POLITECNICO DI TORINO

Corso di Laurea Magistrale in
INGEGNERIA INFORMATICA

Tesi di Laurea Magistrale

Sviluppo di un'Applicazione Web per la Visualizzazione di Dati Georeferenziati con ArcGIS JS API, Spring Boot e React

Relatori	Candidati
Prof. Giovanni MALNATI	Matteo FONTANA
Prof. Daniele APILETTI	Nicolò FONTANA

Ottobre 2024

Abstract

L'analisi dei dati territoriali può essere complessa, poiché la quantità e la varietà delle informazioni raccolte spesso ne ostacolano una facile interpretazione. Le informazioni geografiche, come misurazioni relative a risorse naturali o dati ambientali, si prestano particolarmente bene a una visualizzazione tramite mappe tridimensionali.

Un esempio può essere la rappresentazione di dati relativi ad eventi sismici: l'intensità del sisma viene visualizzata attraverso poligoni che illustrano sia la forza dell'evento che l'estensione dell'area colpita. Ciò facilita l'individuazione di pattern e tendenze, rendendo l'analisi dei fenomeni territoriali più accessibile e immediata.

In questa tesi è stata sviluppata un'applicazione che offre una visualizzazione grafica intuitiva, favorendo analisi interattive e aggregazioni efficaci dei dati. Il sistema è basato su un'architettura a microservizi, che comprende PostgreSQL come database relazionale per una gestione robusta dei dati, e Spring Boot per le funzionalità di backend, necessario a gestire le richieste degli utenti e l'interazione con il database.

L'interfaccia utente, sviluppata con React, utilizza ArcGIS JS per visualizzare simultaneamente più fonti di dati su una mappa 3D, associando a ciascuna uno stile personalizzato. Per evitare che file di grandi dimensioni rallentino le prestazioni dell'applicazione web, abbiamo utilizzato GeoServer per suddividere i file raster in piccole porzioni ("tile") e renderizzare solo quelle necessarie.

L'applicazione è stata sviluppata con un'architettura containerizzata tramite Docker e Kubernetes, garantendo scalabilità e flessibilità. Inoltre, per garantire la sicurezza, è stato implementato un sistema di accesso basato su OAuth 2, con diversi livelli di autorizzazione per amministratori e utenti.

Attualmente, è in fase di sviluppo un'app mobile per facilitare l'inserimento dei dati direttamente sul campo, migliorando l'efficienza e l'accessibilità del sistema.

Il risultato ottenuto è un sistema open source che può essere utilizzato per l'analisi di dati territoriali, applicabile in diversi ambiti, quali il monitoraggio ambientale, la gestione delle risorse naturali e lo studio di fenomeni geofisici.

Sommario

Nell'ambito di questa tesi è stata sviluppata un'applicazione web per la visualizzazione interattiva di dati georeferenziati, con l'obiettivo di facilitare l'analisi e l'interpretazione delle informazioni spaziali. L'applicazione si concentra sull'analisi di dati territoriali complessi, che richiedono una rappresentazione grafica efficace per garantire una comprensione immediata e intuitiva. L'uso di strumenti di visualizzazione avanzati è essenziale per migliorare la fruibilità dei dati e supportare decisioni informate.

Architettura del Sistema

L'architettura del sistema è basata su un approccio a microservizi, che consente modularità e scalabilità. Ciascun microservizio può infatti essere replicato più volte per gestire carichi crescenti di lavoro. Questa architettura è composta da vari componenti chiave, come illustrato nella figura:

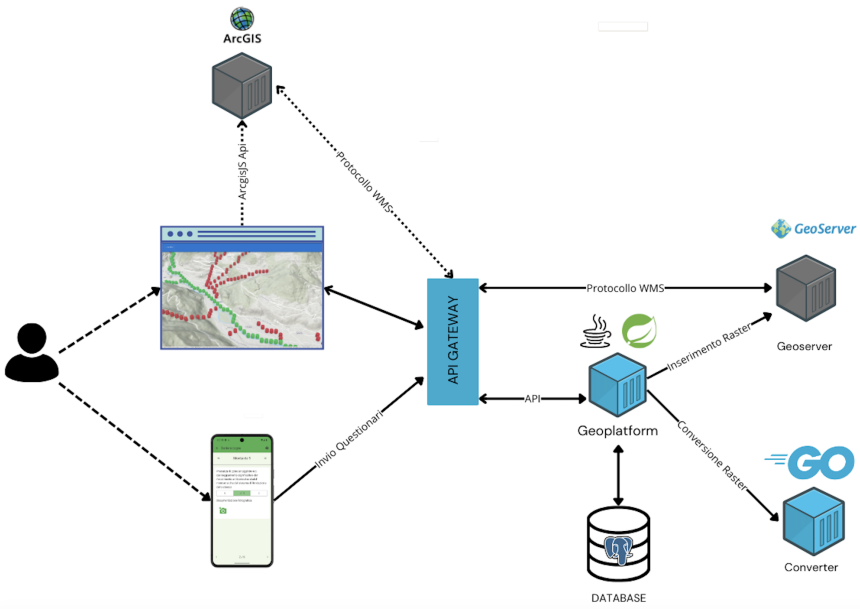


Figura 1: Architettura del sistema

GeoPlatform

GeoPlatform è il server backend dell'applicazione, sviluppato con Spring Boot e Java, e ha il compito di gestire l'interazione tra i vari componenti del sistema. Di seguito sono riportate alcune delle funzionalità principali di GeoPlatform:

- **Gestione login e autenticazione**: implementa la gestione degli utenti con autenticazione OAuth 2, consentendo livelli di accesso differenziati per amministratori, manager e utenti anonimi.
- **Upload**: permette agli utenti di caricare file raster e vettoriali, gestendo il salvataggio dei dati nel file system e nel database. L'upload supporta file di grandi dimensioni attraverso un sistema di upload chunked, che suddivide il file in piccole parti e lo invia al server in modo asincrono, garantendo maggiore stabilità e velocità.
- **Gestione dati**: fornisce funzionalità per visualizzare, aggiornare ed eliminare i dati caricati dagli utenti.
- **Creazione layer vettoriali**: consente la creazione di "geoitem", contenenti dati vettoriali e relative informazioni, come gli stili da applicare e le categorie di appartenenza. I dati vettoriali sono memorizzati in PostgreSQL, con i percorsi dei file nel file system.
- **Creazione layer raster**: coordina la creazione di layer raster in GeoServer per la visualizzazione dei dati sulla mappa. Prima di essere caricati su GeoServer, i file raster vengono convertiti nella corretta proiezione geografica tramite il servizio di conversione.
- **Gestione stili**: gestisce gli stili personalizzati per i dati vettoriali, consentendo agli utenti di creare e applicare stili ai layer vettoriali.

GeoServer

GeoServer è responsabile della visualizzazione e distribuzione dei dati raster e vettoriali tramite il protocollo **WMS** (Web Map Service). Quando viene creato un nuovo *geoitem raster*, GeoPlatform coordina la creazione di un layer nel GeoServer per la visualizzazione sulla mappa.

Converter

Il **Converter** è un microservizio sviluppato in Go, che utilizza **GDAL** (Geospatial Data Abstraction Library) per convertire i file nelle proiezioni geografiche corrette (CRS - Coordinate Reference System). Prima di inviare un dato raster o vettoriale a GeoServer, il Converter si assicura che i dati siano nella CRS richiesta.

PostgreSQL

Il database **PostgreSQL** viene utilizzato per memorizzare dati geospaziali e informazioni non spaziali. I dati vettoriali sono memorizzati all'interno di PostgreSQL, con i relativi percorsi dei file nel file system.

Frontend e Interfaccia Utente

Il frontend dell'applicazione è realizzato con **React**, una libreria JavaScript progettata per creare interfacce utente dinamiche e interattive. Grazie all'integrazione della libreria **ArcGIS JavaScript API**, è possibile visualizzare simultaneamente diverse fonti di dati geospaziali su una mappa 3D interattiva. Gli utenti possono eseguire operazioni di zoom, panoramiche e selezione di elementi specifici, interagendo con i dati in maniera intuitiva. Abbiamo dedicato particolare attenzione alla personalizzazione degli **stili di visualizzazione** dei dati. La nostra implementazione consente agli utenti di creare stili personalizzati per i dati vettoriali, adattando così la visualizzazione alle proprie

esigenze. Ogni tipo di layer può essere rappresentato attraverso poligoni, marker o forme tridimensionali, facilitando la distinzione visiva tra i vari dataset.

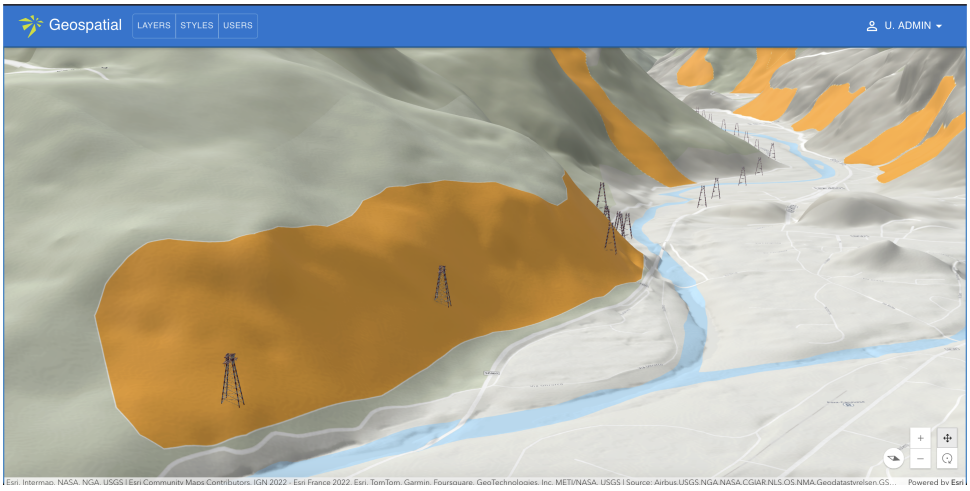


Figura 2: Mappa che mostra contemporaneamente tralicci e zone a rischio frana

In questa immagine, la mappa visualizza simultaneamente i tralicci, rappresentati da icone personalizzate, e le zone a rischio frana, evidenziate con marker arancioni. Questa visualizzazione permette agli utenti di analizzare i dati in modo efficace, identificando facilmente aree dove i tralicci possono essere a rischio.

Integrazione con l'App Mobile

Dopo una fase iniziale di sviluppo dell'applicazione web, il progetto si è evoluto per includere un interfaccia per la raccolta di dati sul campo. Ciò ci è stato richiesto da enti che si occupano di monitoraggio delle barriere paramassi in zone montane. A tal fine, è stata sviluppata un'app mobile per **Android**, che permette di catalogare e gestire informazioni su **barriere paramassi**. L'app mobile, costruita con **Kotlin** e **Jetpack Compose**, offre un'interfaccia intuitiva e moderna, semplificando la raccolta dei dati. Gli utenti possono inserire dati come la posizione geografica e le condizioni strutturali direttamente dal loro dispositivo così come accludere fotografie per documentare lo stato dei fatti. I dati raccolti tramite l'app mobile vengono inviati al nostro server per l'elaborazione e la visualizzazione sulla mappa.

Conclusioni

In sintesi, l'applicazione sviluppata non solo offre strumenti avanzati per la visualizzazione e gestione dei dati georeferenziati, ma si propone anche come una piattaforma versatile per il monitoraggio e la gestione delle risorse territoriali. Grazie all'architettura modulare e scalabile, il sistema è in grado di gestire carichi di lavoro variabili in modo efficiente, assicurando prestazioni elevate. L'integrazione tra i vari componenti permette agli utenti di interagire con i dati in tempo reale, migliorando l'efficacia del monitoraggio e della gestione delle risorse sul campo.

Indice

Elenco delle figure

Glossario

React Una libreria JavaScript per la creazione di interfacce utente modulari e riutilizzabili. Utilizza il Virtual DOM per migliorare l'efficienza del rendering dell'interfaccia utente.
Documentazione ufficiale: `https://reactjs.org/`.

TypeScript Un linguaggio di programmazione sviluppato da Microsoft, che estende JavaScript aggiungendo il controllo statico dei tipi, facilitando la scrittura di codice più sicuro e meno incline a errori.
Documentazione ufficiale: `https://www.typescriptlang.org/`.

GeoServer Un server open-source per la pubblicazione di dati geospaziali su web. Supporta standard come WMS, WFS e WCS, permettendo l'accesso e la visualizzazione di dati geografici in vari formati.
Documentazione ufficiale `https://geoserver.org/`.

GDAL Geospatial Data Abstraction Library, una libreria open-source per la manipolazione di dati geospaziali. Supporta la lettura e la scrittura di diversi formati di file raster e vettoriali.
Documentazione ufficiale: `https://gdal.org/`.

WMS Web Map Service, un protocollo standard per la distribuzione di mappe georeferenziate su web. Consente di richiedere mappe statiche o dinamiche in formato raster.
Documentazione ufficiale: `https://www.ogc.org/standards/wms`.

ArcGIS Una suite di software per la creazione e l'analisi di dati geografici, utilizzata per la visualizzazione di mappe interattive e per l'analisi di dati spaziali.
Sito ufficiale: `https://www.esri.com/en-us/arcgis/products/arcgis-online/overview`.

Esri Un'azienda specializzata nello sviluppo di software per la geoinformatica, tra cui ArcGIS, uno dei software più utilizzati per la creazione e l'analisi di mappe geografiche.
Sito ufficiale: `https://www.esri.com/`.

GIS Sistema informativo geografico, un sistema progettato per acquisire, memorizzare, analizzare, gestire e visualizzare dati geografici.

GeoJSON Un formato di dati geografici basato su JSON, utilizzato per rappresentare informazioni geografiche come punti, linee e poligoni.
Documentazione ufficiale: `https://geojson.org/`.

OCG Open Geospatial Consortium, un consorzio internazionale che sviluppa standard aperti per la geoinformatica, tra cui WMS, WFS e WCS.
Sito ufficiale: `https://www.ogc.org/`.

Shapefile Un formato di file per la memorizzazione di dati geografici, sviluppato da Esri. È composto da diversi file che memorizzano informazioni geometriche e attributi.
Documentazione ufficiale: `https://doc.arcgis.com/en/arcgis-online/reference/shapefiles.htm`.

CSV Comma-Separated Values, un formato di file che memorizza i dati in forma tabellare, separando i campi con virgole. È comunemente utilizzato per l'importazione ed esportazione di dati tra applicazioni.

Raster Un formato di dati geografici che rappresenta le informazioni come una griglia di celle, ognuna contenente un valore. È spesso utilizzato per rappresentare immagini satellitari o mappe topografiche.

GeoTIFF Un formato di file raster georeferenziato, che combina le informazioni di un'immagine raster con i dati di georeferenziazione. È spesso utilizzato per memorizzare mappe topografiche e immagini satellitari.
Documentazione ufficiale: `https://www.ogc.org/standards/geotiff`.

ASC ASCII Grid, un formato di file raster che memorizza i dati in forma testuale, utilizzando spazi o tabulazioni per separare i valori. È spesso utilizzato per rappresentare modelli digitali del terreno.

JSON JavaScript Object Notation, un formato di dati leggero basato su testo, comunemente utilizzato per lo scambio di dati tra applicazioni.

PostgreSQL Un sistema di gestione di database relazionali open-source, noto per la sua affidabilità, scalabilità e conformità agli standard SQL.
Sito ufficiale `https://www.postgresql.org/`.

Spring Boot Un framework Java che semplifica lo sviluppo di applicazioni standalone e basate su microservizi, con configurazioni minime necessarie. Supporta la scalabilità e l'integrazione con il database tramite Spring Data JPA.
Documentazione ufficiale: `https://spring.io/projects/spring-boot`.

Spring Data JPA Un modulo di Spring che semplifica l'accesso ai dati da applicazioni Java, fornendo un'implementazione basata su JPA (Java Persistence API) per la gestione delle operazioni CRUD sul database.

Spring Security Un framework per l'autenticazione e la gestione della sicurezza nelle applicazioni Java. Viene spesso utilizzato per proteggere le API con meccanismi di autenticazione basati su token.

JWT JSON Web Token, uno standard aperto per la creazione di token di accesso basati su JSON. Viene spesso utilizzato per l'autenticazione e l'autorizzazione nelle applicazioni web.
Documentazione dello standard: `https://datatracker.ietf.org/doc/html/rfc7519`

OAuth2 Un protocollo di autorizzazione che permette a un'applicazione di ottenere accesso limitato alle risorse di un altro servizio, senza esporre le credenziali dell'utente.
Documentazione dello standard: `https://datatracker.ietf.org/doc/html/rfc6749/`

BCrypt Un algoritmo di hashing per la crittografia delle password, usato per migliorare la sicurezza dei dati sensibili memorizzati in un database.

Jetpack Compose Un toolkit moderno per lo sviluppo di interfacce utente in Android. Si basa sulla programmazione dichiarativa in Kotlin e offre componenti modulari e riutilizzabili detti composable. Supporta Material Design.
Documentazione ufficiale: `https://developer.android.com/jetpack/compose`.

Hilt Un framework di dependency injection per Android, sviluppato da Google. Si integra con Jetpack e semplifica l'iniezione delle dipendenze nelle applicazioni Android.
Documentazione ufficiale: `https://developer.android.com/training/dependency-injection/hilt-android`.

Retrofit Una libreria per Android che semplifica la creazione di client REST per le chiamate alle API. Si basa su annotazioni per definire le richieste HTTP e i parametri di input.
Documentazione ufficiale: `https://square.github.io/retrofit/`.

Kotlin Un linguaggio di programmazione moderno e conciso, sviluppato da JetBrains, che si basa sulla JVM. È spesso utilizzato per lo sviluppo di applicazioni Android, grazie alla sua interoperabilità con Java.
Documentazione ufficiale: `https://kotlinlang.org/`.

Docker Una piattaforma open-source per lo sviluppo, la distribuzione e l'esecuzione di applicazioni in container. I container Docker sono isolati e leggeri, consentendo l'esecuzione di applicazioni in ambienti virtualizzati.
Sito ufficiale: `https://www.docker.com/`.

Docker Compose Uno strumento per la definizione e l'esecuzione di applicazioni Docker multi-container. Consente di definire i servizi, le reti e i volumi di un'applicazione in un file YAML, semplificando la gestione di ambienti complessi.
Documentazione ufficiale: `https://docs.docker.com/compose/`.

Kubernetes Un sistema open-source per l'automazione della distribuzione, della scalabilità e della gestione di applicazioni containerizzate. Kubernetes consente di orchestrare i container in cluster, garantendo la disponibilità e la scalabilità delle applicazioni.
Sito ufficiale: `https://kubernetes.io/`.

IAM Identity and Access Management, un insieme di processi e tecnologie per la gestione delle identità digitali e dei privilegi di accesso alle risorse.

Keycloak Un server open-source per la gestione dell'identità e degli accessi, sviluppato da Red Hat. Offre funzionalità di autenticazione, autorizzazione e single sign-on per applicazioni web e mobile.
Documentazione ufficiale: `https://www.keycloak.org/`.

Capitolo 1

Introduzione

Il presente progetto nasce dall'esigenza di organizzare e rendere più accessibile la vasta mole di informazioni derivanti dall'analisi dei dati territoriali. Nonostante i dati vengano spesso raccolti con precisione ed accuratezza, la loro interpretazione risulta complessa e poco immediata. Sensori e rilevazioni generano grandi quantità di dati, che però, sotto forma di numeri e tabelle, rimangono di difficile fruizione. Questa situazione ostacola l'analisi ed il monitoraggio efficace degli eventi registrati, rendendo complicata una comprensione approfondita dei fenomeni territoriali osservati. Le misurazioni raccolte includono dati relativi alla posizione geografica, espressi in termini di coordinate del luogo osservato, permettendo una facile rappresentazione su una mappa. L'adozione di una visualizzazione tridimensionale risponde alla necessità di evidenziare anche le variazioni altimetriche, fondamentale in contesti dove due punti geograficamente vicini possono presentare dislivelli significativi. Un esempio rilevante è la rappresentazione di eventi sismici: in questo caso, l'intensità del sisma viene visualizzata tramite poligoni che indicano sia la forza dell'evento sia l'estensione dell'area colpita. Questo approccio consente di individuare con maggiore chiarezza pattern e tendenze, rendendo l'analisi dei fenomeni territoriali più intuitiva e immediata. In questa tesi è stata sviluppata un'applicazione web che offre una visualizzazione grafica intuitiva, facilitando analisi interattive e aggregazioni efficienti dei dati. La scelta di una piattaforma web è motivata dalla sua accessibilità: non richiedendo l'installazione di software, l'applicazione è sempre disponibile online, consentendo un accesso immediato da parte degli utenti. Inoltre, grazie alla condivisione in tempo reale, più utenti possono visualizzare e interagire simultaneamente con i dati, favorendo una collaborazione continua e una gestione dinamica delle informazioni. Abbiamo deciso di supportare dall'inizio i principali formati di dati per adattarci alle necessità degli utenti e per permettere così una migliore fruibilità del servizio.

1.1 Stato dell'arte

Pur esistendo soluzioni alternative sul mercato, come ArcGIS, ArcGIS Online, QGIS e altre piattaforme simili, la nostra applicazione si distingue per la capacità di soddisfare in maniera più precisa le esigenze degli utenti. ArcGIS, ad esempio, è un software avanzato per l'analisi geografica, ma spesso richiede competenze tecniche specializzate e costose licenze d'uso. ArcGIS Online, pur offrendo strumenti di visualizzazione e analisi di dati geospaziali direttamente via web, è comunque legato a una piattaforma proprietaria, limitando la flessibilità dell'utente in termini di personalizzazione e indipendenza nella gestione dei propri dati. QGIS, al contrario, è un software open-source molto versatile,

largamente utilizzato per l'analisi geografica e la creazione di mappe. Tuttavia, non essendo nativamente progettato per funzionare in ambiente web, richiede configurazioni aggiuntive e l'installazione di estensioni per poter supportare l'interazione online e la condivisione immediata dei dati tra più utenti. La nostra soluzione, invece, si distingue proprio per essere completamente online, offrendo un sistema di gestione dei dati geospaziali che non dipende da enti esterni o da piattaforme proprietarie. Gli utenti possono accedere all'applicazione da qualsiasi dispositivo connesso ad Internet, senza necessità di installare software o configurazioni complesse. Questo rende il nostro strumento non solo più flessibile e accessibile, ma anche capace di garantire una maggiore autonomia nella gestione delle informazioni. Inoltre, la possibilità di condividere i dati in tempo reale tra diversi utenti consente una collaborazione immediata e dinamica, migliorando l'efficienza nell'analisi dei fenomeni territoriali e nella presa di decisioni basate sui dati.

Capitolo 2

Architettura del sistema

Questo capitolo si concentrerà sull'analisi dell'architettura del progetto, descrivendo le componenti fondamentali, le loro relazioni e le decisioni di design alla base. L'obiettivo è fornire una comprensione chiara della struttura e del funzionamento dell'applicazione, illustrando in modo dettagliato le basi architetturali e le scelte progettuali che ne hanno guidato lo sviluppo. In particolare si analizzeranno le tecnologie utilizzate, le comunicazioni tra i diversi componenti e le modalità di visualizzazione dei dati geospaziali. Per lo sviluppo del progetto ci siamo basati su un'architettura a microservizi, che permette di suddividere l'applicazione in componenti indipendenti e scalabili, facilitando la gestione e l'aggiornamento delle funzionalità. Questo approccio consente inoltre di isolare le responsabilità di ciascun servizio, semplificando la manutenzione e il testing del codice.

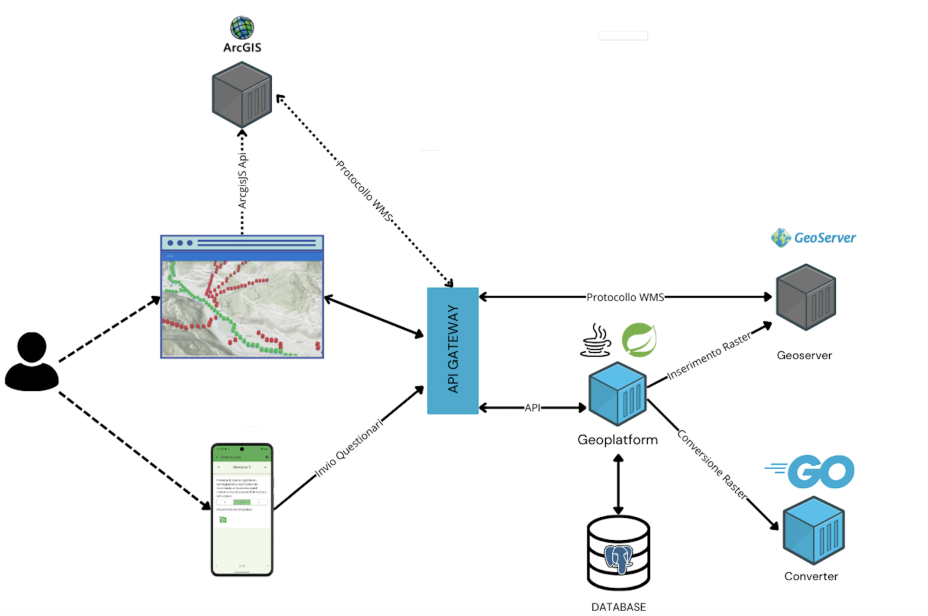


Figura 2.1: Architettura del sistema

Nella figura sono rappresentate le principali componenti dell'architettura del sistema, ciascuna con un ruolo specifico e ben definito. Queste componenti sono interconnesse tra loro attraverso protocolli standard, consentendo una comunicazione efficiente e affidabile. Di seguito verranno analizzate nel dettaglio le funzionalità e le caratteristiche di ciascuna componente, evidenziando il loro ruolo all'interno dell'applicazione e le interazioni con le altre parti del sistema.

2.1 Front-End: React e TypeScript

La parte front-end è realizzata con **React**, una libreria JavaScript. Abbiamo scelto React perché consente di costruire l'interfaccia utente in modo modulare, suddividendo ogni parte dell'interfaccia in piccoli componenti riutilizzabili. Questo approccio non solo rende il codice più pulito e facile da mantenere, ma permette anche a React di essere molto performante in caso di cambiamenti di stato, aggiornando solo i componenti dell'interfaccia che sono stati modificati, senza dover ricaricare l'intera pagina.

Inoltre, abbiamo utilizzato **TypeScript** per scrivere il codice front-end. TypeScript è un linguaggio di programmazione che estende JavaScript aggiungendo tipi statici opzionali. Questo ci ha permesso di ridurre gli errori di programmazione e di rendere il codice più robusto e manutenibile.

2.2 Back-End: Java e Spring Boot

La parte back-end è sviluppata in **Java** usando il framework **Spring Boot**. Abbiamo scelto Java perché è un linguaggio solido e molto usato, specialmente per applicazioni che richiedono scalabilità e robustezza. Spring Boot è un framework che semplifica lo sviluppo di applicazioni Web Java, fornendo funzionalità pronte all'uso per la gestione delle richieste HTTP, la connessione al database e la gestione degli errori. Plugin come Spring Security e Spring Data JPA ci hanno permesso di implementare funzionalità di autenticazione e autorizzazione, e di interagire con il database in modo efficace e riducendo la quantità di codice "boilerplate" necessario.

2.3 Comunicazione tra Microservizi

La comunicazione tra i vari microservizi avviene tramite richieste HTTP e REST API, che rappresentano uno standard per la comunicazione tra servizi web. Questo approccio consente una comunicazione efficiente e scalabile tra i diversi componenti del sistema, permettendo loro di scambiare dati e informazioni in modo rapido e affidabile. Inoltre, l'uso di REST API facilita l'integrazione con altri servizi e applicazioni, consentendo una maggiore flessibilità e interoperabilità del sistema.

2.3.1 Flusso dell'applicazione

Di seguito sono riportati i flussi per la visualizzazione dei dati vettoriali e raster, che illustrano come i dati possano essere caricati, elaborati e visualizzati all'interno dell'applicazione.

Visualizzazione dei Dati Vettoriali

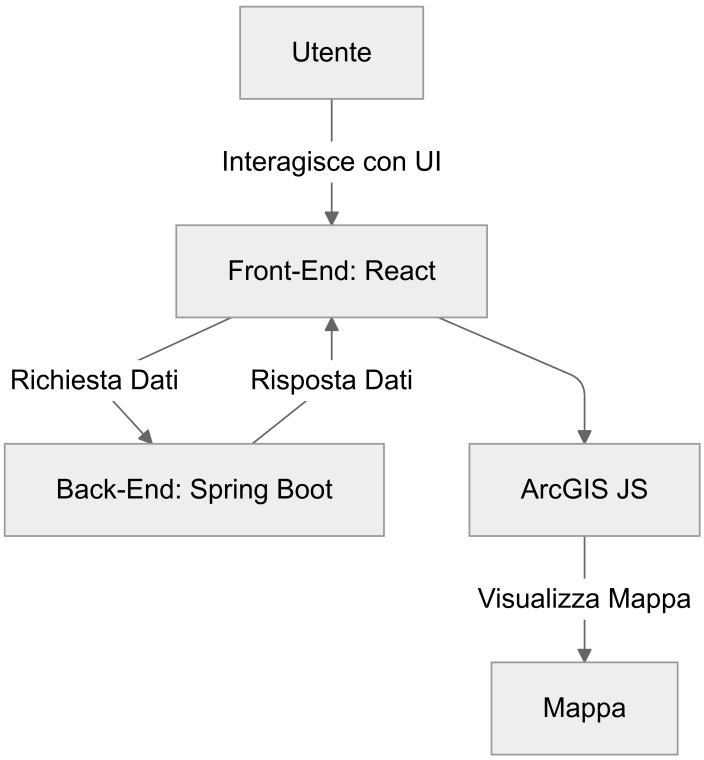


Figura 2.2: Flusso per la visualizzazione dei dati Vettoriali

Per visualizzare i dati vettoriali la sequenza di operazioni è la seguente:

- L'utente carica un file vettoriale (ad esempio, un file shapefile) tramite l'interfaccia utente.
- Il front-end effettua l'upload tramite chunked upload, suddividendo il file in piccole parti e inviandole al back-end in modo asincrono.
- Il back-end elabora i vari chunk e salva il file nel file system e i suoi metadati nel database.
- Il front-end una volta completato l'upload specifica gli stili di visualizzazione da applicare al layer vettoriale.
- Il front-end richiede i dati vettoriali al back-end tramite una richiesta HTTP.
- Il back-end recupera i dati dal database e li invia al front-end nel formato originale (GeoJSON, Shapefile o CSV).
- Il front-end tramite ArcGIS JS visualizza i dati sulla mappa 3D, specificando lo stile di visualizzazione e le proprietà dei dati.

Visualizzazione dei Dati Raster

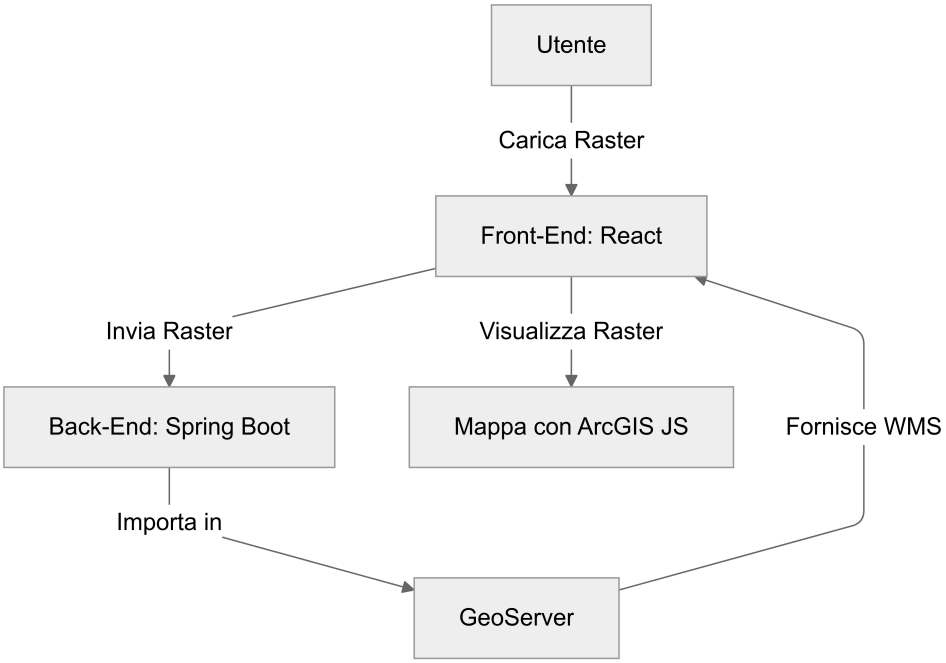


Figura 2.3: Flusso per la visualizzazione dei dati Raster

Per visualizzare i dati raster la sequenza di operazioni è la seguente:

- L'utente carica un file raster (ad esempio, un file GeoTIFF) tramite l'interfaccia utente.
- Il front-end effettua l'upload tramite chunked upload, suddividendo il file in piccole parti e inviandole al back-end in modo asincrono.
- Il back-end elabora i vari chunk e salva il file nel file system e i suoi metadati nel database.
- Il converter si occupa di convertire il file raster nella proiezione geografica corretta, utilizzando GDAL.
- Il back-end coordina la creazione di un layer raster in GeoServer per la visualizzazione sulla mappa.
- Il front-end richiede i dati raster al back-end tramite una richiesta HTTP.
- Il back-end riconosce la richiesta e invia i tile del file raster al front-end tramite il protocollo WMS.
- Il front-end visualizza i dati raster sulla mappa 3D, utilizzando i tile ricevuti con il protocollo WMS generati da GeoServer.
- Ogni volta che la camera della mappa viene spostata o ingrandita, il front-end richiede nuovi tile al geoserver tramite il protocollo WMS per garantire una visualizzazione fluida e dettagliata.

2.4 Libreria per la Visualizzazione delle Mappe

Per la visualizzazione di dati georeferenziati su mappe 3D, abbiamo scelto di utilizzare **ArcGIS API per JavaScript**. Questa decisione è stata presa dopo aver valutato diverse alternative disponibili:

- **Leaflet**: Una libreria semplice e leggera per la realizzazione di mappe in JavaScript. È estendibile tramite plugin per eseguire operazioni più complesse, ma potrebbe non offrire le funzionalità GIS avanzate di cui abbiamo bisogno.
- **Mapbox GL JS**: Una libreria open-source che utilizza WebGL per la visualizzazione su mappa, con supporto per la visualizzazione 3D. Rispetto a **ArcGIS JS**, **Mapbox GL JS** offre meno supporto per funzionalità GIS avanzate, come l'analisi spaziale e la geocodifica.
- **OpenLayers**: Una libreria open-source che fornisce un supporto completo per la visualizzazione di mappe, ma non supporta nativamente il 3D.
- **CesiumJS**: Specializzata nella visualizzazione 3D, **CesiumJS** è un'ottima alternativa open-source per la gestione di dati tridimensionali. Tuttavia, rispetto ad **ArcGIS JS**, è meno performante nelle funzionalità GIS avanzate.

Pur potendo ottenere risultati simili utilizzando altre librerie, abbiamo scelto **ArcGIS** per la sua completezza, il supporto, e la documentazione fornita da **Esri**.

2.4.1 ArcGIS

ArcGIS è una piattaforma GIS (Geographic Information System) sviluppata da **Esri**, leader mondiale nel settore delle tecnologie geospaziali. ArcGIS offre un insieme completo di strumenti per la raccolta, gestione, analisi e visualizzazione di dati geografici. Questi strumenti vengono utilizzati in una vasta gamma di settori, dall'urbanistica alla gestione delle risorse naturali, dal monitoraggio ambientale alla logistica. La piattaforma consente la creazione di mappe interattive, l'esecuzione di analisi spaziali complesse e il geoprocessamento di grandi quantità di dati, il tutto integrato in un sistema scalabile e flessibile che può essere utilizzato sia localmente che nel cloud.

ArcGIS si distingue per la sua capacità di gestire dati geospaziali provenienti da diverse fonti, come immagini satellitari, sensori, droni e dataset open data, permettendo agli utenti di ottenere informazioni dettagliate sul territorio. Le sue funzionalità si estendono dalla semplice visualizzazione di mappe alla modellazione spaziale avanzata, supportando flussi di lavoro GIS sia per utenti occasionali che per professionisti GIS esperti.

2.4.2 ArcGIS API per JavaScript

ArcGIS API per JavaScript è una libreria basata su JavaScript che permette agli sviluppatori web di integrare le potenti funzionalità di ArcGIS direttamente nelle loro applicazioni. Questa API consente la creazione di mappe interattive, la gestione di layer geografici, l'interazione con dati 2D e 3D, e l'esecuzione di operazioni GIS come il routing, la misurazione di distanze, e le analisi di sovrapposizione spaziale, tutto tramite codice JavaScript.

L'uso di ArcGIS API per JavaScript presenta numerosi vantaggi per gli sviluppatori:

- **Integrazione di mappe interattive**: permette di visualizzare mappe dinamiche con funzionalità di zoom, panoramica e selezione di elementi, offrendo un'esperienza utente coinvolgente e intuitiva.
- **Supporto per visualizzazioni avanzate**: offre supporto nativo per mappe 3D, permettendo di rappresentare dati in modo realistico e interattivo con l'uso di SceneView.
- **Accesso a servizi GIS avanzati**: consente di accedere a servizi GIS avanzati come geocodifica, geoprocessing e analisi spaziale, offrendo funzionalità di alto livello per l'elaborazione e la visualizzazione dei dati.
- **Personalizzazione completa**: la libreria è altamente configurabile, consentendo agli sviluppatori di personalizzare ogni aspetto della visualizzazione e dell'interazione con i dati, offrendo flessibilità nella creazione di interfacce utente e flussi di lavoro specifici.
- **Strumenti di analisi integrati**: include strumenti pronti all'uso per eseguire buffer, analisi di prossimità, clustering spaziale e molto altro.
- **Aggiornamenti e supporto costante**: sviluppato da Esri è continuamente aggiornata con nuove funzionalità e miglioramenti per garantire prestazioni ottimali e compatibilità con le tecnologie web più recenti.

2.5 Dati Vettoriali e Raster

Per comprendere meglio le difficoltà relative alla gestione dei dati, è utile introdurre le due categorie di dati che dovranno essere visualizzati sulle mappe:

2.5.1 Dati Raster

I dati raster rappresentano immagini o superfici continue come una griglia di celle o pixel, ciascuno con un valore specifico. Ogni cella può rappresentare un colore in un'immagine o un valore in una mappa, come l'altitudine. Questa loro struttura causa:

- **Alte Dimensioni del File**: Poiché ogni cella della griglia deve essere memorizzata singolarmente, le immagini raster tendono a richiedere molto spazio di archiviazione, specialmente ad alta risoluzione. Questo può rendere difficoltoso il loro utilizzo in applicazioni che richiedano un'elaborazione rapida o che dispongano di spazio limitato.
- **Perdita di Dettaglio con l'Ingrandimento**: A differenza dei dati vettoriali, i raster sono soggetti a una perdita di qualità visibile quando vengono ingranditi oltre la loro risoluzione originale. Questo effetto rende i bordi sfocati e riduce la chiarezza dei dettagli

2.5.2 Dati Vettoriali

I dati vettoriali sono un modello di dati basato su coordinate che rappresentano le caratteristiche geografiche come punti, linee e poligoni. Ogni elemento puntuale è rappresentato da una singola coppia di coordinate, mentre le caratteristiche lineari e poligonali sono rappresentate come elenchi ordinati di vertici. Questa loro struttura causa:

- **Elevata Precisione Geometrica**: I dati vettoriali permettono una rappresentazione accurata delle forme geometriche e delle relazioni spaziali tra gli oggetti, garantendo una qualità costante indipendentemente dall'ingrandimento o dalla riduzione.
- **Efficienza nello Spazio di Archiviazione**: Poiché solo i vertici principali vengono memorizzati, i dati vettoriali tendono a occupare meno spazio rispetto ai raster per rappresentare oggetti complessi, specialmente quando si tratta di mappe o dati con molti dettagli.
- **Minor Adattabilità alla Visualizzazione di Dati Continui**: I dati vettoriali non sono ideali per rappresentare superfici continue o gradienti, come altitudini, temperature o altre variabili che cambiano gradualmente nello spazio. Essi rappresentano le informazioni in forma discreta attraverso punti, linee e poligoni, rendendo difficile catturare le variazioni sottili e continue presenti in questi tipi di dati.

2.6 Visualizzazione dei dati Raster

Per la visualizzazione dei dati raster, abbiamo scelto di utilizzare un server dedicato che gestisce la generazione dei tile. Questi tile vengono trasmessi al frontend tramite il protocollo WMS (Web Map Service).

2.6.1 Importanza di un Server per la Generazione dei Tile

Avere un server dedicato alla generazione dei tile è fondamentale per diverse ragioni:

- **Alleggerimento del Carico sul Forntend**: Utilizzando un server esterno, si riduce il carico computazionale e di memoria sul dispositivo dell'utente. Questo permette un'esperienza più fluida, poiché il client non deve gestire operazioni intensive come la generazione e l'elaborazione delle immagini raster.
- **Scalabilità**: Un server dedicato può gestire richieste multiple e simultanee da parte di più utenti, consentendo di scalare facilmente l'applicazione in base al numero di richieste.
- **Ottimizzazione della Performance**: I server specializzati nella generazione di tile possono implementare tecniche di caching e ottimizzazione delle immagini, migliorando ulteriormente le prestazioni di caricamento e visualizzazione.

2.6.2 Protocollo WMS

Il Web Map Service (WMS) è uno standard sviluppato da OpenGIS® che fornisce un'interfaccia HTTP semplice per richiedere immagini di mappe georeferenziate da uno o più database geospaziali distribuiti. Una richiesta WMS definisce i layer geografici e l'area di interesse da elaborare. La risposta alla richiesta consiste in una o più immagini di mappa georeferenziate, restituite in formati come JPEG o PNG, che possono essere visualizzate in un'applicazione browser. Il protocollo WMS supporta anche la possibilità di specificare se le immagini restituite debbano essere trasparenti, consentendo la sovrapposizione di layer provenienti da più server wms.

2.6.3 Alternative per la Generazione di Tile e Visualizzazione di Dati Raster

Esistono diversi prodotti in grado di gestire la generazione di tile e la visualizzazione di dati raster attraverso il protocollo WMS. Tra questi, abbiamo scelto GeoServer per i seguenti motivi:

- **Compatibilità con Standard OGC:** GeoServer è conforme agli standard OGC (Open Geospatial Consortium), inclusi WMS e WFS (Web Feature Service), il che ne facilita l'integrazione con altre applicazioni e strumenti GIS.
- **Facilità d'Uso:** Offre un'interfaccia web intuitiva per la gestione dei dati e dei layer, consentendo agli utenti di configurare facilmente le proprie mappe e servizi.
- **Supporto per Diversi Formati di Dati:** GeoServer supporta una vasta gamma di formati di dati raster e vettoriali, permettendo una grande flessibilità nella gestione delle informazioni geospaziali.
- **Caching e Ottimizzazione:** GeoServer offre funzionalità di caching che migliorano la velocità di caricamento delle immagini e la performance generale del server.
- **Comunità Attiva:** Essendo open-source, GeoServer ha una comunità attiva di sviluppatori e utenti che contribuiscono al suo miglioramento e forniscono supporto.

Capitolo 3

Implementazione

In questo capitolo, descriviamo l'implementazione dell'applicazione Geospatial, concentrandoci sui principali aspetti tecnici e architetturali.

3.1 Data store

Per la gestione e il salvataggio dei dati all'interno della nostra applicazione, utilizziamo una combinazione di **File System** (FS) e **PostgreSQL**. Sebbene avessimo potuto utilizzare **PostGIS** per gestire dati geospaziali, abbiamo scelto una soluzione più semplice per evitare complessità eccessive.

3.1.1 PostgreSQL

PostgreSQL è un sistema di gestione di database relazionali open-source, noto per la sua robustezza, flessibilità e conformità agli standard SQL. Supporta una vasta gamma di tipi di dati, inclusi tipi geospaziali, e offre funzioni avanzate come transazioni e controlli di integrità. Nella nostra applicazione, utilizziamo **Docker** per eseguire PostgreSQL in un ambiente isolato.

3.1.2 Configurazione del Database

La configurazione del database nel file `application.yml` è la seguente:

```yaml
spring:
  datasource:
    url: jdbc:postgresql://${DATABASE_HOST}:5432/geodb
    username: username
    driver-class-name: org.postgresql.Driver
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: false
    properties:
      hibernate:
        format_sql: true
    database: postgresql
    database-platform: org.hibernate.dialect.PostgreSQLDialect
    defer-datasource-initialization: true
```

3.1.3 Struttura del Database

Utilizziamo **JPA** (Java Persistence API) per mappare le classi Java agli oggetti del database. Le classi principali includono:

- **GeoItemRepository**: Estende `JpaRepository` per interagire con il database, fornendo metodi come `findAll()`, `findById()`, e `existsByTitle()`.

```java
public interface GeoItemRepository extends JpaRepository<GeoItem, Integer> {
    List<GeoItem> findAll();
    Optional<GeoItem> findById(Integer id);
    boolean existsByTitle(String title);
}
```

- **GeoItem**: Rappresenta un oggetto geospaziale, contenendo attributi come `id`, `title`, `storageRelativePath`, e relazioni con altre entità.

```java
@Data
@Builder
@Entity(name = "geoitem")
@Table(name = "geoitem")
public class GeoItem {
    @Id
    @GeneratedValue
    public Integer id;

    @Column(unique = true)
    public String title;

    @Column(name = "storage_relative_path", length = 500)
    public String storageRelativePath = null;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    public User user;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "category_id")
    public Category category;
}
```

3.1.4 Salvataggio dei Dati

Il salvataggio è gestito dal metodo `save()`, annotato con `@Transactional` per indicare che è eseguito atomicamente. Ecco i passaggi principali del processo di salvataggio:

- **Verifica di Aggiornamento o Creazione**: Determiniamo se il salvataggio riguarda un nuovo **GeoItem** o un aggiornamento di uno esistente.
- **Controllo di Nome**: Verifichiamo se esiste già un **GeoItem** con lo stesso titolo e solleviamo un'eccezione in caso di conflitto.
- **Controllo della Proprietà**: Assicuriamo che l'utente abbia i diritti per modificare o creare l'oggetto.
- **Verifica della Categoria**: Recuperiamo la categoria associata, che deve essere valida.
- **Conversione da DTO a Entità**: I dati vengono convertiti da un oggetto DTO (`GeoItemEditRequestDTO`) a un'entità **GeoItem**. Se stiamo aggiornando un **GeoItem** senza caricare un nuovo file, manteniamo il percorso del file precedente.
- **Salvataggio nel Database**: Infine, il **GeoItem** viene salvato nel database tramite `geoItemRepository.save(entity)`.

```
@Transactional
public GeoItem save(User user, GeoItemEditRequestDTO req)
        throws CategoryNotFoundException, UserNotFoundException,
        ↪  GeoItemNameConflictException {

    boolean isUpdate = req.getId().isPresent();
    GeoItem previous = null;

    if (isUpdate) {
        previous = geoItemRepository.findById(req.getId().get()).orElseThrow(
                () -> new GeoItemNotFoundException("GeoItem not found"));
    }

    boolean nameExists = geoItemRepository.existsByTitle(req.getTitle());
    if (nameExists && (isUpdate && !req.getTitle().equals(previous.getTitle()))) {
        throw new GeoItemNameConflictException("GeoItem with name already exists");
    }

    Integer ownerUserId = req.getUserId().orElse(user.getId());
    super.guardOwnership(user.getId(), ownerUserId, user.isAdmin(), isUpdate);

    Category cat = categoryService.getCategory(req.getCategoryId());
    GeoItem entity = req.toEntity(!isUpdate, user, userService.getUser(ownerUserId),
    ↪  cat);

    if (isUpdate && entity.storageRelativePath == null) {
        entity.setStorageRelativePath(previous.getStorageRelativePath());
    }

    return geoItemRepository.save(entity);
}
```

3.2 Tipi di Formati Supportati

In questa sezione analizziamo in dettaglio i formati di dati geospaziali supportati, suddivisi in vettoriali e raster, e descriviamo come vengono gestiti all'interno della nostra applicazione.

3.2.1 Dati Vettoriali

GeoJSON

GeoJSON è un formato per lo scambio di dati geospaziali basato su *JavaScript Object Notation (JSON)*. Rappresenta dati relativi a caratteristiche geografiche, le loro proprietà e le loro estensioni spaziali. A ogni dato vengono associati il tipo di geometria rappresentata (come `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, e `MultiPolygon`), le coordinate e le proprietà che lo riguardano

```
{
    "type": "Feature",
    "geometry": {
        "type": "Point",
        "coordinates": [125.6, 10.1]
    },
    "properties": {
        "name": "Dinagat Islands"
    }
}
```

Per importare un file `GeoJSON` in ArcGIS, utilizziamo il seguente codice:

```
case "GEOJSON":
    layer = new GeoJSONLayer({
        url: url,
        title: title,
        popupTemplate: graphInfo ?
            getPopupTemplateWithLineGraph(title,
                graphInfo.graphKeys,
                graphInfo.graphXLabel,
                graphInfo.graphYLabel,
                graphInfo.graphTitle) :
            getPopupTemplate(title),
        elevationInfo: {
            mode: "on-the-ground"
        }
    });
    break;
```

CSV

Il **CSV** è un formato semplice per rappresentare dati in forma tabellare, dove i campi sono separati da virgole e i record da interruzioni di linea. Questo formato è particolarmente utile per gestire grandi quantità di dati strutturati. Di seguito un esempio di come sono organizzati i dati di potenziale idroelettrico:

Lat	Long	Potenziale idroelettrico (MWh)	Area Bacino (Km^2)	Max salto geodetico (m)
45.58639324	7.56558644	1002.93	0.80	765
45.58640933	7.566868	1085.91	0.88	753
45.5864254	7.56814955	1025.14	0.89	703

Tabella 3.1: Esempio di dati in formato CSV per l'analisi del potenziale idroelettrico

Per importare un file `CSV` in ArcGIS, utilizziamo il `CSVLayer` come segue:

```
case "CSV":
    layer = new CSVLayer({
        url: url,
        title: title,
        popupTemplate: graphInfo ?
            getPopupTemplateWithLineGraph(title,
                graphInfo.graphKeys,
                graphInfo.graphXLabel,
                graphInfo.graphYLabel,
                graphInfo.graphTitle) :
            getPopupTemplate(title),
        elevationInfo: {
            mode: "on-the-ground"
        }
    });
    break;
```

Shapefile

Uno **Shapefile** è un formato di dati vettoriali comunemente utilizzato nelle applicazioni GIS per rappresentare punti, linee o poligoni, insieme agli attributi associati. Uno shapefile è costituito da almeno quattro file principali:

- `.shp`: contiene la geometria delle entità.
- `.shx`: file di indice per l'accesso rapido ai dati.
- `.dbf`: contiene gli attributi delle entità in formato tabellare.
- `.prj`: specifica il sistema di coordinate e la proiezione.

Per importare uno shapefile, utilizziamo una funzione asincrona che invia il file a un endpoint di ArcGIS per la generazione di una `FeatureCollection`, come mostrato di seguito:

```
async function generateFeatureCollection(fileName) {
    const portalUrl = "https://www.arcgis.com";
    const params = {
        'name': fileName,
        'maxRecordCount': 1000,
        'enforceInputFileSizeLimit': true,
        'enforceOutputJsonSizeLimit': true,
        'generalize': true,
        'maxAllowableOffset': 10,
        'reducePrecision': true,
        'numberOfDigitsAfterDecimal': 0
    };

    const myContent = {
        'filetype': 'shapefile',
        'publishParameters': JSON.stringify(params),
        'f': 'json',
    };

    let formData = new FormData();
    return fetch(fileName)
        .then(response => response.blob())
        .then(blob => {
            var file = new File([blob], './shapefile.zip');
            formData.append('file', file);

            return request(portalUrl + '/sharing/rest/content/features/generate', {
                query: myContent,
                body: formData,
                responseType: 'json'
            });
        });
}
```

Una volta ricevuta la `FeatureCollection` dalla risposta dell'API, viene generato un `FeatureLayer` per visualizzare i dati sulla mappa:

```
function genLayerList(featureCollection) {
    const layers = featureCollection.layers.map((layer) => {
        const graphics = layer.featureSet.features.map((feature) => {
            return Graphic.fromJSON(feature);
        });
        const featureLayer = new FeatureLayer({
            objectIdField: "FID",
            source: graphics,
            fields: layer.layerDefinition.fields.map((field) => {
                return Field.fromJSON(field);
            })
        });
        return featureLayer;
    });
    return layers[0];
}
```

3.2.2 Dati Raster

Per rappresentare dati ad alta risoluzione, come immagini satellitari o mappe topografiche, utilizziamo formati raster come **GeoTIFF** e **ASC** (ASCII Grid). Questi formati sono ampiamente utilizzati nell'analisi di dati spaziali e nella creazione di mappe tematiche in quanto consentono di rappresentare informazioni dettagliate su una superficie geografica.

GeoTIFF

Il formato **GeoTIFF** è una versione estesa del formato TIFF (Tagged Image File Format) che include informazioni geospaziali. Un file GeoTIFF contiene dati raster, metadati e informazioni sulla proiezione e sul sistema di coordinate. Questo formato è particolarmente utile per rappresentare immagini satellitari, mappe topografiche e altri tipi dati raster georeferenziati.

ASC (ASCII Grid)

Il formato **ASC** (ASCII Grid) è un formato di dati raster che memorizza le informazioni in un file di testo ASCII. Ogni riga del file rappresenta una riga della griglia raster, con i valori separati da spazi o tabulazioni. Questo formato è particolarmente utile per rappresentare dati di elevazione, pendenza o altre informazioni topografiche.

Import dei Dati Raster

I file GeoTIFF e ASC possono essere caricati direttamente in GeoServer tramite le sue API, consentendo la pubblicazione di informazioni georeferenziate accessibili tramite servizi web, come il `WMS` (Web Map Service). Il processo di import è facilitato dal plugin Importer di GeoServer, che semplifica la gestione e la pubblicazione dei dati raster.
La visualizzazione di tali dati viene gestita in `ArcGIS` attraverso la creazione di un `WMSLayer`. In seguito è riportato un esempio di codice per la creazione di un `WMSLayer`:

```
case "RASTER":

layer = new WMSLayer({
    url: url,
    sublayers: [{
        name: layerWMSName
    }]
});
break;
```

3.3 Gestione degli Stili per la Visualizzazione dei Layer

Un aspetto fondamentale nella visualizzazione dei dati geografici in un'applicazione GIS è la rappresentazione grafica degli oggetti sulla mappa. Ogni elemento può essere stilizzato per renderlo più comprensibile, a seconda delle sue caratteristiche e del contesto in cui è visualizzato. In questo progetto, è stato sviluppato un sistema che consente di applicare diversi stili grafici ai layer geografici attraverso un meccanismo chiamato *renderer*.

3.3.1 Definizione di Stile

Uno stile è un insieme di attributi che definiscono come un oggetto geografico debba essere visualizzato sulla mappa. Gli stili includono proprietà come:

- **Forma**: icone, coni, cilindri, ecc.;
- **Colore**: sia per il riempimento che per i contorni;
- **Dimensioni**: altezza, larghezza, profondità, ecc.

Questi attributi visivi possono variare a seconda del tipo di oggetto da rappresentare e servono a migliorare la comprensione e l'interpretazione visiva dei dati. Ad esempio, un layer che rappresenta edifici in 3D può essere reso più intuitivo usando poligoni estrusi per rappresentare l'altezza degli edifici, mentre un layer che rappresenta punti di interesse può utilizzare *marker* più semplici o icone predefinite. L'applicazione degli stili avviene attraverso la funzione `getRendererFromStyle`, che si occupa di generare un *renderer* basato sui parametri forniti dall'oggetto `Style`. Il *renderer* è responsabile dell'applicazione di uno specifico stile agli oggetti geografici e permette di differenziare la visualizzazione degli elementi sulla mappa.

Di seguito vengono descritti i principali tipi di stili supportati dall'applicazione.

3.3.2 Icone ESRI (ESRI_ICON)

Figura 3.1: Esempio Icona ESRI

Quando lo stile specifica l'uso di un'icona ESRI, viene impiegato il *WebStyleSymbol*, una serie di simboli predefiniti forniti da ESRI. Questi possono essere personalizzati per adattarsi alle necessità visive del layer. In particolare, la funzione recupera l'icona dal servizio, ne clona una copia e ne modifica le dimensioni (altezza e larghezza) in base ai parametri specificati nello stile.

```
const webStyleSymbol = new WebStyleSymbol({
    styleName: iconDetails[0],
    name: iconDetails[1]
});
const newSymbol = await webStyleSymbol.fetchSymbol()
    .then(function (pointSymbol3D) {
    // Modifica delle dimensioni e creazione di un nuovo simbolo
});
```

3.3.3 Forme tridimensionali (CONO e CILINDRO)

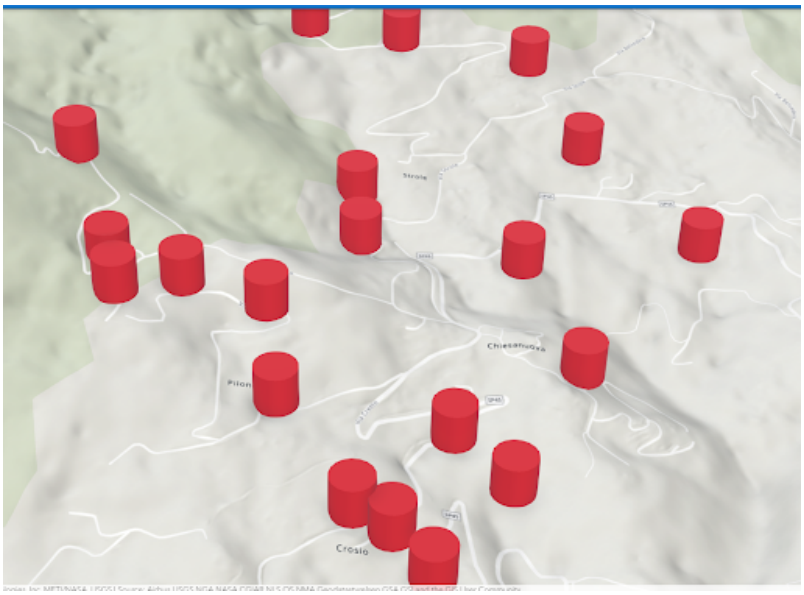


Figura 3.2: Esempio stile Cilindri

Per forme tridimensionali come coni e cilindri, viene utilizzato un simbolo tridimensionale (`PointSymbol3D`) che rappresenta l'oggetto 3D sulla mappa. La forma viene specificata nel parametro `resource` e il colore, la larghezza e l'altezza vengono impostati in base ai valori forniti. Questo permette una rappresentazione chiara e visivamente efficace di elementi tridimensionali in contesti di visualizzazione 3D.

```
    new SimpleRenderer({
        symbol: new PointSymbol3D({
            symbolLayers: [{
                type: "object",
                resource: {primitive: "cone"}, // o "cylinder"
                width: width,
                height: height,
                material: {color: color}
            }]
        })
    });
```

3.3.4 Simple Marker

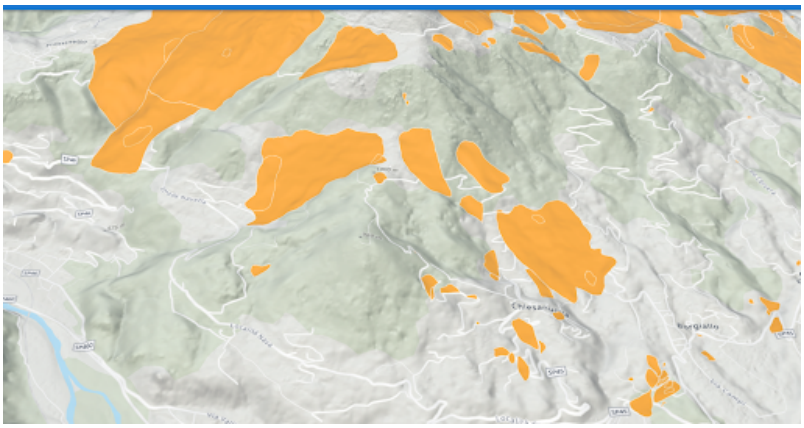


Figura 3.3: Esempio Simple Marker sui dati delle frane

In alcuni casi, può essere sufficiente utilizzare marker semplici per rappresentare oggetti o forme semplici sulla mappa. Questi marker possono essere personalizzati in base al colore, alla dimensione e ad altri attributi visivi per migliorare la leggibilità e l'interpretazione dei dati. In questo scenario, viene impiegato un `SimpleFillSymbol`, che definisce il colore di riempimento del marker, con un contorno bianco per migliorarne la visibilità.

```
new SimpleRenderer({
    symbol: new SimpleFillSymbol({
        color: color,
        outline: {
            color: "white"
        }
    })
});
```

3.3.5 Polygon

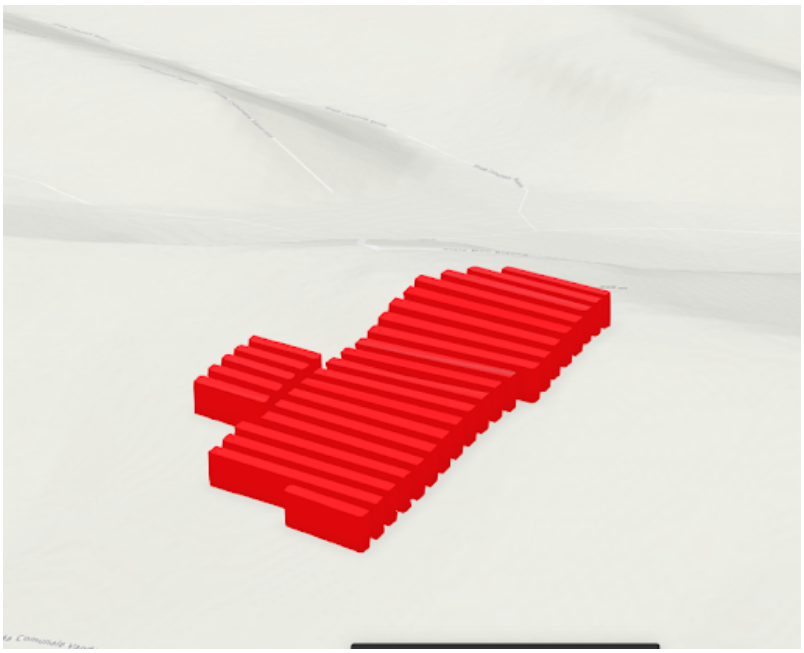


Figura 3.4: Esempio Polygon -> Pannelli solari

Per rappresentare aree o volumi, lo stile può includere un poligono tridimensionale. In questo caso, viene utilizzato un `PolygonSymbol3D`, che consente di estrudere il poligono, rendendolo visivamente simile ad un prisma. Il colore può essere personalizzato per distinguere dati diversi. In questo esempio, ogni poligono rappresenta un pannello solare in modo da evidenziare la distribuzione geografica delle risorse rinnovabili.

```
new SimpleRenderer({
    symbol: new PolygonSymbol3D({
        symbolLayers: [{
            type: "extrude",
            size: 10,  // altezza del prisma
            material: {color: color}
        }]
    })
});
```

3.3.6 Cilindri con Variabili Visive (Color e Height Stop)

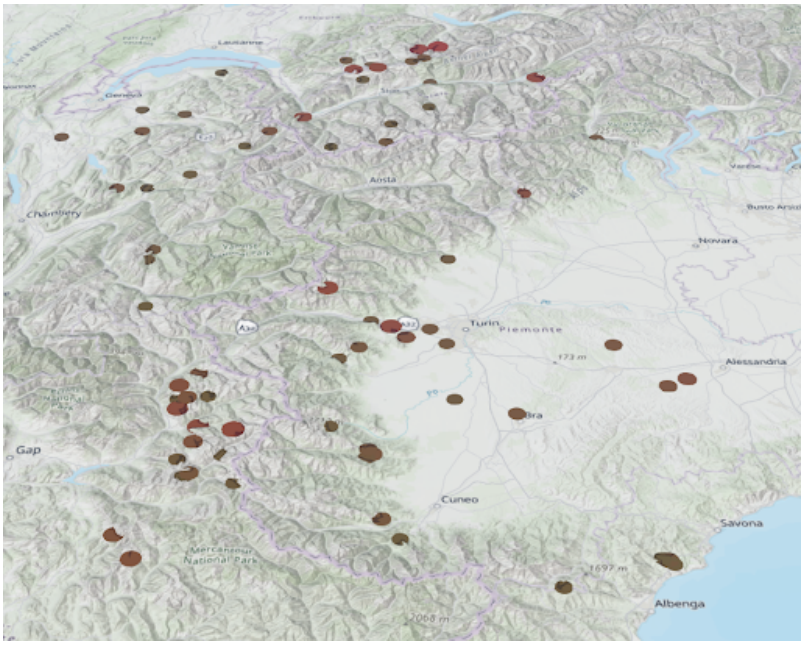


Figura 3.5: Terremoti con colore variabile

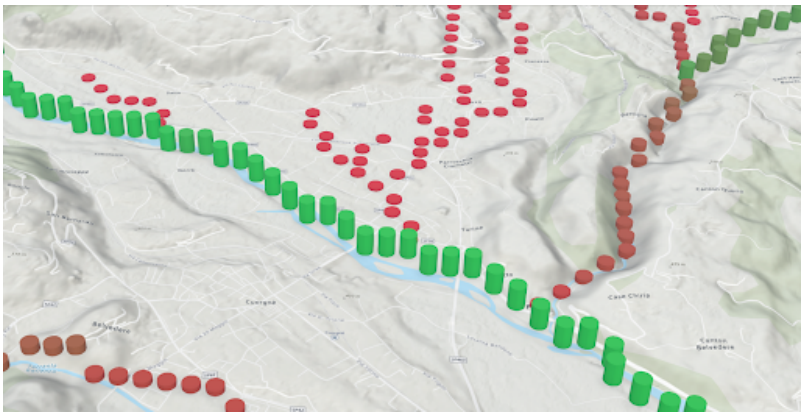


Figura 3.6: Potenziale idroelettrico con altezza e colore variabili

Abbiamo implementato un sistema di variabili visive che consente di personalizzare la rappresentazione dei cilindri in base ai dati associati. Utilizzando la funzione `getVisualVariables`, è possibile generare cilindri con colori, altezze e larghezze variabili a seconda dei valori di specifici campi.

- **Color Variable**: Se lo stile include `colorStop`, viene creata una `ColorVariable` che associa il campo specificato all'interpolazione dei colori.
- **Height Variable**: Se lo stile include `heightStop`, viene generata una `SizeVariable` per definire l'altezza dei cilindri.
- **Width Variable**: Se lo stile include `widthStop`, viene generata una `SizeVariable` per definire la larghezza dei cilindri.

Ecco un esempio di implementazione del codice:

```
export const getVisualVariables = (style: Style, visualVariableFields: string) => {
    const vvf = JSON.parse(visualVariableFields);
    const visualVariables: VisualVariable[] = [];

    vvf.forEach((it: { type: string; key: string; }) => {
        switch (it.type) {
            case "COLOR":
                if (style.colorStop) {
                    visualVariables.push(new ColorVariable({
                        field: it.key,
                        stops: generateColorStopArray(style.colorStop)
                    }));
                }
                break;
            case "HEIGHT":
                if (style.heightStop) {
                    visualVariables.push(new SizeVariable({
                        axis: "height",
                        field: it.key,
                        valueUnit: "meters",
                        stops: generateSizeStopArray(style.heightStop)
                    }));
                }
                break;
            case "WIDTH":
                if (style.widthStop) {
                    visualVariables.push(new SizeVariable({
                        axis: "width-and-depth",
                        field: it.key,
                        valueUnit: "meters",
                        stops: generateSizeStopArray(style.widthStop)
                    }));
                }
                break;
        }
    });

    return visualVariables;
};
```

3.3.7 Applicazione dello Stile al Layer

Il processo di applicazione degli stili avviene nel momento in cui viene creato un layer tramite la funzione `getRenderedLayer`. Dopo aver recuperato tutte le informazioni necessarie sul layer, la funzione verifica se è possibile applicare uno stile personalizzato. Se il layer non è di tipo `WMS` (Web Map Service), viene utilizzato un renderer personalizzato per applicare lo stile desiderato.

La funzione `getRendererFromGeoItem` è responsabile di associare un renderer all'oggetto geografico in base alle specifiche dello stile associato. Se viene trovato un renderer adeguato, esso viene applicato al layer e ne modifica la visualizzazione in base ai parametri definiti. I layer WMS, invece, non possono essere stilizzati direttamente, poiché il protocollo WMS fornisce solo immagini statiche e non permette di modificare dinamicamente lo stile dei dati.

3.3.8 Personalizzazione degli Stili

Molti stili utilizzano campi che vengono popolati dinamicamente tramite variabili (ad esempio, `width` e `height`). Questo permette all'utente di personalizzare in modo flessibile la visualizzazione dei layer, non solo modificando le proprietà esistenti, ma anche cambiando completamente lo stile, adattandolo alle proprie esigenze specifiche.

Nell'immagine seguente, è mostrata l'interfaccia per la modifica di un `GeoItem`, dove l'utente può configurare le diverse proprietà del layer.

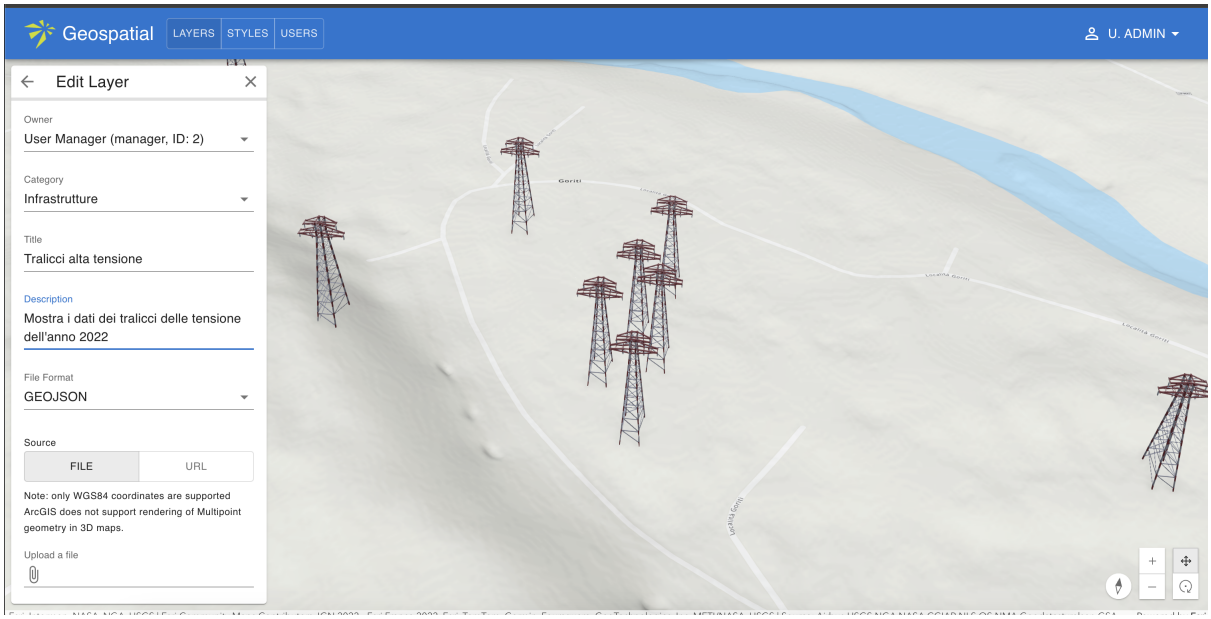


Figura 3.7: Modifica Stile

Per la gestione delle icone ESRI, è stato sviluppato un Icon Picker che offre un'interfaccia intuitiva con anteprime delle icone disponibili, facilitando la selezione e la personalizzazione dell'aspetto grafico degli oggetti.

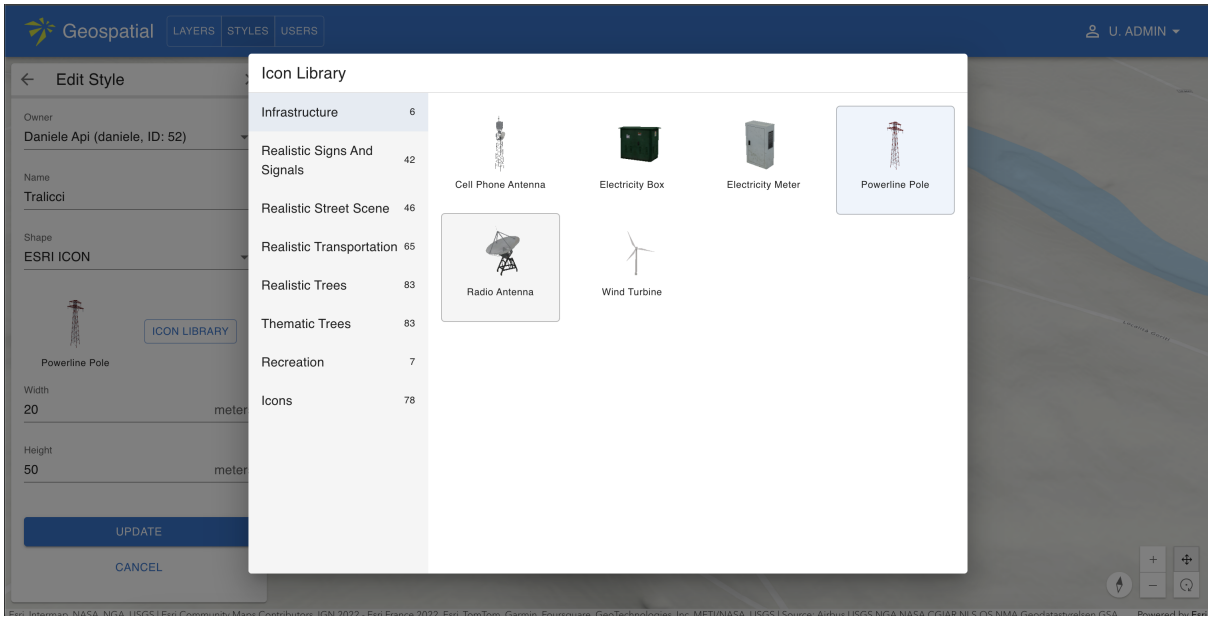


Figura 3.8: Icon Picker

3.4 Integrazione di GeoServer

Per integrare GeoServer nella nostra applicazione, abbiamo deciso di utilizzare la versione Docker. Questa scelta ci consente di gestire facilmente l'ambiente di esecuzione e di semplificare il processo di distribuzione. In particolare, abbiamo utilizzato l'immagine GeoServer Docker nella versione 2.25.2, includendo il plugin Importer per facilitare l'import dei dati geospaziali. La configurazione dell'immagine è stata effettuata utilizzando il seguente Dockerfile:

```
FROM docker.osgeo.org/geoserver:2.25.2
# Create the additional_libs directory  uncomment the lines in the web.xml file
↪  that enable the CORS filter
RUN mkdir -p /opt/additional_libs \
    && sed -i '166s/<!--//g'
    ↪  /opt/apache-tomcat-9.0.89/webapps/geoserver/WEB-INF/web.xml \
    && sed -i '183s/-->//g'
    ↪  /opt/apache-tomcat-9.0.89/webapps/geoserver/WEB-INF/web.xml \
    && sed -i '196s/<!--//g'
    ↪  /opt/apache-tomcat-9.0.89/webapps/geoserver/WEB-INF/web.xml \
    && sed -i '201s/-->//g'
    ↪  /opt/apache-tomcat-9.0.89/webapps/geoserver/WEB-INF/web.xml
# Download the plugin
RUN wget -N https://build.geoserver.org/geoserver/2.25.x/ext-latest
/geoserver-2.25-SNAPSHOT-importer-plugin.zip -P /tmp

# Unzip the plugin into the additional_libs directory
RUN unzip -o /tmp/geoserver-2.25-SNAPSHOT-importer-plugin.zip -d
↪  /opt/additional_libs
```

Funzionalità del Plugin Importer

Il plugin Importer di GeoServer è uno strumento che facilita l'import di dati geospaziali da vari formati e fonti. Le sue principali funzionalità includono:

- **Import di Dati:** Permette di caricare facilmente dati raster e vettoriali da file locali o da fonti remote.
- **Elaborazione dei Dati:** Durante l'import, il plugin gestisce automaticamente la creazione dei layer e la configurazione delle loro proprietà, rendendo il processo più veloce e intuitivo.
- **API Complete:** Permette di gestire l'import dei dati tramite chiamate alle API ben documentate.

CORS Filter

Per garantire una corretta comunicazione tra il frontend e il backend, abbiamo disabilitato il CORS (Cross-Origin Resource Sharing) filter nel file `web.xml` di GeoServer. Il CORS filter controlla quali origini possono accedere alle risorse sul server. Disabilitarlo permette di evitare problemi di accesso quando il frontend della nostra applicazione React deve interagire con GeoServer da un dominio diverso. Tuttavia, è importante utilizzare questa configurazione con attenzione, poiché disabilitare il CORS può comportare rischi per la sicurezza.

Perché Utilizzare la versione Docker?

La scelta di utilizzare un'immagine Docker per GeoServer è stata motivata da diverse considerazioni:

- **Consistenza dell'Ambiente:** Utilizzando Docker, possiamo garantire che GeoServer venga eseguito in un ambiente controllato e replicabile, riducendo il rischio di problemi di compatibilità tra le diverse macchine di sviluppo e produzione.
- **Facilità di Deployment:** Le immagini Docker semplificano il processo di distribuzione e gestione dei servizi. Possiamo facilmente avviare, fermare o aggiornare GeoServer senza doverci preoccupare delle configurazioni ambientali.
- **Integrazione con Altri Servizi:** Abbiamo già implementato PostgreSQL, Spring e il frontend in React all'interno di contenitori Docker. Questo approccio consente una gestione coerente dei servizi e facilita la comunicazione tra di essi all'interno di un ecosistema Docker.

3.4.1 Comunicazione con GeoServer

Sicurezza e Isolamento

Isolamento del GeoServer: Il container Docker che ospita il GeoServer è stato configurato in modo da essere nascosto e non accessibile direttamente dall'esterno. Questo approccio migliora significativamente la sicurezza del sistema, impedendo l'accesso diretto da parte degli utenti finali e riducendo il rischio di potenziali attacchi esterni. **Gestione Centralizzata degli Utenti:** Un ulteriore vantaggio di questa scelta è la possibilità di gestire in modo centralizzato l'autenticazione e il controllo degli accessi. Poiché il GeoServer non è esposto pubblicamente, è possibile mantenere un unico sistema di gestione degli utenti e di controllo dei permessi, sfruttando la stessa base di utenti per tutte le applicazioni collegate, semplificando la gestione della sicurezza e riducendo le vulnerabilità.

3.4.2 Metodi di Accesso a GeoServer

Sono stati implementati due principali metodi per comunicare con GeoServer:

1. **API Chiamate dal Server Spring:** Le chiamate API per la gestione dei raster vengono eseguite dal server Spring. Gli utenti già autenticati nel server Spring possono accedere a queste API, garantendo che tutte le richieste siano validate.
2. **Spring Cloud Gateway:**
   - **Redirect delle Richieste WMS:** Tutte le chiamate relative al protocollo WMS (Web Map Service) sono gestite tramite Spring Cloud Gateway. Questo componente funge da proxy, reindirizzando le richieste al GeoServer.
   - **Filtri di Routing** Solo le comunicazioni che corrispondono ai percorsi specificati vengono inoltrate al GeoServer. Le altre richieste non vengono elaborate, il che riduce il rischio di accessi non autorizzati.

3.4.3 Configurazione di Spring Cloud Gateway

La configurazione YAML di Spring Cloud Gateway specifica le rotte e i filtri da applicare. Ecco un esempio della configurazione fornita:

```yaml
cloud:
  gateway:
    mvc:
      http-client:
        type: autodetect
    routes:
      - id: raster_route
        uri: ${RASTER_SERVER_URL:http://localhost:9090}
        predicates:
          - Path=/raster/geoserver/Polito/wms**
        filters:
          - StripPrefix=1
          - TokenRelay
      - id: ows
        uri: ${RASTER_SERVER_URL:http://localhost:9090}
        predicates:
          - Path=/raster/geoserver/Polito/ows**
        filters:
          - StripPrefix=1
          - TokenRelay
```

Dettagli della Configurazione

- **uri:** Indica l'URL di GeoServer, che è accessibile solo tramite il server Spring.
- **predicates:** Specificano quali percorsi devono essere inoltrati al GeoServer. Solo quelli che iniziano con `/raster/geoserver/Polito/wms` o `/raster/geoserver/Polito/ows` vengono considerati.
- **filters:**
  - `StripPrefix:` Rimuove il prefisso dalla richiesta prima di inoltrarla al GeoServer.
  - `TokenRelay:` Consente di passare il token di autenticazione dell'utente dalla richiesta originale a quella inoltrata al GeoServer, mantenendo così il contesto di autenticazione.

3.5 Stili Raster

Anche per i file raster è possibile gestire gli stili. Poiché il dato viene renderizzato dal GeoServer, è necessario creare degli stili predefiniti su di esso e poi selezionare quale stile utilizzare. La creazione degli stili avviene tramite l'invio di file `.sld` al GeoServer, che descrivono come renderizzare il layer. All'interno di questi stili è possibile specificare diverse regole per la visualizzazione dei dati, come la scala di colori, la trasparenza, ecc. Questo permette di personalizzare la visualizzazione dei raster in base alle esigenze specifiche dell'utente.

3.5.1 Esempio di Stili Applicati

Di seguito sono riportate due immagini dello stesso raster, ciascuna visualizzata con uno stile diverso:

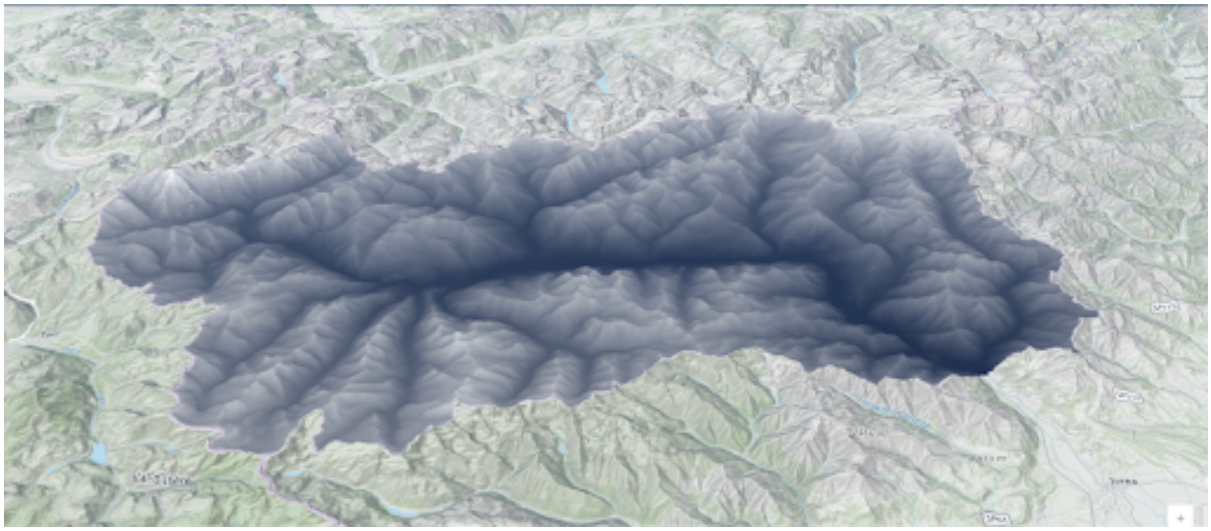


Figura 3.9: Stile Raster Grayscale

In questo caso, il raster è stato stilizzato con una scala di grigi per evidenziare le differenze di intensità dei valori. Questo stile è particolarmente versatile in quanto non necessita di una scala di colori specifica e può essere utilizzato per rappresentare dati di vario tipo.

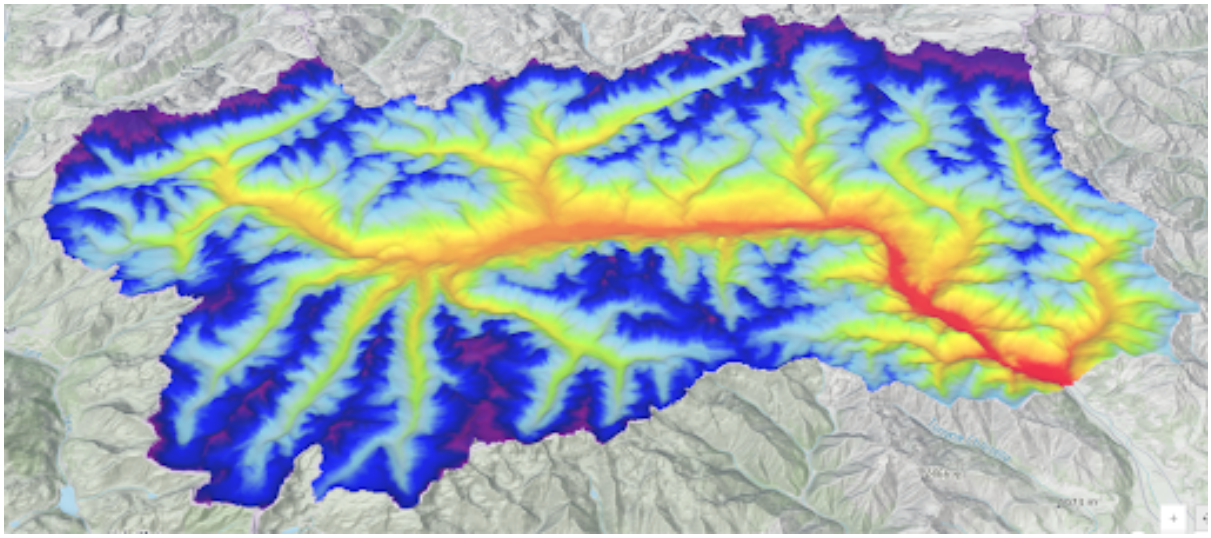


Figura 3.10: Stile Raster ColorStop

Questo stile utilizza una scala di colori per rappresentare i valori del raster. Ogni colore corrisponde a un intervallo di valori, consentendo di visualizzare in modo chiaro le differenze tra le varie zone del raster. anche se necessita di avere conoscenze pregresse sui dati per essere interpolato correttamente.

3.6 Converter

La presenza di molti sistemi di riferimento dei dati rende difficile la loro rappresentazione senza conoscere a priori come sono stati raccolti. Per risolvere questo problema, è stata adottata la libreria GDAL per convertire i dati nel sistema di riferimento `EPSG:23032` prima di fornire i file al GeoServer. A tal fine, è stato implementato un server in Go dedicato a gestire questa conversione.

La scelta di Go è motivata dalla sua leggerezza e velocità, che lo rendono particolarmente adatto per applicazioni server-side. La sua concorrenza nativa e le prestazioni elevate consentono di gestire in modo efficiente le richieste di conversione dei dati.

3.6.1 Codice Essenziale del Server Go

Di seguito viene presentato il codice essenziale del server Go:

```
func (s *Server) convert(w http.ResponseWriter, r *http.Request) {
    // Recupero dell'ID dalla query string
    id := r.URL.Query().Get("id")
    if id == "" {
        http.Error(w, "Missing id in query string", http.StatusBadRequest)
        return
    }

    // Esecuzione del comando gdalwarp per la conversione
    gdalCmd := exec.Command("gdalwarp", "-t_srs", "EPSG:23032", "/files/"+id,
    ↪  "/files/result")
    if err := gdalCmd.Run(); err != nil {
        http.Error(w, "Error executing gdalwarp command: "+err.Error(),
        ↪  http.StatusInternalServerError)
        return
    }

    // Rimozione del file originale
    if err := exec.Command("rm", "/files/"+id).Run(); err != nil {
        http.Error(w, "Error executing rm command: "+err.Error(),
        ↪  http.StatusInternalServerError)
        return
    }

    // Spostamento del file convertito
    if err := exec.Command("mv", "/files/result", "/files/"+id).Run(); err !=
    ↪  nil {
        http.Error(w, "Error executing mv command: "+err.Error(),
        ↪  http.StatusInternalServerError)
        return
    }
}
```

3.6.2 Spiegazione del Codice

- **Recupero dell'ID:** Il codice estrae l'ID del file dalla query string della richiesta HTTP. Se l'ID non è fornito, restituisce un errore di richiesta non valida (400 Bad Request).

- **Esecuzione del Comando gdalwarp:** Viene eseguito il comando gdalwarp per convertire il file nel sistema di riferimento `EPSG:23032`. Se si verifica un errore durante l'esecuzione, viene restituito un errore di server interno (500 Internal Server Error).
- **Rimozione del File Originale:** Dopo la conversione, il file originale viene rimosso utilizzando il comando `rm`. Se si verifica un errore, viene restituito un errore di server interno.
- **Spostamento del File Convertito:** Infine, il file convertito viene spostato nella posizione del file originale, sovrascrivendolo. Eventuali errori generati durante questa operazione sono gestiti con un errore di server interno.

3.6.3 GDAL

GDAL (Geospatial Data Abstraction Library) è una libreria open-source progettata per la lettura e scrittura di formati raster e vettoriali geospaziali. Supporta numerosi formati di file e consente operazioni complesse come la conversione tra diversi sistemi di riferimento spaziale. GDAL è ampiamente utilizzata nei sistemi GIS (Geographic Information Systems) per gestire e trasformare dati geospaziali in maniera efficiente.

Essendo rilasciata sotto licenza MIT, GDAL offre grande flessibilità agli sviluppatori. La licenza MIT è permissiva, il che significa che consente di utilizzare, modificare e distribuire il software anche in progetti proprietari, senza molte restrizioni. Questa caratteristica è vantaggiosa perché permette di integrare GDAL facilmente in applicazioni commerciali e open-source, senza preoccuparsi di violare licenze restrittive o di dover pubblicare il proprio codice.

Fonte: GDAL Official Documentation

3.7 Creazione di un Layer

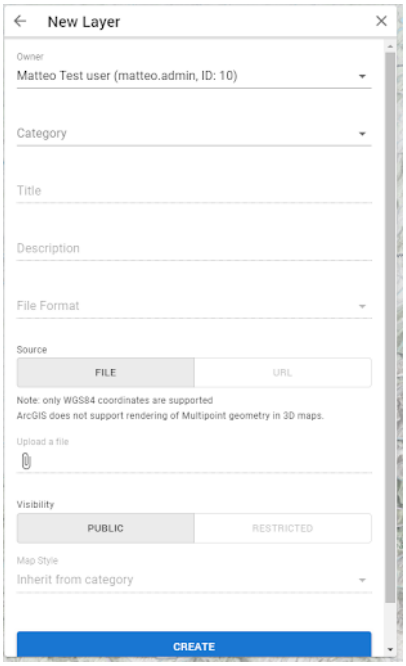


Figura 3.11: Interfaccia Creazione Layer

Quando si avvia la creazione di un nuovo layer (tramite il pulsante "+"), il primo passo consiste nel compilare le informazioni generali relative al dato geografico che si desidera rappresentare. Questi campi includono il titolo, la descrizione e il formato del file (GeoJSON, CSV, Shapefile o Raster).

Successivamente, occorre stabilire se il layer sarà pubblico o privato, il che determinerà la sua visibilità o meno per gli utenti non autenticati. I dati possono essere caricati direttamente dal dispositivo dell'utente come file, oppure forniti tramite un URL. In quest'ultimo caso, il layer si aggiornerà automaticamente ogni volta che i dati subiranno modifiche. A questo punto, viene selezionata una categoria specifica per organizzare il layer all'interno del sistema.

3.7.1 Formati Vettoriali e Raster

Dati Vettoriali: Se il file caricato è di tipo vettoriale (come GeoJSON o Shapefile), una volta premuto il pulsante `Create`, viene mostrata un'anteprima del layer. Questo consente all'utente di verificare visivamente i dati e, se soddisfatto, di confermare la creazione del layer.

Dati Raster: Se il layer è un dato raster (immagini georeferenziate come TIFF), viene richiesto all'utente di scegliere il formato di visualizzazione (ad esempio, grayscale, color stop, black and white). Una volta configurato, il dato raster viene inviato al GeoServer, che ne esporrà il contenuto attraverso il protocollo WMS (Web Map Service), permettendo la visualizzazione dinamica sulla mappa.

3.8 Cloud

L'applicazione è stata progettata per essere eseguita interamente in cloud, con l'obiettivo di rendere accessibili i servizi all'utente finale senza la necessità di installare software specifico sui propri dispositivi. L'accesso principale avviene tramite un'interfaccia web, direttamente utilizzabile tramite browser. Questo approccio cloud-native offre numerosi vantaggi, tra cui l'eliminazione delle complessità di installazione, la facilità di aggiornamento e manutenzione centralizzata, e la possibilità di scalare l'applicazione in base alle necessità dell'utente.

3.8.1 Docker Compose

Attualmente l'infrastruttura dell'applicazione è gestita tramite Docker Compose, uno strumento che consente di definire ed eseguire applicazioni multi-container in modo semplice ed efficace. Docker Compose permette di orchestrare l'intero stack applicativo attraverso un singolo file di configurazione YAML, in cui è possibile definire i servizi, le reti e i volumi necessari al funzionamento dell'applicazione.

Prima di approfondire Docker Compose, è importante comprendere i concetti di **container** e immagini Docker. I container sono ambienti isolati e leggeri che permettono di eseguire un'applicazione con tutte le sue dipendenze, librerie e configurazioni necessarie. A differenza delle macchine virtuali, i container condividono il kernel del sistema operativo host, il che li rende molto più efficienti in termini di risorse. Le **immagini Docker**, invece, rappresentano il modello immutabile di un container: contengono tutto il necessario per eseguire un'applicazione, inclusi il codice sorgente, le librerie e le dipendenze. Quando si avvia un container, questo viene creato a partire da un'immagine Docker.

Uno dei principali vantaggi di Docker Compose è la sua semplicità d'uso e la possibilità di eseguire applicazioni containerizzate su ambienti di sviluppo locali, nonché in produzione per piccoli cluster. Tuttavia, Docker Compose presenta alcune limitazioni significative, specialmente in termini di scalabilità e gestione delle risorse. In particolare, Docker Compose non supporta nativamente il bilanciamento del carico e la scalabilità automatica dei servizi in risposta al variare del carico di lavoro, il che può portare a inefficienze o addirittura a interruzioni del servizio durante i picchi di utilizzo.

Creazioni immagini docker

Per la creazione delle immagini Docker, abbiamo sviluppato uno script bash che automatizza il processo per ciascun servizio, generando un'immagine partendo da un Dockerfile. Abbiamo utilizzato un approccio a due fasi per ottimizzare la costruzione dei container. Nella prima fase, utilizziamo un'immagine più completa che include tutte le dipendenze necessarie per la build dell'applicazione. Nella seconda fase, invece, viene creata l'immagine finale, ottimizzata per l'esecuzione in un ambiente minimale. Di seguito è riportato un esempio di Dockerfile relativo al frontend di un'applicazione basata su Node e React, che implementa questo approccio.

```
# Stage 1: Build React App
FROM node:21 AS builder
ENV NODE_OPTIONS="--max_old_space_size=4096"
WORKDIR /app
# Copy package.json and package-lock.json to leverage Docker cache
COPY package*.json ./
# Install dependencies
RUN npm install
# Copy the rest of the application code
COPY . .
# Build the React app
RUN npm run build
# Stage 2: Create Nginx Container
FROM nginx:alpine
# Copy custom Nginx configuration file
COPY nginx.conf /etc/nginx/conf.d/default.conf
# Copy the built React app from the previous stage
COPY --from=builder /app/dist /usr/share/nginx/html
# Expose port
EXPOSE 80
# Command to run Nginx in the foreground
CMD ["nginx", "-g", "daemon off;"]
```

Struttura Docker Compose

Il file docker compose contiene la lista di servizi che compongono un deployment e le informazioni relative alla configurazione di ogni container.

```
services:
 server:
   container_name: "backend-server"
   image: geoplatform-server
   environment:
     - CONVERTER_URL=http://converter:8383/
     - SPRING_PROFILES_ACTIVE=sa2
     - RASTER_SERVER_URL=http://geoserver:8080
     - RASTER_STYLES_FOLDER=/geoserver/styles
   ports:
     - "8081:8080"
   depends_on:
     - "postgres"
     - "geoserver"
   volumes:
     - ${UPLOADS_FOLDER}:/files
     - ${ROOT_FOLDER}/import:/import
     - ${ROOT_FOLDER}/import.json:/import.json
     - ${LOGS_FOLDER}:/tmp/logs
     - ${ROOT_FOLDER}/geoserver/styles:/geoserver/styles
```

Le varie sezioni presenti in un docker compose sono:

1. **services**: Questa sezione contiene la lista dei servizi (contenitori) che compongono l'applicazione. Ogni servizio è definito da un nome (in questo caso, server) e può avere diverse configurazioni associate.

2. **server**: Questo è il nome del servizio, in questo caso rappresenta la parte SpringBoot di geospatial.
3. **container_name**: Questo campo specifica il nome del container che verrà creato per il servizio. In questo caso, il nome del container è "backend-server".
4. **image**: Questo campo specifica l'immagine Docker da utilizzare per il servizio. In questo caso, l'immagine è "geoplatform-server".
5. **environment**: Questo campo definisce le variabili d'ambiente che verranno passate al servizio. In questo caso, vengono passate alcune variabili d'ambiente necessarie per la configurazione del servizio.
6. **ports**: Questo campo definisce le porte che verranno esposte dal servizio. In questo caso, la porta 8080 del servizio verrà mappata sulla porta 8081 dell'host.
7. **depends_on**: Questo campo definisce le dipendenze del servizio da altri servizi. In questo caso, il servizio dipende dai servizi "postgres" e "geoserver".
8. **volumes**: Questo campo definisce i volumi che verranno montati nel servizio. In questo caso, vengono montati alcuni volumi necessari per il funzionamento del servizio.

3.8.2 Kubernetes

Per superare le limitazioni di Docker Compose, è comune migrare verso strumenti di orchestrazione più avanzati, come Kubernetes. Kubernetes è un sistema open-source progettato per automatizzare la distribuzione, il ridimensionamento e la gestione di applicazioni containerizzate. Esso permette di gestire in modo efficiente ambienti di produzione su larga scala, garantendo elevata disponibilità, scalabilità e resilienza.

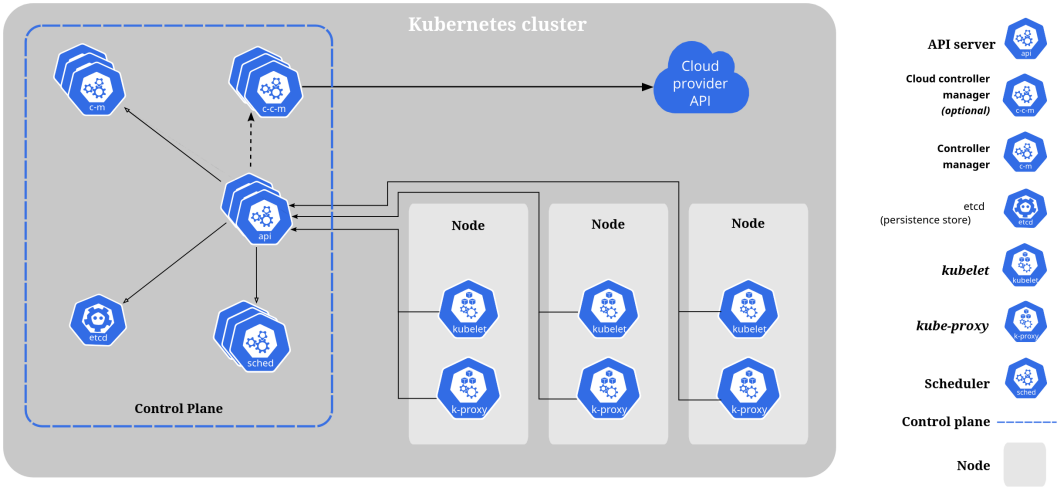


Figura 3.12: Struttura logica di Kubernetes

Kubernetes opera gestendo una serie di nodi facenti parte di un cluster. I componenti principali sono:

1. **Control Plane**: Si occupa di schedulare i pods sui vari worker node, garantendo la massima efficienza e la scalabilità dei servizi. Solitamente il control plane è distribuito su più host, in modo da garantire resilienza.

2. **Worker Node**: Hanno il compito di eseguire i pod. Ogni Worker Node include due componenti fondamentali: il Kubelet, che assicura l'esecuzione corretta dei pod comunicando con il Control Plane, e il Kube-proxy, che gestisce il networking del nodo, facilitando la comunicazione tra i servizi e i pod all'interno e all'esterno del cluster.

La transizione da Docker Compose a Kubernetes richiede diverse considerazioni, poiché i due strumenti utilizzano approcci e formati di configurazione differenti. Mentre Docker Compose utilizza un singolo file YAML per definire l'intero stack, Kubernetes richiede la suddivisione della configurazione in più file YAML per gestire vari aspetti come i deployment, i servizi, i volumi persistenti e le risorse.

3.8.3 Struttura logica di Kubernetes

Nella logica di Kubernetes, i principali concetti strutturali includono:

- **Pod**: L'unità base di esecuzione in Kubernetes, rappresenta una singola istanza di un'applicazione che può contenere uno o più container strettamente correlati.
- **Deployment**: Un Deployment è un oggetto Kubernetes che gestisce il ciclo di vita dei Pod, compreso il controllo delle repliche e gli aggiornamenti progressivi. Esso consente di dichiarare lo stato desiderato dei Pod e Kubernetes si occupa di mantenerlo.
- **Service**: Un Service è un'astrazione che definisce un endpoint di rete stabile per un insieme di Pod selezionati tramite etichette. I Service possono essere esposti internamente al cluster o esternamente tramite Load Balancer, assicurando che le richieste vengano bilanciate tra le repliche dei Pod.
- **ConfigMap**: Utilizzato per separare la configurazione dell'applicazione dal codice dell'immagine del container, i ConfigMap contengono dati di configurazione non sensibili che possono essere utilizzati dai Pod. Questi dati possono includere variabili d'ambiente, parametri di avvio e file di configurazione.
- **Secrets**: Simili ai ConfigMap, ma utilizzati per gestire informazioni sensibili come chiavi SSL, credenziali di accesso, e altre informazioni riservate. I Secrets sono crittografati e possono essere montati come volumi o esposti come variabili d'ambiente.
- **PersistentVolume (PV) e PersistentVolumeClaim (PVC)**: Kubernetes supporta la gestione della persistenza dei dati tramite PersistentVolume e PersistentVolumeClaim. Un PV è una risorsa di storage nel cluster, mentre un PVC è una richiesta di storage da parte di un utente. Questo meccanismo permette di mantenere i dati persistenti indipendenti dai Pod, garantendo che i dati siano accessibili anche in caso di riavvio o ridistribuzione dei Pod.

3.8.4 Transizione da Docker Compose a Kubernetes

La transizione da Docker Compose a Kubernetes richiede una serie di passaggi, tra cui:

1. **Decomposizione del Docker Compose**: Il primo passo consiste nel decomporre il file Docker Compose in più file YAML, ciascuno dei quali rappresenta un concetto specifico di Kubernetes, come Deployment, Service, ConfigMap, Secrets, PersistentVolume e PersistentVolumeClaim.

2. **Creazione dei manifesti Kubernetes**: Per ciascun servizio definito nel Docker Compose, è necessario creare un file YAML corrispondente che definisca il Deployment, il Service e gli altri oggetti necessari per il funzionamento del servizio.

3. **Configurazione dei volumi persistenti**: Se il Docker Compose utilizza volumi per memorizzare dati persistenti, è necessario creare PersistentVolume e PersistentVolumeClaim per garantire che i dati siano accessibili anche in un ambiente Kubernetes.

4. **Configurazione delle variabili d'ambiente e dei Secrets**: Le variabili d'ambiente e i Secrets definiti nel Docker Compose devono essere trasformati in ConfigMap e Secrets di Kubernetes per garantire che le informazioni sensibili siano protette e accessibili ai Pod.

5. **Creazione del cluster Kubernetes**: Una volta che i manifesti Kubernetes sono stati creati, è necessario creare un cluster Kubernetes e applicare i manifesti per distribuire l'applicazione. Questo processo può essere eseguito su un cluster locale o su un ambiente cloud gestito come Google Kubernetes Engine (GKE), Amazon Elastic Kubernetes Service (EKS) o Microsoft Azure Kubernetes Service (AKS).

6. **Monitoraggio e gestione**: Una volta che l'applicazione è stata distribuita su Kubernetes, è possibile utilizzare strumenti come Kubectl, Kubernetes Dashboard, Prometheus e Grafana per monitorare e gestire l'applicazione in modo efficiente.

7. **Configurazione Ingress e Load Balancer**: Per esporre l'applicazione all'esterno del cluster, è possibile configurare un Ingress Controller o un Load Balancer per instradare il traffico alle repliche dei Pod. Questo permette di garantire l'accesso all'applicazione da parte degli utenti esterni.

3.8.5 Esempio di configurazione Kubernetes

Di seguito è riportato un esempio di configurazione Kubernetes per un Deployment e un Service che corrispondono al servizio postgres definito nel Docker Compose.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: postgres
 namespace: geospatial
spec:
 replicas: 1
 selector:
   matchLabels:
     app: postgres
 template:
   metadata:
     labels:
       app: postgres
   spec:
     containers:
       - name: postgres
         image: postgres:16
         ports:
           - containerPort: 5432
         env:
           - name: POSTGRES_USER
             value: "username"
           - name: POSTGRES_PASSWORD
             value: "password"
           - name: PGDATA
             value: "/data/postgres"
           - name: POSTGRES_DB
             value: "geodb"
         volumeMounts:
           - name: postgres-storage
             mountPath: /data/postgres
           - name: init-sql
             mountPath: /docker-entrypoint-initdb.d/init.sql
             subPath: init.sql
     volumes:
       - name: postgres-storage
         hostPath:
           path: /home/volumes/postgres
       - name: init-sql
         configMap:
           name: postgres-init-sql
```

In questo file viene definito il namespace da usare, l'immagine del pod e le sue varie configurazioni, tra cui volumi montati e variabili di ambiente.

```yaml
apiVersion: v1
kind: Service
metadata:
 name: postgres
 namespace: geospatial
spec:
 selector:
   app: postgres
 ports:
   - protocol: TCP
     port: 5432
     targetPort: 5432
 type: ClusterIP
```

Questo file definisce il servizio che espone il pod PostgreSQL all'interno del cluster Kubernetes. Il servizio è configurato per utilizzare il protocollo TCP sulla porta 5432, che è la porta standard per PostgreSQL. Il tipo di servizio è ClusterIP, il che significa che il servizio è accessibile solo all'interno del cluster Kubernetes. Questo approccio garantisce che il database sia isolato e protetto da accessi esterni non autorizzati, migliorando la sicurezza complessiva dell'applicazione.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
 name:  postgres-init-sql
 namespace: geospatial
data:
 init.sql: |
   DO
   $$
      BEGIN
          IF NOT EXISTS (SELECT FROM pg_database WHERE datname = 'geodb') THEN
              PERFORM dblink_exec('dbname=postgres', 'CREATE DATABASE geodb');
          END IF;
      END
   $$;
```

Il ConfigMap contiene il file init.sql che verrà eseguito all'avvio del pod postgres per inizializzare il database.

3.9 Import dati da Web App

GeoSpatial offre la funzionalità di caricare file di grandi dimensioni, che possono raggiungere anche diversi GB. Tuttavia, i browser impongono limiti sulla dimensione delle richieste HTTP; ad esempio, Chrome non consente di eseguire POST con più di 4 GB di dati. Per superare questo problema, è stato necessario sviluppare un uploader personalizzato basato su chunk, che divide il file in parti più piccole per consentire il caricamento senza superare i limiti imposti dai browser.

3.9.1 Chunk Upload

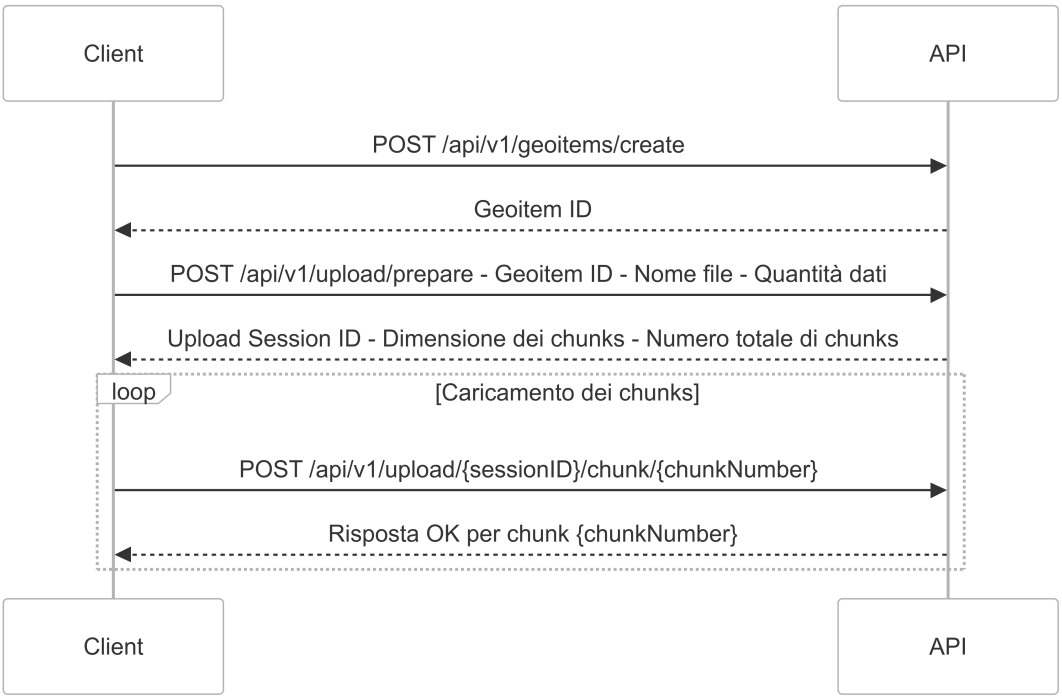


Figura 3.13: Chunk Upload

Per creare un layer e caricare il file corrispondente, è necessario seguire una serie di passaggi attraverso le API di GeoSpatial:

1. **Creazione del Geoitem**: Prima di tutto, è necessario creare un geoitem utilizzando l'API dedicata. Se la creazione ha successo, l'API restituisce un codice identificativo univoco, chiamato Geoitem ID. Questo ID sarà utilizzato nei passaggi successivi per collegare il file al geoitem creato.
2. **Inizializzazione dell'upload**: Successivamente, bisogna inviare una richiesta all'endpoint `/api/v1/upload/prepare` per inizializzare il processo di upload del file. In questa richiesta, è necessario specificare:
   - La quantità di dati da inviare.
   - Il nome del file.
   - Il Geoitem ID ottenuto in precedenza.

3. In risposta, l'API fornisce diverse informazioni cruciali, tra cui:
   - **Upload Session ID**: l'identificativo della sessione di upload.
   - **Dimensione dei chunks**: la dimensione di ciascun blocco di dati in cui il file sarà suddiviso.
   - **Numero totale di chunks**: quanti chunks saranno necessari per completare l'upload.
4. **Caricamento dei chunks**: Per caricare il file, è necessario inviare i singoli chunks tramite una richiesta POST all'endpoint `/api/v1/upload/{sessionID}/chunk/{chunkNumber}`. Qui:
   - {sessionID} rappresenta l'Upload Session ID ottenuto precedentemente.
   - {chunkNumber} è il numero del chunk che si sta inviando.

   È possibile inviare più chunks contemporaneamente e non è necessario inviarli in ordine sequenziale. Ogni chunk caricato correttamente riceve una risposta con stato 200 OK.
5. **Cancellazione dell'upload**: In qualsiasi momento, se è necessario interrompere il processo di upload, è possibile farlo inviando una richiesta DELETE all'endpoint `/api/v1/upload/{sessionID}`. Questo rimuove la sessione di upload e cancella i dati caricati fino a quel momento.

3.9.2 Chunk Upload Frontend

Il frontend di GeoSpatial è stato progettato per supportare l'upload chunked dei file. Riporto di seguito un esempio di codice TypeScript che illustra come implementare l'upload chunked in un'applicazione web.

```
const uploadChunk = async (chunk: Blob, currentChunk: number, id: string) => {
    const formData = new FormData();
    formData.append('file', chunk);
    return axios.post(makeGeoItemUploadUrl(id, currentChunk), formData);
 }


 const uploadFile = async (geoItemID: number) => {
    if (!file)
        return;


    const response = await axios.post(makePrepareUploadUrl(), {
        fileName: file.name,
        fileSize: file.size,
        geoItemID: geoItemID
    })
    const chunkSize = response.data.chunkSize;
    const totalChunks = response.data.totalChunks;
    const id = response.data.id;
    uploadSessionId.current = id;
    setUploadProcess([0, totalChunks]);
    let startByte = 0;
```

```
        for (let i = 0; i < totalChunks; i++) {
            const endByte = Math.min(startByte + chunkSize, file.size);
            const chunk = file.slice(startByte, endByte);
            const response = await uploadChunk(chunk, i, id);
            startByte = endByte;
            setUploadProcess([i, totalChunks]);
            if (!isUploading.current || response.status !== 200)
                break;
        }
    };
```

In questo esempio si presuppone l'esistenza di uno stato `file` che rappresenta il file da caricare e uno stato `uploadProcess` che tiene traccia del progresso dell'upload.

3.9.3 Chunk Upload - Cleanup

Per garantire che i file temporanei non utilizzati non occupino spazio inutilmente, è necessario implementare un meccanismo di cleanup che rimuova i file temporanei dopo un certo periodo di tempo. Questo processo può essere eseguito in modo automatico impostando una Scheduled Task su Spring Boot, che esegue periodicamente un'operazione di pulizia per rimuovere i file temporanei scaduti.

```
@Transactional
@Scheduled(fixedRate = CHECK_TIMER)
public void cancelExpiredSessions() {
   System.out.println("Running scheduled operation");
   LocalDateTime time = LocalDateTime.now().minus(FILE_SESSION_TIMEOUT,
   ↪  ChronoUnit.MILLIS);
   System.out.println("Deleting sessions before " + time);
   fileInfoRepository.getFileInfoByCreatedOnBeforeAndStatus(time,
   ↪  "IN_PROGRESS").forEach(fileInfo -> {
       System.out.println("Deleting session " + fileInfo);
       String extensionWithDot = FileUtil.getExtension(fileInfo.getFileName());
       File filePath = new File(destinationFolderPath + "/" + fileInfo.getId() +
       ↪  extensionWithDot);
       filePath.delete();
       fileInfoRepository.deleteById(fileInfo.getId());
   });
}
```

In questo esempio, viene definito un metodo `cancelExpiredSessions` che viene eseguito periodicamente in base a un intervallo di tempo specificato. Il metodo recupera tutti i file temporanei che sono stati creati prima di un certo periodo di tempo e che sono ancora in stato "IN_PROGRESS". Per ciascun file, il metodo elimina il file dal disco e rimuove l'entry corrispondente dal database.

3.10 Sicurezza

Un aspetto fondamentale nella gestione di un'applicazione web è garantire la sicurezza, la gestione degli accessi e la differenziazione degli utenti in base ai loro ruoli. In Geospatial, questi aspetti sono affrontati attraverso la gestione di tre tipi principali di utenze:

- **Utente anonimo**: Questa è l'utenza di base che può solo visualizzare i dati pubblici. Non ha la possibilità di modificare né cancellare alcun dato, mantenendo un accesso limitato e sicuro alle sole informazioni accessibili al pubblico.
- **Utente manger**: Gli utenti con il ruolo di Manager hanno un accesso più ampio. Possono creare nuove categorie, layer o stili, visualizzare tutti i tipi di dati presenti nel sistema e gestire i dati che hanno creato, inclusa la modifica e l'eliminazione di tali dati. Questo ruolo è essenziale per mantenere e aggiornare le risorse del sistema senza compromettere la sicurezza dei dati creati da altri utenti.
- **Utente amministratore**: Gli Admin hanno il massimo livello di controllo all'interno dell'applicazione. Possono effettuare ogni tipo di modifica, inclusa la creazione, cancellazione e modifica degli utenti. Questo ruolo è responsabile della gestione complessiva del sistema, inclusi gli aspetti di sicurezza e configurazione.

Per gestire questi aspetti, è stato scelto di utilizzare Spring Security, un modulo altamente personalizzabile per l'autenticazione e il controllo degli accessi nelle applicazioni basate su Spring. Spring Security è lo standard de facto per la sicurezza delle applicazioni Spring ed è progettato per offrire sia autenticazione che autorizzazione. La vera forza di Spring Security risiede nella sua capacità di estendersi facilmente per soddisfare requisiti personalizzati.

3.10.1 OAuth2 Password Grant

Per l'autenticazione degli utenti, è stato scelto di utilizzare il protocollo OAuth2 con il grant type Password. Questo approccio consente agli utenti di autenticarsi utilizzando le proprie credenziali (username e password) e di ottenere un token di accesso che può essere utilizzato per accedere alle risorse protette. Il token di accesso è valido per un periodo di tempo limitato e può essere utilizzato per effettuare richieste alle API protette.
Anche se questo metodo semplifica l'implementazione e l'integrazione, è considerato meno sicuro rispetto ad altri flussi di OAuth 2.0, come l'Authorization Code Grant, poiché richiede che il client gestisca direttamente le credenziali dell'utente. Tuttavia, la scelta è stata fatta per la sua semplicità di implementazione e per soddisfare le esigenze specifiche dell'applicazione.

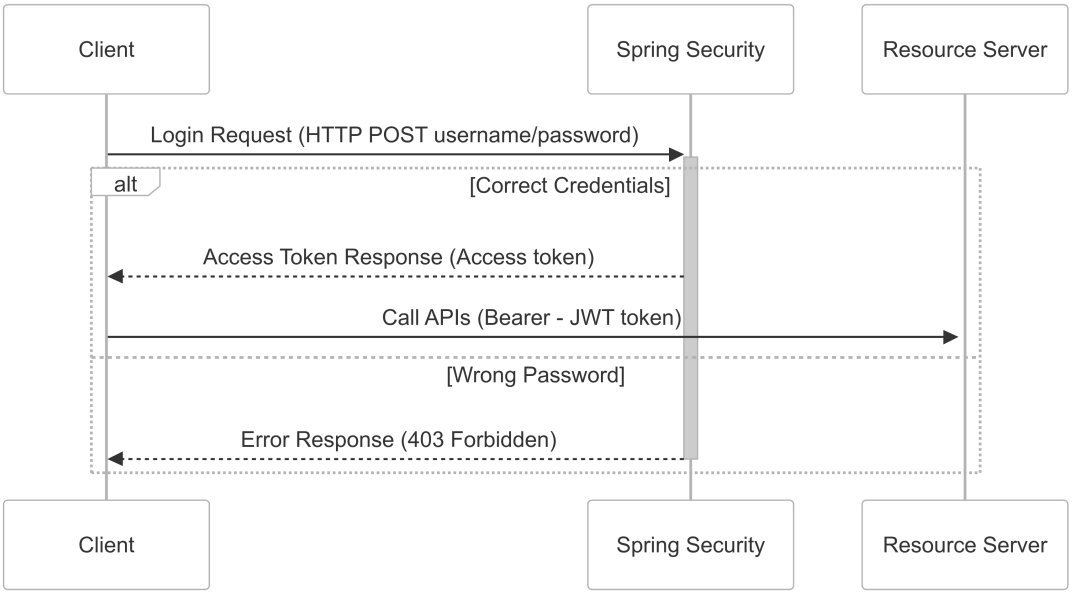


Figura 3.14: OAuth2 Password Grant

Il flusso di autenticazione si sviluppa come segue:

1. **Inserimento delle credenziali**: L'utente compila un modulo di login con nome utente e password.
2. **Invio della richiesta**: Il browser invia una richiesta POST all'endpoint `/login` del server web.
3. **Verifica delle credenziali**:
   - Se le credenziali sono corrette, il server risponde con un token JWT (JSON Web Token).
   - Se le credenziali sono errate, il server restituisce un errore 403, richiedendo all'utente di ripetere il processo di inserimento delle credenziali.
4. **Utilizzo del token**: Per ogni richiesta successiva al server, il client deve includere il token JWT ricevuto nel passo precedente nell'header della richiesta come Bearer Token. Questo consente al server di identificare e autenticare l'utente.

3.10.2 Password Encryption

Per garantire la conformità alle normative sulla sicurezza dei dati e proteggere le informazioni degli utenti in caso di data breach, si è scelto di utilizzare l'algoritmo bcrypt per la crittografia delle password. Rispetto ad altri algoritmi, bcrypt offre significativi vantaggi in termini di sicurezza, specialmente quando si tratta di proteggere le password dagli attacchi.

3.10.3 BCrypt

Algoritmi come MD5 e le varie versioni di SHA sono stati progettati per essere "general purpose", cioè per gestire grandi quantità di dati nel modo più rapido possibile. Questa velocità li rende eccellenti per garantire l'integrità dei file, ma non li rende ideali per proteggere le password. Infatti, uno dei principali obiettivi di bcrypt è esattamente il contrario: rendere l'operazione di hashing delle password il più lenta e onerosa possibile, complicando notevolmente gli attacchi di brute force.

Un server moderno può calcolare l'hash MD5 di circa 330MB di dati al secondo. Questo significa che un hacker potrebbe testare ogni possibile combinazione di password alfanumeriche a 6 caratteri in circa 40 secondi. E la situazione peggiora se l'attaccante dispone di hardware più potente. Con un investimento di circa 2000 dollari e qualche settimana di pratica con il framework CUDA, è possibile costruire un cluster di supercomputer in grado di provare circa 700 milioni di password al secondo, permettendo di craccare una password al secondo.

Bcrypt, invece, è stato progettato per essere deliberatamente lento. Utilizza una variante del processo di chiave dell'algoritmo Blowfish e introduce un "work factor", un parametro che può essere regolato per aumentare il tempo necessario per calcolare l'hash. Ad esempio, con un work factor di 12, bcrypt potrebbe impiegare circa 0,3 secondi per calcolare l'hash di una password semplice come "yaaa" su un laptop moderno, mentre MD5 richiederebbe meno di un microsecondo per la stessa operazione, una differenza di 5 ordini di grandezza. Per contesto se con MD5 si riesce ad esplorare tutte le password di 6 caratteri in 40 secondi con bcrypt ci si metterebbe 12 anni.

3.11 Applicazione Android

Una delle principali limitazioni della nostra web app è la necessità di utilizzare dati preesistenti, come file provenienti da sensori o censimenti, senza offrire la possibilità di inserire dati manualmente, ad esempio tramite questionari o input diretto da parte degli utenti. Per superare questa limitazione, abbiamo avviato lo sviluppo di una mobile app che opererà in parallelo alla web app, permettendo una raccolta dati più flessibile e integrata, anche da fonti esterne.

Questa necessità è emersa dalla richiesta di integrare il processo di revisione delle barriere paramassi presenti in Piemonte e in Valle d'Aosta, con l'obiettivo di migliorare la raccolta e gestione dei dati sul campo, che attualmente vengono rilevati tramite questionari cartacei.

Con lo sviluppo della mobile app, sarà possibile digitalizzare completamente questo processo: gli operatori potranno compilare questionari digitali, tracciare la posizione GPS delle barriere, scattare foto delle problematiche rilevate e inviare i dati. Questi verranno saranno visualizzabili sulla mappa, insieme alla classificazione del livello di attenzione, rendendo il processo di revisione più efficiente e preciso.

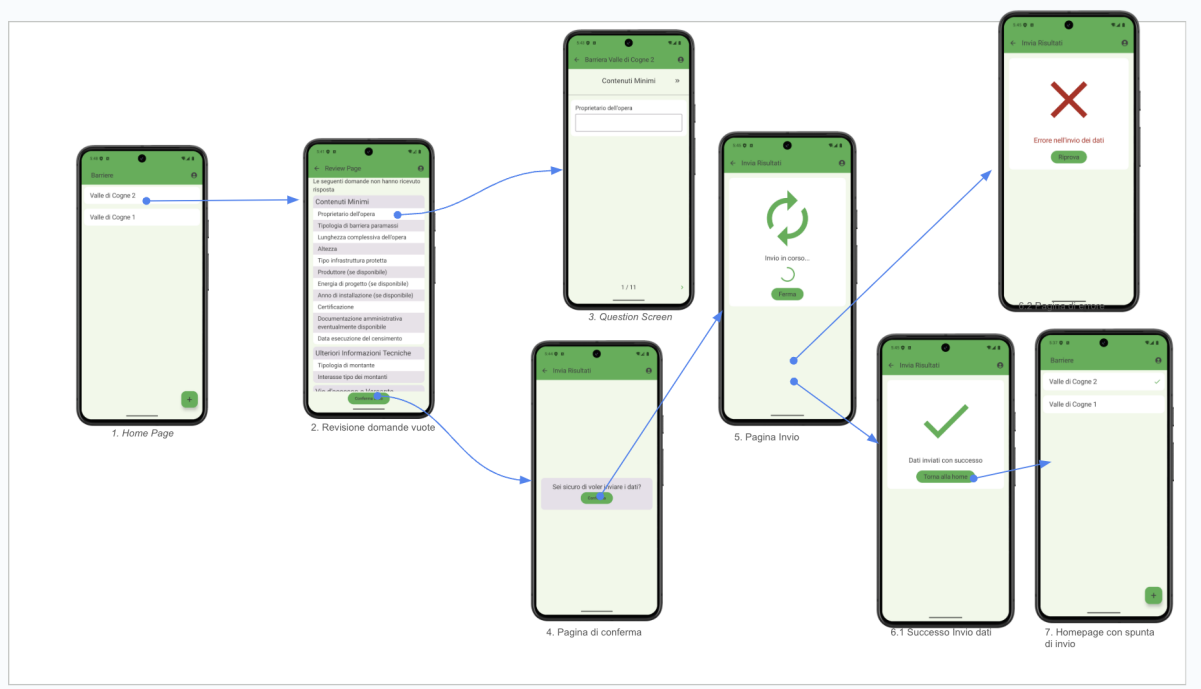


Figura 3.15: Schermata riempilogo questionario mobile app

3.11.1 Manifesto Android

Il file **AndroidManifest.xml** è un elemento fondamentale per qualsiasi app Android, in quanto specifica le componenti principali dell'applicazione, le configurazioni globali e i permessi necessari per accedere alle risorse di sistema. È importante notare che nel manifesto sono anche indicate le versioni di Android supportate dall'app.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.READ_MEDIA_IMAGES" />
    <uses-permission
    ↪  android:name="android.permission.READ_MEDIA_VISUAL_USER_SELECTED" />
    <uses-permission android:name="android.permission.INTERNET" />

    <uses-feature android:name="android.hardware.camera" />

    <application
        android:name=".db.MyApp"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:networkSecurityConfig="@xml/network_security_config"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.BarriereForm"
        android:usesCleartextTraffic="true"
        tools:targetApi="31">

        <provider
            android:name="androidx.core.content.FileProvider"
            android:authorities="${applicationId}.provider"
            android:exported="false"
            android:grantUriPermissions="true">
            <meta-data
                android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/path_provider" />
        </provider>
        ...
    </application>
</manifest>
```

Il manifesto include diverse dichiarazioni di permessi, essenziali per garantire l'accesso a funzionalità critiche del dispositivo:

- **ACCESS_COARSE_LOCATION e ACCESS_FINE_LOCATION**: Necessari per ottenere la posizione del dispositivo, utile per tracciare le barriere paramassi sul campo.
- **ACCESS_NETWORK_STATE**: Consente di monitorare lo stato della connessione di rete, assicurando che i dati vengano sincronizzati solo quando è disponibile una connessione.
- **CAMERA**: Permette l'accesso alla fotocamera del dispositivo per scattare foto delle barriere o delle problematiche.

- **READ_MEDIA_IMAGES e READ_MEDIA_VISUAL_USER_SELECTED**: Consentono all'app di leggere le immagini archiviate nel dispositivo, offrendo la possibilità di utilizzare file multimediali già presenti.
- **INTERNET**: Permette l'accesso a Internet, essenziale per sincronizzare i dati raccolti con il server centrale.

3.11.2 Jetpack Compose

Nel nostro progetto, abbiamo scelto di utilizzare Jetpack Compose per sviluppare l'interfaccia utente della nostra applicazione Android. Jetpack Compose è il toolkit moderno raccomandato da Android per la creazione di interfacce utente native.

Le principali caratteristiche di Jetpack Compose includono:

- **Programmazione Dichiarativa**: Jetpack Compose ci consente di costruire l'interfaccia utente attraverso funzioni Kotlin, anziché utilizzare layout XML tradizionali. Questo approccio permette di concentrarsi sul contenuto e sul comportamento, rendendo il codice più chiaro e facile da comprendere. Ad esempio, possiamo definire un pulsante direttamente nel nostro codice senza dover gestire complessi file XML.
- **Componenti Riutilizzabili**: Con Compose, creiamo componenti UI chiamati "composables". Questi componenti sono altamente riutilizzabili e modulari, permettendoci di costruire interfacce complesse a partire da parti più semplici. Ad esempio, possiamo definire un componente "Greeting" che accetta un nome come parametro e restituisce un saluto personalizzato.
- **Gestione dello Stato**: Jetpack Compose gestisce automaticamente gli aggiornamenti dell'interfaccia utente in base ai cambiamenti di stato. Se i dati cambiano, l'interfaccia si aggiorna automaticamente per riflettere queste modifiche, senza la necessità di gestire manualmente le riconfigurazioni dell'interfaccia.
- **Supporto per Material Design**: Compose offre un supporto integrato per Material Design, consentendo di creare interfacce utente gradevoli e moderne. Possiamo facilmente utilizzare componenti come bottoni, schede e menu che seguono le linee guida di design di Google.

Inoltre 'utilizzo di Jetpack Compose offre diversi vantaggi rispetto all'approccio tradizionale basato su XML:

- **Semplificazione dello Sviluppo**: Riduce la complessità del codice, permettendo di costruire interfacce in modo più diretto e intuitivo.
- **Migliore Manutenibilità**: La struttura modulare rende il codice più facile da mantenere e aggiornare, facilitando eventuali modifiche future.
- **Aumento della Produttività**: Gli sviluppatori possono dedicare più tempo alla logica dell'applicazione e meno alla gestione di layout complessi.

Esempio di Jetpack Compose

Figura 3.16: Interfaccia utente in Jetpack Compose

3.11.3 Esempio di utilizzo di Jetpack Compose per costruire una lista di domande

In questo esempio, vedremo come Jetpack Compose viene utilizzato per creare una lista di domande, gestire lo stato e le interazioni dell'utente. Il codice è spezzato in sezioni per evidenziare ciascuna parte.

Definizione della funzione composable `QuestionList`

```
@Composable
fun QuestionList(router: Router, id: String, vm: MainViewModel) {
    var questionList by remember { mutableStateOf(emptyList<QuestionInfo>()) }
    LaunchedEffect(Unit) {
        questionList = vm.getQuestionByBarrierId(id)
    }

    BackHandler {
        router.toHome()
    }
```

- **@Composable**: La funzione è annotata con `@Composable`, che indica che è un composable, ossia una funzione utilizzabile per definire l'interfaccia utente in modo dichiarativo.

- **Gestione dello stato**: Viene utilizzato `remember` con `mutableStateOf` per gestire lo stato della lista delle domande (`questionList`). Questo permette di aggiornare lo stato reattivamente quando cambiano i dati.
- **LaunchedEffect**: Con `LaunchedEffect(Unit)`, viene eseguita un'operazione asincrona per recuperare le domande attraverso il `ViewModel` (qui il metodo `getQuestionByBarrierId(i`
- **BackHandler**: Intercetta l'evento del pulsante "indietro" dell'utente, reindirizzandolo alla schermata principale (`router.toHome()`).

Layout principale con Box e LazyColumn

```
Box(modifier = Modifier.fillMaxSize().padding(horizontal = 8.dp)) {
    LazyColumn(modifier = Modifier.fillMaxWidth()) {
        items(questionList.groupBy { it.section }.toList()) { (section, questions)
        ↪ ->
            var showAccordion by rememberSaveable { mutableStateOf(false) }
```

- **Box**: Un contenitore che permette di sovrapporre e posizionare elementi. Qui viene utilizzato con il modificatore `fillMaxSize()` per occupare tutto lo spazio disponibile.
- **LazyColumn**: Componente efficiente per visualizzare liste di grandi dimensioni. Invece di caricare tutti gli elementi subito, carica solo quelli visibili, migliorando le prestazioni.
- **Raggruppamento per sezione**: Le domande sono raggruppate per sezione (`groupBy { it.section }`), e ogni gruppo viene visualizzato come una scheda (card).
- **Stato per la visualizzazione dell'accordion**: `rememberSaveable` salva lo stato di `showAccordion`, che determina se mostrare o nascondere le domande all'interno di ogni sezione.

Creazione della card per ogni sezione

```
Card(
    modifier = Modifier.padding(vertical = 8.dp),
    colors = CardDefaults.cardColors(
        containerColor = MaterialTheme.colorScheme.surface
    )
) {
    Row(modifier = Modifier.padding(8.dp)) {
        Text(
            text = section.sectionName,
            style = MaterialTheme.typography.titleMedium,
            modifier = Modifier
                .padding(8.dp)
                .weight(1f)
                .clickable {
                    router.toQuestionCard(id, section.sectionName, 0)
                },
            textAlign = TextAlign.Left
        )
```

- **Card**: Un componente che rappresenta una "scheda" con bordi arrotondati e ombra, ideale per contenere le sezioni delle domande. La card ha un padding verticale per separare le sezioni.
- **Row**: Una riga orizzontale che contiene il titolo della sezione (nel componente `Text`) e il contatore delle risposte. Il `Modifier.weight(1f)` è usato per distribuire lo spazio proporzionalmente.
- **Text**: Viene visualizzato il nome della sezione. Il modificatore `clickable` permette all'utente di cliccare e navigare alla schermata della domanda (`router.toQuestionCard`).

Contatore delle risposte completate

```
val response = questions.map { it.toQuestionEntity(id).toResponse() }
val counter = response.count { isResponseFilled(it) }
Row(modifier = Modifier.clickable { showAccordion = !showAccordion }) {
    Text(
        text = "$counter/${questions.size}",
        style = MaterialTheme.typography.titleMedium,
        modifier = Modifier.padding(vertical = 8.dp),
        color = MaterialTheme
            .colorScheme
            .onSurface.copy(alpha = 0.6f)
    )
    Icon(
        imageVector = ImageVector.vectorResource(
            id = if (showAccordion)
                    R.drawable.baseline_keyboard_arrow_up_24
                else R.drawable.baseline_keyboard_arrow_down_24
        ),
        contentDescription = "Arrow",
        modifier = Modifier.align(Alignment.CenterVertically)
    )
}}
```

- **Contatore risposte**: La variabile `counter` conta quante domande sono state risposte all'interno di una sezione. Questo è ottenuto mappando ogni domanda a un'entità risposta (`toResponse()`) e contando quelle che sono compilate con `isResponseFilled(it)`.
- **Toggle accordion**: Un'altra riga (`Row`) con un `Modifier.clickable`, che permette all'utente di espandere o ridurre l'accordion per mostrare o nascondere le domande.
- **Arrow Icon**: L'icona cambia a seconda che l'accordion sia aperto o chiuso, grazie all'uso condizionale dell'icona della freccia (`arrow_up` o `arrow_down`).

Espansione dell'accordion e visualizzazione delle domande

```
if (showAccordion) {
    Column {
        questions.forEachIndexed { index, question ->
            Row(
                modifier = Modifier
                    .fillMaxWidth()
                    .background(
                        if (index % 2 == 0)
                            MaterialTheme.colorScheme.surface
                        else MaterialTheme.colorScheme.secondary
                    )
                    .padding(vertical = 8.dp, horizontal = 16.dp),
                verticalAlignment = Alignment.CenterVertically
            ) {
                Text(
                    text = question.question,
                    style = MaterialTheme.typography.bodyLarge.copy(fontSize =
                    ↪  20.sp),
                    modifier = Modifier
                        .weight(3f)
                        .clickable {
                            router.toQuestionCard(id, section.sectionName, index)
                        }
                )
                if (response[index].isNotEmpty()) {
                    CircleImage(color = MaterialTheme.colorScheme.primary)
                } else {
                    CircleImage(color = warningYellow)
                }
            }
        }
    }
}
...
```

- **Condizionale `if (showAccordion)`**: Se l'accordion è espanso (`showAccordion == true`), viene renderizzato un elenco di domande all'interno di una `Column`.
- **Alternanza dei colori delle righe**: Ogni riga che contiene una domanda alterna il colore di sfondo (`surface` e `secondary`) per migliorare la leggibilità.
- **Domanda cliccabile**: Ogni domanda è cliccabile, e quando viene cliccata, porta alla visualizzazione dettagliata di quella domanda specifica.
- **Indicatori di risposta**: A destra di ogni domanda, un cerchio colorato indica se la risposta è stata compilata (colore primario) o se manca (colore giallo di avviso).

3.11.4 Navigazione nell'Applicazione

All'interno della nostra applicazione, gestiamo la navigazione tra le schermate utilizzando una classe chiamata Router. Questa classe si occupa di interfacciarsi con il NavHostController, che è parte dell'architettura di navigazione di Jetpack Compose. Di seguito, vediamo come funziona il Router e come viene utilizzato nella navigazione dell'app.
Di seguito è riportato un esempio di configurazione del Router:

```
    class Router(private val navController: NavHostController) {

    fun toHome() {
        Log.d("XXX", "toHome() with navController.graph.id =
        ↪  ${navController.graph.id}")
        navController.navigate(HomeRoute) {
            popUpTo(navController.graph.id)
        }
    }

    fun toLogin() {
        navController.navigate(LoginRoute)
    }

    fun toQuestionCard(id: String, section: String, startIndex: Int) {
        navController.navigate(FormRoute(id, section, startIndex))
    }

    fun toUserPage() {
        navController.navigate(UserPageRoute)
    }

    fun toQuestionList(id: String) {
        Log.d("XXX", "toQuestionList($id)")
        navController.navigate(QuestionListRoute(id))
    }

    fun getId(): String {
        return navController.currentBackStackEntry?.arguments?.getString("id") ?:
        ↪  "1"
    }

    fun back() {
        if (navController.previousBackStackEntry != null) {
            navController.popBackStack()
        } else {
            navController.navigate(HomeRoute)
        }
    }

    fun toReviewPage(id: String) {
        navController.navigate(ReviewPageRoute(id))
    }

    fun toSendDataPage(id: String) {
        navController.navigate(SendDataPageRoute(id))
    }
}
```

I ruoli della classe Router definita sono i seguenti:

1. **Navigazione a Schermate Specifiche**:
   - **toHome()**: Questo metodo naviga alla schermata principale dell'app. Usa popUpTo per rimuovere tutte le schermate precedenti dallo stack, assicurando che l'utente non possa tornare indietro.
   - **toLogin()**: Naviga alla schermata di login senza rimuovere le schermate precedenti.
   - **toQuestionCard()**: Naviga a una schermata specifica di una domanda, passando l'ID della barriera, la sezione e l'indice iniziale. Questo metodo è chiamato quando un utente seleziona una domanda dalla lista.
2. **Gestione dello Stato**:
   - **getId()**: Restituisce l'ID corrente dalla back stack, consentendo di accedere ai dati associati alla schermata corrente.
   - **back()**: Questo metodo gestisce la navigazione indietro. Se ci sono schermate precedenti, utilizza popBackStack() per tornare indietro, altrimenti naviga alla home.
3. **Navigazione con Dati**:
   - Le funzioni di navigazione possono anche ricevere parametri, come id, che possono essere utilizzati per caricare dati specifici nella nuova schermata.

Esempio di Utilizzo del Router

L'esempio seguente illustra l'uso del router per navigare verso una schermata di dettaglio quando un utente fa clic su una domanda nella lista delle domande.:

```
Text(
    text = question.question,
    style = MaterialTheme.typography.bodyLarge.copy(fontSize = 20.sp),
    modifier = Modifier.weight(3f).clickable {
        router.toQuestionCard(id, sectionName, index)
    }
)
```

In questo frammento di codice, il modifier.clickable sull'oggetto Text consente all'utente di fare clic sul testo della domanda. Quando l'utente fa clic, viene chiamato il metodo toQuestionCard del router, che gestisce la navigazione alla schermata di dettaglio della domanda, passando l'ID della barriera, il nome della sezione e l'indice della domanda.

3.11.5 Gestione dei Dati nell'Applicazione

Nel nostro progetto, abbiamo implementato un sistema robusto per la gestione dei dati utilizzando Room, una libreria di persistenza per Android che semplifica le operazioni di database, e Hilt, una libreria per la dependency injection che facilita la gestione delle dipendenze. Questo approccio ci consente di mantenere il codice pulito, organizzato e facilmente manutenibile.

Configurazione del Database con Room

Per gestire la persistenza dei dati, abbiamo creato un database chiamato `AppDatabase`. Questa classe estende `RoomDatabase` e serve come punto centrale per accedere ai vari DAO (Data Access Object) della nostra applicazione.

Ecco come è strutturato il database:

```
@Database(entities = [Barrier::class, QuestionEntity::class], version = 14,
↪  exportSchema = false)
@TypeConverters(OptionsConverter::class)
abstract class AppDatabase : RoomDatabase() {
    abstract fun barrierDao(): BarrierDao
    abstract fun questionDao(): QuestionDao

    companion object {
        @Volatile
        private var INSTANCE: AppDatabase? = null
    }
}
```

Spiegazione del codice:

- `@Database`: Annotazione che definisce il database, specificando le entità che contiene (in questo caso, `Barrier` e `QuestionEntity`), la versione del database e altre configurazioni.
- `DAO`: I metodi astratti `barrierDao()` e `questionDao()` ci forniscono l'accesso ai metodi per interagire con le rispettive entità.

Definizione delle Entità

Le entità rappresentano le tabelle del database. Abbiamo definito due entità: `Barrier` e `QuestionEntity`.

`Barrier`:

```
@Entity(tableName = "barriers")
data class Barrier(
    @PrimaryKey
    val id: String,
    val numPillars: Int,
    var sent: Boolean = false
)
```

`QuestionEntity`:

```
@Entity(
    tableName = "question",
    primaryKeys = ["questionId", "barrierId"],
    indices = [Index(value = ["barrierId"])],
    foreignKeys = [ForeignKey(
        entity = Barrier::class,
        parentColumns = ["id"],
        childColumns = ["barrierId"],
        onDelete = ForeignKey.CASCADE
    )]
)
data class QuestionEntity(
    val questionId: Int,
    val barrierId: String,
    val section: String,
    val question: String,
    val optionLabel: String,
    val options: String,
    val imageLabel: String,
    val positionLabel: String,
    val repeatable: Boolean,
    val dateLabel: String,
    val textLabel: String,
    val numberLabel: String,
    var response: String,
    val importanceClass: Int,
    val isOptional: Boolean
)
```

Spiegazione del codice:

- `@Entity`: Annotazione per definire che la classe rappresenta un'entità del database.
- `@PrimaryKey`: Indica il campo che funge da chiave primaria nella tabella.
- `@ForeignKey`: Definisce una relazione tra `QuestionEntity` e `Barrier`, assicurando che ogni domanda sia collegata a una barriera specifica.

Repository per l'Accesso ai Dati

Abbiamo implementato i repository per gestire l'accesso ai dati in modo strutturato. I repository forniscono un'interfaccia per recuperare e manipolare i dati, isolando la logica di accesso al database.

`BarrierRepository`:

```
class BarrierRepository @Inject constructor(private val barrierDao: BarrierDao) {

    fun getAllBarriers(): List<Barrier> {
        return barrierDao.getAllBarriers()
    }

    suspend fun insertBarrier(barrier: Barrier) {
        barrierDao.insertBarrier(barrier)
    }

    suspend fun deleteBarrier(barrierId: String) {
        barrierDao.getAllBarriers().find { it.id == barrierId }?.let {
            barrierDao.deleteBarrier(it)
        }
    }

    suspend fun markBarrierAsSent(barrierId: String) {
        barrierDao.markBarrierAsSent(barrierId)
    }

    fun getBarrierById(id: String): Barrier {
        return barrierDao.getBarrierById(id)
    }
}
```

Spiegazione del codice:

- `@Inject`: Utilizziamo l'annotazione `@Inject` per abilitare l'iniezione delle dipendenze tramite Hilt, permettendo al repository di ricevere il DAO senza doverlo creare manualmente.
- `getAllBarriers()`: Recupera tutte le barriere dal database.
- `insertBarrier(barrier: Barrier)`: Inserisce una nuova barriera nel database in modo asincrono.
- `deleteBarrier(barrierId: String)`: Elimina una barriera specifica dal database.
- `getBarrierById(id: String)`: Recupera una barriera per ID.

Integrazione di Hilt per la Dependency Injection

Abbiamo utilizzato Hilt per facilitare la gestione delle dipendenze. Annotando la classe principale dell'applicazione con `@HiltAndroidApp`, Hilt viene configurato per gestire l'iniezione delle dipendenze in tutto il progetto, inclusi i repository e i DAO. Questo approccio ci consente di avere un codice più pulito e modulare, migliorando la manutenibilità dell'applicazione.

```
@HiltAndroidApp
class MyApp : Application()
```

3.11.6 Invio Dati al Server con Retrofit

È stato utilizzato Retrofit per gestire la comunicazione con il server e inviare richieste di login. Di seguito viene mostrato un esempio di come è stata implementata la chiamata per l'autenticazione degli utenti, evidenziando come questo rappresenti solo uno dei vari metodi API utilizzati nel progetto.

Esempio di Autenticazione dell'Utente

Nel nostro codice, utilizziamo il metodo `login` dell'interfaccia `ApiService` per inviare la richiesta al server:

```
apiService.login(loginRequest).enqueue(object : Callback<LoginResponse> {
    override fun onResponse(call: Call<LoginResponse>, response:
    ↪  Response<LoginResponse>) {
        Log.d(TAG, "onResponse: ${response.body()}")
        if (response.body()?.access_token?.isNotEmpty() == true) {
            resultState.value = Jwt(response.body()!!.access_token,
            ↪  response.body()!!.refresh_token)
            viewModel.setJwt(resultState.value)
        } else {
            resultState.value = Jwt("", "")
            viewModel.setJwt(resultState.value)
        }
    }

    override fun onFailure(call: Call<LoginResponse>, t: Throwable) {
        resultState.value = Jwt("", "")
    }
})
```

Spiegazione del Codice:

- **Chiamata API**: Utilizziamo `apiService.login(loginRequest)` per inviare la richiesta di login al server.
- **Gestione della Risposta:**
  - `onResponse`: Se riceviamo una risposta valida con un token di accesso, aggiorniamo lo stato del risultato con il token e il refresh token. Se il token non è presente o è vuoto, impostiamo un JWT vuoto.
  - `onFailure`: In caso di errore nella richiesta, anche qui impostiamo un JWT vuoto.

3.12 Barriere nella Applicazione Web

Per garantire una visualizzazione chiara e intuitiva dei dati delle barriere all'interno dell'applicazione web, abbiamo implementato una tabella interattiva che consente agli utenti di visualizzare, filtrare e ordinare i dati in base alle proprie esigenze. La tabella è raggiungibile tramite la pagina principale dell'applicazione web e mostra i dati delle barriere in modo strutturato e organizzato.

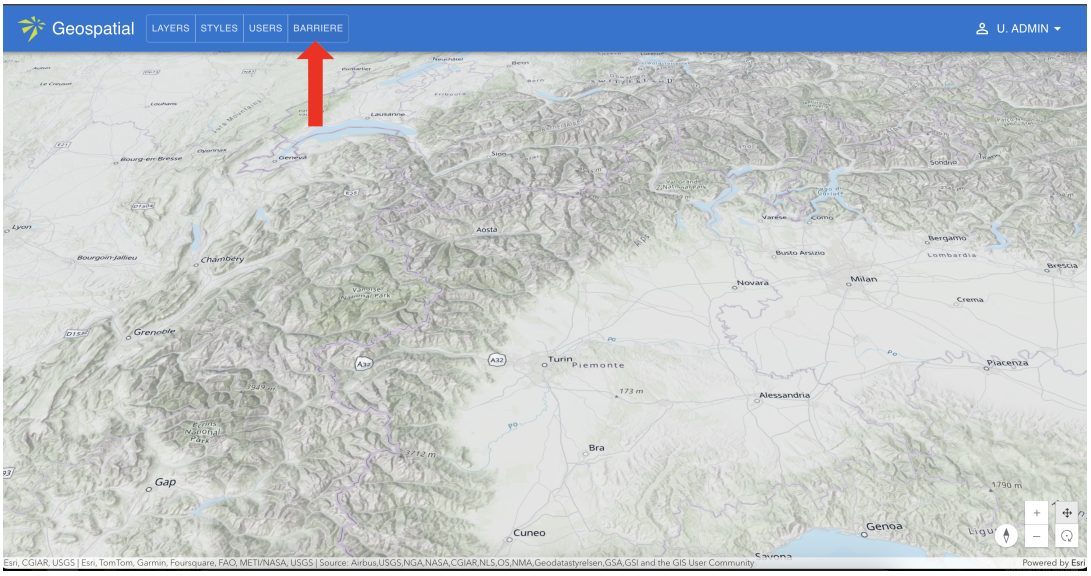


Figura 3.17: Pagina principale dell'applicazione web

Per visualizzare la tabella, l'utente, dopo aver fatto l'accesso, deve fare clic sul pulsante "Barriere" nella pagina principale. Questo pulsante reindirizza l'utente alla pagina della tabella, dove è possibile visualizzare e interagire con i dati delle barriere.

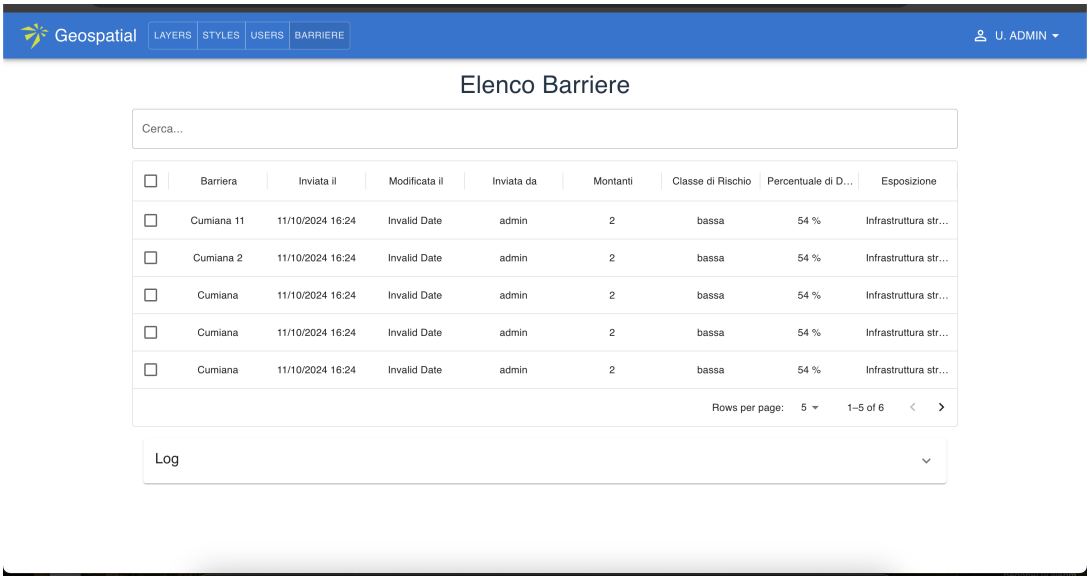


Figura 3.18: Tabella delle barriere

Oltre alla possibilità di ordinare e filtrare i dati, la tabella offre funzionalità aggiuntive come la visualizzazione dei dettagli di una barriera, la modifica dei dati e l'eliminazione di una barriera. Queste funzionalità consentono agli utenti di gestire facilmente i dati delle barriere e di effettuare le operazioni necessarie per mantenere aggiornato il database.

Per visualizzare i dettagli delle risposte al questionario di una barriera, l'utente può fare clic sulla riga della barriera interessata. Questo fa si che l'utente venga reindirizzato a una nuova pagina contenente i dettagli della barriera, inclusi i dati del questionario e le risposte fornite. Con la possibilità di modifica delle risposte, l'utente può apportare modifiche ai dati del questionario e salvare le modifiche nel database.

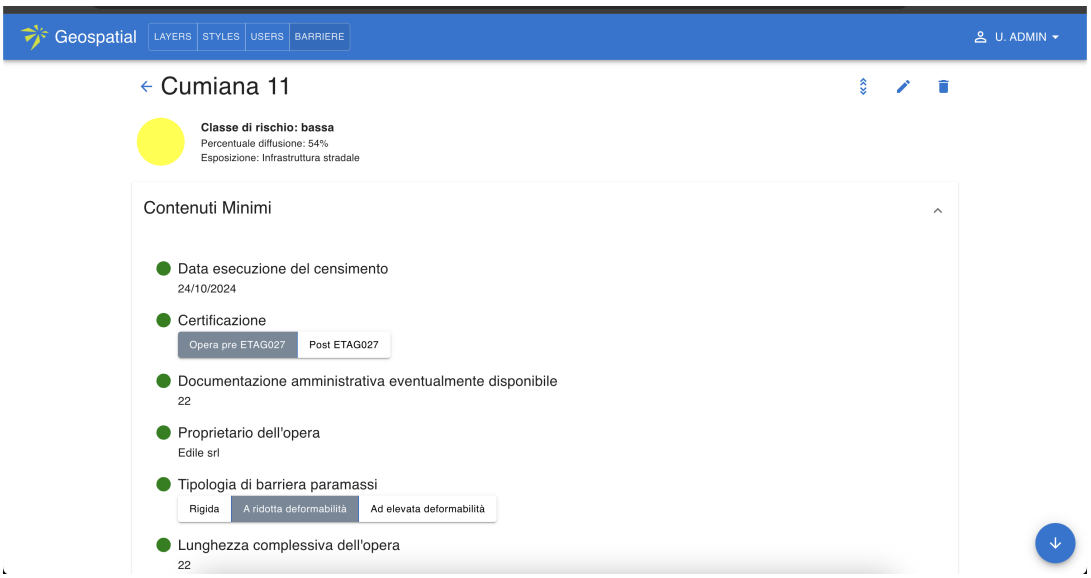


Figura 3.19: Dettagli di una barriera

Alla fine della revisione di una barriera l'utente può decidere se salvare le modifiche apportate. Il salvataggio delle modifiche comporta l'invio dei dati aggiornati al server e la creazione di un nuovo layer sulla mappa che rappresenterà la barriera con le nuove informazioni.

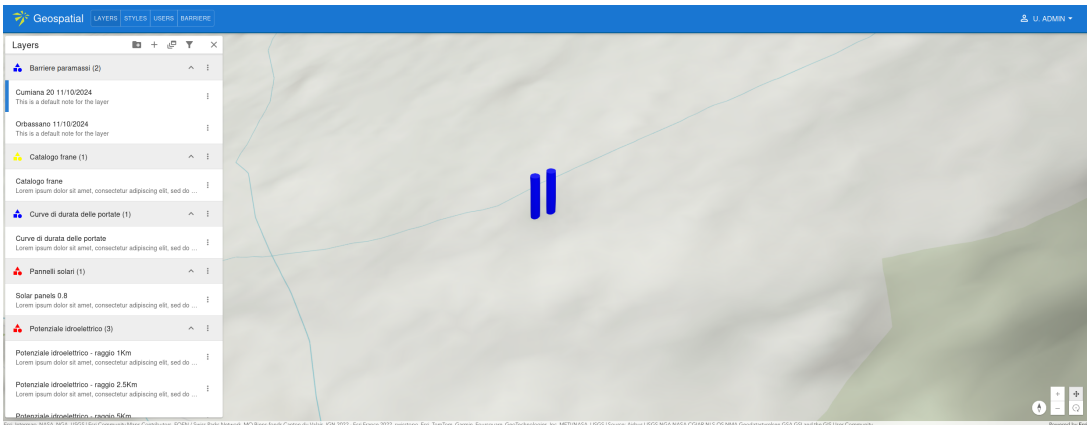


Figura 3.20: Immagine di un layer creato sulla mappa

Capitolo 4

Conclusioni e Sviluppi Futuri

4.1 Sicurezza con Authentication code flow

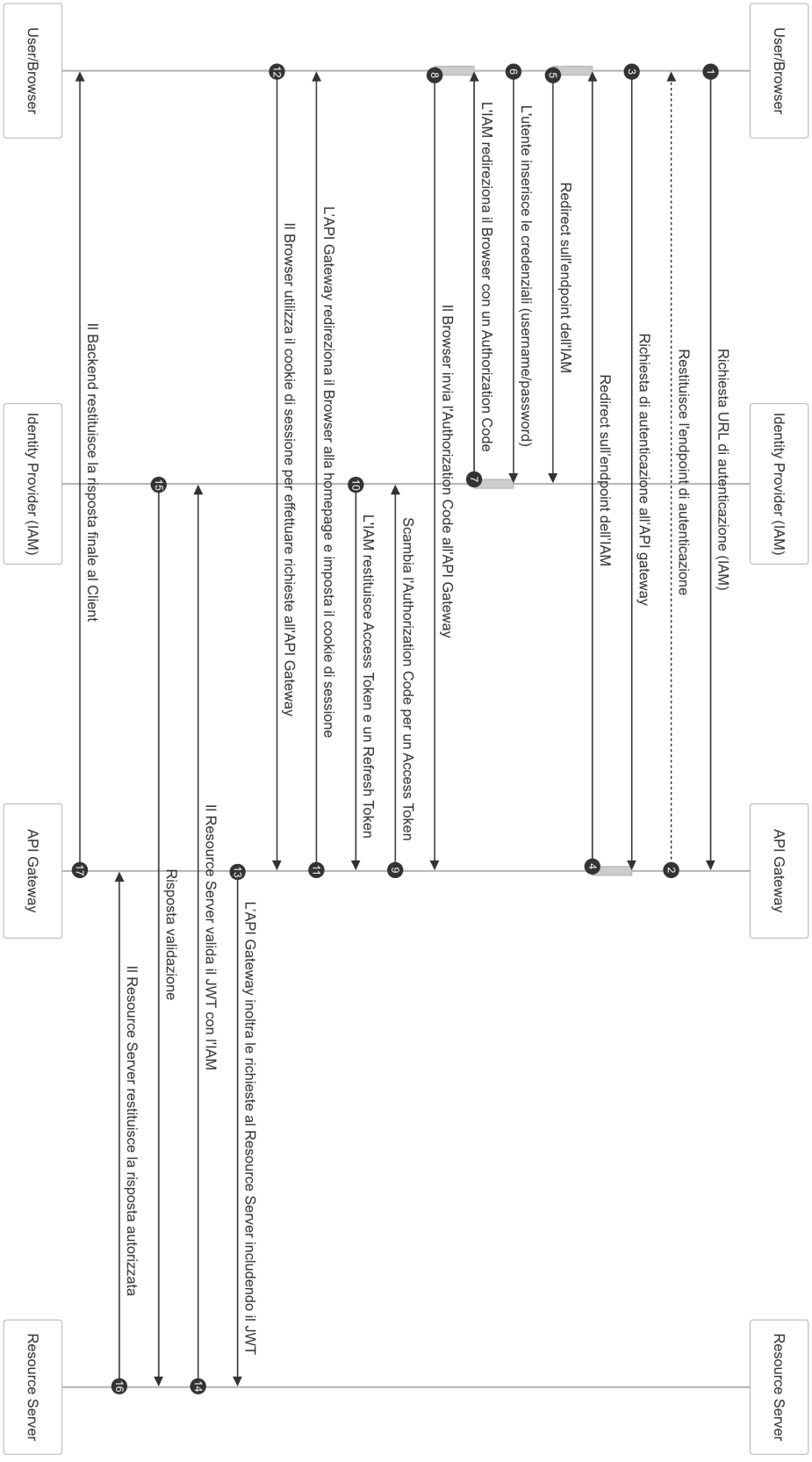
Una delle aree che potrebbero essere significativamente migliorate in GeoSpatial riguarda la sicurezza, in particolare per quanto concerne il flusso di autenticazione. Attualmente, il flusso utilizzato è stato dichiarato deprecato già nel 2018 e sarà completamente rimosso nella versione 2.1 di OAuth. Questa decisione è stata presa a causa di diversi problemi:

- **Gestione delle Credenziali**: Il flusso richiede che l'applicazione client raccolga direttamente le credenziali dell'utente, inclusa la password. Questo rappresenta un rischio significativo, poiché il client potrebbe essere un'applicazione non sicura, aumentando le probabilità che tali informazioni sensibili vengano compromesse.
- **Sicurezza del Token**: Il client gestisce l'access token, spesso memorizzandolo in locale, ad esempio nello storage del browser. Se un utente malintenzionato riuscisse a rubare questo token, potrebbe ottenere accesso non autorizzato alle risorse protette.
- **Assenza di Autenticazione Multifattore**: Il flusso attuale non supporta facilmente l'implementazione di autenticazioni più avanzate, come l'autenticazione a due fattori (2FA) o con dati biometrici, strumenti oggi considerati essenziali per garantire un livello di sicurezza più elevato.

Il flusso di autenticazione raccomandato dallo standard è l'**Authentication Code Flow**, che si basa sull'interazione tra le seguenti entità:

1. **Authentication Server**: Questo componente gestisce il processo di autenticazione dell'utente. È responsabile della verifica delle credenziali e del rilascio di un codice di autorizzazione all'OAuth2 Client una volta che l'utente ha confermato la propria identità.
2. **OAuth2 Client**: Si tratta dell'applicazione che richiede l'accesso alle risorse protette. Il client inoltra la richiesta di autenticazione all'Authentication Server e, successivamente, scambia il codice di autorizzazione ricevuto per un token di accesso. Questo token permette al client di interagire con il Resource Server per accedere alle risorse.
3. **Resource Server**: È il server che ospita le risorse protette. Riceve le richieste dal client e, utilizzando il token di accesso fornito, verifica che quest'ultimo sia autorizzato ad interagire con le risorse richieste.

4.1.1 Sequenza login



Per eseguire il login, lo standard prevede i seguenti passi:

1. **Richiesta di URL per l'autenticazione**: L'utente, tramite il proprio browser (Client/Browser), invia una richiesta all'API Gateway per ottenere l'URL necessario per avviare il processo di autenticazione con l'Identity Provider (IAM).
2. **Restituzione dell'endpoint di autenticazione**: L'API Gateway risponde con l'URL dell'endpoint di autenticazione fornito dall'IAM, dove l'utente dovrà inserire le credenziali.
3. **Richiesta di autenticazione all'API Gateway**: Il browser invia una richiesta all'API Gateway per avviare il processo di autenticazione.
4. **Redirect all'IAM**: L'API Gateway risponde con una HTTP 301 (Redirect) verso la pagina di login dell'IAM
5. **Redirect all'IAM**: Il browser, ricevuta la risposta dall'API gateway, effettua il redirect
6. **Inserimento delle credenziali**: L'utente inserisce le proprie credenziali (username e password) direttamente sull'IAM.
7. **Restituzione Authorization Code**: Dopo una verifica delle credenziali, l'IAM invia un Authorization Code al browser tramite un redirect.
8. **Invio Authorization Code all'API Gateway**: Il browser invia questo Authorization Code all'API Gateway, che lo utilizzerà per ottenere i token di accesso.
9. **Scambio Authorization Code con Access Token**: L'API Gateway invia l'Authorization Code all'IAM, richiedendo un Access Token e un Refresh Token.
10. **Restituzione Access Token e Refresh Token**: L'IAM risponde restituendo all'API Gateway l'Access Token e il Refresh Token, che permetteranno all'utente di accedere alle risorse protette.
11. **Redirect alla homepage e impostazione del cookie di sessione**: L'API Gateway reindirizza l'utente alla homepage e imposta un cookie di sessione nel browser, che contiene informazioni utili per mantenere la sessione attiva.
12. **Invio richieste con il cookie di sessione**: Ora che il browser ha un cookie di sessione, può inviare richieste successive all'API Gateway per accedere alle risorse protette.
13. **Inoltro delle richieste al Resource Server con JWT**: L'API Gateway inoltra queste richieste al Resource Server, includendo nel messaggio il JWT (JSON Web Token) che rappresenta l'Access Token.
14. **Validazione del JWT con IAM**: Il Resource Server, per assicurarsi che il JWT sia valido, lo invia all'IAM per la validazione.
15. **Risultato della validazione**: L'IAM risponde al Resource Server con il risultato della validazione del JWT (valido o non valido).
16. **Risposta autorizzata**: Se il JWT è valido, il Resource Server invia la risposta autorizzata all'API Gateway.
17. **Restituzione della risposta finale al browser**: Infine, l'API Gateway invia la risposta finale al browser dell'utente, permettendogli di visualizzare o interagire con la risorsa richiesta.

4.1.2 IAM e Keycloak

È buona prassi utilizzare strumenti di Identity and Access Management (IAM) preesistenti per gestire il flusso di autenticazione, evitando la necessità di implementare manualmente gli standard di sicurezza. Tra le opzioni disponibili, Keycloak si distingue come una scelta significativa. Fornito da Red Hat, Keycloak può essere ospitato su un server dedicato e, essendo open source, offre un alto livello di personalizzazione. Questa flessibilità consente, ad esempio, di sviluppare plugin per gestire operazioni personalizzate, come la migrazione di utenti da servizi legacy.

Keycloak è progettato per integrarsi facilmente all'interno di un'infrastruttura basata su container. È disponibile un'immagine ufficiale su registri Docker pubblici, che consente di avviare rapidamente un'istanza con una configurazione di base. Per farlo, è sufficiente eseguire il seguente comando:

```
docker run -p 8080:8080 \
-e KC_BOOTSTRAP_ADMIN_USERNAME=admin \
-e KC_BOOTSTRAP_ADMIN_PASSWORD=admin \
quay.io/keycloak/keycloak:26.0.0 start-dev
```

Eseguendo questo comando e visitando tramite browser l'indirizzo http://localhost:8080, si può accedere alla console di amministrazione di Keycloak. Dopo aver effettuato il login con le credenziali di default (username: admin, password: admin), ci si troverà di fronte a una dashboard intuitiva.

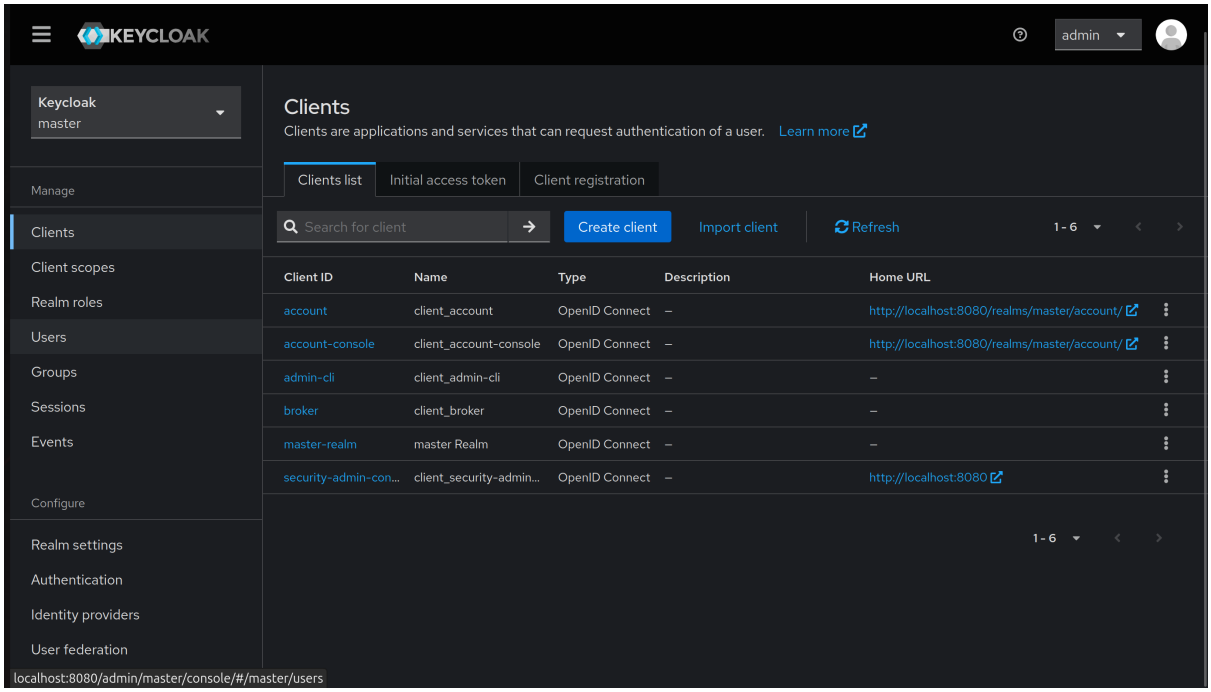


Figura 4.1: Console Amministrazione Keycloak

Panoramica della Console di Amministrazione

Nella sezione **Clients**, è possibile gestire le entità che possono richiedere a Keycloak di autenticare un utente. Di solito, per "Client" si intendono servizi e applicazioni che desiderano utilizzare un sistema IAM per implementare un login basato su single sign-on. Un altro tipo di Client comprende entità che sono interessate esclusivamente a ottenere un token, da poter sfruttare per generare una comunicazione sicura tra i membri dei servizi gestiti da Keycloak.

Attraverso le sezioni **Groups** e **Users**, è possibile gestire i gruppi di utenti e le loro autorizzazioni, semplificando notevolmente l'implementazione di un sistema di **Role-Based Access Control** (RBAC).

Altre sezioni importanti includono:

- **Realm Settings**: Consente di abilitare funzionalità come il modulo di registrazione degli utenti, nonché di definire la durata e la struttura dei JSON Web Token (JWT) generati, migliorando la sicurezza e la gestione delle sessioni.
- **Events**: Offre la possibilità di visualizzare il log delle operazioni effettuate dagli utenti, come login, registrazione e cambio password, fornendo un utile strumento per monitorare e analizzare l'attività degli utenti.

Keycloak supporta vari protocolli di autenticazione, come OAuth2 e OpenID Connect, e consente l'integrazione con servizi di autenticazione esterni, come Google e Facebook, semplificando ulteriormente il processo di accesso per gli utenti.

Con queste caratteristiche, Keycloak rappresenta una soluzione robusta e versatile per la gestione dell'identità e dell'accesso in ambienti moderni, contribuendo a garantire la sicurezza e l'efficienza delle applicazioni.

Inoltre, Keycloak permette di gestire più "Realm" in una singola installazione. Per realm si intende uno spazio dedicato in cui è possibile gestire oggetti come utenti, applicazioni, ruoli e gruppi. Ogni utente appartiene a un realm specifico e si autentica al suo interno.

4.2 Espandibilità dell'app mobile

L'app mobile è progettata con un design modulare, consentendo di generare qualsiasi tipo di questionario. Questo fa sì che un questionario sia composto da più sezioni che possono essere assemblate e configurate per creare un qualsiasi tipo di form.

Descrizione dei tipi di Moduli

1. **TextQuestion**: Permette di aggiungere una domanda di tipo testuale.
2. **NumberQuestion**: Permette di aggiungere una domanda di tipo numerico.
3. **PositionQuestion**: Consente di chiedere all'utente la posizione GPS del dispositivo.
4. **OptionImageQuestion**: Permette di selezionare opzioni e caricare immagini.
5. **DateQuestion**: Chiede all'utente di inserire una data.

Questo sistema modulare porta alla creazione di un prototipo funzionale che può essere ulteriormente espanso per includere altri tipi di domande e sezioni, a seconda delle necessità del progetto.

Futuro di KMM e Compose Multiplatform

In un'ottica di sviluppo cross-platform, **Kotlin Multiplatform Mobile** (KMM) sta guadagnando sempre più attenzione. KMM consente di condividere la logica di business tra le app Android e iOS, migliorando l'efficienza e riducendo il tempo di sviluppo.

Con l'emergere di **Jetpack Compose Multiplatform**, si prevede un futuro in cui gli sviluppatori possano creare interfacce utente in modo coerente su più piattaforme, utilizzando la stessa base di codice. Questa integrazione promette di semplificare ulteriormente il processo di sviluppo, consentendo la realizzazione di un'unica applicazione per Android e iOS.

In questo contesto, l'app sviluppata, realizzata in Kotlin e basata su Jetpack Compose, risulta particolarmente compatibile con KMM e Jetpack Compose Multiplatform. Questo aspetto non solo garantisce un'ottimizzazione dei tempi e delle risorse, ma apre anche la strada a future evoluzioni e miglioramenti nell'esperienza utente su entrambe le piattaforme.

4.3 Funzionalità aggiuntive Raster

Durante la fase di test da parte degli utenti, è emersa la necessità di integrare funzionalità aggiuntive per la gestione dei dati raster. Le nuove funzionalità previste sono le seguenti:

- **Tooltip per layer raster**: Per consentire l'accesso rapido alle informazioni relative a una coordinata specifica, verrà implementato un tooltip per i layer raster. Questo strumento permetterà di visualizzare i valori dei pixel in una posizione precisa della mappa, fornendo all'utente dettagli sui dati raster visualizzati.
- **Creazione stili personalizzati**: Attualmente l'applicazione permette di visualizzare i dati raster utilizzando un set di stili predefiniti. Per offrire maggiore flessibilità, verrà sviluppato un editor di stili per i layer raster. Questo permetterà agli utenti di creare stili personalizzati, adattandoli alle proprie esigenze, attraverso un'interfaccia simile a quella già disponibile per i dati vettoriali.
- **Supporto per MultiDimensional Raster**: Verranno integrate funzionalità per la gestione e visualizzazione dei dati raster multidimensionali. Questi dati, utilizzati in ambiti come meteorologia, ambiente e geofisica, richiedono strumenti dedicati per un'analisi e interpretazione efficaci.

4.4 Conclusioni

Il progetto ha raggiunto gli obiettivi prefissati, portando alla realizzazione di un'applicazione web capace di gestire e visualizzare dati geospaziali in maniera efficace e intuitiva. Le tecnologie scelte hanno garantito prestazioni adeguate ed un'interfaccia user-friendly, sia per l'uso desktop che mobile.
Durante lo sviluppo, abbiamo affrontato alcune sfide tecniche, soprattutto nella gestione delle risorse e nell'integrazione di componenti che richiedevano documentazione non sempre aggiornata. Tuttavia, grazie a soluzioni implementate per superare queste difficoltà, il sistema si è rivelato stabile e performante.
L'applicazione offre ora agli utenti la possibilità di visualizzare dati geospaziali in tempo reale, facilitando la collaborazione e l'analisi in un ambiente completamente online. Questo progetto rappresenta un passo avanti verso una gestione autonoma e accessibile dei dati territoriali, dimostrando come le tecnologie attuali possano essere sfruttate per migliorare l'efficienza e la condivisione delle informazioni.
L'applicazione web sviluppata è attualmente disponibile al seguente link: summer.polito.it.
In futuro, l'obiettivo è quello di continuare a migliorare e ampliare le funzionalità dell'applicazione, integrando nuove tecnologie e implementando nuovi moduli per rispondere alle esigenze degli utenti. Inoltre, l'adozione di standard di sicurezza come l'Authentication Code Flow e l'integrazione con strumenti IAM come Keycloak contribuiranno a garantire un ambiente sicuro e affidabile per gli utenti.

Ringraziamenti

Ringraziamenti

Un ringraziamento speciale va ai nostri relatori, Prof. Giovanni Malnati e Prof. Daniele Apiletti, per la loro disponibilità e i preziosi consigli che ci hanno accompagnato durante tutto il percorso. Il loro supporto è stato determinante per il successo di questo progetto.

Un grazie di cuore va ai nostri colleghi, in particolare a Daniele, Alessandra, David, Gabriele e Giovanni S., per il continuo confronto, il supporto e la collaborazione. Il loro contributo ha arricchito profondamente questa esperienza.

Un pensiero speciale va ai nostri amici e compagni di studi, che ci sono stati accanto in ogni momento, condividendo con noi sfide, difficoltà e successi.

Alla nostra famiglia, il nostro grazie più profondo. Senza il loro costante sostegno e incoraggiamento, nulla di tutto questo sarebbe stato possibile.

Infine, ringraziamo tutti coloro che, in modo diretto o indiretto, hanno contribuito alla realizzazione di questa tesi. Ogni consiglio e incoraggiamento è stato prezioso per raggiungere questo traguardo.

Matteo e Nicolò