Firebase come stringhe), una per la latitudine, una per la longitudine ed una per l'eventuale altitudine:

```
coord = {
    lat: "float",
    lon: "float",
    ele: "float"
}
```

Figura 15: Struttura di una coordinata

2- Liste di coordinate *(figura 16)*: sono come un array di array, dove ogni elemento interno rappresenta una singola coordinata:

```
listOfCoords = [
    coord1: {
        lat: "float",
        lon: "float",
        ele: "float"
    },
    ...
]
```

Figura 16: Struttura di una lista di coordinate

3- Itinerari *(figura 17)*: sono oggetti contenenti una serie di informazioni, inclusi i dettagli dei marker, le statistiche, le istruzioni di navigazione ed altri metadati:

```
itinerary = {
    name: "string",
    markers: {
        markersList: listOfCoords,
        markersName: "string"
    },
    instructions: "string",
    coordinates: listOfCoords,
    stats: {
        elevationProfile: "JSON",
        terrainProfile: "JSON",
    }
    summary: "string"
}
```

Figura 17: Struttura di un itinerario

4- Poligoni *(figura 18)*: sono oggetti contenenti il nome del poligono, il tipo di indirizzo e le coordinate che delineano il confine del poligono, salvandoli sullo storage sotto forma di file JSON e convertendoli per utilizzarlo nell'applicazione nel seguente formato:

```
polygon = {
    name: "string",
    addresstype: "string",
    coordinates: listOfCoords
}
```

Figura 18: Struttura di un poligono

5- File audio *(figura 19)*: sono salvati con la propria estensione all'interno di Google Drive (quindi ".mp3" o ".wav"), per poi recuperarli come oggetti contenenti informazioni sul nome, il genere e l'URL del file audio:

```
audio = {
    name: "string",
    genre: "string",
    url: "string"
}
```

Figura 19: Struttura di un audio

Queste strutture dati consentiranno di organizzare e salvare le informazioni nel database in modo efficace e veloce, consentendo un facile recupero ed utilizzo all'interno dell'applicazione.

4.3. Back-end

All'interno della struttura del progetto è stato creato separatamente un file scritto in Node.js, chiamato "GoogleDriveServer", in cui utilizzando il framework Express è possibile connettersi ai server di Google Drive.

Innanzitutto, bisogna richiedere i moduli npm, come "express", "stream", "multer", "path", abilitare i "cors" per evitare il blocco da parte del browser e "googleapis" per gestire il server, il caricamento di file, la gestione delle richieste HTTP, il routing e l'interazione con l'API di Google Drive. Viene creata poi un'istanza di Express e definita la porta su cui il server sarà in ascolto e specificato il percorso della cartella chiave con i relativi permessi necessari per accedervi con l'API di Google Drive.

Di seguito, sono definite diverse route:

- "/uploadToGoogleDrive", che permette di effettuare l'upload di un singolo audio alla volta nella cartella di Google Drive. Dopo aver effettuato l'accesso alla cartella specifica con le necessarie autorizzazioni, il file audio in input verrà salvato in un file con i relativi metadata ed il media sotto forma di buffer stream; se l'upload verrà effettuato con successo, verranno generati due URL per la visualizzazione e la riproduzione dell'audio e verrà restituito un messaggio di successo da parte del server;

- "/getAudioFiles", con metodo GET, che recupera i file audio presenti nella cartella specificata su Google Drive e restituisce un elenco di file con i rispettivi ID e nomi. Avendo la possibilità di uploadare un file con estensione ".wav" o ".mp3", verranno

effettuate due chiamate alla stessa cartella per recuperarli entrambi. La response ricevuta ed i rispettivi dati verranno raggruppati in una struttura unica e restituita al client con un messaggio di successo;

- "/downloadAudio/:fileId", con metodo GET, che consente di scaricare un file audio da Google Drive utilizzando l'ID specifico di quest'ultimo. Accedendo alla cartella, verrà ricercato il relativo file e restituito attraverso una response type di tipo "stream": accedendo al file data di quest'ultimo, sarà possibile utilizzare il file audio;

- "/deleteAudio/:fileId", con metodo DELETE, che permette di eliminare un file audio specifico all'interno di Google Drive. Anche qui, accedendo alla cartella, dopo aver recuperato il relativo file, attraverso una chiamata "delete" verrà eliminato l'audio e restituito un messaggio di successo.

Infine, il server viene avviato sulla porta specificata, stampando un messaggio di log per confermare che il server è in esecuzione. Lo schema *(figura 20)* rappresenta il funzionamento di una richiesta effettuata lato client e le possibili risposte del server.

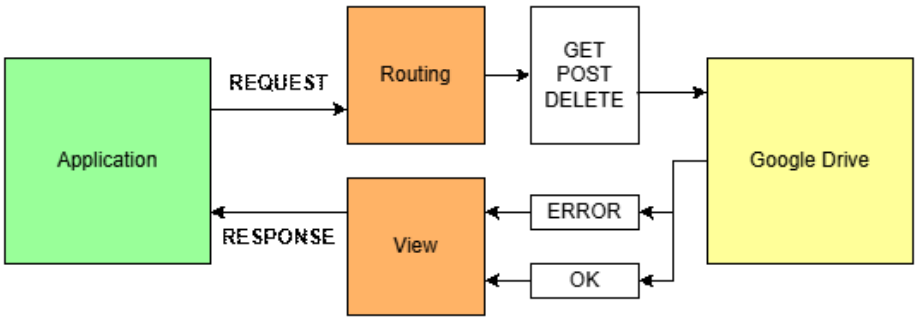


Figura 20: Funzionamento delle richieste di un server Express

Questa parte di codice consente quindi all'applicazione di gestire operazioni come il caricamento, il recupero, il download e l'eliminazione di file audio su Google Drive. Le richieste dal lato client, quindi, verranno inviate al back-end, che a sua volta interagisce con l'API di Google Drive per eseguire le operazioni richieste.

4.4. App.js

Rappresenta il punto d'ingresso dell'applicazione lato client e svolge diversi compiti cruciali. Innanzitutto, è responsabile per l'inizializzazione dei dati, recuperandoli sia dal server Express che da Firebase, preparando così l'esperienza utente.

All'interno del componente "App", sono definiti gli stati per gestire i dati principali dell'applicazione, tra cui gli itinerari, i poligoni ed i file audio. Utilizzando l'hook "useEffect", viene effettuata una chiamata asincrona per recuperare i dati online. Una volta ottenuti i dati, vengono impostati negli stati corrispondenti. Successivamente, viene definito il routing dell'applicazione, che permetterà la gestione della navigazione all'interno dell'applicazione, passando i dati ad i vari componenti.

Complessivamente, "App.js" funge da punto di partenza per l'applicazione, gestendo il recupero dei dati e la navigazione tra le diverse pagine.

4.5. Navigazione

La navigazione all'interno dell'applicazione sarà strutturata con un approccio basato su routing, dove diverse pagine o interfacce dell'applicazione saranno associate a specifici percorsi URL. Questo permetterà agli utenti di spostarsi a un'interfaccia all'altra in modo rapido ed intuitivo semplicemente navigando attraverso l'URL nel browser o utilizzando controlli di navigazione integrati nell'applicazione stessa.

4.5.1. Browser Router

Per controllare ed effettuare la navigazione, è stato utilizzato il componente <BrowserRouter> fornito da React Router **[22]**. Questo gestisce la navigazione utilizzando il browser's History API ed assicura che l'URL corrente sia sincronizzato con l'applicazione. Sono state definite le route per i tre URL principali utilizzando il componente <Route>. Ogni Route è stata configurata con un path corrispondente all'URL desiderato e un componente da renderizzare quando esso corrisponde al percorso specificato. Inoltre, inizializzando alcuni dati al partire dell'applicazione, il router ha permesso il passaggio dei dati da un componente all'altro senza troppe difficoltà, mantenendo ed aggiornando il suo stato in maniera asincrona.

I componenti principali dell'applicazione, inseriti nel router nel file "App.js", sono:

- Map.js;
- Statistics.js;
- Music.js;
- Emulator.js;
- Navbar.js.

4.5.2. Navbar

Per effettuare la navigazione è stata posta una barra di navigazione in alto, in modo tale da poter essere sempre in vista per poter cambiare interfaccia velocemente, utilizzando il

componente <BottomNavigation> della libreria MUI. Composta da quattro action differenti, ognuna per una specifica URL, si permette di navigare al click dell'utente attraverso il metodo "useNavigate" di React Router.

4.6. Firebase

Per l'interazione con il database e lo storage di Firebase sono stati integrati due file che collaborano tra loro per l'inizializzazione e le chiamate API: "Firebase.js" e "FirebaseController.js".

Il primo è utilizzato per inizializzare l'applicazione Firebase e per esportare le funzionalità di base per l'accesso al database in tempo reale ed allo storage. Contiene anche le credenziali di accesso e le configurazioni dell'app Firebase.

D'altra parte, il secondo contiene una serie di funzioni specifiche che vengono utilizzate per eseguire operazioni su database e storage. Queste funzioni includono operazioni come il salvataggio di itinerari, poligoni e statistiche, nonché il recupero dei dati.

I path principali per raccogliere i dati e salvarli in delle strutture specifiche, sono:

- "itineraries/", dove verranno salvate le caratteristiche e statistiche degli itinerari;
- "polygons/", dove verranno salvate le informazioni relative agli audio associati ai poligoni;
- "audios/", dove verranno salvati le informazioni dei file audio come nome e genere;
- "polygons/" (nello storage), dove verranno salvati gli effettivi file JSON relativi ai poligoni.

Di seguito, l'elenco delle funzioni del controller:

- saveItinerary(markersList, itinerary, name):

  Salva un itinerario nel Realtime Database di Firebase: prende in input una lista di marker, l'itinerario ed il nome scelto; i dati vengono poi strutturati e salvati nel database nel path "itineraries/<nomeitinerario>". I dati verranno salvati in una struttura JSON, che comprende il nome, i marker (ovvero i checkpoint) con i rispettivi nomi, le istruzioni con le rispettive coordinate ed il sommario;

- getItineraries(setItineraries):

  Recupera la lista degli itinerari dal database e la imposta nello stato dell'applicazione attraverso una chiamata che recupera lo snapshot dalla reference "itineraries/";

- deleteItinerary(name):

  Elimina un itinerario dal database utilizzando il nome come identificativo, ricercando il path corrispondente in "itineraries/";

- savePolygon(polygon, name, addresstype):

  Salva un poligono nello storage, insieme al suo nome e al tipo di indirizzo

(città/stato). All'inizio, le informazioni vengono strutturate in un formato JSON; poi si accede allo storage nel path "polygons/" e si trasforma il JSON dei dati in una stringa;

- getPolygons(setPolygons):

  Recupera i poligoni dallo storage accedendo al path "polygons/" e restituisce una promessa della lista dei poligoni presenti; poi, effettua il download della URL, recupera il risultato in formato JSON e lo imposta nello stato dell'applicazione.

- deletePolygons(polygoName):

  Elimina un poligono dallo storage attraverso la ricerca del nome.

- saveStatistics(itineraryName, stats):

  Salva le statistiche di un itinerario nel database, effettuando l'update delle informazioni nel relativo path. Le statistiche includono informazioni come il profilo altimetrico ed il terreno lungo il percorso.

- getStatistics(itineraryName, setElevationProfile, setTerrainProfile):

  Recupera le statistiche di un itinerario dal database e le restituisce all'applicazione per l'elaborazione e la visualizzazione. Si accede al path "itineraries/<nomeitinerario>/stats/" e si effettua il download dello snapshot; lo snapshot, se esiste, comprenderà un profilo altimetrico ed uno relativo al terreno, che verranno salvati nei corrispettivi stati;

- saveAudio(audio):

  Salva un file audio nello storage con il nome ed il genere specificato, sottoforma di file BLOB, un tipo di dato usato nei database per identificare oggetti binari di grandi dimensioni **[23]**. Il file deve essere di tipo "audio/wav" ed esso verrà salvato attraverso la funzione "uploadBytes";

- getAudios(setAudios):

  Recupera l'elenco dei file audio dallo storage e li imposta nello stato dell'applicazione. Accedendo al path "audios/", si effettua una scansione delle sottocartelle con i rispettivi generi, ed effettuando il download dell'URL dei corrispettivi audio, i dati verranno salvati in una lista ed ogni file comprenderà: nome, BLOB data, genere, url;

- addAudioToPolygon(polygonName, newAudios):

  Aggiunge un file audio ad un poligono nel database, consentendo di associare audio specifici a determinati luoghi. Accedendo al path di un poligono specifico con "polygons/<nomepoligono>/audios/", è possibile associare uno o più audio ad esso;

- getAudiosFromPolygon(polygons):

  Recupera i file audio associati ad un poligono dal database e li restituisce all'applicazione. Accedendo al path "polygons/<nomepoligono>/audios/", si

effettua il download dello snapshot e si organizzano gli audio in una lista appropriata;

- saveAudioData(audioData, genre):

  Salva i dati del file audio nel database sotto la cartella del corrispettivo genere. Accedendo al path "audios/<genere>/", si avrà una organizzazione precisa per genere; la scelta di tenere traccia delle informazioni relative agli audio anche su Firebase, oltre che su Google Drive, è per pura organizzazione strutturale e perché GoogleDrive non permette di salvare informazioni utili come il genere;

- getAudioData(setAudioData):

  Recupera i dati dei file audio dal database e li restituisce all'applicazione. Accedendo al path "audios/" e poi ad ognuno sottocartella specifica che rappresenta un genere particolare, è possibile salvare i dati degli audio categorizzati per generi in una lista;

- deleteAudioData(audioId, genre):

  Elimina i dati di un file audio specifico nel database, accedendo al path "audios/<genere>/" e ricercandolo tramite ID.

Le funzionalità relative al salvataggio e manipolazione di file audio all'interno dello storage (sottoforma di BLOB) non verranno utilizzate, poiché nel corso della progettazione, caricare e scaricare una lista di file così pesanti pone un limite di chiamate giornaliere molto basso, motivazione per cui è stato integrato Google Drive. Nonostante ciò, queste funzioni saranno mantenute all'interno del codice nel caso in cui si volessero utilizzare in futuro.

Queste funzioni forniscono un'interfaccia semplice ed intuitiva per interagire con il database e lo storage di Firebase all'interno dell'applicazione.

4.7. Map

Il componente <Map> è un container per renderizzare e gestire diverse funzionalità legate agli itinerari e ad operazioni con luoghi, città, nazioni e coordinate.

Tutte le funzioni e chiamate ad API esterne che restituiscono dati particolari sono state raggruppate in un file chiamato "Actions.js".

4.7.1. Controllo della mappa

Il controllo della mappa è affidato al componente <Maps>, chiamato così poiché, in un futuro sviluppo dell'applicazione, potrebbe essere possibile visualizzare più tipologie di mappe. Per renderizzare la mappa si è utilizzata Leaflet **[24]**, una libreria open source di JavaScript utilizzata per mostrare delle mappe su applicazioni web. Infatti, utilizzando un componente <MapContainer> verrà renderizzata una mappa con un determinato URL. In questo caso si è scelto di utilizzare una mappa di OpenStreetMap basilare, che mostri sia una panoramica generale sulla tipologia di terreno, sia le reti stradali. La libreria Leaflet mette a disposizione molti metodi per poter interagire con la mappa, mentre altri sono stati ricercati online, creati da terzi sviluppatori ed integrati direttamente nel progetto. I dettagli di ogni singola funzionalità verranno approfonditi in seguito.

Un metodo che si è tenuto fondamentale da inserire è stato quello di poter inserire direttamente all'interno della mappa un "marker", ovvero un segnale che indicasse un luogo specifico all'interno della mappa. Quando si inserisce un marker in un punto, i primi dati restituiti dalla mappa saranno le sue coordinate geografiche, latitudine e longitudine. Questi verranno poi processati da altre funzioni per elaborare nuovi dati.

Quando si aggiungono due o più marker, essi creeranno un percorso all'interno della mappa, seguendo le reti stradali e calcolando il percorso più veloce. Il risultato verrà mostrato direttamente nella mappa, con le relative indicazioni stradali, il tempo di impiego per percorrerlo in automobile e la lunghezza del tragitto.

Inoltre, è stato implementato un sistema per l'aggiunta di marker in luoghi specifici, come città o nazioni, tramite una casella di ricerca.

Nello stesso componente è stata inserita una lista di varie funzionalità, chiamate "Actions", in cui è possibile effettuare chiamate API per mandare o recuperare particolari dati, la quale comprende:

- aggiungere un marker (attraverso le coordinate o manualmente);
- calcolare la distanza tra due marker (in linea d'aria);
- trovare l'altitudine di una particolare coordinata;
- salvare un nuovo itinerario su Firebase;
- mostrare la lista degli itinerari salvati su Firebase;
- ricercare il poligono inerente a città o nazioni salvate sul database più vicino ad una coordinata specifica.

Infine, si è ritenuto opportuno inserire la funzionalità di geolocalizzare il dispositivo, sotto opportune autorizzazioni concesse dall'utente, e di mostrarlo direttamente nella mappa, creando un marker e centro la mappa su di esso.

4.7.2. Searching

La ricerca di luoghi specifici è gestita dal componente <SearchBox>, che prevede un input dove gli utenti hanno la possibilità di inserire il nome completo o parziale di un luogo, o altri dettagli rilevanti, ed un bottone di ricerca che richiama la funzione "searchLocationWithName": è progettata per effettuare una ricerca di luoghi utilizzando il servizio Nominatim di OpenStreetMap. Definendo l'URL di base per le richieste di servizio, invia una richiesta HTTP con parametri il testo di ricerca ed il formato della risposta (in questo caso JSON). La richiesta restituirà quindi una lista di luoghi inerenti alla ricerca, escludendo i dati poligonali. Utilizza la tecnica di geocodifica degli indirizzi, ovvero il processo di prendere una descrizione testuale di una posizione, come un indirizzo o il nome di un luogo, e restituisce le coordinate geografiche. I risultati verrano elaborati ed inseriti all'interno di una lista, che verrà visualizzata dinamicamente sotto alla search box. Cliccando su una delle alternative in lista, è possibile aggiungere un marker nella mappa inerente alle sue coordinate.

4.7.3. Marker

I marker vengono collezionati ed aggiunti in una lista dinamica, il cui stato viene conservato nel ciclo di vita del componente <Map>. Un marker può essere inserito attraverso vari metodi:

- ricercando la posizione corrente del dispositivo utente tramite GPS: la web app chiederà l'autorizzazione al browser che si sta utilizzando;
- selezionando uno dei risultati in lista dopo aver effettuato una ricerca;
- inserendo manualmente delle coordinate (latitudine e longitudine);
- attivando la modalità "Manual Mode" e cliccando direttamente in un punto della mappa interattiva.

Ogni volta che si vuole aggiungere un marker alla lista, le sue coordinate verranno passate ad una funzione chiamata "createMarker", che a sua volta eseguirà una chiamata API a Nominatim **[25]**: questa volta la ricerca non verrà effettuata attraverso delle parole chiave, ma attraverso le sole coordinate, grazie alla tecnica di reverse geocoding, restituendo così tutti i dati inerenti a quelle coordinate. Questi ultimi verranno inseriti all'interno della lista "markersList".

Grazie al componente <MarkerController> si potrà visualizzare la lista dei marker temporanei inseriti durante l'utilizzo dell'interfaccia, con le relative informazioni, in cui è possibile anche eliminarli.

Nel caso in cui siano presenti due o più marker all'interno della lista, è possibile salvare direttamente sul database l'itinerario che si è creato in automatico.

4.7.4. Routing

Utilizzando la libreria "Leaflet Routing Machine" **[26]**, è possibile creare dinamicamente un percorso all'interno di una mappa Leaflet. Il componente <Routing> si occupa di renderizzare il percorso stradale tra due o più marker. La lista dei marker viene interpretata come dei waypoint ai quali il controllo del routing trova un percorso, lo restituisce sotto forma di un itinerario, che verrà poi salvato nel database una volta che l'utente è soddisfatto. L'itinerario includerà le coordinate dei marker e dei waypoint relativi alle indicazioni stradali lungo il percorso, come ad esempio i punti in cui è necessario svoltare a destra o imboccare altre strade. Saranno fornite anche informazioni sulla lunghezza del percorso e sul tempo stimato di percorrenza in macchina.

4.7.5. Actions

Tutte le azioni implementate inerenti a coordinate, itinerari e poligoni sono state raggruppate in un componente chiamato <ActionController> e vengono presentate come opzioni selezionabili dall'utente. Tutte queste funzionalità sono state implementate con l'obiettivo di fornire agli utenti ed agli sviluppatori una base solida di funzioni semplici per iniziare ad interagire con i dati ricavati dalla mappa e dalle coordinate.

Nel dettaglio, si andranno ad analizzare tutte le funzionalità:

1- <AddMarker>: grazie a questo componente, l'utente potrà inserire delle coordinate (latitudine e longitudine) manualmente; in questo caso verrà invocata, come già descritto, la funzione "createMarker" e nella mappa verrà posizionato un marker corrispondente a quelle coordinate. In alternativa, l'utente potrà attivare la modalità "Manual Mode" attraverso un bottone ed è un'opzione che permette agli utenti di cambiare il comportamento della mappa in base alle proprie esigenze. Quando attivata, questa modalità trasforma l'interazione dell'utente con la mappa: anziché consentire agli utenti di navigarci attraverso il trascinamento ed i click per esplorare l'area, la modalità manuale consente di posizionare manualmente un marker sulla mappa. Quindi, una volta attivata la modalità, l'utente potrà cliccare in un punto specifico della mappa ed un marker verrà posizionato esattamente in quella posizione, ricavando in automatico le coordinate e creando un nuovo marker all'interno della lista. Una volta che il marker sarà stato creato, la modalità verrà disinserita.

2- <CalculateDistance>: consente agli utenti di calcolare la distanza in linea d'aria tra due coordinate geografiche, attraverso l'inserimento manuale di due coordinate desiderate dall'utente. Una volta inserite, il sistema calcola automaticamente la distanza tra i due punti e restituisce il risultato sotto forma di valore numerico espresso in chilometri. Tutto ciò avviene attraverso la funzione "calculateDistance", che accetta due coordinate geografiche come parametri e le elabora utilizzando operazioni matematiche appropriate, tenendo conto della curvatura della Terra.

3- <GetAltitude>: in questo caso l'utente potrà inserire manualmente delle coordinate geografiche che verranno elaborate da API online per restituire il valore della relativa altitudine in metri. In questo caso, la funzione che svolge questo compito è "getAltitude": viene innanzitutto specificato il server a cui fare richiesta, che in questo caso è "api.open-elevation.com" **[27]**, poi viene costruito l'URL completo ed impostati gli header, che indicano che il client accetta risposte in formato JSON e che invia dati in formato JSON; viene costruito l'oggetto delle opzioni per la richiesta, che include il corpo (le posizioni geografiche da interrogare), gli header ed il metodo (POST); infine, viene eseguita la richiesta fetch verso l'URL specificato, che verrà poi gestita dal server elaborando un risultato, controllando che non ci siano eventuali errori di connessione o di sintassi durante il processo.

4- <NearestPolygon>: consente agli utenti di trovare il poligono di una città più vicino rispetto ad una coordinata specifica. La funzione che svolgerà questo compito è "findNearestPolygon": viene inizializzato un array per memorizzare le distanze tra le coordinate specifiche ed i centri dei vari poligoni presenti nella lista; utilizzando un'altra funzione "calculateCenterID", viene iterata la lista dei poligoni, calcolate le coordinate del suo centro ed infine, attraverso "calculateDistance" viene calcolata la distanza tra la coordinata scelta dall'utente e tutte le coordinate del centro dei poligoni. I risultati verranno aggiunti all'array, su cui verrà eseguita una ricerca per trovare la distanza minima tra tutte, che verrà restituita come risultato. La funzione "calculateCenterID", per calcolare le coordinate del centro di un poligono, utilizza il package "Turf.js" **[28]**, che fornisce una varietà di funzioni per l'analisi geospaziale.

5- <SaveData>: durante l'utilizzo dell'applicazione, una volta che l'utente abbia

inserito nella mappa due o più marker e la routing machine abbia creato il percorso, attraverso questo componente è possibile salvare i dati dell'itinerario nel Realtime Storage di Firebase. L'utente deve assegnare un nome all'itinerario; inoltre, per le limitazioni e restrizioni di Firebase, si è preferito imporre un controllo sulla lunghezza dell'array delle coordinate della route: se superiore a 2000, l'itinerario verrà ritenuto troppo lungo per essere salvato sul database: Il salvataggio verrà effettuato attraverso il richiamo della funzione "saveItinerary" nel controller di Firebase, che creerà un path univoco per ogni percorso.

6- <ItineraryList>: fornisce una visualizzazione organizzata ed accessibile degli itinerari disponibili salvati su Firebase, consentendo agli utenti di ricercare specifici percorsi tramite testo. Una volta selezionato un itinerario, il componente si occupa di aggiornare dinamicamente la mappa per mostrare il percorso, con i relativi marker e route.

4.8. Statistics

Il componente <Statistics> è un container che permette di renderizzare e gestire varie funzionalità inerenti agli itinerari. Quindi, fornisce un'interfaccia per esaminare varie statistiche relative al terreno e ai file audio associati agli itinerari. In particolare, questo componente consente agli utenti di:

- visualizzare statistiche dettagliate del profilo altimetrico lungo il percorso;
- ottenere una sequenza di note musicali attraverso la lettura del profilo altimetrico ed effettuare un'analisi su di esse, come la nota più alta o quella più frequente;
- visualizzare statistiche dettagliate del terreno in un intorno specifico e delimitato lungo il percorso;
- associare le varie tipologie del terreno a dei generi musicali prestabiliti ed ottenere una playlist random di file audio a partire da essi, in modo da generare una lista di canzoni o di suoni naturali per accompagnare durante il viaggio in maniera suggestiva;
- visualizzare statistiche inerenti ai poligoni (città) più vicini rispetto all'itinerario selezionato;
- osservare quali file audio sono stati associati ai poligoni più vicini rispetto all'itinerario selezionato.

Tutte queste funzionalità sono state raggruppate e divise in tre componenti annidati: <Itineraries>, <Sounds> e <Stats>.

4.8.1. Itinerari

Il componente <Itineraries> permette di visualizzare in una lista tutti gli itinerari che sono stati salvati sul database di Firebase. Nello specifico, la funzionalità principale è quella di generare un profilo altimetrico e del terreno attraverso l'utilizzo della lista di coordinate all'interno del percorso selezionato. All'interno della lista è presente un'icona di download che permette, qualora non fosse già presente, di creare da zero questo profilo.

Infatti, quando si cercherà di cliccare sull'icona di download del profilo di un itinerario presente in lista, l'applicazione tenterà di controllare se effettivamente è presente all'interno del database attraverso la funzione "getStatistics" all'interno del controller di Firebase. Se la call non darà risultati, allora verranno istanziate due variabili per i due profili, che verranno inizializzate attraverso dei metodi e delle API terze. Al termine delle operazioni e quando tutte le informazioni saranno state ricavate, le statistiche verranno salvate nel Realtime Database attraverso la funzione "saveStatistics", anch'essa all'interno del controller di Firebase. Il recupero dei dati relativi alle statistiche degli itinerari viene effettuato nel file "profileControl.js", che fa riferimento a tutte le funzioni per recuperare informazioni e statistiche sul profilo altimetrico e del terreno attraverso delle chiamate ad API esterne. Le seguenti funzioni sono state raggruppate in questo file in modo da semplificare la struttura del progetto ed ottenere maggiore pulizia nel codice.

Per la creazione di un profilo del terreno vengono utilizzate delle API che fanno riferimento al server "https://overpass-api.de/api/interpreter" **[29]**, delle chiamate di sola lettura che funzionano come un grande database sul web: il client invia una query ed ottiene il set di dati che corrisponde alla query. Overpass è ottimizzata per i consumatori di dati che necessitano di pochi elementi in un attimo o fino a circa dieci milioni di elementi in alcuni minuti, entrambi selezionati da criteri di ricerca come ad esempio posizione, tipo di oggetti, proprietà dei tag, prossimità o combinazioni di essi. Le funzioni utilizzate per creare un profilo per il terreno sono le seguenti:

- drawTerrainProfile(coordinates, setTerrainProfile):

  Viene passata come parametro una lista di coordinate che, utilizzando delle API esterne, restituiranno dei dati inerenti ad un raggio intorno al percorso creato dalle coordinate, che verranno suddivise in tre macrocategorie: "Nature", "Water" ed "Artificial". Per non effettuare troppe chiamate si è dovuto imporre una limitazione anche qui, ovvero a seconda della lunghezza della lista delle coordinate, il codice deciderà su quale strategia di batching basarsi:

  - se la lunghezza delle coordinate è inferiore a 51, la elabora una volta sola, poiché la quantità di dati è relativamente piccola e può essere gestita senza rischi di sovraccarico dell'API;
  - se la lunghezza è compresa tra 51 e 100, il codice elabora le coordinate a lotti di 10, consentendo comunque una buona precisione;
  - se la lunghezza è compresa tra 101 e 500, le coordinate vengono elaborate in lotti di 50;
  - se la lunghezza supera le 500 coordinate, il codice elabora le coordinate a lotti di 100: questo riduce significativamente il numero di chiamate all'API, mentre cerca di mantenere una rappresentazione accurata del terreno.

  Per distinguere i vari casi, verranno chiamate delle funzioni specifiche, che effettueranno un processo di analisi di una coordinata specifica o di una lista di coordinate.

- fetchAndProcessProfileData(coord, radius, terrainProfile, overpassAPIUrl):

Attraverso una query opportunamente formattata, effettuerà una richiesta all'API di Overpass per recuperare dati relativi agli elementi di nodi, relazioni e strade (rappresentati da "nwr") con i tag "landuse", "landcover", "natural" e "water". La query sarà circoscritta alla vicinanza della coordinata fornita con un raggio di 200 metri. Infine, una volta recuperate le informazioni, si procede con l'analisi di tutti gli elementi e dei relativi tag. Questo approccio permette di aggregare gli elementi recuperati in tre macrocategorie distinte, ovvero "Nature" (natura), "Water" (acqua) e "Artificial" (artificiale). Attraverso questa suddivisione, è possibile organizzare ed interpretare in maniera più chiara e significativa le informazioni sul terreno. Ad esempio, gli elementi con tag come "farmland", "forest" o "grass" potrebbero essere raggruppati sotto la categoria "Nature" se indicano aree naturali, mentre gli elementi con tag "water", "beach", "aquaculture" verranno inclusi nella categoria "Water". Questo processo viene attivati nel caso in cui la lista di coordinate abbia una lunghezza inferiore a 51;

- fetchAndProcessBatchProfileData(batchCoordinates, radius, terrainProfile, overpassAPIUrl):

  La funzione sarà chiamata in tutti gli altri casi, prendendo in considerazione una lista di coordinate raggruppate in batch, che verranno processati dalla stessa query. Questo approccio, come già descritto, permetterà di ridurre il numero di chiamate all'API di Overpass, mantenendo comunque un'adeguata copertura dei dati del terreno;

- isNature(element):

  Passando come parametro un elemento con dei tag, verrà effettuato un controllo per controllare se appartengono e potranno essere raggruppati della macrocategoria "Nature";

- isWater(element):

  Come la precedente, ma per la macrocategoria "Water". Inoltre, se i tag non faranno riferimento a nessuna delle due, gli elementi verranno raggruppati automaticamente sotto la macrocategoria "Artificial".

La decisione di raggruppare i vari tag in queste tre macrocategorie deriva da due ragioni fondamentali: innanzitutto, rappresentarli separatamente, oltre alla quantità elevata di tag diversi, potrebbe non permettere di ottenere una comprensione completa di come il terreno sia effettivamente occupato, poiché alcuni tag sono troppo specifici ed isolati e considerarli singolarmente potrebbe rendere difficile cogliere il quadro generale della distribuzione del suolo; in secondo luogo, categorizzando gli elementi in queste macrocategorie, è possibile adottare suoni ed audio più generali per ciascuna. Questo non solo semplifica l'analisi e l'interpretazione dei dati, ma permette anche una rappresentazione più intuitiva ed accessibile del paesaggio sonoro. Di seguito, una tabella con la suddivisione effettuata per la distinzione di tutti i tag *(figura 21)*:

landuse	landcover	water	natural
Nature	Water		Artificial
allotments	aquaculture		commercial
farmland	port		construction
paddy	basin		education
animal_keeping	salt_pond		fairground
flowerbed	bay		industrial
forest	beach		residential
greenhouse_horticulture	blowhole		retail
meadow	cape		institutional
orchard	coastline		farmyard
plant_nursery	isthmus		brownfield
vineyard	peninsula		cemetery
grass	reef		depot
greenfield	shoal		garages
recreation_ground	strait		landfill
village_green	water		mlitary
winter_sports	wetland		port
natural	water		quarry
landcover			railway
			recreation_ground
			religious

Figura 21: Tabella delle categorie dei tag

Per la creazione di un profilo altimetrico, inizialmente, sono state riscontrate molte difficoltà. Infatti, il server "api.open-elevation.com" permette di effettuare una singola call ogni tot secondi, imponendo un limite eccessivo per quello che l'applicazione avrebbe dovuto svolgere, dato che un itinerario potrà avere un numero massimo di coordinate pari a 2000. Dopo molta ricerca online, è stato trovato un metodo su Github da un utente che, attraverso delle funzioni matematiche, è riuscito a scannerizzare una mappa globale e ricavarne l'altitudine del terreno **[30]**. La mappa a cui si fa riferimento è "Terrain RGB V2", del sito "Maptiler" **[31]**, che contiene un modello digitale di elevazione (DEM) a copertura globale: la mappa viene suddivisa in dei set di riquadri, disponibili fino ad un livello di zoom pari a 12, che forniscono l'altezza sopra al livello del mare in metri per ogni singolo luogo del mondo, misurata per regioni di circa 30x30 metri. Quindi, il modello di elevazione è suddiviso in riquadri, ognuno codificato nel formato immagine RGB standard, in cui ogni pixel di colore diverso viene trasformato in elevazione utilizzando questa formula:

"elevation = -10000 + ((R * 256 * 256 + G * 256 + B) * 0.1)"

Il formato consente di memorizzare le elevazioni in modo molto efficiente (3 byte), con sufficiente precisione. Il vantaggio principale è il basso traffico per applicazioni web, poiché il browser carica solo i riquadri nella visualizzazione corrente con la risoluzione appropriata e scarica i dati aggiuntivi nel caso in cui l'utente ingrandisca o esegua una panoramica della mappa. Di seguito, la colorazione della mappa RGB *(figura 22)*:

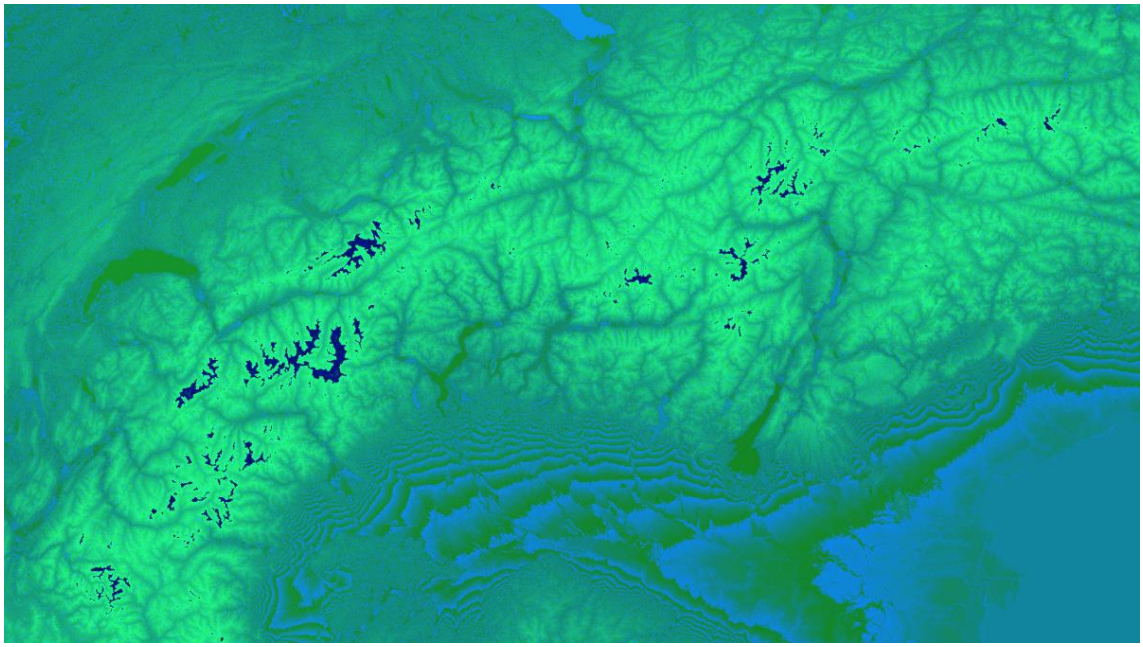


Figura 22: Mappa Terrain RGB V2, MapTiler

Inizialmente, la funzione a cui si fa riferimento è la seguente:

- drawElevationProfile(coordinates, setElevationProfile, elevationProvider):

  Viene passata come parametro una lista di coordinate che, grazie ad un provider, ricaverà l'altitudine di ognuna di esse, generando così un array di coordinate composte da [lat, lon, ele] che andranno ad aggiornare le statistiche dell'itinerario. Il codice fa riferimento a due classi: "ElevationProvider" e "GlobalMercator":

  - "ElevationProvider": questa classe è un provider per i dati di elevazione forniti da MapTiler RGB-Terrain. La funzione principale è quella di accettare delle coordinate con latitudine e longitudine e restituire l'elevazione corrispondente in base ai dati ottenuti dalle chiamate API. La classe utilizza la libreria "globalMercator" per gestire le trasformazioni tra coordinate geografiche e coordinate dei tile. Le funzioni principali, quindi, sono:

    - getElevation(lat, lon):

      Questa funzione prende le coordinate di latitudine e longitudine come input e restituisce l'elevazione corrispondente. Prima calcola l'indice della tile contenente le coordinate fornite, quindi recupera i dati della tile tramite "getOrFetchTile" e li decodifica utilizzando "decodeElevation";

    - getOrFetchTile(tileIndex):

      Gestisce il recupero dei dati della tile. Controlla se la tile è già stata memorizzata in memoria; se sì, restituisce la tile memorizzata, altrimenti la recupera tramite "fetchTile";

- fetchTile(tileIndex):

  Effettua una richiesta HTTP per ottenere i dati dell'immagine in formato utilizzabile;

- loadImage(url):

  Restituisce una promessa che si risolve con un oggetto immagine caricato dall'URL specificato;

- getImageData(image):

  Prende un oggetto immagine come input e restituisce i dati dell'immagine in formato ImageData, che può essere utilizzato per l'analisi dei colori;

- createTileKey(tileIndex):

  Crea una chiave univoca per identificare una tile in base alle sue coordinate ed al livello di zoom;

- decodeElevation(lat, lon, tileIndex, tileData):

  Decodifica I dati della tile per ottenere l'elevazione corrispondente alla coppia di coordinate fornite. Utilizza le informazioni sul colore dei pixel nell'immagine della tile per calcolare l'elevazione.

- "GlobalMercator": questa classe fornisce funzioni per la conversione tra coordinate geografiche e coordinate delle tile, seguendo le specifiche di mercator globale. Queste funzioni sono utilizzate dalla classe "ElevationProvider" per gestire le conversioni di coordinate necessarie durante il recupero dei dati di elevazione:

  - LatLonToMeters(lat, lon):

    Converte la coordinata in coordinate in metri rispetto all'origine della proiezione mercator globale;

  - MetersToPixel(mx, my, zoom):

    Converte le coordinate in metri in coordinate di pixel nel contesto di un determinato livello di zoom;

  - Resolution(zoom):

    Ottiene la risoluzione metri/pixel da un determinato livello di zoom misurato a partire dall'equatore;

  - PixelsToMeters(px, py, zoom):

    Converte le coordinate pixel in metri nel contesto di un determinato livello di zoom;

- PixelsToTile(px, py):

  Ritorna una tile che ricopre la regione delle coordinate pixel in input;

- PixelsToRoaster(px, py, zoom):

  Muove le coordinate pixel originali all'angolo in alto a sinistra;

- LatLonToTile(lat, lon, zoom):

  Converte una coordinata in una tile utilizzando i metodi precedenti;

- MetersToTile(mx, my, zoom):

  Converte delle coordinate in metri in una tile utilizzando i metodi precedenti.

Grazie ad una catena di montaggio contribuita da tutte queste funzioni, è possibile quindi ricavare una quantità elevata di altitudini partendo da una lista di coordinate in pochi secondi, rispettando i limiti delle API. Le funzioni di ElevationProvider gestiscono efficacemente il recupero dei dati di elevazione dalle tile fornite da MapTiler, mentre le funzioni di GlobalMercator consentono di gestire le conversioni di coordinate necessarie durante questo processo. Questo approccio consente di massimizzare l'efficienza e la velocità nell'ottenere le altitudini desiderate. L'unico problema riscontrato utilizzando questo approccio è la mancanza di considerazione delle peculiarità del terreno, come il fatto che un percorso stradale potrebbe attraversare una galleria: in questo caso le API potrebbero fornire l'altitudine della montagna o della collina sopra la galleria anziché quella effettiva del percorso stradale. Questo può portare a discrepanze nelle altitudini misurate rispetto alla realtà del terreno.

4.8.2. Stats

Il componente <Stats> funge da container per visualizzare le statistiche degli itinerari attraverso dei charts attraverso la libreria "Chart.js" **[32]**: è una libreria JavaScript libera per la creazione di grafici basati su HTML con una vastità di grafici di diversa tipologia, interattivi, dinamici e semplici da inserire. Le statistiche analizzate dai grafici sono inerenti alle caratteristiche degli itinerari, quindi dal proprio profilo altimetrico, profilo del terreno e dei poligoni più vicini ad esso. Infatti, il componente permette la renderizzazione di tre charts:

- <ElevationChart>:

  Il componente è responsabile della rappresentazione del profilo altimetrico dell'itinerario selezionato. Il grafico è configurato per visualizzare le altitudini corrispondenti alle coordinate lungo il percorso.

- <TerrainChart>:

  Questo componente rappresenta il profilo del terreno intorno all'itinerario selezionato. Il grafico a barre mostra la percentuale di ciascun uso del suolo

nell'itinerario, diviso nelle tre macrocategorie: Natura, Acqua ed Artificiale. La percentuale di ciascuna viene calcolata sommando il numero di elementi corrispondenti a ciascuna categoria e poi viene calcolata la percentuale rispetto al totale di ognuna di esse.

- <PolygonsChart>:

  Il componente permette di visualizzare una lista dei dieci poligoni più vicini all'itinerario selezionato, utilizzando un grafico a barre. Calcolando le distanze di tutti i poligoni rispetto all'inizio del percorso, utilizzando la funzione "calculateCenterID" per determinare il centro di ciascun poligono, e la funzione "calculateDistance" per calcolare la distanza tra la partenza ed il centro di ciascun poligono. Successivamente, vengono selezionati i dieci poligoni più vicini e le loro distanze sono utilizzate per popolare il grafico a barre. I nomi dei poligoni vengono tagliati per rendere il grafico più leggibile.

Ogni volta che viene selezionato un itinerario dalla lista, i grafici si aggiorneranno automaticamente con i dati delle corrispondenti statistiche, permettendo una visualizzazione immediata ed accurata delle informazioni rilevanti all'interno dei grafici. L'integrazione dei tre grafici statistici fornisce agli utenti un'analisi dettagliata e multi-prospettica del percorso geografico, offrendo un resoconto utile per la pianificazione o una futura esplorazione.

4.8.3. Sounds

Il componente <Sounds> funge da contenitore per la visualizzazione della sezione relativa all'associazione di suoni e tracce audio alle statistiche presentate nei grafici. Per mostrare la flessibilità e la versatilità nell'elaborazione dei dati geografici per creare un'esperienza sonora coinvolgente e personalizzata, sono stati scelti tre esempi funzionali. Questi offrono una panoramica di come i dati geografici possano essere interpretati ed associati ad elementi sonori in modi creativi ed utili per gli utenti. Il componente, quindi, comprenderà tre componenti figli:

- <ElevationSounds>:

  Questo componente sfrutta il profilo altimetrico del percorso scelto, trasformando le varie altezze delle coordinate del percorso in note musicali. Il risultato è la creazione di una melodia unica e personalizzata, che segue fedelmente le variazioni di altitudine lungo il tragitto selezionato. Effettuando poi un'analisi della melodia, è possibile ricavarne alcune statistiche. Nello specifico, il componente è formato da un Music player, per il controllo della riproduzione della melodia e da un altro componente per visualizzarne le statistiche.

  Per creare musica e suoni interattivi nel browser è stata utilizzata la libreria "Tone.js", un web audio framework che offre familiarità per musicisti e programmatori per la creazione di applicazioni audio basate sul web. Inoltre, Tone.js offre funzionalità DAW (workstation audio digitale) comuni come un trasporto globale per la sincronizzazione e la pianificazione di eventi, sintetizzatori ed effetti predefiniti ma anche complessi e personalizzabili. **[33]**

  I metodi per la creazione della melodia partendo dal profilo altimetrico dell'itinerario

sono stati raggruppati in un file chiamato “Functions.js”, che comprende:

- getNoteFromElevation(elevation):

  La funzione calcola la nota musicale più vicina all’altitudine specificata: la variabile “nearestNote” viene inizializzata con la nota di base “C0”, che è la nota più bassa nella scala musicale. La variabile “minDiff” viene invece inizializzata con la differenza tra l’altitudine specificata e l’altitudine corrispondente alla nota. Poi, la funzione itera su tutte le noti presenti nell’oggetto “notes” dove, per ogni nota, calcola la differenza assoluta tra l’altitudine specificata e l’altitudine corrispondente alla nota. Se la differenza calcolata è minore della differenza minima finora registrata, aggiorna la “nearestNote” con la nota corrente e la “minDiff” con la nuova differenza minima. Alla fine dell’iterazione, la funzione restituisce la nota più vicina all’altitudine specificata.

  L’oggetto “notes” associa le note musicali alle loro rispettive frequenze in Hertz: ogni chiave rappresenta quindi una nota musicale, mentre il valore corrispondente rappresenta la frequenza della nota in Hertz. Ad esempio, la chiave “C4” rappresenta la nota Do di quarta ottava, che ha una frequenza di 261.63 Hertz ed una corrispettiva altezza di 261.63 metri. Questa associazione è stata progettata in modo tale che le altezze delle varie coordinate geografiche possano essere correlate alle note musicali, consentendo così la creazione di una melodia basata sull’altitudine e trasformando le caratteristiche geografiche in elementi musicali. Di seguito, la lista delle note associate alle frequenze *(figura 23)*:

```
"C1": 32.70,    "C2": 65.41,    "C3": 130.81,   "C4": 261.63,
"C#1": 34.65,   "C#2": 69.30,   "C#3": 138.59,  "C#4": 277.18,
"D1": 36.71,    "D2": 73.42,    "D3": 146.83,   "D4": 293.66,
"D#1": 38.89,   "D#2": 77.78,   "D#3": 155.56,  "D#4": 311.13,
"E1": 41.20,    "E2": 82.41,    "E3": 164.81,   "E4": 329.63,
"F1": 43.65,    "F2": 87.31,    "F3": 174.61,   "F4": 349.23,
"F#1": 46.25,   "F#2": 92.50,   "F#3": 185.00,  "F#4": 369.99,
"G1": 49.00,    "G2": 98.00,    "G3": 196.00,   "G4": 392.00,
"G#1": 51.91,   "G#2": 103.83,  "G#3": 207.65,  "G#4": 415.30,
"A1": 55.00,    "A2": 110.00,   "A3": 220.00,   "A4": 440.00,
"A#1": 58.27,   "A#2": 116.54,  "A#3": 233.08,  "A#4": 466.16,
"B1": 61.74,    "B2": 123.47,   "B3": 246.94,   "B4": 493.88,

"C5": 523.25,   "C6": 1046.50,  "C7": 2093.00,  "C8": 4186.01,
"C#5": 554.37,  "C#6": 1108.73, "C#7": 2217.46, "C#8": 4434.92,
"D5": 587.33,   "D6": 1174.66,  "D7": 2349.32,  "D8": 4698.63,
"D#5": 622.25,  "D#6": 1244.51, "D#7": 2489.02, "D#8": 4978.03,
"E5": 659.26,   "E6": 1318.51,  "E7": 2637.02,  "E8": 5274.04,
"F5": 698.46,   "F6": 1396.91,  "F7": 2793.83,  "F8": 5587.65,
"F#5": 739.99,  "F#6": 1479.98, "F#7": 2959.96, "F#8": 5919.91,
"G5": 783.99,   "G6": 1567.98,  "G7": 3135.96,  "G8": 6271.93,
"G#5": 830.61,  "G#6": 1661.22, "G#7": 3322.44, "G#8": 6644.88,
"A5": 880.00,   "A6": 1760.00,  "A7": 3520.00,  "A8": 7040.00,
"A#5": 932.33,  "A#6": 1864.66, "A#7": 3729.31, "A#8": 7458.62,
"B5": 987.77,   "B6": 1975.53,  "B7": 3951.07,  "B8": 7902.13
```

Figura 23: Lista delle note associate alle frequenze

- createMelody(elevationProfile, bpm, synth, synthPart, setMelodyData):

  Prende in input il profilo altimetrico dell'itinerario, il valore dei BPM (battiti per minuto), un oggetto synth che rappresenta il sintetizzatore sonoro utilizzato per generare i suoni, un oggetto synthPart per salvarne lo stato (entrambi forniti dalla libreria Tone.js). La funzione crea quindi una melodia basata sul profilo altimetrico, mappando ogni coordinata in un oggetto che contiene la nota corrispondente ottenuta tramite la funzione "getNoteFromElevation", imponendo una durata fissa di "4n" (quattro note). Successivamente, la funzione calcola alcune statistiche sulla melodia appena creata, maxPitch, minPitch, mostFrequentPitch e leastFrequentPitch, attraverso le funzioni descritte di seguito. La funzione crea un nuovo oggetto "synthParth" che rappresenta la vera e propria sequenza sonora che andrà ad eseguire le note della melodia con il sintetizzatore specificato. Infine, la funzione imposta il valore dei BPM sulla timeline della libreria, avvia la sequenza sonora e restituisce l'oggetto "synthParth" creato. Inoltre, aggiorna i dati della melodia sullo stato del componente tramite la funzione "setMelodyData";

- findMaxPitch(melody):

  La funzione ritorna il valore massimo di pitch trovato nella melodia passata come parametro, mappandone tutte le note;

- findMinPitch(melody):

  La funzione ritorna il valore minimo di pitch trovato nella melodia passata come parametro, mappandone tutte le note;

- findMostFrequentlyPitch(melody):

  La funzione ritorna il valore di pitch più frequente all'interno della melodia passata come parametro, analizzandone il numero attraverso delle semplici iterazioni;

- findLeastFrequentlyPitch(melody):

  La funzione ritorna il valore di pitch meno frequente all'interno della melodia passata come parametro, analizzandone il numero attraverso delle semplici iterazioni.

Il componente viene inizializzato nel suo stato di partenza, includendo se la melodia è in riproduzione, i BPM ed i dati della melodia, che verranno aggiornate appena un itinerario sarà stato scelto dall'utente. Viene anche inizializzato un sintetizzatore audio. Per gestire la riproduzione della melodia è stato inserito un componente <MusicPlayer>, dove viene cambiato lo stato di "Tone.Transport", richiamando i metodi "start()" per avviarla e "stop()" per fermarla. Per aggiornare i BPM dinamicamente, invece, è possibile trascinare uno slider. Infine, il componente <ShowMelodyData> si occupa della visualizzazione delle statistiche della melodia, quali la nota più alta, la nota più bassa, la nota più frequente e quella meno frequente;

- <TerrainSounds>:

  Il componente sfrutta il profilo del terreno di un itinerario per generare una playlist di audio e canzoni in maniera casuale e consente di riprodurla attraverso un player. Nello specifico, nel file "Genres.js" sono stati raggruppati degli oggetti che associano la conformità del terreno, suddivisa nelle tre macrocategorie "Natura"; "Acqua" e "Artificiale", ad una lista di generi musicali che potessero richiamare le sensazioni, le emozioni o le associazioni tipiche di ciascun ambiente. Ad esempio, dei paesaggi naturali potrebbero essere associati a genere musicali come "acoustic", "folk" o "reggae", invocando sensazioni di serenità, tranquillità e connessione con la natura; dei paesaggi acquatici, invece, potrebbero essere correlati a generi come "ocean waves", "ambient techno" o "chillout", richiamando immagini di paesaggi marini, il suono delle onde misto a sensazioni di calma e rilassamento; infine, la categoria "Artificiale" potrebbe essere legata a generi musicali più movimentati come "techno", "hip hop" o "dance", suggerendo un'atmosfera più urbana, energetica e moderna, associata ad ambienti costruiti dall'uomo come città o strutture industriali. In sintesi, l'associazione dei generi musicali alle categorie di terreno offre un modo creativo per arricchire l'esperienza utente durante l'esplorazione degli itinerari, fornendo una colonna sonora che si integra con le caratteristiche ambientali del percorso. Di seguito, la tabella per la suddivisione dei generi *(figura 24)*:

Nature	Water	Artificial
Wind Sound	Ocean Waves	Techno
New Age	Downtempo	Jazz
Acoustic	Chillout	Pop
Classical	Ambient Techno	Hip Hop
Country	Deep House	Punk
Folk	Lo-Fi	Dance
Reggae	Hawaiian	R & B

Figura 24: Suddivisione dei generi in base al terreno

  Il componente, quindi, riceverà i dati del profilo del terreno dell'itinerario selezionato che, aggiornandosi, attraverso la funzione "randomPlaylist" genererà una nuova playlist casuale: la funzione utilizza le percentuali dei diversi tipi di terreno nel profilo per determinare il numero di brani per ciascuna tipologia, utilizzando a sua volta la funzione "getRandomSong" per ottenere una traccia audio casuale per un determinato genere musicale in lista. La funzione "percentages", invece, si occupa di calcolare le percentuali dei diversi tipi di terreno.

  La playlist ottenuta verrà visualizzata nel componente <Playlist> all'interno di una lista, mostrando il nome di ciascuna traccia audio insieme al genere corrispondente.

Infine, il componente <Player> gestisce la riproduzione audio e fornisce i controlli del lettore audio, includendo riproduzione/pausa, traccia successiva/precedente, modalità casuale e modalità di ripetizione. Per eseguire il download asincrono dell'audio ed ascoltarlo in streaming mentre si utilizza l'applicazione, viene eseguita una fetch al server

Express dell'URL "/downloadAudio/fileID", ottenendo così il BLOB che verrà convertito in un HTMLAudioElement, il quale potrà essere riprodotto poiché di formato compatibile con il browser;

- <PolygonsSounds>:

  Il componente è progettato per essere un contenitore per il rendering e la gestione dell'elenco di audio associati ai poligoni vicini ad un determinato itinerario. Ricevendo in input i dati dell'itinerario e dei poligoni più vicini, viene eseguita la funzione "retrieveData", che innanzitutto controlla se "itinerary" e "nearPolygons" sono definiti, per poi richiamare "getAudiosFromPolygon" per recuperare i dati audio associati ai poligoni vicini. Una volta ottenuti, vengono impostati nello stato di "polygonsAudio". Successivamente, per ogni poligono nell'elenco viene visualizzata una lista di audio associati a quel poligono.

  I poligoni più vicini vengono calcolati con una distanza massima di 100 metri dalla partenza dell'itinerario nel componente <PolygonsChart>, per un massimo di dieci elementi.

  In sintesi, questo componente è progettato per mostrare in modo ordinato e comprensibile l'elenco di audio associati ai poligoni più vicini ad un itinerario specifico. L'idea futura è quella di effettuare un fade-in e fade-out di una traccia audio associata ad un poligono quando l'utente si avvicina e si allontana da esso calcolando la posizione corrente del dispositivo utilizzato.

4.9. Music

Il componente <Music> fornisce un'interfaccia che permette la gestione di audio e poligoni ed è organizzato in modo da integrare due sottocomponenti principali: <Songs> e <Polygons>, ciascuno dedicato a delle funzionalità specifiche. Il primo gestisce il caricamento di un file audio su Google Drive e permette di visualizzarne la lista dinamicamente, mentre il secondo permette di ricercare un poligono per nome, caricarlo su Firebase ed associare file audio a questi ultimi.

4.9.1. Gestione file audio

Per quanto riguarda il componente <Songs>, ad esso vengono passati in input la lista degli audio restituiti dal server Express durante l'inizializzazione dell'applicazione e le relative informazioni salvate su Firebase. L'interfaccia è suddivisa in due funzionalità principali, le quali sono raggruppate in un file chiamato "GoogleDrive.js":

- <GoogleUpload>:

  Consente di caricare un file audio con estensione ".mp3/.wav" direttamente dalla memoria del dispositivo, per poi effettuare una richiesta al server Express alla URL "/uploadToGoogleDrive". Se la chiamata avrà successo, il file audio verrà salvato direttamente sulla cartella di Google Drive, poi verrà effettuata una nuova chiamata alla URL "/getAudioFiles" in modo da aggiornare la lista degli audio nello stato del dispositivo. Inoltre, le informazioni relative al file caricato verranno caricate anche su Firebase attraverso il richiamo delle funzioni "saveAudioData" (per il caricamento) e "getAudioData" (per aggiornare i dati nello stato

dell'applicazione).

- <GoogleDownload>:

  Consente di visualizzare la lista dei file audio caricati sulla cartella di Google Drive salvati nello stato dell'app. In questo caso viene utilizzato il widget "Table" della libreria MUI, a cui è stato integrato un metodo di ricerca e filtraggio delle canzoni per titolo. Le colonne mostrano le informazioni di ogni singolo file audio e comprendono il titolo, il genere, la tipologia (ovvero l'estensione del file), un bottone per effettuare il download della canzone nel dispositivo ed un altro bottone per l'eliminazione. Queste ultime due operazioni sono gestite da due richieste al server Express: la prima effettua una fetch alla URL "/downloadAudio/fileID", restituendo il file BLOB dell'audio scelto che verrà poi convertito in un elemento HTML e scaricato in automatico nella memoria del dispositivo; la seconda invece, tenta una fetch alla URL "/deleteAudio/fileID" e darà come risultato l'eliminazione del file audio nella cartella di Google Drive, l'eliminazione dei relativi dati su Firebase attraverso la funzione "deleteAudioData" ed il relativo aggiornamento dei dati nello stato dell'applicazione.

4.9.2. Gestione poligoni

Questo componente si occupa della gestione dei poligoni all'interno del database di Firebase e delle relative informazioni per l'assegnazione di file audio ad essi.

Per quanto riguarda l'upload, il componente <SavePolygon> si occupa di mostrare una lista di città/stati attraverso una ricerca su base input: viene richiamata la funzione "searchCitiesAndCountries", che effettua una query a Nominatim per geolocalizzarne i risultati corrispondenti. I poligoni corrispondenti verranno visualizzati in una lista e da qui, cliccando su quello desiderato, verrà richiamata la funzione "searchPolygonByOsmId", dato che l'ID viene salvato in automatico con la prima query. Il risultato di quest'ultima sarà il GeoJSON del poligono con l'ID corrispondente nel database di OSM, che verrà salvato nello storage di Firebase attraverso la funzione "savePolygon".

La lista dei poligoni salvati su Firebase verrà mostrata attraverso il componente <PolygonList>, il quale implementa la funzionalità di eliminare un poligono dal database richiamando la funzione "deletePolygons" ed aggiornando lo stato dell'applicazione.

Infine, l'ultima funzionalità riguarda l'assegnazione di uno o più file audio ad uno specifico poligono: cliccando su uno di essi, si aprirà un pop-up contenente una lista a scelta multipla di tutti i file audio salvati su Google Drive. Una volta confermato, i file audio selezionati verranno salvati richiamando la funzione "addAudioToPolygon".

4.10. Emulator

Il componente <Emulator> fornisce un'interfaccia per effettuare una vera e propria evoluzione temporale di un itinerario scelto dall'utente, mostrando all'interno della mappa il routing dell'itinerario e la posizione corrente durante la simulazione. Inoltre, verranno mostrati a schermo alcune delle statistiche principali delle coordinate e verrà

creata una playlist di audio che partirà quando l'utente inizierà la simulazione. L'interfaccia, quindi, è suddivisa in tre sottocomponenti:

- <EmulatorMap>: come per il componente <Maps>, renderizza la mappa da Leaflet e gestisce la lista dei marker a seconda dell'itinerario selezionato, creando una route. Inoltre, quando viene selezionato un itinerario, l'utente potrà cliccare un pulsante di start/stop per iniziare e fermare la simulazione del percorso. Tenendo traccia dell'index della coordinata corrente durante l'evoluzione, verranno richiamate le funzioni "getCurrentElevation" e "getCurrentNote" per mostrare a schermo le proprie specifiche, a mano a mano che la simulazione avverrà. Infine, verrà generata una nuova playlist random passando la lista dei file audio presenti nel database, che potrà essere controllata attraverso il player;

- <EmulatorRoute>: come per il componente <Routing>, permette di utilizzare Leaflet Routing Machine per creare la route del percorso;

- <EmulatorStats>: come per il componente <Stats>, mostra in una lista tutti gli itinerari salvati sul database e permette di effettuare il download e salvataggio sullo stato dell'applicazione dei profili altimetrici e del terreno del percorso scelto.

Per questo componente si è deciso di riutilizzare del codice già presente all'interno del progetto, ma separandolo definitivamente per poter apportare dei cambiamenti necessari per la simulazione, consentendo una facilitazione per eventuali future modifiche.

4.11. Problematiche e soluzioni

Durante la fase di progettazione, una delle principali problematiche riscontrate è stata la limitata disponibilità di risorse online riguardanti l'uso e l'implementazione delle API di OpenStreetMap. Sebbene esso sia un progetto collaborativo di notevole ampiezza e diversità, il suo sviluppo continuo implica una documentazione non sempre esaustiva o facilmente accessibile. Un punto di svolta significativo è stato l'impiego di Leaflet Routing Machine per la creazione di percorsi. Questo strumento si è rivelato fondamentale per calcolare il tragitto più breve tra due punti all'interno della mappa interattiva, superando così le limitazioni incontrate con altre risorse e migliorando l'efficienza del progetto.

Un'ulteriore problematica riscontrata è stata l'estrazione dei dati sul terreno utilizzando i tag "landuse" e "landcover" da OpenStreetMap. La complessità deriva dalla vasta gamma di tag, nodi, aree e relazioni, che rende estremamente difficile organizzare tali informazioni in macrocategorie di tipologie di terreno distinte. Questa varietà complica non solo la classificazione, ma anche la gestione ed il salvataggio dei dati nel database sotto forma di JSON, poiché tali dati non si conformano facilmente ad un modello predefinito. Attraverso un'analisi dettagliata di ciascun tag ed una conseguente organizzazione delle tre macrocategorie, è stato possibile ottenere una classificazione più strutturata e gestibile.

Oltre alle difficoltà riscontrate nella ricerca di soluzioni per l'utilizzo delle API di OpenStreetMap, in particolare per ottenere informazioni sull'altitudine e sulle caratteristiche del terreno, sono emerse anche delle limitazioni operative. Per evitare di superare il limite di richieste al secondo imposto dalle API durante la generazione del

profilo altimetrico di un percorso, è stato necessario impostare un limite massimo di 2000 coordinate per ogni itinerario creato. Una strategia analoga è stata adottata per l'acquisizione dei dati relativi al terreno circostante una coordinata specifica, suddividendo la lista delle coordinate in batch, al fine di rispettare i limiti imposti dalle API e garantire un funzionamento efficiente del sistema.

La gestione dei file audio ha presentato notevoli complessità. Inizialmente, il database utilizzato era esclusivamente Firebase, con i file audio salvato nello storage insieme al file JSON dei poligoni. Tuttavia, la necessità di accedere anche a soli dieci file audio al giorno risultava problematica, a causa del limite di 1 GiB di trasferimento giornaliero, un ostacolo significativo durante la fase di progettazione. Questo limite era particolarmente restrittivo poiché l'applicazione richiedeva frequenti riavvii per apportare modifiche o correggere errori. La soluzione individuata è stata l'integrazione di un server Express connesso a Google Drive, il quale offre un numero significativamente maggiore di chiamate in lettura e scrittura giornaliere, facilitando così lo sviluppo e la manutenzione dell'applicazione.

Infine, la ricerca di soluzioni per la lettura e la scrittura di un file audio all'interno di una cartella di Google Drive si è rivelata particolarmente onerosa. La documentazione disponibile online su questo argomento è scarsa e frammentaria, rendendo difficile individuare best practices ed implementazioni efficienti. L'integrazione di Google Drive per la gestione del file audio ha richiesto un'analisi approfondita delle API di Google Drive e lo sviluppo di metodi personalizzati per gestire l'autenticazione, le autorizzazioni concesse dalla Google Developer Console e le operazioni di I/O. Questo processo ha comportato numerosi tentativi ed errori per garantire che i file audio potessero essere caricati, letti e gestiti in modo coerente all'interno dell'applicazione.

5. SCREENSHOT DELLA WEB APP

In questo capitolo verranno mostrati gli screenshot delle interfacce e delle sue funzionalità durante il funzionamento dell'applicazione.

Routing tra due marker inseriti nella mappa interattiva nell'interfaccia "Map"

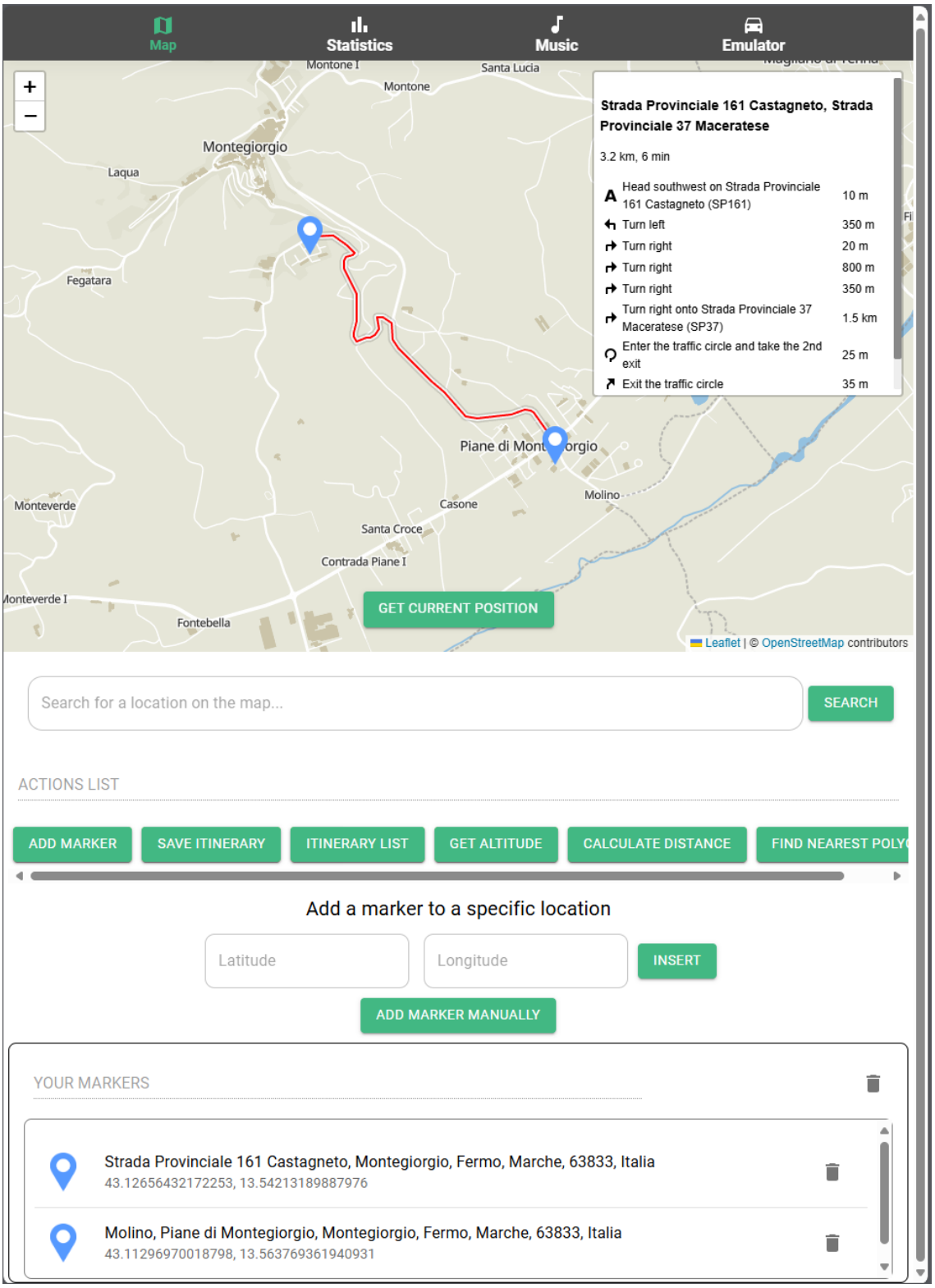


Grafico del profilo altimetrico dell'itinerario nell'interfaccia "Statistics"

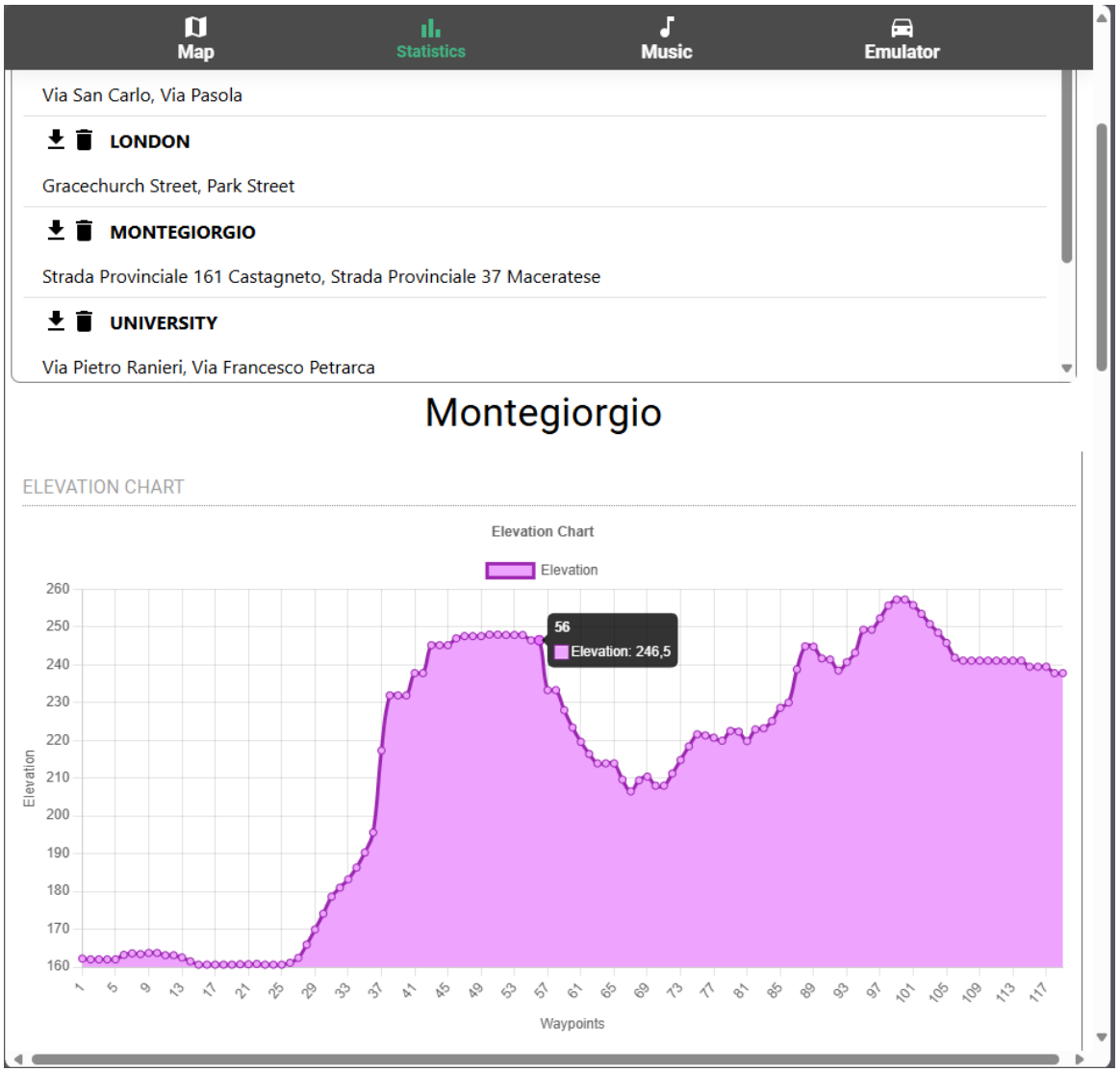


Grafico del profilo del terreno dell'itinerario

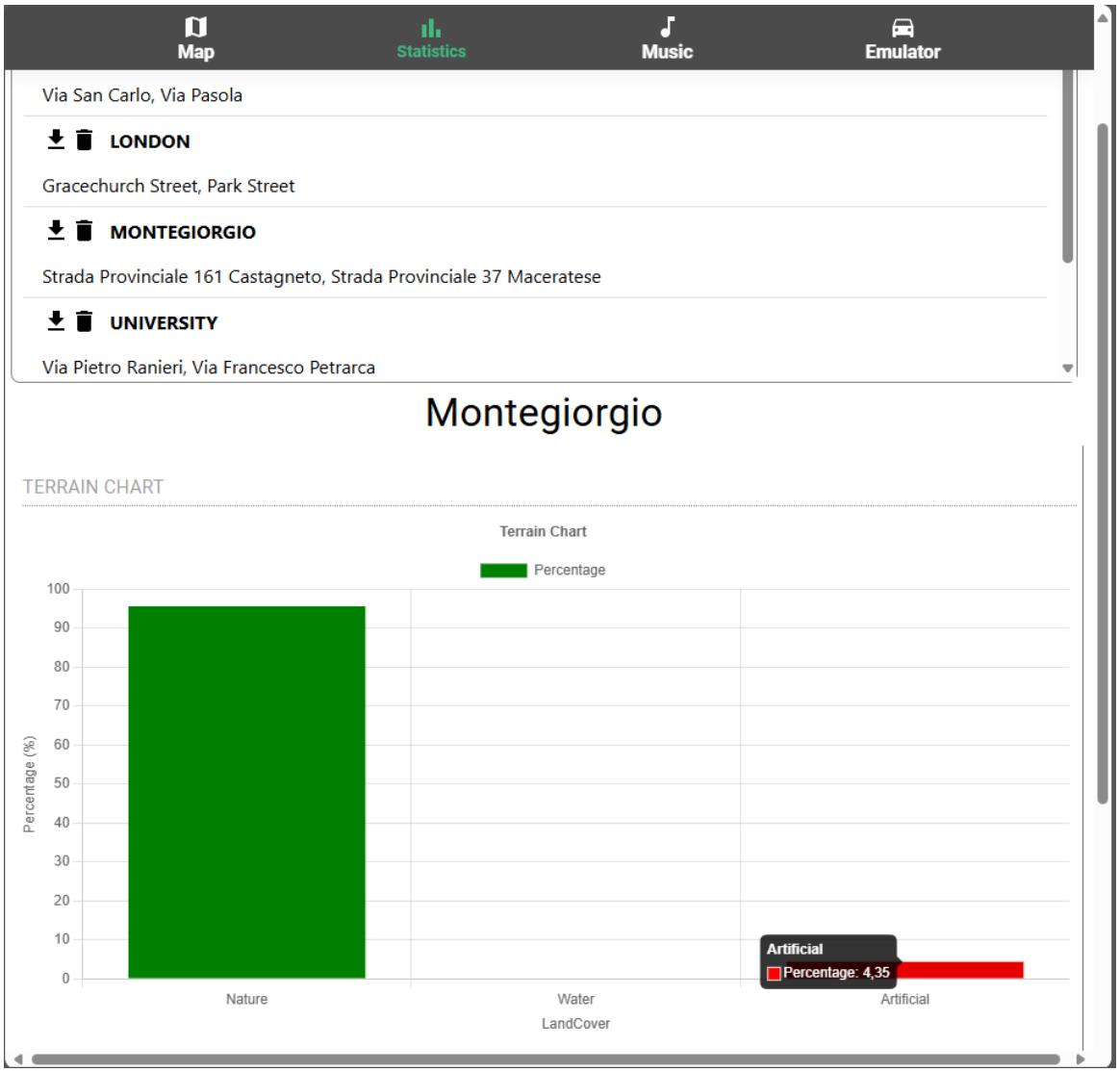
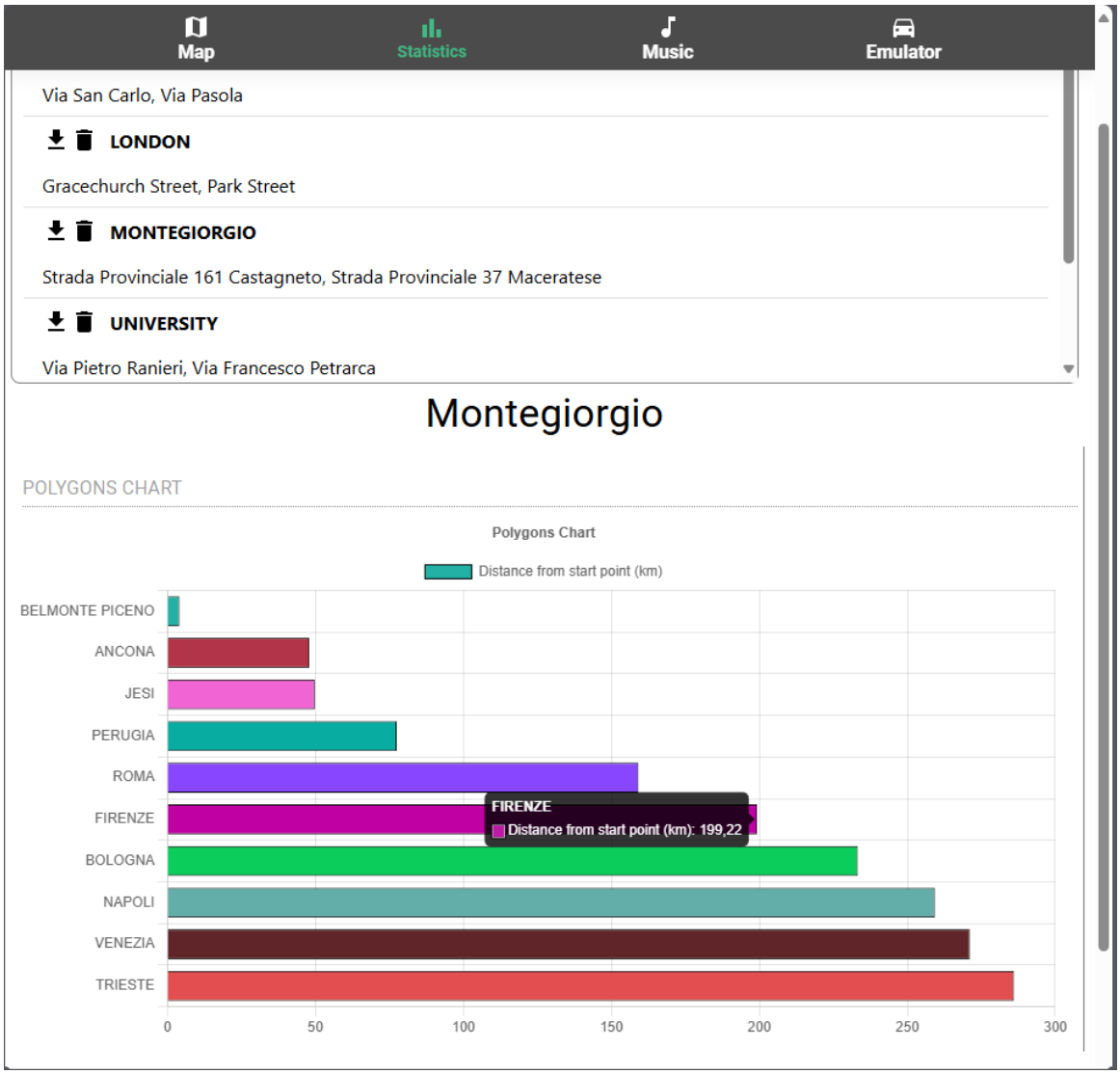
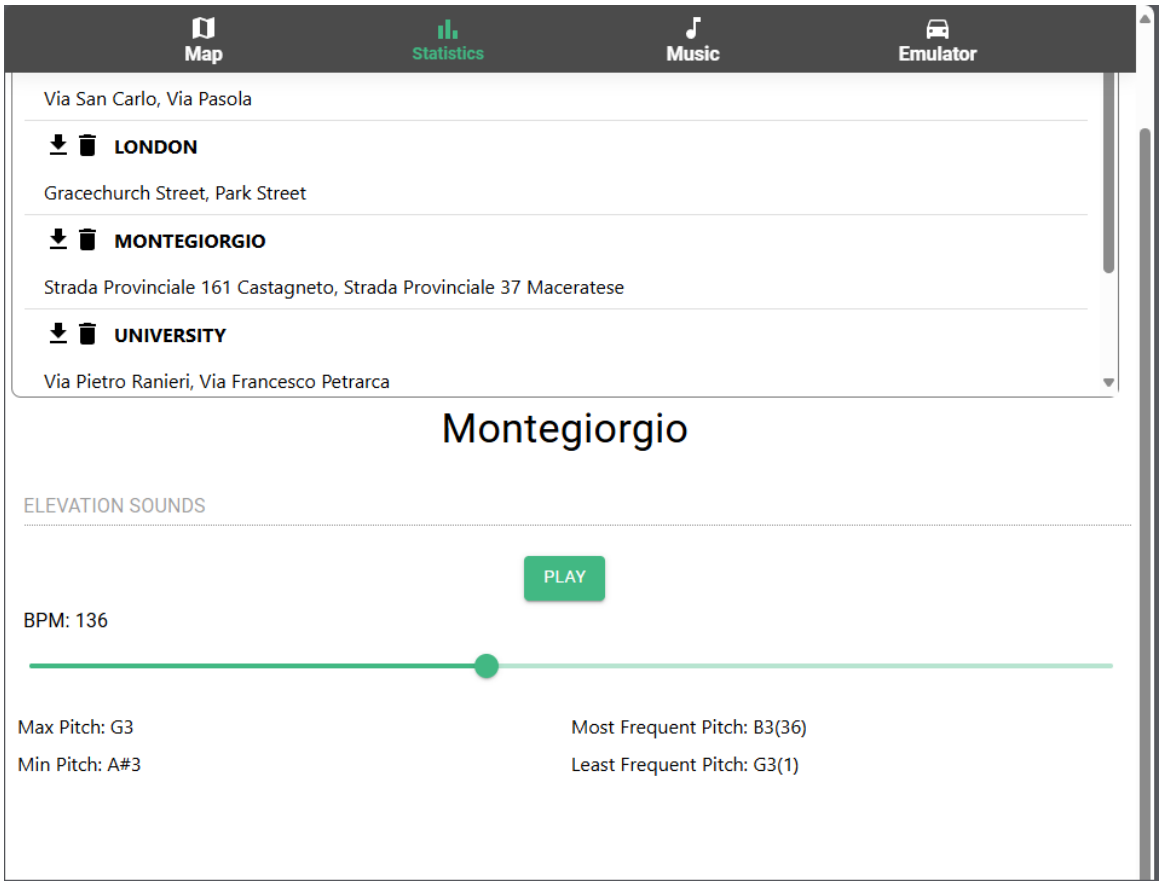


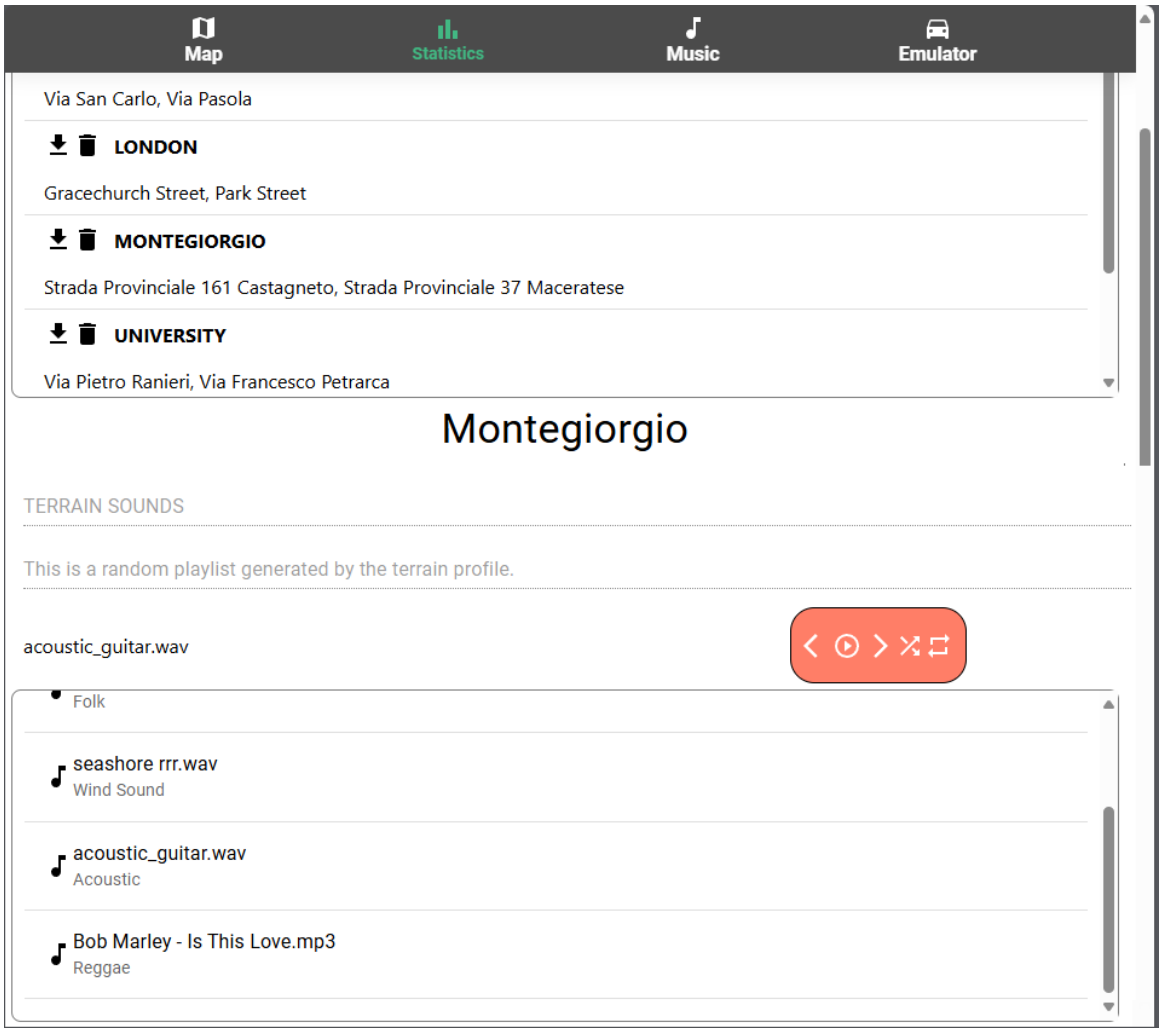
Grafico dei poligoni più vicini rispetto all'itinerario



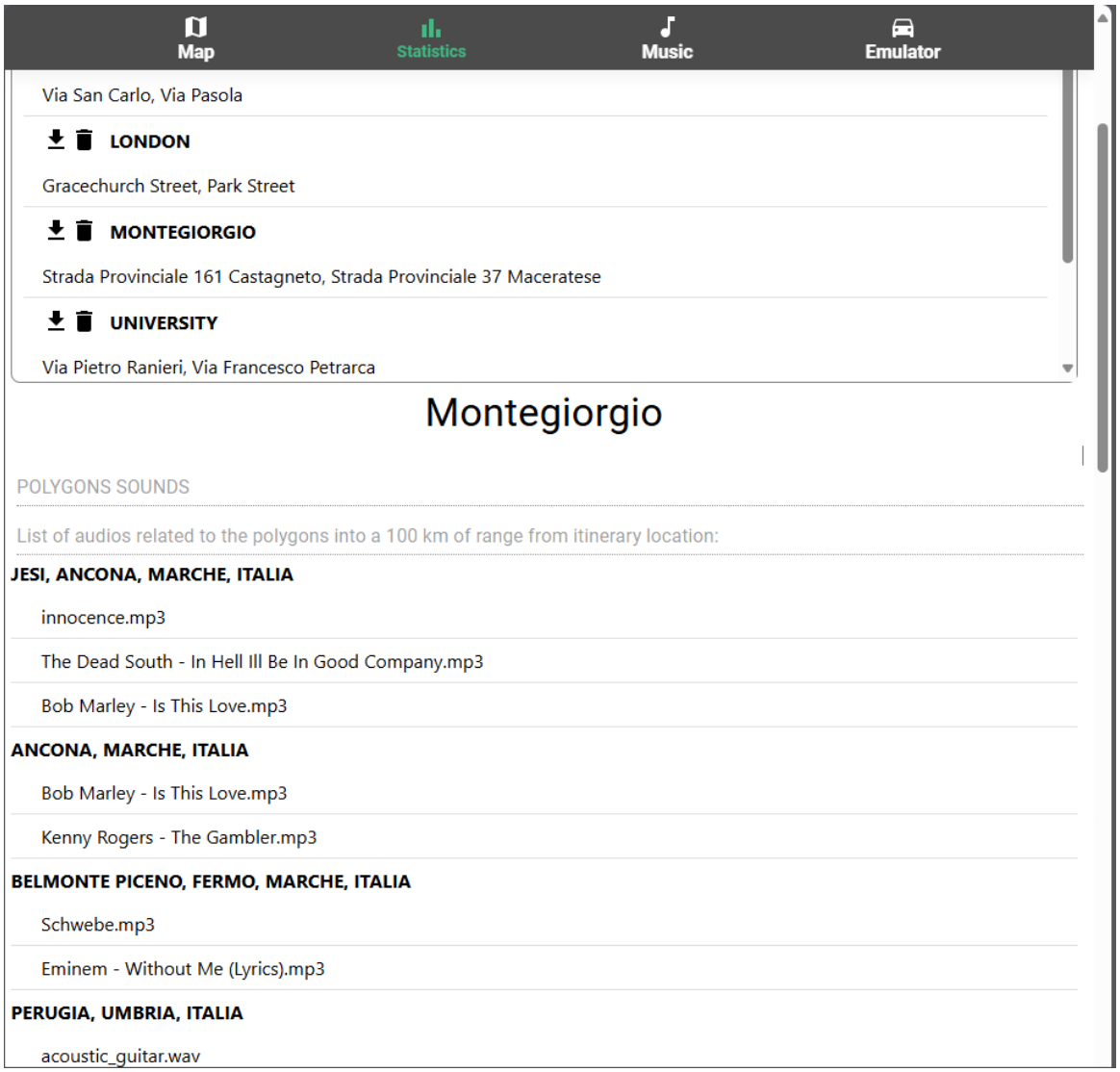
Suoni e statistiche relative al profilo altimetrico dell'itinerario



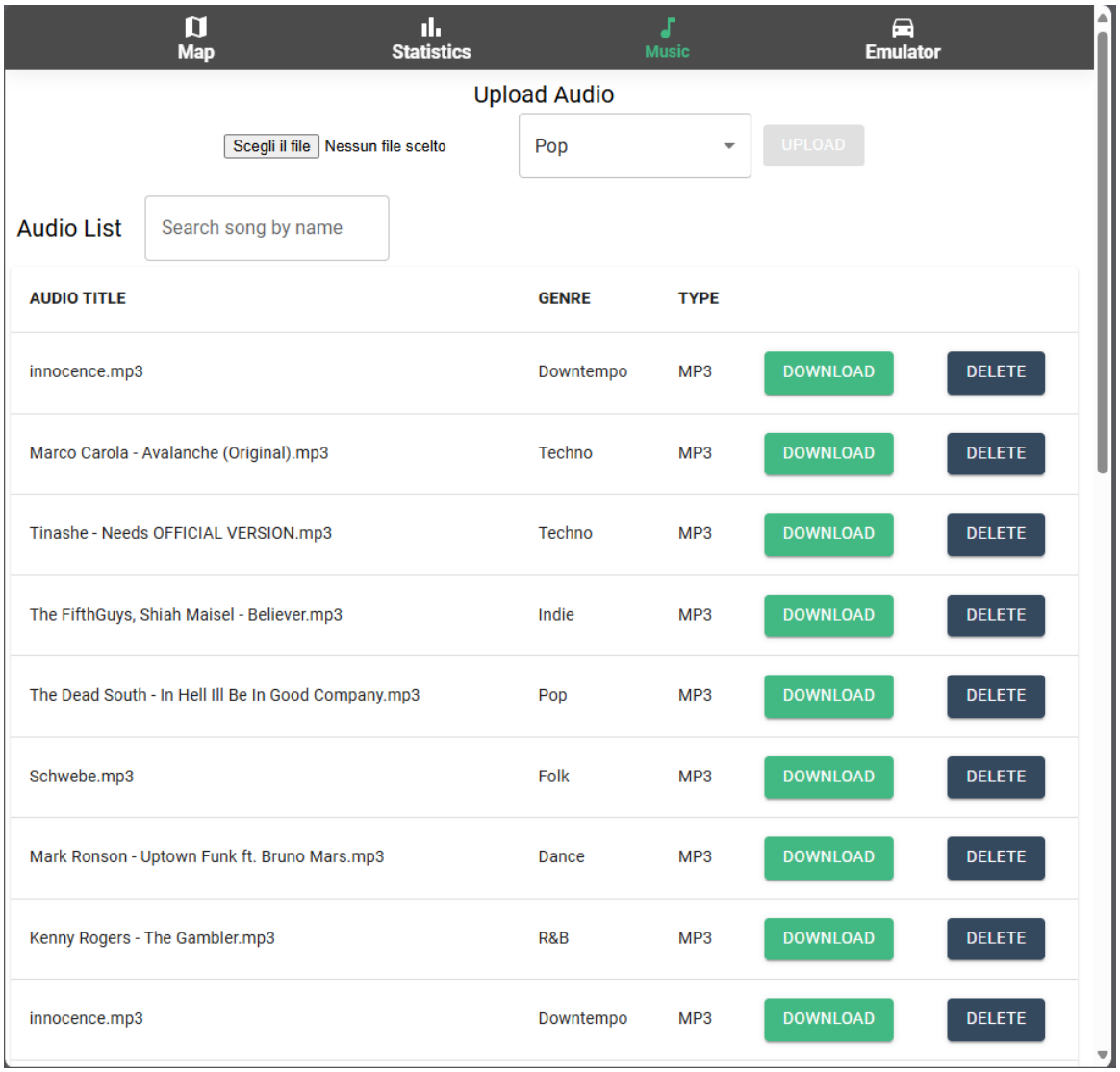
Playlist generata dal profilo del terreno dell'itinerario



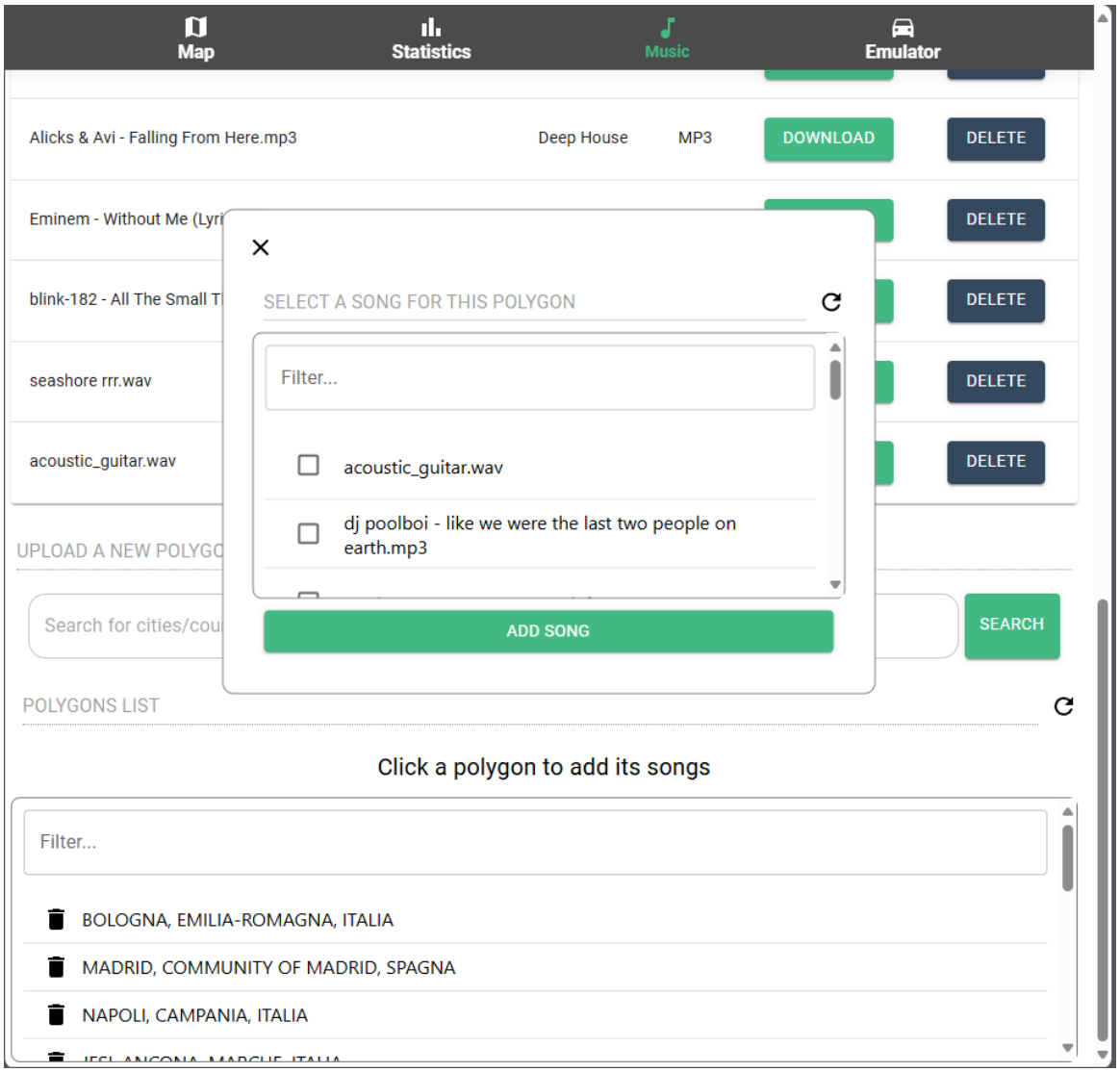
File audio associati ai poligoni più vicini all'itinerario



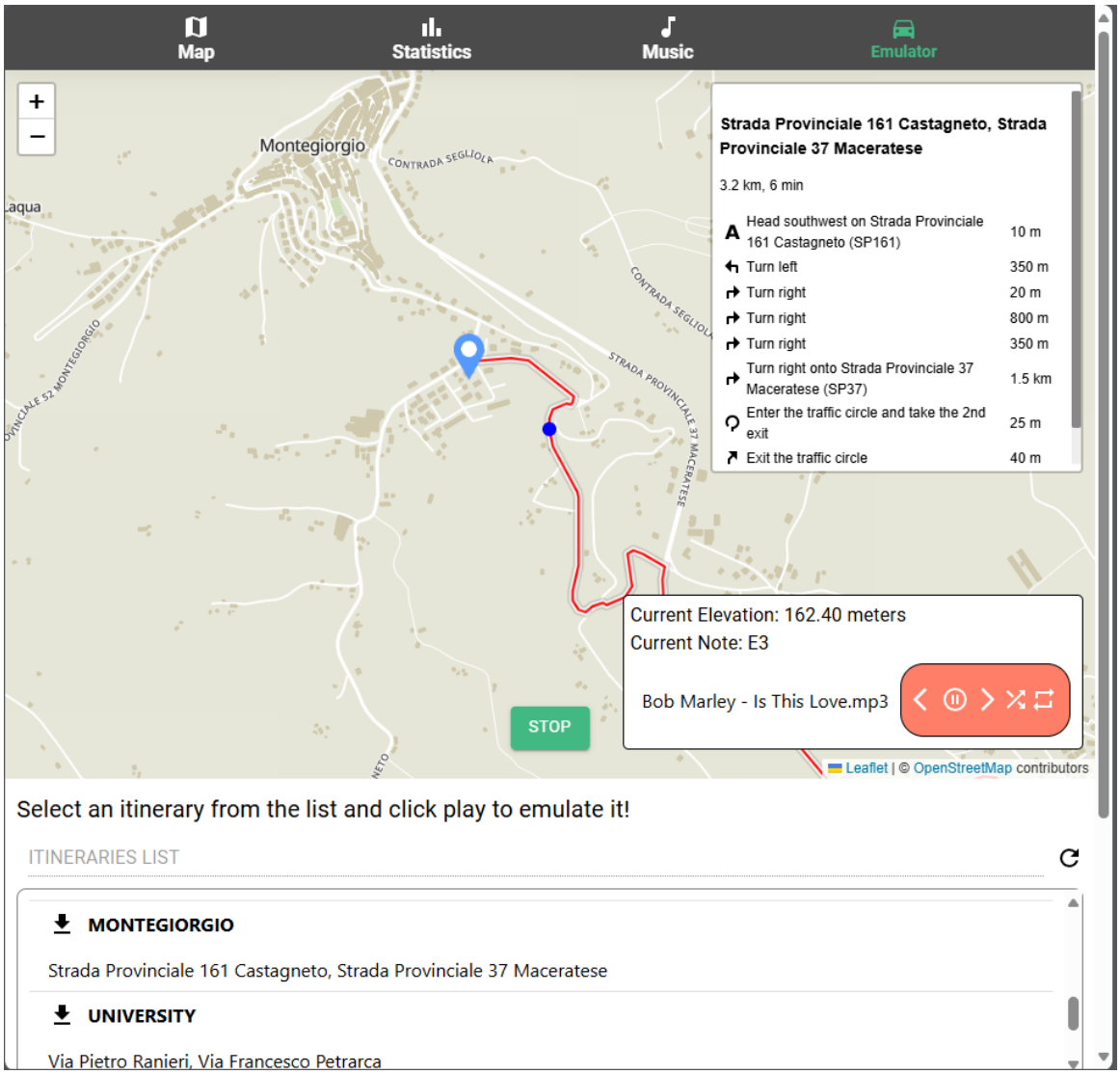
Caricamento di un file audio e lista degli audio salvati nell'interfaccia "Music"



Gestione dei poligoni ed associazione dei file audio



Simulazione dell'itinerario nell'interfaccia "Emulator"



6. CONCLUSIONI

In questo elaborato è stato presentato lo sviluppo di una web app innovativa dedicata alla sonorizzazione di itinerari geografici. La web app mira a trasformare i dati geografici in esperienze sonore, permettendo agli utenti di vivere i percorsi in modo interattivo e coinvolgente. Il progetto ha coinvolto una serie di fasi complesse, dalla definizione dei requisiti alla progettazione ed implementazione delle funzionalità principali. Inizialmente, è stato necessario scegliere le architetture ed i linguaggi di programmazione più appropriati da utilizzare. La scelta è ricaduta su React per il front-end, nonostante non fosse stato trattato durante il corso di studi, e su JavaScript per la parte di script del codice. React è stato selezionato per la sua capacità di creare interfacce utente dinamiche ed efficienti, nonostante la necessità di un ulteriore sforzo per apprenderne l'uso.

Durante lo sviluppo, sono state riscontrate molte difficoltà tecniche. Tra le principali vi è stata sicuramente l'integrazione dei file audio, resa complessa prima dalle limitazioni di trasferimento dati giornaliero di Firebase, poi dalla necessità di implementare soluzioni personalizzate per la gestione di questi file tramite Google Drive. Inoltre, la creazione di un'interfaccia utente intuitiva che permettesse una facile interazione con la mappa e le funzionalità sonore è stata cruciale per il successo dell'applicazione.

A conclusione del lavoro, si può dire che il prodotto finale soddisfi tutti i requisiti che sono stati scelti di integrare durante la fase di progettazione. Gli obiettivi futuri per questa web app includono l'espansione delle sue funzionalità per coprire una gamma più ampia di scenari, grazie alla possibilità di estendere l'applicazione facilmente attraverso la modularità della struttura, favorendo una collaborazione con una comunità più ampia di sviluppatori, utenti, studenti e ricercatori. Tra le possibilità, si considerano: integrazione con la realtà aumentata (AR), per sovrapporre informazioni visive e sonore sugli itinerari fisici, arricchendo ulteriormente l'esperienza utente; supporto multilingue, che permetta di utilizzare l'app in diverse lingue; personalizzazione avanzata dei percorsi, permettendo agli utenti di condividere i propri itinerari sonori personalizzati, integrando punti di interesse specifici e preferenze musicali; analisi dei dati, per fornire agli utenti statistiche più dettagliate sui loro percorsi; integrazione con dispositivi wearable, come ad esempio orologi, per offrire una navigazione sonora più immersiva ed hands-free.

In sintesi, questo progetto rappresenta un passo significativo verso l'integrazione della tecnologia sonora con i dati geografici, aprendo nuove possibilità per la fruizione interattiva di itinerari e percorsi. Le difficoltà incontrate hanno fornito preziose lezioni e spunti per il miglioramento continuo, ponendo solide basi per ulteriori sviluppi ed innovazioni future.

7. BIBLIOGRAFIA

[1] «Wikipedia,» [Online]. Available: https://it.wikipedia.org/wiki/Applicazione_web. [Consultato il giorno 13 Giugno 2024].

[2] «Visual Studio Code,» Microsoft, [Online]. Available: https://code.visualstudio.com/. [Consultato il giorno 13 Giugno 2024].

[3] «Wikipedia,» [Online]. Available: https://it.wikipedia.org/wiki/Visual_Studio_Code. [Consultato il giorno 13 Giugno 2024].

[4] «React,» Meta Open Source, [Online]. Available: https://react.dev/. [Consultato il giorno 13 Giugno 2024].

[5] «Wikipedia,» [Online]. Available: https://it.wikipedia.org/wiki/React_(web_framework). [Consultato il giorno 13 Giugno 2024].

[6] «HTML,» WHATWG, [Online]. Available: https://html.spec.whatwg.org/multipage/. [Consultato il giorno 13 Giugno 2024].

[7] «Wikipedia,» [Online]. Available: https://it.wikipedia.org/wiki/HTML. [Consultato il giorno 13 Giugno 2024].

[8] B. Bos, «W3,» W3C, 30 Maggio 2024. [Online]. Available: https://www.w3.org/Style/CSS/. [Consultato il giorno 13 Giugno 2024].

[9] «Node.js,» OpenJS Fundation, [Online]. Available: https://nodejs.org/en. [Consultato il giorno 13 Giugno 2024].

[10] «Wikipedia,» [Online]. Available: https://it.wikipedia.org/wiki/JavaScript. [Consultato il giorno 13 Giugno 2024].

[11] «OpenStreetMap,» OpenStreetMap Foundation, [Online]. Available: https://www.openstreetmap.org/about. [Consultato il giorno 13 Giugno 2024].

[12] «Firebase,» Google, [Online]. Available: https://firebase.google.com/. [Consultato il giorno 13 Giugno 2024].

[13] «Wikipedia,» [Online]. Available: https://it.wikipedia.org/wiki/Firebase. [Consultato il giorno 13 Giugno 2024].

[14] «Google Drive,» Google, [Online]. Available: https://drive.google.com/drive/u/0/home. [Consultato il giorno 13 Giugno 2024].

[15] «Wikipedia,» [Online]. Available: https://it.wikipedia.org/wiki/Google_Drive. [Consultato il giorno 13 Giugno 2024].

[16] «Express.js,» Fondazione OpenJS, [Online]. Available: https://expressjs.com/it/. [Consultato il giorno 13 Giugno 2024].

[17] «Wikipedia,» [Online]. Available: https://it.wikipedia.org/wiki/Express.js. [Consultato il giorno 13 Giugno 2024].

[18] «Wikipedia,» [Online]. Available: https://it.wikipedia.org/wiki/Node.js. [Consultato il giorno 13 Giugno 2024].

[19] «Github,» Github Inc, [Online]. Available: https://github.com/. [Consultato il giorno 13 Giugno 2024].

[20] «Material-UI,» Material UI SAS, [Online]. Available: https://mui.com/. [Consultato il giorno 13 Giugno 2024].

[21] «JSON,» [Online]. Available: https://www.json.org/json-it.html. [Consultato il giorno 13 Giugno 2024].

[22] «React Router DOM,» Remix Software Inc, [Online]. Available: https://reactrouter.com/en/main. [Consultato il giorno 13 Giugno 2024].

[23] «Wikipedia,» [Online]. Available: https://it.wikipedia.org/wiki/Binary_large_object. [Consultato il giorno 13 Giugno 2024].

[24] V. Agafonkin, «Leaflet,» OpenStreetMap, [Online]. Available: https://leafletjs.com/. [Consultato il giorno 13 Giugno 2024].

[25] «Nominatim,» OpenStreetMap Foundation, [Online]. Available: https://nominatim.org/. [Consultato il giorno 13 Giugno 2024].

[26] P. Liedman, «Leaflet Routing Machine,» Creative Commons, [Online]. Available: https://www.liedman.net/leaflet-routing-machine/. [Consultato il giorno 13 Giugno 2024].

[27] J. R. Lourenço, «Open-Elevation,» Creative Commons, [Online]. Available: https://open-elevation.com/. [Consultato il giorno 13 Giugno 2024].

[28] «Turf.js,» Turf org, [Online]. Available: https://turfjs.org/. [Consultato il giorno 13 Giugno 2024].

[29] «Overpass API,» Creative Commons, [Online]. Available: https://wiki.openstreetmap.org/wiki/Overpass_API. [Consultato il giorno 13 Giugno 2024].

[30] Vaclavjanak e Bololllo, «Github,» [Online]. Available: https://github.com/maptiler/samples/tree/main/cloud/elevation-profile. [Consultato il giorno 13 Giugno 2024].

[31] «MapTiler,» MapTiler, [Online]. Available: https://www.maptiler.com/. [Consultato il giorno 13 Giugno 2024].

[32] «Chart.js,» [Online]. Available: https://www.chartjs.org/. [Consultato il giorno 13 Giugno 2024].

[33] «Tone.js,» [Online]. Available: https://tonejs.github.io/. [Consultato il giorno 13 Giugno 2024].