

1 Fase (8 giornate)

1. Definizione del diagramma E/R e delle specifiche relative per la creazione dell'infrastruttura DB (1 giornata) – **in sede ISPRA** (già realizzata) – **21 marzo**
2. Installazione di PostGIS, definizione di un ambiente di lavoro, creazione delle tabelle ritenute necessarie per il progetto. Test funzionale. (1/2 giornata) - **9 maggio**
3. Installazione di Geoserver, definizione dei layer e integrazione delle tabelle. Test funzionale. (1/2 giornata) **9 maggio**
TEST ON LINE E ATTESTATO 2 GIORNATE: 12 ORE
- 9 Importazione dei dati da file Shape o altri db georeferenziati per la costruzione degli elementi verdi e blu (1 e ½ giornate) - **15 maggio, 22 maggio**
- 10 Importazione dei dati della popolazione su raster fornito da ISPRA (1 e ½ giornate) – **22 maggio, 29 maggio**
- 11 Installazione ambiente di lavoro usando Eclipse, definizione classi per accesso DB (2 giornate) – **12 giugno, 19 giugno**
- 12 Confronto su sviluppo del geodatabase - (1 giornata) - in sede ISPRA – **24 giugno**
TEST ON LINE E ATTESTATO 6 GIORNATE: 36 ORE

2 fase (24 giornate)

1. Calcolo dei punti di ingresso delle aree verdi e blu (4 giornate) – **3 luglio, 10 luglio, 17 luglio, 2 settembre**
2. Calcolo della distanza minima fra i centroidi dei poligoni rappresentanti la popolazione e i punti di ingresso delle aree verdi e blu (2 giornate) **4 settembre, 9 settembre**
TEST ON LINE E ATTESTATO 6 GIORNATE: 36 ORE



Administrator



vebs



Questo PC



Progetto_VeBS



Rete



DATA



Cestino



QGIS 3.42.2



Postman

Attiva Windows
Passa a Impostazioni per attivare Windows.

1 Fase (8 giornate)

1. Definizione del diagramma E/R e delle specifiche relative per la creazione dell'infrastruttura DB (1 giornata) – **in sede ISPRA** (già realizzata) – **21 marzo**
2. Installazione di PostGIS, definizione di un ambiente di lavoro, creazione delle tabelle ritenute necessarie per il progetto. Test funzionale. (1/2 giornata) - **9 maggio**
3. Installazione di Geoserver, definizione dei layer e integrazione delle tabelle. Test funzionale. (1/2 giornata) **9 maggio**
TEST ON LINE E ATTESTATO 2 GIORNATE: 12 ORE
- 9 Importazione dei dati da file Shape o altri db georeferenziati per la costruzione degli elementi verdi e blu (1 e ½ giornate) - **15 maggio, 22 maggio**
- 10 Importazione dei dati della popolazione su raster fornito da ISPRA (1 e ½ giornate) – **22 maggio, 29 maggio**
- 11 Installazione ambiente di lavoro usando Eclipse, definizione classi per accesso DB (2 giornate) – **12 giugno, 19 giugno**
- 12 Confronto su sviluppo del geodatabase - (1 giornata) - in sede ISPRA – **24 giugno**
TEST ON LINE E ATTESTATO 6 GIORNATE: 36 ORE

2 fase (24 giornate)

1. Calcolo dei punti di ingresso delle aree verdi e blu (4 giornate) – **3 luglio, 10 luglio, 17 luglio, 2 settembre**
2. Calcolo della distanza minima fra i centroidi dei poligoni rappresentanti la popolazione e i punti di ingresso delle aree verdi e blu (2 giornate) **4 settembre, 9 settembre**
TEST ON LINE E ATTESTATO 6 GIORNATE: 36 ORE

Gestione Database e Pubblicazione Geospaziale con PostGIS e GeoServer

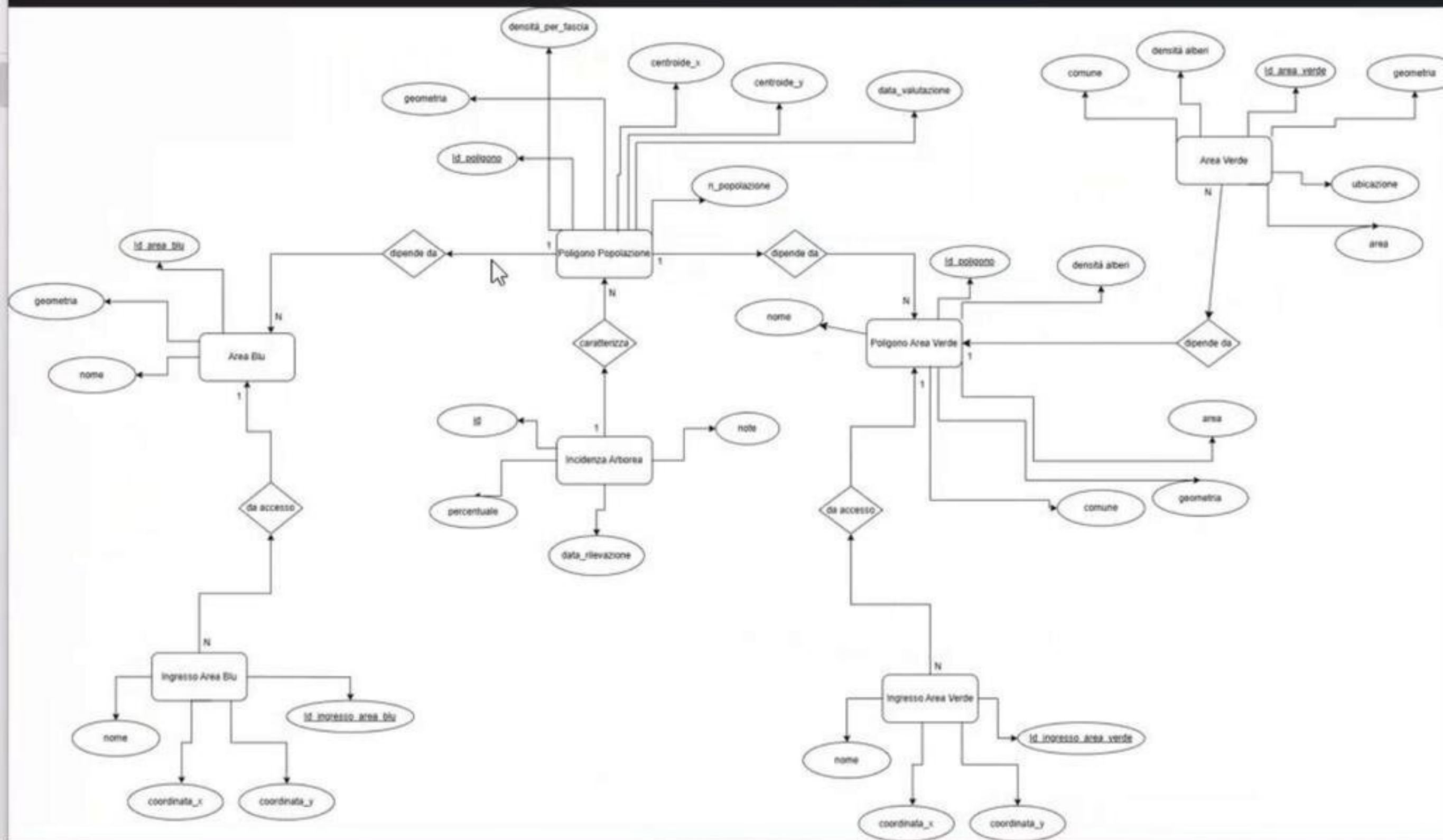
1. Definizione del Diagramma E/R e Specifiche per la Creazione dell'Infrastruttura DB

Obiettivi:

- Comprendere la struttura logica dei dati
- Definire le entità, relazioni e attributi
- Produrre un Diagramma E/R coerente con i requisiti del progetto
- Redigere specifiche tecniche per la realizzazione fisica del database

Attività previste:

1. Analisi dei requisiti informativi del progetto
2. Identificazione delle entità principali (es. aree verdi, aree blu, poligono per popolazione, ingressi per aree)
3. Definizione delle relazioni (es. un'area verde può avere molti ingressi)
4. Rappresentazione E/R con strumenti visuali (es. draw.io)
5. Produzione di uno schema relazionale



- 7
- 8
- 9
- 10
- 11

Population That Has Access to a GLUP

MC	OSM				UA			
	Within 300 m		Within 400 m		Within 300 m		Within 400 m	
	Number of Inhab.	% *	Number of Inhab.	% *	Number of Inhab.	% *	Number of Inhab.	% *
Torino	980,640	37.9	859,799	53.9	279,268	17.9	420,090	27
Genova	124,725	19.1	192,809	29.5	101,234	15.8	159,562	24.4
Milano	1,484,951	49.4	1,878,162	66.4	891,382	29.9	1,280,206	43.3
Vercelli	114,030	28.1	135,779	42.1	79,664	20.4	113,489	31.1
Belluno	104,235	36.4	411,987	73.2	217,860	38.7	288,491	51.4
Florence	147,748	39.4	347,600	85.5	163,729	36.2	336,388	84.2
Roma	994,034	29.4	1,407,314	42.2	739,062	21.6	1,001,374	30.3
Naples	103,286	12.3	314,773	19	469,901	17.3	677,338	25
Bari	193,773	18.8	308,011	29.9	96,421	5.5	96,897	8.4
Brescia	11,958	6.3	25,114	10	4800	1.7	10,101	4
Calabria	85,910	6.9	110,860	15.4	108,735	12.3	166,963	19.2
Pesento	14,836	7	32,444	11.4	-	-	-	-
Messina	38,901	7.3	96,027	12.3	33,929	4.2	94,310	6.8
Cagliari	67,043	19.1	107,180	30.9	77,038	22	117,362	33.4
Total	4,639,989	28.2	6,374,084	40	3,222,055	19.6	4,650,153	28.3

(*) Percentage of population that has access, compared to the total population of the urban area

Implementazione della metodologia per il calcolo dell'accessibilità alle aree verdi urbane

Nota la classificazione delle aree in funzione della distanza, occorre valutare la popolazione che ricade nell'area posta a 300 metri rispetto ai punti di accesso.

L'accessibilità verrebbe calcolata nel modo seguente:

$$Accessibilità\ area\ verde = \frac{Popolazione_{300}}{Popolazione_T}$$

In cui $Popolazione_{300}$ = popolazione rappresentata dagli abitanti localizzati nelle celle della griglia che hanno accesso ad un'area verde se quest'ultima può essere raggiunta camminando per non più di 300 metri;
 $Popolazione_T$ = popolazione totale all'interno della municipalità

- Spiegazione e sviluppo delle classi Repository
- Spiegazione e sviluppo delle classi di configurazione
- Spiegazione e sviluppo delle classi Service
- Spiegazione e sviluppo della classe Application
- Spiegazione e sviluppo delle classi Controller

Spring MVC:

Spring MVC è un framework per realizzare applicazioni web basate sul modello MVC sfruttando i punti di forza offerti dal framework Spring come l'inversion of control (tramite dependency injection) e la aspect oriented programming.

Esso si occupa di mappare metodi e classi Java con determinati url, di gestire differenti tipologie di "viste" restituite al client, di realizzare applicazioni internazionalizzate e di gestire i cosiddetti temi per personalizzare al massimo l'esperienza utente.

Cos'è il pattern MVC?

Per comprendere al meglio il framework è necessario introdurre il pattern teorico che esso implementa ovvero il modello MVC. MVC rappresenta un acronimo per Model View

ovvero le tre componenti principali di un'applicazione web. Grazie a questo pattern i compiti vengono separati verso questi componenti:

- i **Model** si occupano di accedere ai dati necessari alla logica di business implementata nell'applicazione (nel caso di un'applicazione per una biblioteca potrebbero essere le classi Libro, Autore, Scaffale);
- le **View** si occupano di creare l'interfaccia utilizzabile dall'utente e che espone i dati da esso richiesti (nel caso bibliotecario potrebbero essere le pagine HTML del catalogo, le form di ricerca oppure i PDF contenenti ricerche o appunti);
- i **Controller** si occupano di implementare la vera logica di business dell'applicazione integrando le due componenti precedenti, ricevendo gli input dell'utente, gestendo i modelli per la ricerca dei dati e la creazione di viste da restituire all'utente (nel nostro caso potrebbero essere il motore di ricerca interno oppure il sistema di login al sito Internet della biblioteca).

Come funziona MVC in Spring?

Spring MVC implementa perfettamente questo approccio mantenendo sia i concetti che la nomenclatura del pattern. All'interno di una applicazione Spring MVC avremo quindi: i **Model** sono rappresentati dalle classi che a loro volta rappresentano gli oggetti gestiti e le **classi di accesso al database**;

le **View** sono rappresentate dai vari file Javascript (che vengono compilati in HTML) e da eventuali classi per l'esportazione in formati diversi da HTML (PDF, XLS, CSV...);

i **Controller** sono rappresentati da **classi** (chiamate appositamente Controller) che rimangono "in ascolto" su un determinato URL e, grazie ai Model e alle View, si occupano di gestire la richiesta dell'utente.



caso potrebbero essere il motore di ricerca interno oppure il sistema di login di sito Internet della biblioteca).

Come funziona MVC in Spring?

Spring MVC implementa perfettamente questo approccio mantenendo sia i concetti che la nomenclatura del pattern. All'interno di una applicazione Spring MVC avremo quindi:
i **Model** sono rappresentati dalle classi che a loro volta rappresentano gli oggetti gestiti e le **classi di accesso al database**;
le **View** sono rappresentate dai vari **file Javascript** (che vengono compilati in HTML) e da eventuali classi per l'esportazione in formati diversi da HTML (PDF, XLS, CSV...);
i **Controller** sono rappresentati da **classi (chiamate appositamente Controller)** che rimangono "in ascolto" su un determinato URL e, grazie ai Model e alle **View**, si occupano di gestire la richiesta dell'utente.

Secondo la documentazione ufficiale Spring MVC presenta inoltre molti altri vantaggi oltre alla netta separazione tra le funzionalità:

- è adattabile, flessibile e non intrusivo grazie alla presenza di comode e chiare **Java Annotations**;
- permette di scrivere codice riusabile;
- possibilità di essere esteso tramite adattatori e validatori scritti ad hoc per le nostre esigenze;
- **url** dinamici, SEO-friendly e personalizzabili;
- gestione integrata dell'internazionalizzazione e dei temi;
- libreria JSP sviluppata ad hoc per facilitare alcune operazioni ripetitive;
- nuovi scope per i **bean (request e session)** che permettono di adattare i container base di Spring anche al mondo web.



Maven:

Maven è un progetto open source, sviluppato dalla Apache, che permette di organizzare in modo molto efficiente un progetto java.

Può essere paragonato all'altro progetto più conosciuto della Apache, Ant, ma fornisce funzionalità più avanzate. I vantaggi principali di Maven sono i seguenti:

- standardizzazione della struttura di un progetto compilazione;
- test ed esportazione automatizzate;
- gestione e download automatico delle librerie necessarie al progetto;
- creazione automatica di un semplice sito di gestione del progetto contenente informazioni.

Successivamente andremo ad importare attraverso il file pom.xml automaticamente generato le seguenti dipendenze (<https://mvnrepository.com/repos/central>):

- Jackson (per serializzare e deserializzare i dati JSON utile con dati geografici)
- Geotools (libreria per gestire dati GIS)
- JTS Core (libreria per lavorare con le geometrie)

Le classi Entity:

- **Definizione:** Classe che rappresenta una tabella nel database.
- **Annotazione principale:** @Entity
- **Responsabilità:** Mappare i dati tra il database e l'applicazione.
- **Esempio:**

Spiegazione strumenti OpenStreetMap e Overpass API e continuazione sviluppi

Obiettivi:

- Estrazione dati e continuazione sviluppi backend

Attività previste:

- Introduzione all'utilizzo di OpenStreetMap
- Approfondimento dell'uso dell'Overpass API
- Continuazione sviluppo delle classi per le API REST e dei relativi servizi

Introduzione all'utilizzo di OpenStreetMap

OpenStreetMap è un progetto collaborativo per creare una **mappa libera del mondo**, accessibile e modificabile da chiunque, un po' come Wikipedia per i dati geografici.

- **Fondato nel 2004**, OSM raccoglie dati geografici come strade, edifici, fiumi, parchi, punti di interesse, ecc.
- I dati sono **open data**, rilasciati con licenza **ODbL (Open Database License)**: cioè si possono usare, modificare, integrare anche in progetti commerciali, citando la fonte.



- Approfondimento dell'uso dell'Overpass API
- Continuazione sviluppo delle classi per le API REST e dei relativi servizi

Introduzione all'utilizzo di OpenStreetMap

OpenStreetMap è un progetto collaborativo per creare una **mappa libera del mondo**, accessibile e modificabile da chiunque, un po' come Wikipedia per i dati geografici.

- **Fondato nel 2004**, OSM raccoglie dati geografici come strade, edifici, fiumi, parchi, punti di interesse, ecc.
- I dati sono **open data**, rilasciati con licenza **ODbL (Open Database License)**: cioè si possono usare, modificare, integrare anche in progetti commerciali, citando la fonte.

OSM è basato su tre elementi principali:

Nodo (node) Un punto con coordinate (es. una fermata autobus, un ingresso di un parco)

Via (way) Una linea o poligono composto da nodi (es. una strada o un edificio)

Relazione (relation) Struttura per collegare oggetti (es. un itinerario bus con più vie)

Ogni oggetto può avere **tag** del tipo **chiave = valore**, ad esempio:

highway=primary

name=Via Roma



OSM è basato su tre elementi principali:

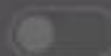
- Nodo (node)** Un punto con coordinate (es. una fermata autobus, un ingresso di un parco)
- Via (way)** Una linea o poligono composto da nodi (es. una strada o un edificio)
- Relazione (relation)** Struttura per collegare oggetti (es. un itinerario bus con più vie)

Ogni oggetto può avere **tag** del tipo **chiave = valore**, ad esempio:

highway=primary
name=Via Roma



I dati vengono salvati in un **database globale** che chiunque può esplorare, interrogare, aggiornare o estrarre.



Cosa è possibile fare su OpenStreetMap

1. Esplorazione e visualizzazione

<https://www.openstreetmap.org>
CTRL+clic per aprire collegam.

- Vai su [openstreetmap.org](https://www.openstreetmap.org)
- Puoi cercare luoghi, esplorare i dati mappa e vedere i dettagli di ogni oggetto (tag, cronologia modifiche, utenti, ecc.)

2. Modificare dati

Si può contribuire a OSM (da utente registrato) con:

- **iD editor**: editor online facile da usare, direttamente dal sito OSM
- **JOSM**: editor avanzato desktop (Java) per modifiche massive
- **Field Papers** o app mobile (Vespucci, StreetComplete)

3. Scaricare dati

Come ottenere dati OSM:

- In formato **.osm (XML)** tramite Geofabrik
- In formato **GeoJSON, KML** usando [Overpass Turbo](#)
- Con strumenti GIS come QGIS (tramite plugin QuickOSM)

4. Sviluppo software

È possibile integrare OSM in app/web usando:



VISUALIZZAZIONE PROTETTA Attenzione. I file provenienti da Internet possono contenere virus. A meno che non sia necessario modificare il file, è consigliabile restare in Visualizzazione protetta.

Abilita modifica

- Puoi cercare luoghi, esplorare i dati mappa e vedere i dettagli di ogni oggetto (tag, cronologia modifiche, utenti, ecc.)

2. Modificare dati

Si può contribuire a OSM (da utente registrato) con:

- **iD editor**: editor online facile da usare, direttamente dal sito OSM
- **JOSM**: editor avanzato desktop (Java) per modifiche massive
- **Field Papers** o app mobile (Vespucci, StreetComplete)

3. Scaricare dati

Come ottenere dati OSM:

- In formato **.osm (XML)** tramite Geofabrik
- In formato **GeoJSON, KML** usando [Overpass Turbo](#)
- Con strumenti GIS come QGIS (tramite plugin QuickOSM)

4. Sviluppo software

È possibile integrare OSM in app/web usando:

- **Leaflet.js + Tile Server**: per visualizzare mappe
- **OpenLayers**: libreria GIS frontend potente
- **osmnx (Python)**: per analisi di reti stradali
- **Nominatim API**: per geocoding/reverse geocoding



Cosa è possibile fare su OpenStreetMap

1. Esplorazione e visualizzazione

- Vai su openstreetmap.org
- Puoi cercare luoghi, esplorare i dati mappa e vedere i dettagli di ogni oggetto (tag, cronologia modifiche, utenti, ecc.)

2. Modificare dati

Si può contribuire a OSM (da utente registrato) con:

- **iD editor**: editor online facile da usare, direttamente dal sito OSM
- **JOSM**: editor avanzato desktop (Java) per modifiche massive
- **Field Papers** o app mobile (Vespucci, StreetComplete)

3. Scaricare dati

Come ottenere dati OSM:

- In formato **.osm (XML)** tramite Geofabrik
- In formato **GeoJSON, KML** usando [Overpass Turbo](#)
- Con strumenti GIS come QGIS (tramite plugin QuickOSM)

4. Sviluppo software

È possibile integrare OSM in app/web usando:



VISUALIZZAZIONE PROTETTA

Attenzione. I file provenienti da Internet possono contenere virus. A meno che non sia necessario modificare il file, è consigliabile restare in Visualizzazione protetta.

Abilita modifica

- Puoi cercare luoghi, esplorare i dati mappa e vedere i dettagli di ogni oggetto (tag, cronologia modifiche, utenti, ecc.)

2. Modificare dati

Si può contribuire a OSM (da utente registrato) con:

- **iD editor**: editor online facile da usare, direttamente dal sito OSM
- **JOSM**: editor avanzato desktop (Java) per modifiche massive
- **Field Papers** o app mobile (Vespucci, StreetComplete)

3. Scaricare dati

Come ottenere dati OSM:

- In formato **.osm (XML)** tramite Geofabrik
- In formato **GeoJSON, KML** usando [Overpass Turbo](#)
- Con strumenti GIS come QGIS (tramite plugin QuickOSM)

4. Sviluppo software

È possibile integrare OSM in app/web usando:

- **Leaflet.js + Tile Server**: per visualizzare mappe
- **OpenLayers**: libreria GIS frontend potente
- **osmnx (Python)**: per analisi di reti stradali
- **Nominatim API**: per geocoding/reverse geocoding



VISUALIZZAZIONE PROTETTA

Attenzione. I file provenienti da Internet possono contenere virus. A meno che non sia necessario modificare il file, è consigliabile restare in Visualizzazione protetta.

Si può contribuire a OSM (da utente registrato) con:

- **iD editor**: editor online facile da usare, direttamente dal sito OSM
- **JOSM**: editor avanzato desktop (Java) per modifiche massive
- **Field Papers** o app mobile (Vespucci, StreetComplete)

3. Scaricare dati

Come ottenere dati OSM:

- In formato **.osm (XML)** tramite [Ge](https://overpass-turbo.eu/) **https://overpass-turbo.eu/**
CTRL+clik per aprire collegam.
- In formato **GeoJSON, KML** usando [Overpass Turbo](#)
- Con strumenti GIS come QGIS (tramite plugin QuickOSM)

4. Sviluppo software

È possibile integrare OSM in app/web usando:

- **Leaflet.js + Tile Server**: per visualizzare mappe
- **OpenLayers**: libreria GIS frontend potente
- **osmnx (Python)**: per analisi di reti stradali
- **Nominatim API**: per geocoding/reverse geocoding
- **Overpass API**: per query personalizzate sui dati
- **PostGIS**: puoi importare i dati OSM nel tuo DB spaziale per analisi avanzate





VISUALIZZAZIONE PROTETTA

Attenzione. I file provenienti da Internet possono contenere virus. A meno che non sia necessario modificare il file, è consigliabile restare in Visualizzazione protetta.

Abilita modifica



È possibile integrare OSM in app/web usando:

- **Leaflet.js + Tile Server:** per visualizzare mappe
- **OpenLayers:** libreria GIS frontend potente
- **osmnx (Python):** per analisi di reti stradali
- **Nominatim API:** per geocoding/reverse geocoding
- **Overpass API:** per query personalizzate sui dati
- **PostGIS:** puoi importare i dati OSM nel tuo DB spaziale per analisi avanzate

I

Approfondimento dell'uso dell'Overpass API

Overpass API è un'API che consente di **interrogare i dati di OpenStreetMap (OSM)** in modo avanzato e personalizzato.

È progettata per estrarre *solo i dati che servono*, senza dover scaricare l'intero database OSM.

Non è un API REST. Il metodo di richiesta è GET (e non POST, come si potrebbe pensare).





VISUALIZZAZIONE PROTETTA

Attenzione. I file provenienti da Internet possono contenere virus. A meno che non sia necessario modificare il file, è consigliabile restare in Visualizzazione protetta.

Abilita modifica



Approfondimento dell'uso dell'Overpass API

Overpass API è un'API che consente di **interrogare i dati di OpenStreetMap (OSM)** in modo avanzato e personalizzato.

È progettata per estrarre *solo i dati che servono*, senza dover scaricare l'intero database OSM.

Non è un'API REST "standard" tipo GET /endpoint, ma un motore di query geospaziali su un database OSM.

Funzionalità di Overpass Turbo:

- - Creazione ed esecuzione di query Overpass QL
- - Visualizzazione su mappa degli oggetti ottenuti
- - Esportazione in vari formati (GeoJSON, KML, GPX, ecc.)

Overpass API lavora su una copia aggiornata del database OSM e consente di interrogarlo tramite due linguaggi:

1. Overpass QL (query language compatto)
2. Overpass XML (più verboso, in formato XML)

Concetti base

- Gli oggetti OSM sono: node, way, relation.
- Ogni oggetto ha **tag** del tipo chiave=valore (es: amenity=school).



Approfondimento dell'uso dell'Overpass API

Overpass API è un'API che consente di **interrogare i dati di OpenStreetMap (OSM)** in modo avanzato e personalizzato.

È progettata per estrarre *solo i dati che servono*, senza dover scaricare l'intero database OSM.

Non è un'API REST "standard" tipo GET /endpoint, ma un motore di query geospaziali su un database OSM.

Funzionalità di Overpass Turbo:

- - Creazione ed esecuzione di query Overpass QL
- - Visualizzazione su mappa degli oggetti ottenuti
- - Esportazione in vari formati (GeoJSON, KML, GPX, ecc.)

Overpass API lavora su una copia aggiornata del database OSM e consente di interrogarlo tramite due linguaggi:

1. Overpass QL (query language compatto)
2. Overpass XML (più verboso, in formato XML)

Concetti base

- Gli oggetti OSM sono: node, way, relation.
- Ogni oggetto ha **tag** del tipo chiave=valore (es: amenity=school).
- Si può filtrare per:

Overpass API lavora su una copia aggiornata del database OSM e consente di interrogarlo tramite due linguaggi:

1. Overpass QL (query language compatto)
2. Overpass XML (più verboso, in formato XML)

Concetti base

- Gli oggetti OSM sono: node, way, relation.
- Ogni oggetto ha **tag** del tipo chiave=valore (es: amenity=school).
- Si può filtrare per:
 - **Tag** (es: tutti i parchi)
 - **Area geografica** (box lat/lon)
 - **Tipo di oggetto** (nodo/poligono)
 - **Tempo** (modifiche recenti)

Esempio di Query in Overpass QL

```
[out:json][timeout:25];  
// seleziona tutti i parchi in un'area o specificando le coordinate dell'area
```



```
//es:" (45.42,10.95,45.45,10.98)"
(
  node["leisure"="park"](45.42,10.95,45.45,10.98);
  way["leisure"="park"](45.42,10.95,45.45,10.98);
  relation["leisure"="park"](45.42,10.95,45.45,10.98);
);

[out:json][timeout:25];
//o passandogli il nome es:"Catanzaro"
area["name"="Catanzaro"]["boundary"="administrative"]["admin_level"="8"]->.catanzaro;
(
  node["leisure"="park"](area.catanzaro);
  way["leisure"="park"](area.catanzaro);
  relation["leisure"="park"](area.catanzaro);
) ->.parchi;

// trova i nodi con barrier=gate o entrance nelle aree verdi (via map_to_area)
.parchi map_to_area -> .aree_parco;

(
  node["barrier"="gate"](area.aree_parco);
  node["barrier"="entrance"](area.aree_parco);
);

out body;
>;
out skel qt;
// oppure out geom;
```

Spiegazione:

- out:json: formato di output (può essere anche XML)

```
// trova i nodi con barrier=gate o entrance nelle aree verdi (via map_to_area)
.parchi map_to_area -> .aree_parco;
```

```
(
  node["barrier"="gate"](area.aree_parco);
  node["barrier"="entrance"](area.aree_parco);
);
```

```
out body;
```

```
>;
```

```
out skel qt;
```

```
// oppure out geom;
```

I

Spiegazione:

- out:json: formato di output (può essere anche XML)
- timeout:25: tempo massimo di esecuzione in secondi
- node, way, relation: tipi di oggetti cercati
- (lat1, lon1, lat2, lon2): bounding box
- out body: restituisce i dati principali
- >; segue le referenze (es. nodi delle way)
- out skel qt; output compatto di chiusura
- out geom; output integrale che contiene le relazioni con nodi e figli delle ways


```
way["leisure"="park"](45.42,10.95,45.45,10.98);
relation["leisure"="park"](45.42,10.95,45.45,10.98);
);

[out:json][timeout:25];
//o passandogli il nome es:"Catanzaro"
area["name"="Catanzaro"]["boundary"="administrative"]["admin_level"="8"]->.catanzaro;
(
  node["leisure"="park"](area.catanzaro);
  way["leisure"="park"](area.catanzaro);
  relation["leisure"="park"](area.catanzaro);
) ->.parchi;

// trova i nodi con barrier=gate o entrance nelle aree verdi (via map_to_area)
.parchi map_to_area -> .aree_parco;

(
  node["barrier"="gate"](area.aree_parco);
  node["barrier"="entrance"](area.aree_parco);
);

out body;
>;
out skel qt;
// oppure out geom;
```

Spiegazione:

- out:json: formato di output (può essere anche XML)
- timeout:25: tempo massimo di esecuzione in secondi
- node, way, relation: tipi di oggetti cercati

```
// trova i nodi con barrier=gate o entrance nelle aree verdi (via map_to_area)
.parchi map_to_area -> .aree_parco;
```

```
(
  node["barrier"="gate"](area.areas_parco);
  node["barrier"="entrance"](area.areas_parco);
);
```

I

```
out body;
>;
out skel qt;
// oppure out geom;
```

Spiegazione:

- out:json: formato di output (può essere anche XML)
- timeout:25: tempo massimo di esecuzione in secondi
- node, way, relation: tipi di oggetti cercati
- (lat1, lon1, lat2, lon2): bounding box
- out body: restituisce i dati principali
- >; segue le referenze (es. nodi delle way)
- out skel qt; output compatto di chiusura
- out geom; output integrale che contiene le relazioni con nodi e figli delle ways


```
//es:" (45.42,10.95,45.45,10.98)"
(
  node["leisure"="park"](45.42,10.95,45.45,10.98);
  way["leisure"="park"](45.42,10.95,45.45,10.98);
  relation["leisure"="park"](45.42,10.95,45.45,10.98);
);

[out:json][timeout:25];
//o passandogli il nome es:"Catanzaro"
area["name"="Catanzaro"]["boundary"="administrative"]["admin_level"="8"]->.catanzaro;
(
  node["leisure"="park"](area.catanzaro);
  way["leisure"="park"](area.catanzaro);
  relation["leisure"="park"](area.catanzaro);
) ->.parchi;

// trova i nodi con barrier=gate o entrance nelle aree verdi (via map_to_area)
.parchi map_to_area -> .aree_parco;

(
  node["barrier"="gate"](area.aree_parco);
  node["barrier"="entrance"](area.aree_parco);
);

out body;
>;
out skel qt;
// oppure out geom;
```

Spiegazione:

```
//es:" (45.42,10.95,45.45,10.98)"
(
  node["leisure"="park"](45.42,10.95,45.45,10.98);
  way["leisure"="park"](45.42,10.95,45.45,10.98);
  relation["leisure"="park"](45.42,10.95,45.45,10.98);
);
```

```
[out:json][timeout:25];
//o passandogli il nome es:"Catanzaro"
area["name"="Catanzaro"]["boundary"="administrative"]["admin_level"="8"]->.catanzaro;
(
  node["leisure"="park"](area.catanzaro);
  way["leisure"="park"](area.catanzaro);
  relation["leisure"="park"](area.catanzaro);
) ->.parchi;

// trova i nodi con barrier=gate o entrance nelle aree verdi (via map_to_area)
.parchi map_to_area -> .aree_parco;

(
  node["barrier"="gate"](area.aree_parco);
  node["barrier"="entrance"](area.aree_parco);
);

out body;
>;
out skel qt;
// oppure out geom;
```

Spiegazione:

- out:json: formato di output (può essere anche XML)





Integrazione delle informazioni disponibili

Nonostante l'obiettivo sia quello di procedere con un approccio bottom-up partendo da fonti ufficiali, la sovrapposizione con layer estratti da OpenStreetMap e da Copernicus Urban Atlas Land Cover/Land Use 2018 (UA) diventa utile per integrare il contenuto informativo dei layer resi disponibili dalle Amministrazioni locali.

Inoltre, se i dati dei punti di accesso non fossero disponibili, si potrebbero identificare attraverso l'analisi spaziale nei seguenti modi:

- Considerare solo i nodi estratti da OpenStreetMap con la parola chiave "barrier" ed i tag "gate" e "entrance", localizzati all'interno o sul perimetro dell'area verde considerata
- Considerare l'intersezione tra il layer "strade" e "aree verdi urbane", scaricati da OpenStreetMap.

Laddove le condizioni precedenti non possono essere soddisfatte, una soluzione potrebbe essere rappresentata dalla individuazione di punti ogni 100 metri lungo il perimetro.

Fare clic per inserire le note

Esegui

Condividi

Esporta

Wizard

Salva

Carica

Impostazioni

[Aiuto](#)

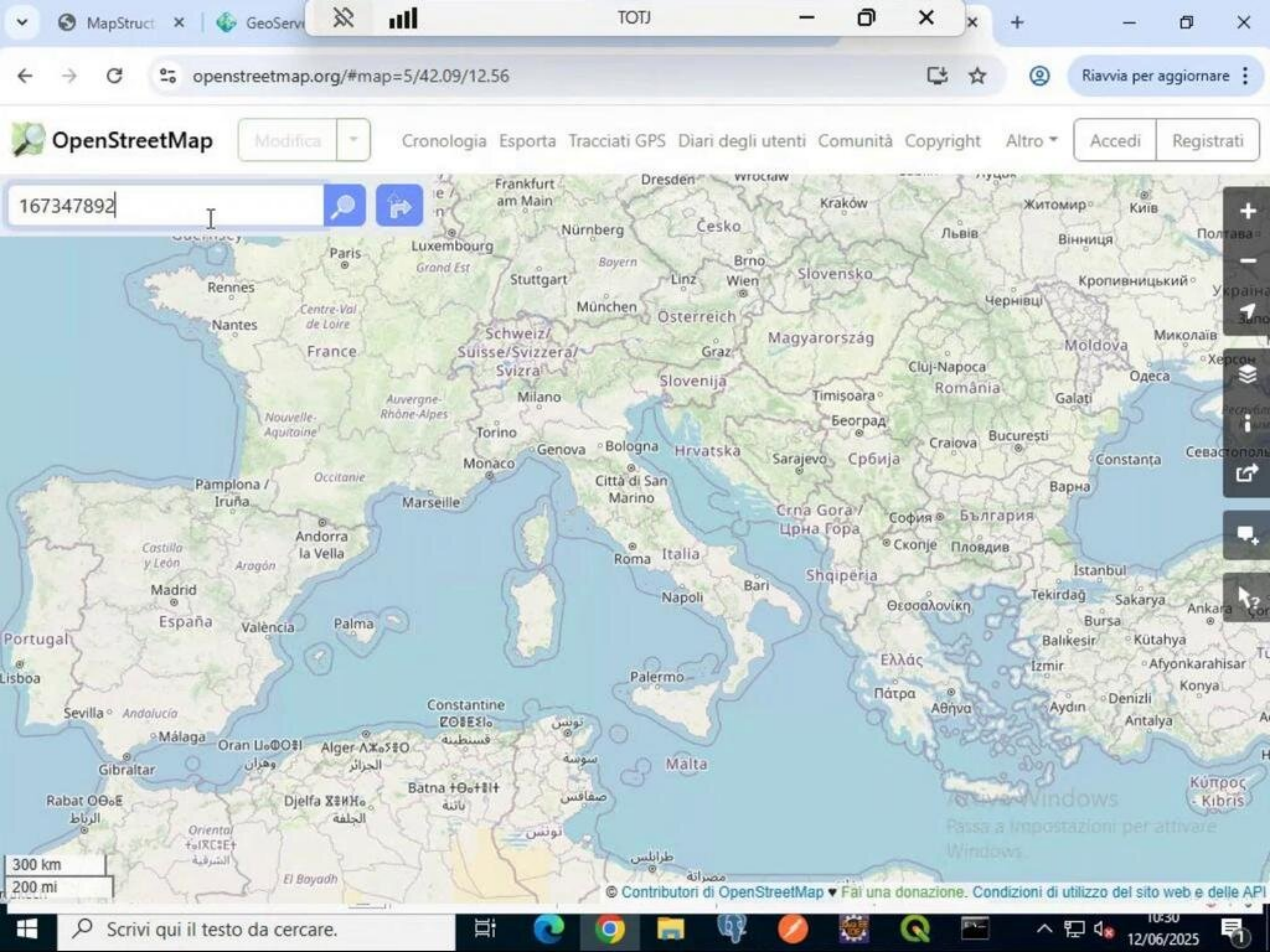
overpass turbo 

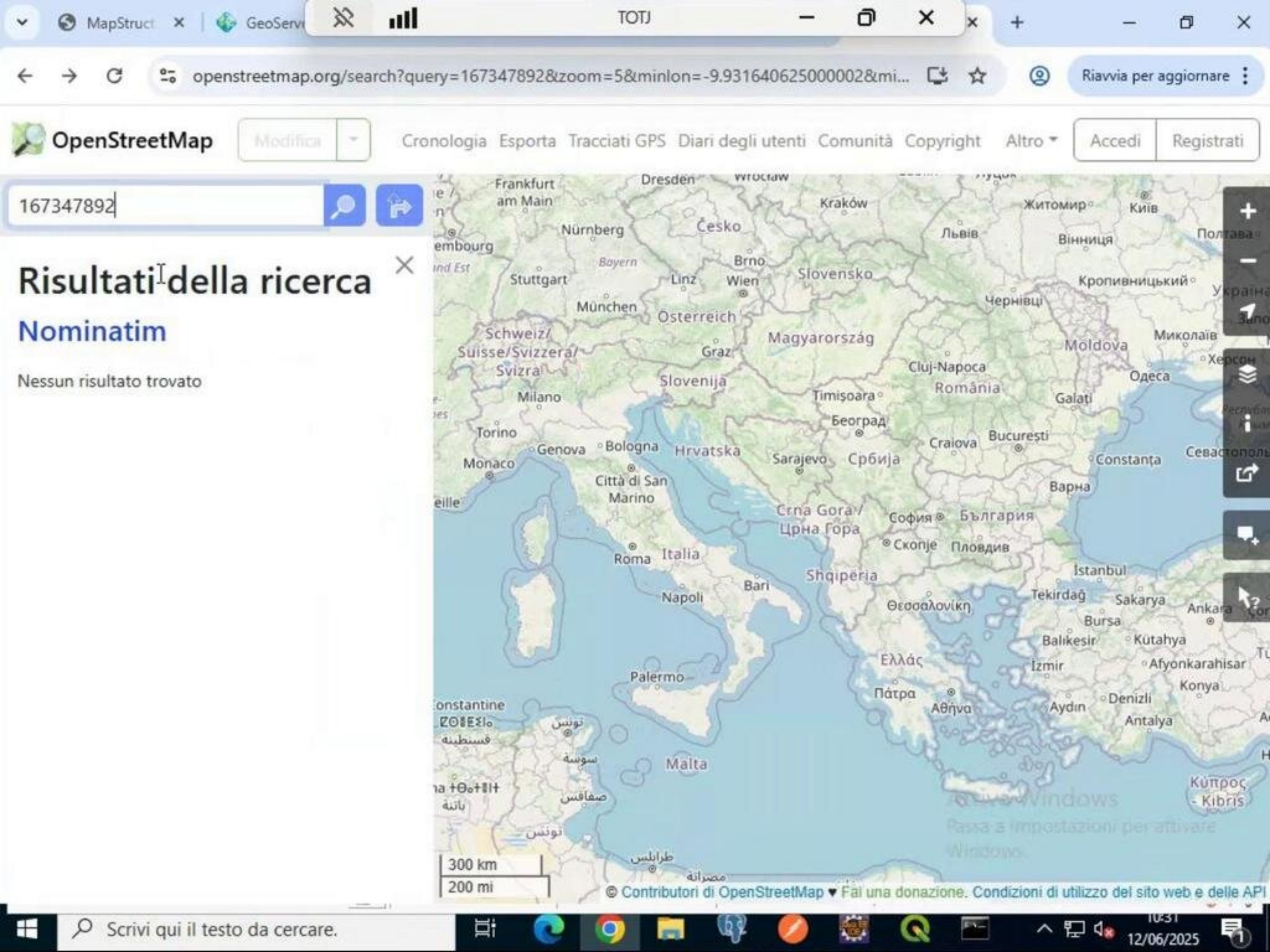
```

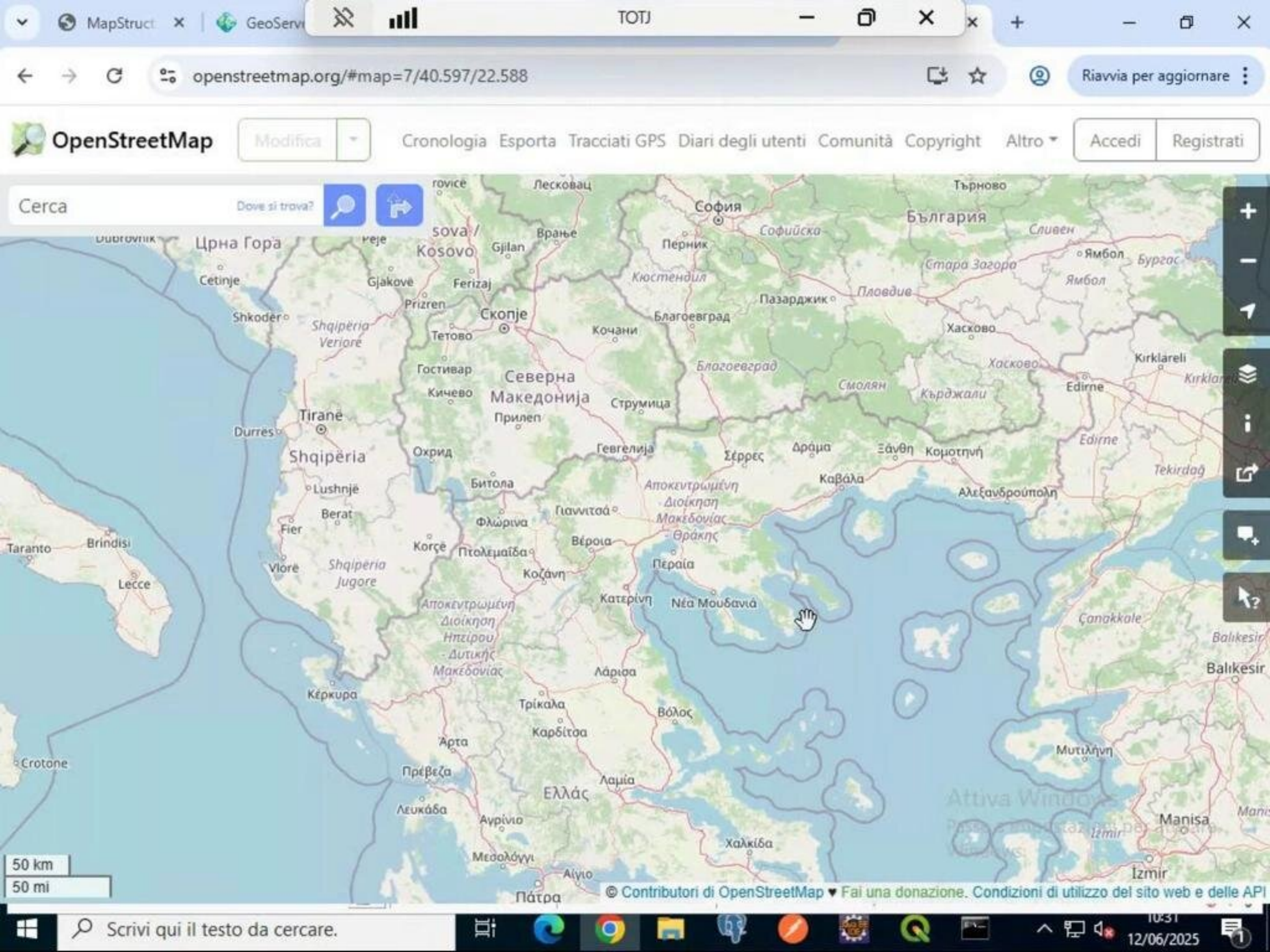
1[out:json][timeout:25];
2//o passandogli il nome es:"Catanzaro"
3area["name"="Catanzaro"]["boundary"="administrative"]["admin_level"="8"]->.catanzaro;
4(
5  node["leisure"="park"](area.catanzaro);
6  way["leisure"="park"](area.catanzaro);
7  relation["leisure"="park"](area.catanzaro);
8);
9
10
11out body;
12>;
13out skel qt;
14

```











Commenti

Consente di visualizzare e rispondere ai commenti inclusi nel documento.

```
//es:" (45.42,10.95,45.45,10.98)"
(
  node["leisure"="park"](45.42,10.95,45.45,10.98);
  way["leisure"="park"](45.42,10.95,45.45,10.98);
  relation["leisure"="park"](45.42,10.95,45.45,10.98);
);

[out:json][timeout:25];
//o passandogli il nome es:"Catanzaro"
area["name"="Catanzaro"]["boundary"="administrative"]["admin_level"="8"]->.catanzaro;
(
  node["leisure"="park"](area.catanzaro);
  way["leisure"="park"](area.catanzaro);
  relation["leisure"="park"](area.catanzaro);
) ->.parchi;

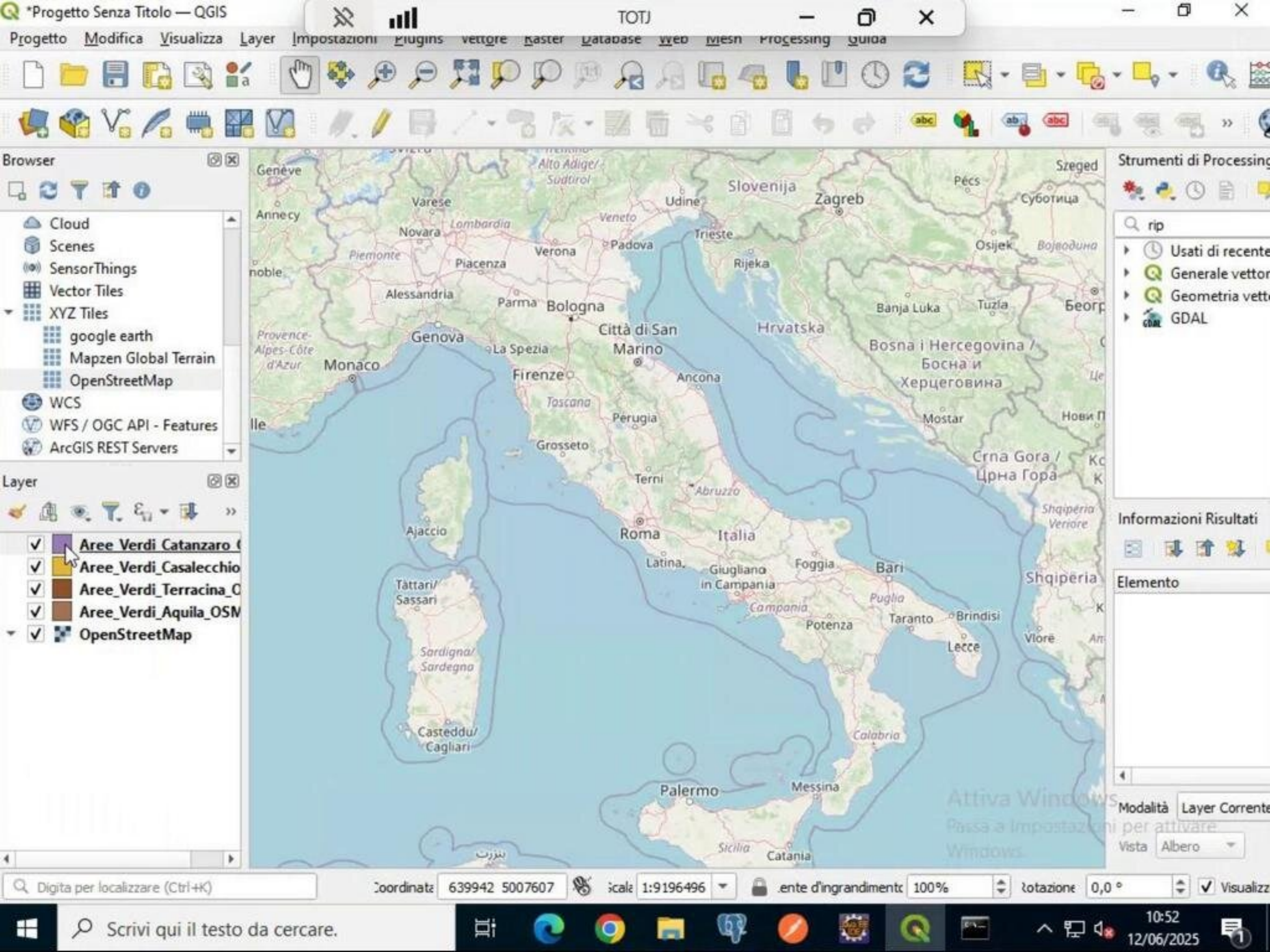
// trova i nodi con barrier=gate o entrance nelle aree verdi (via map_to_area)
.parchi map_to_area -> .aree_parco;

(
  node["barrier"="gate"](area.aree_parco);
  node["barrier"="entrance"](area.aree_parco);
);

out body;
>;
out skel qt;
// oppure out geom;
```

Spiegazione:





Progett

Zoom sui Layer

Zoom alla Selezione

Mostra nella Panoramica

Mostra Conteggio Elementi

Mostra Etichette

Copia Layer

Rinomina Layer

Execute SQL...

Duplica Layer

Rimuovi Layer...

Sposta in Cima

Sposta in Fondo

Apri Tabella Attributi

Attiva Modifiche

Filtro...

Cambia Sorgente Dati...

Imposta Scala e Visibilità Layer...

SR del layer

Esporta

Stili

Aggiungi Note Layer...

Proprietà...

Aree Verdi Aquila

OpenStreetMap

Digita per localizzare (Ctrl+K)

TOTJ

Plugins vettore raster Database Web Mesh Processing Guida

Copia lo stile

Aggiungi...

Rinomina Stile Corrente...

predefinito

Colori recenti

Modifica Simbolo...

Copia Simbolo

Incolla Simbolo

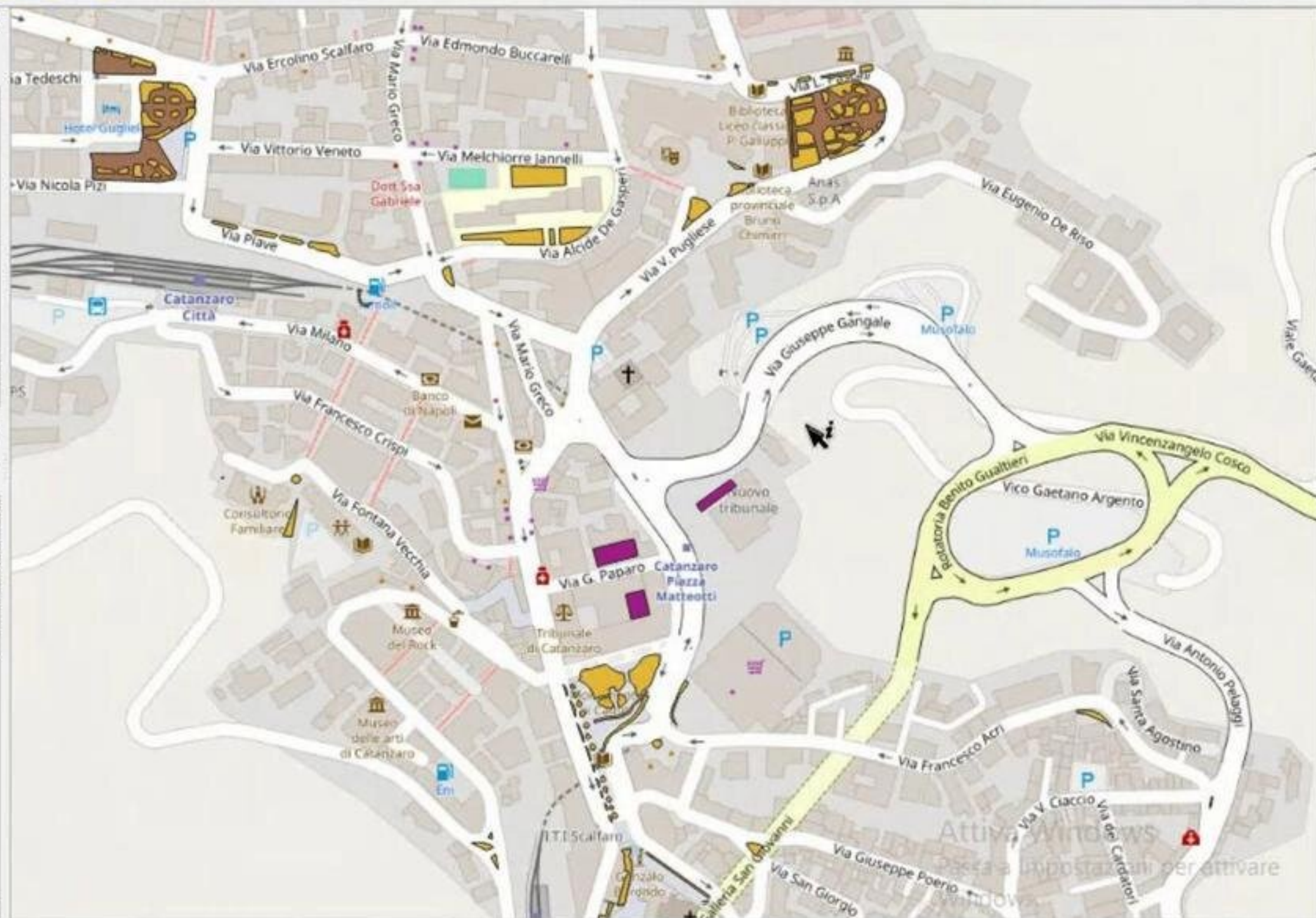
Attiva Windows

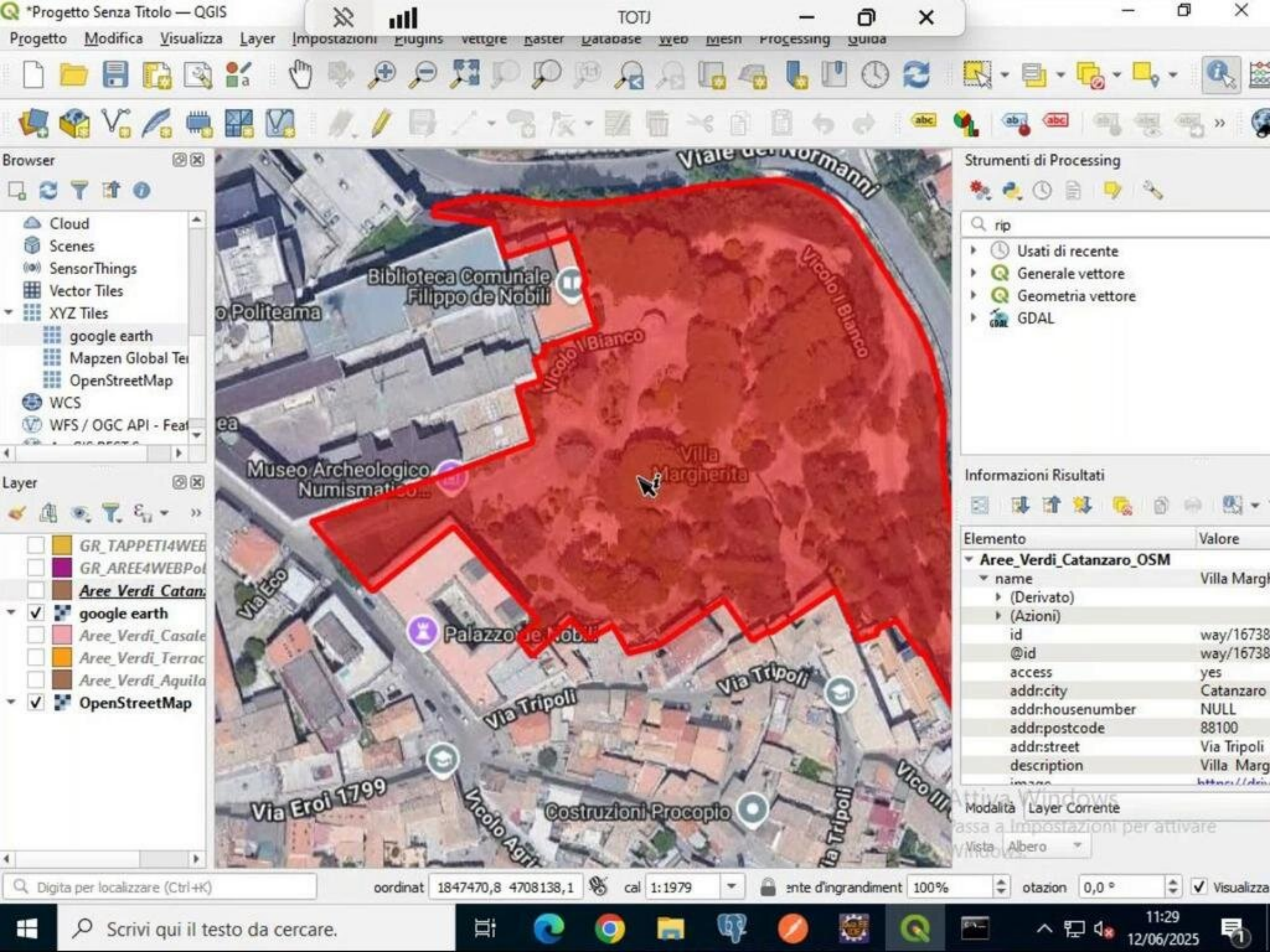
Passa a Impostazioni per attivare Windows.

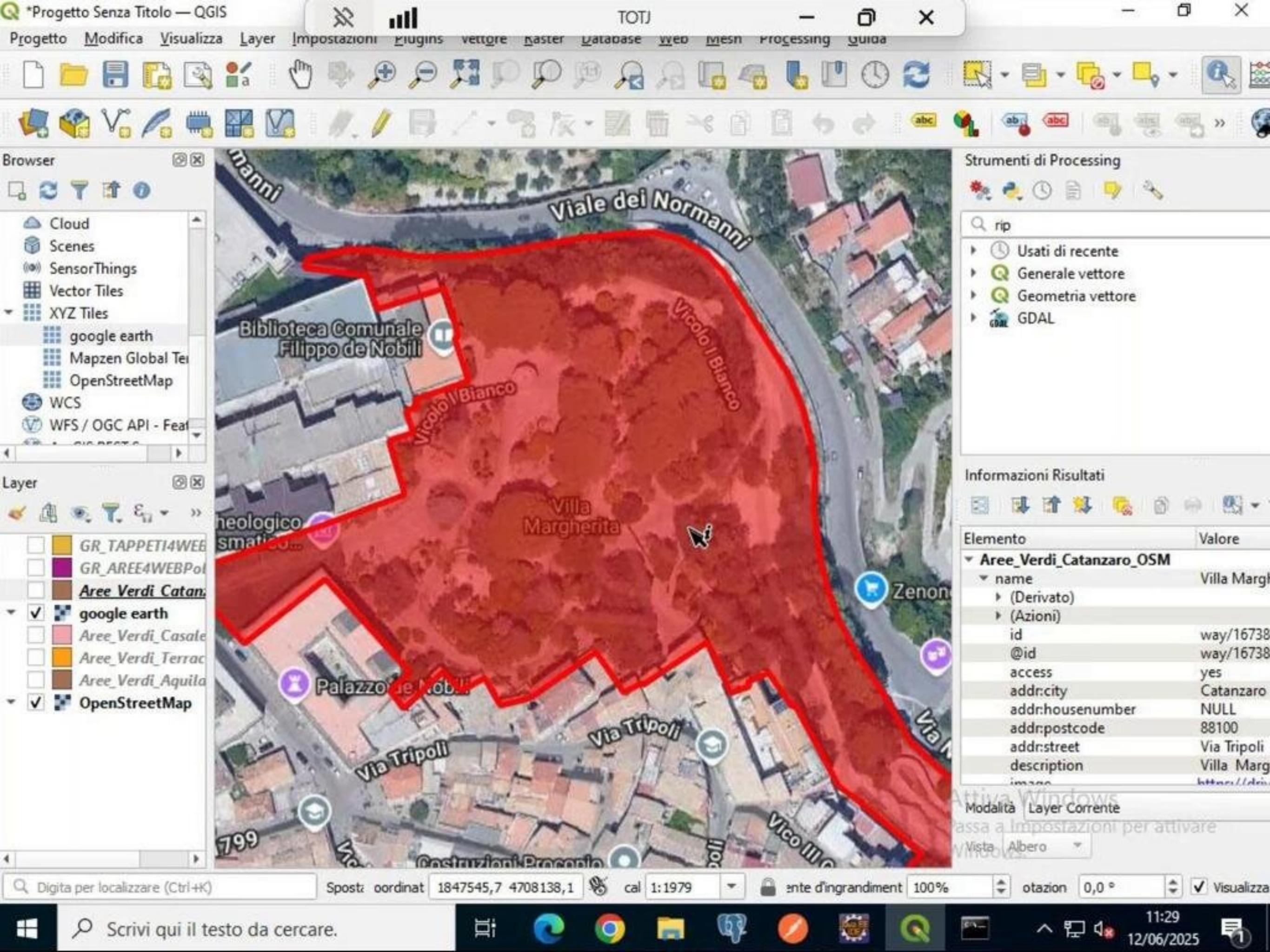
ente d'ingrandimento 100%

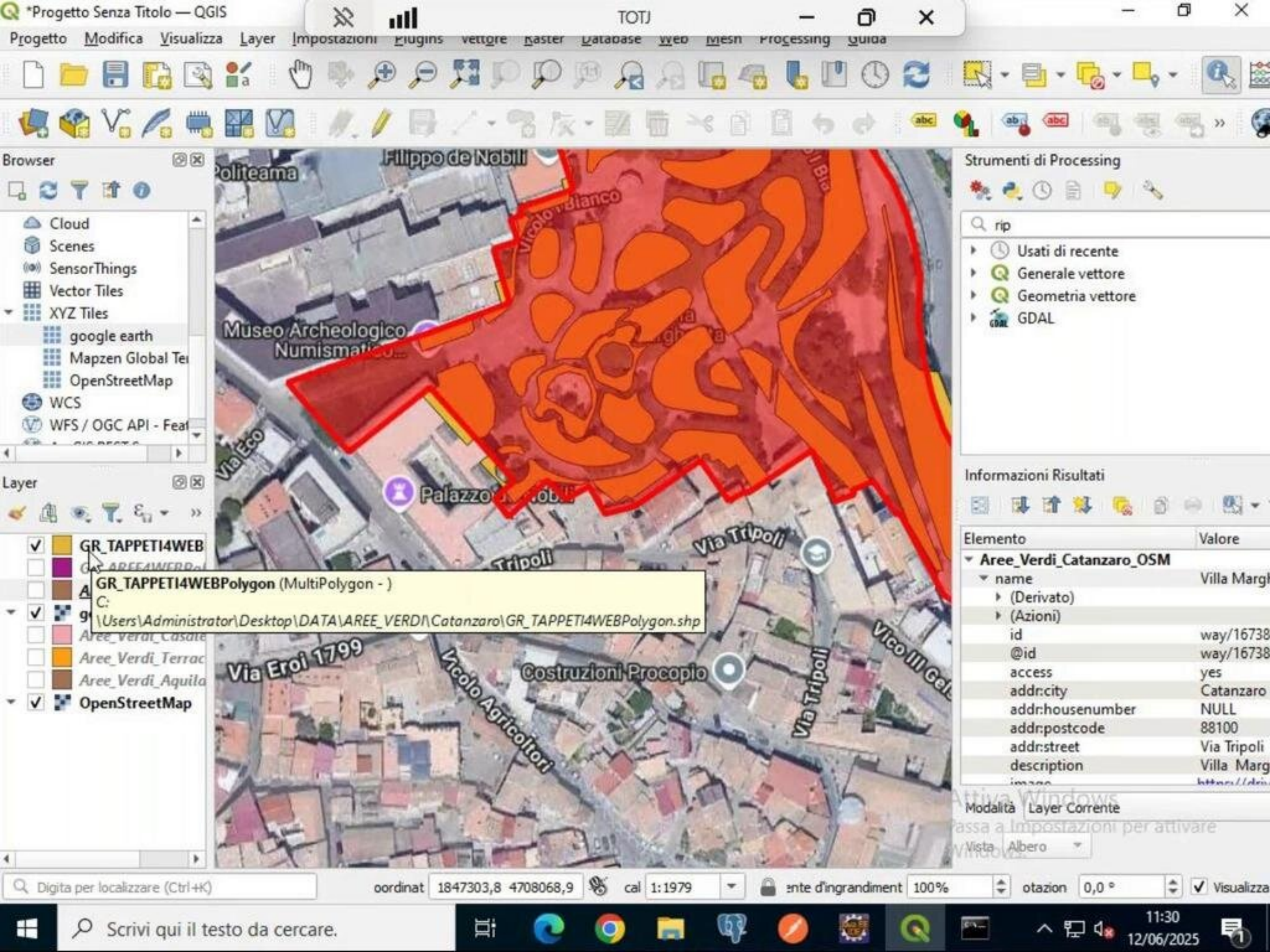
Rotazione 0,0°

Visualizza









Browser

- Cloud
- Scenes
- SensorThings
- Vector Tiles
- XYZ Tiles
 - google earth
 - Mapzen Global Ter
 - OpenStreetMap
- WCS
- WFS / OGC API - Feat

Layer

- ☒ GR_TAPPETI4WEB
- ☐ GR_TAPPETI4WEBPolygon (MultiPolygon -)
- ☒ Aree_Verdi_Casale
- ☐ Aree_Verdi_Terrac
- ☐ Aree_Verdi_Aquila
- ☒ OpenStreetMap



Strumenti di Processing

rip

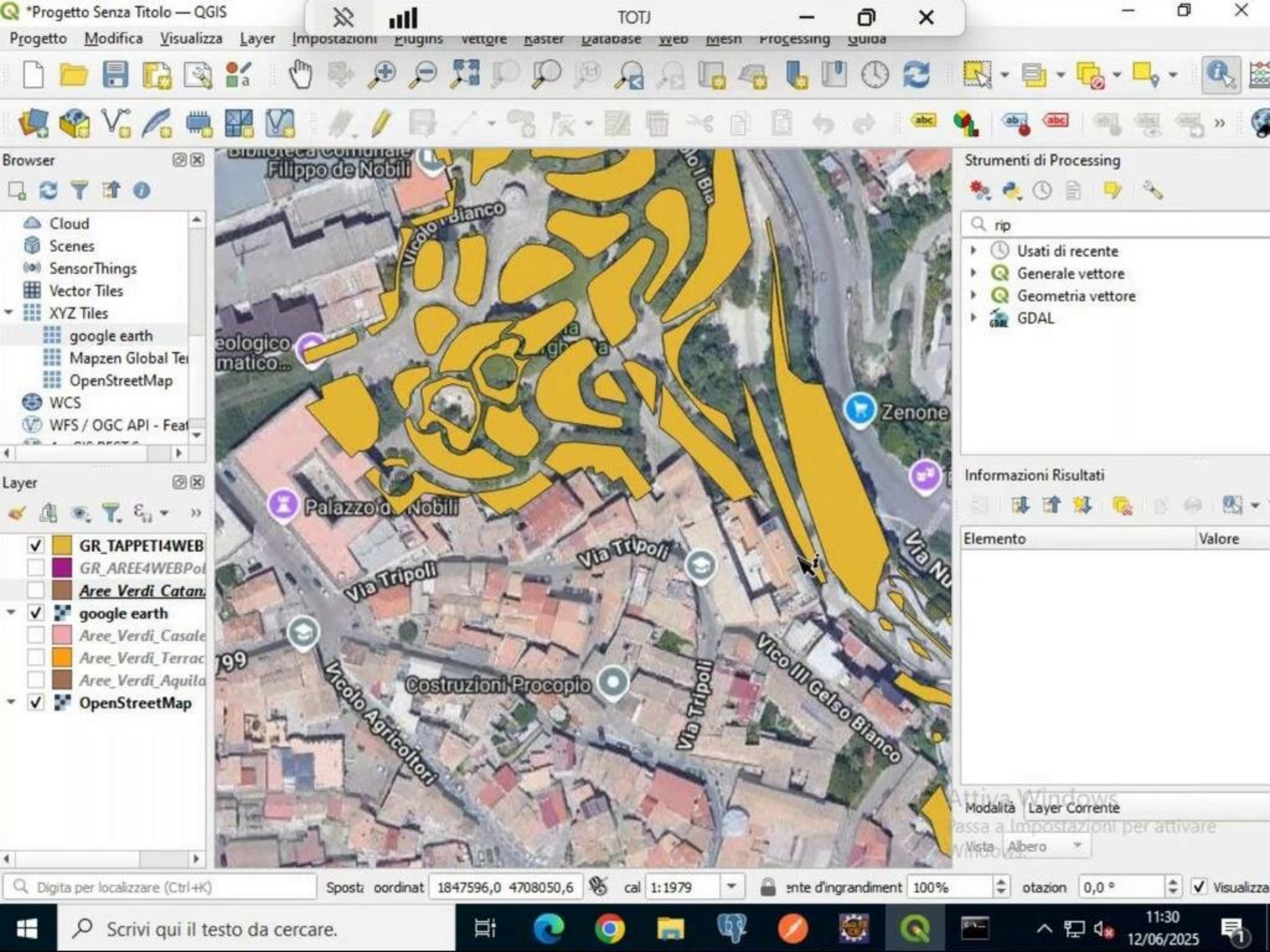
- Usati di recente
- Generale vettore
- Geometria vettore
- GDAL

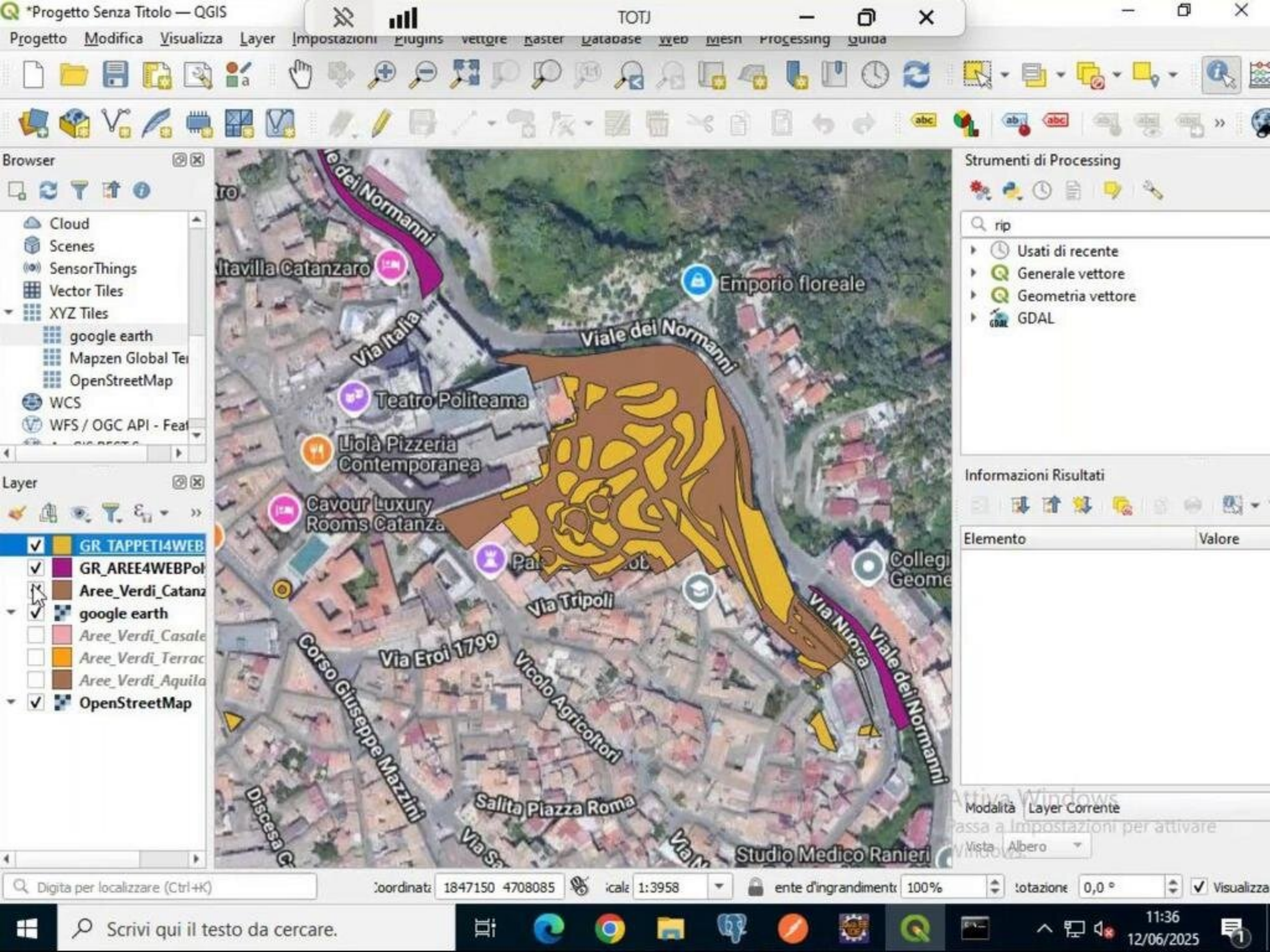
Informazioni Risultati

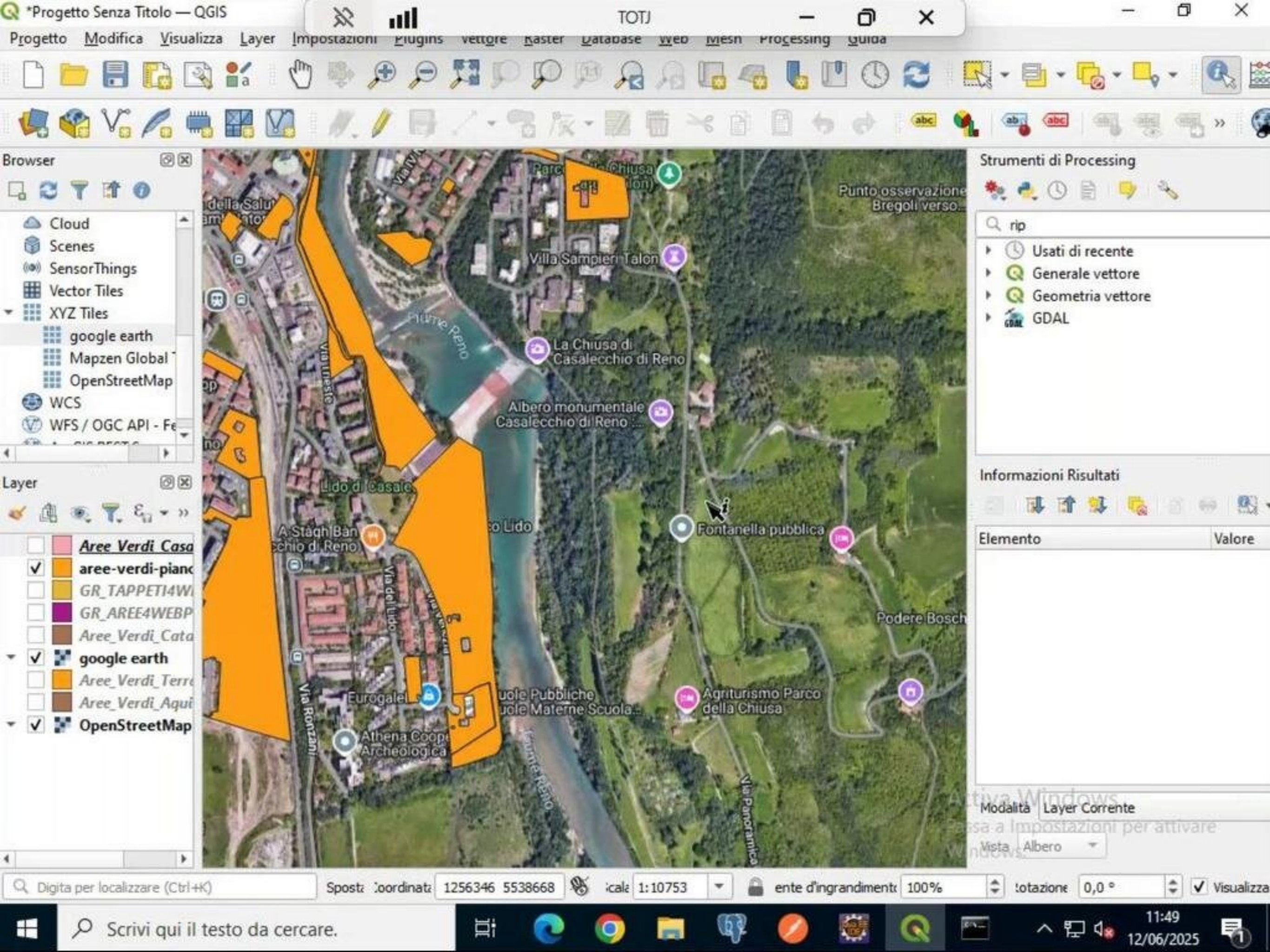
Elemento	Valore
Aree_Verdi_Catanzaro_OSM	
name	Villa Margh
(Derivato)	
(Azioni)	
id	way/16738
@id	way/16738
access	yes
addr:city	Catanzaro
addr:housenumber	NULL
addr:postcode	88100
addr:street	Via Tripoli
description	Villa Marg
image	https://...

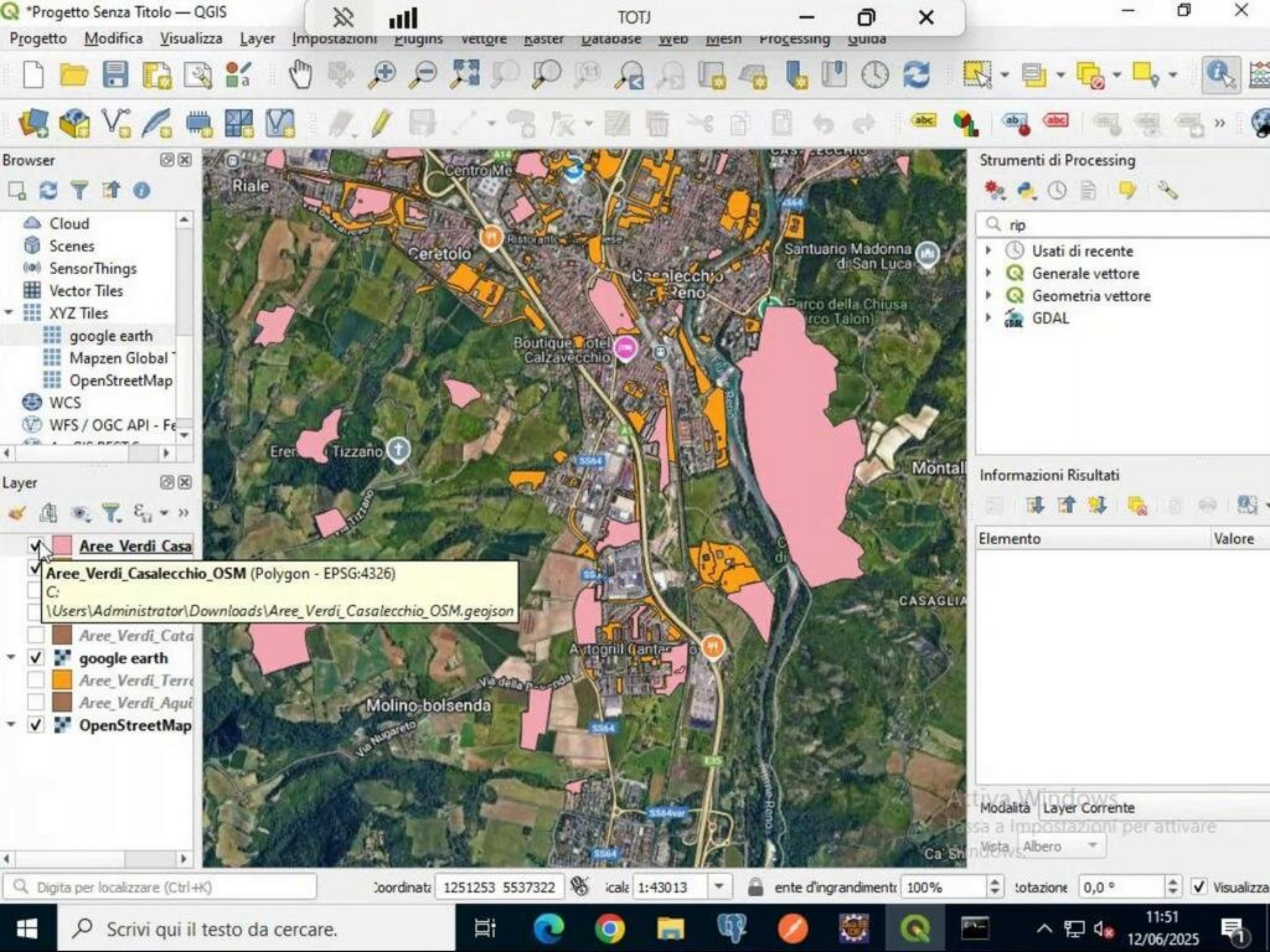
Modalità Layer Corrente

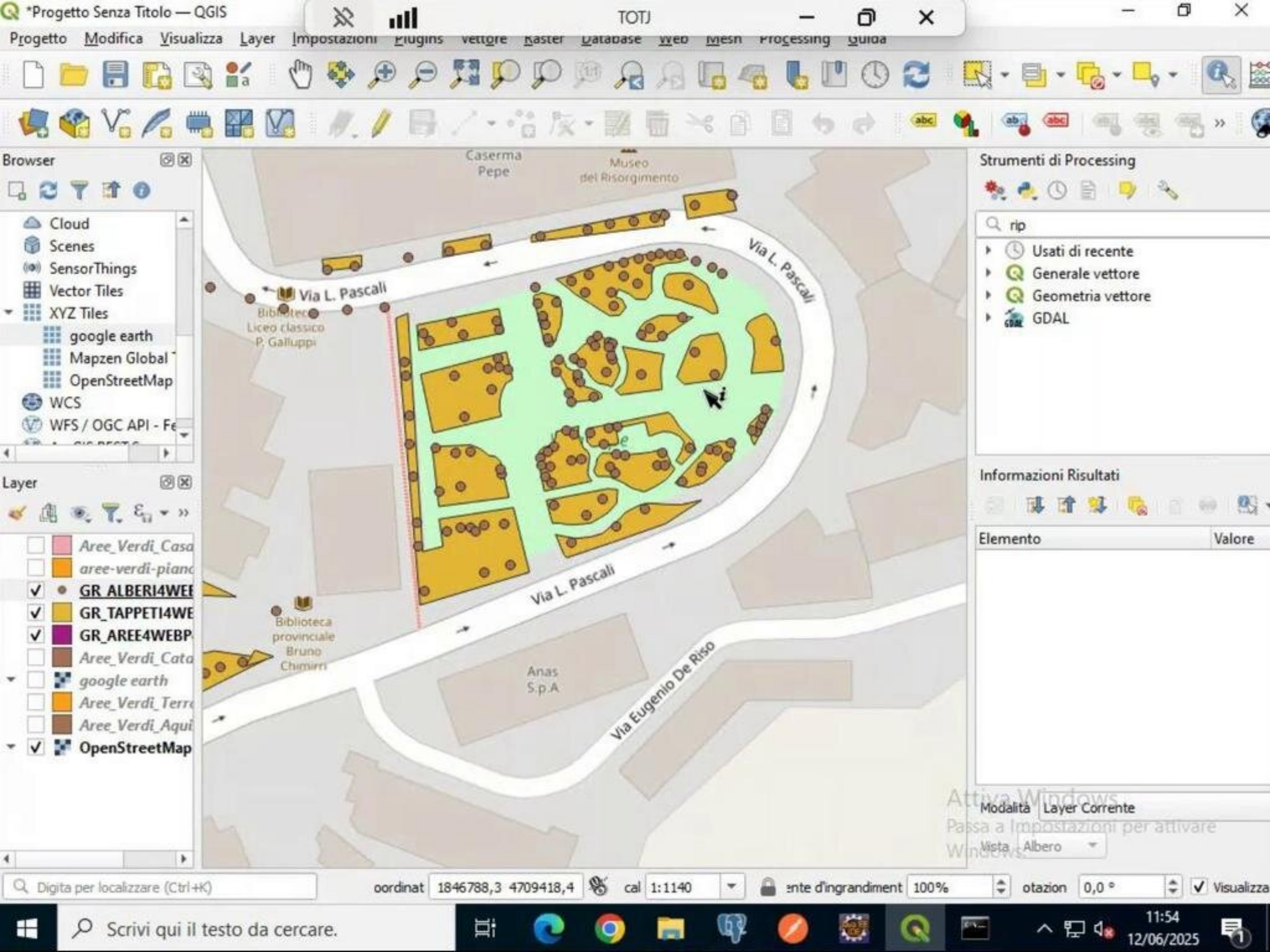
Vista Albero



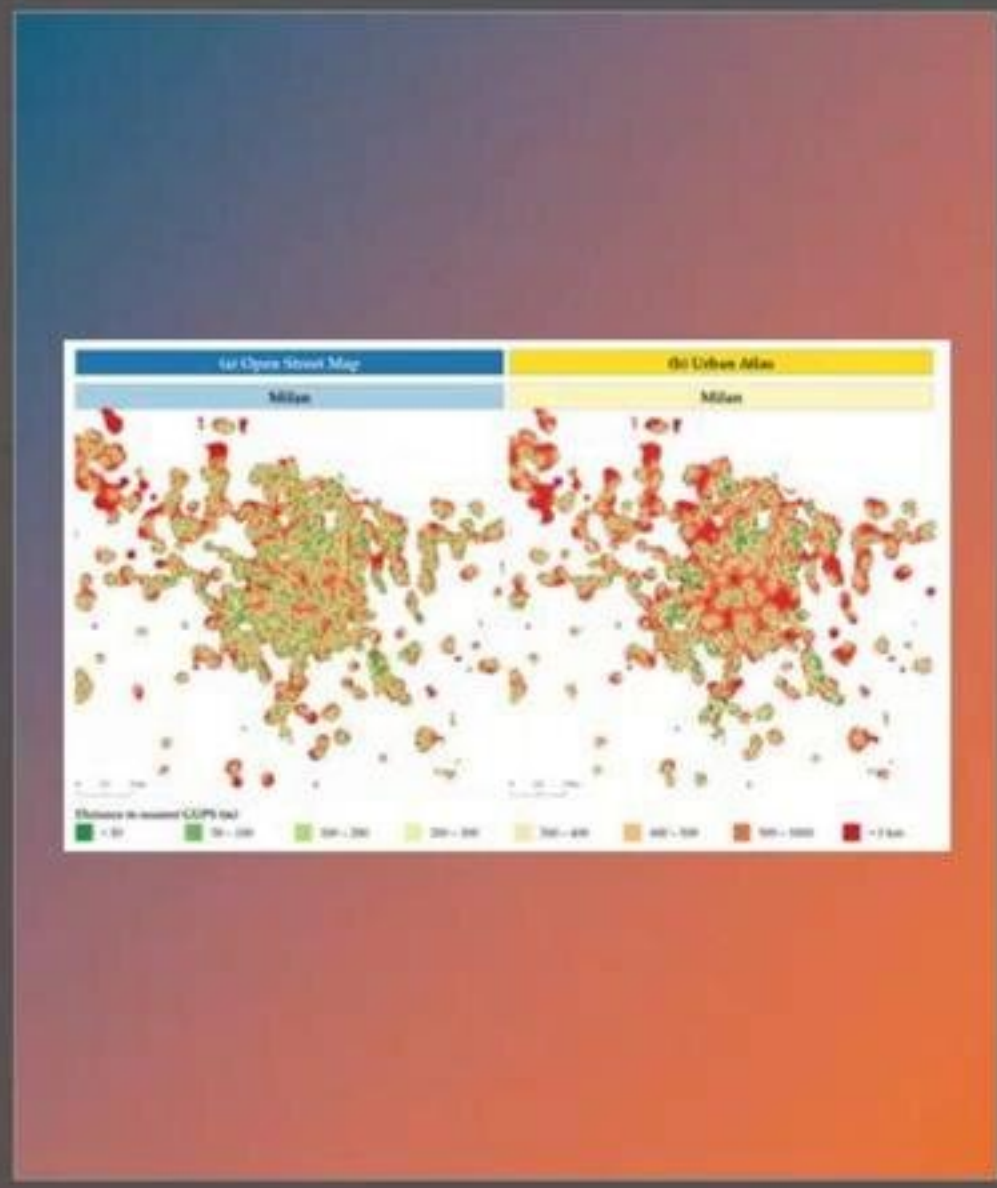


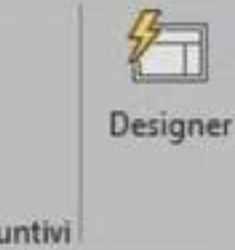






- 7
- 8
- 9
- 10
- 11







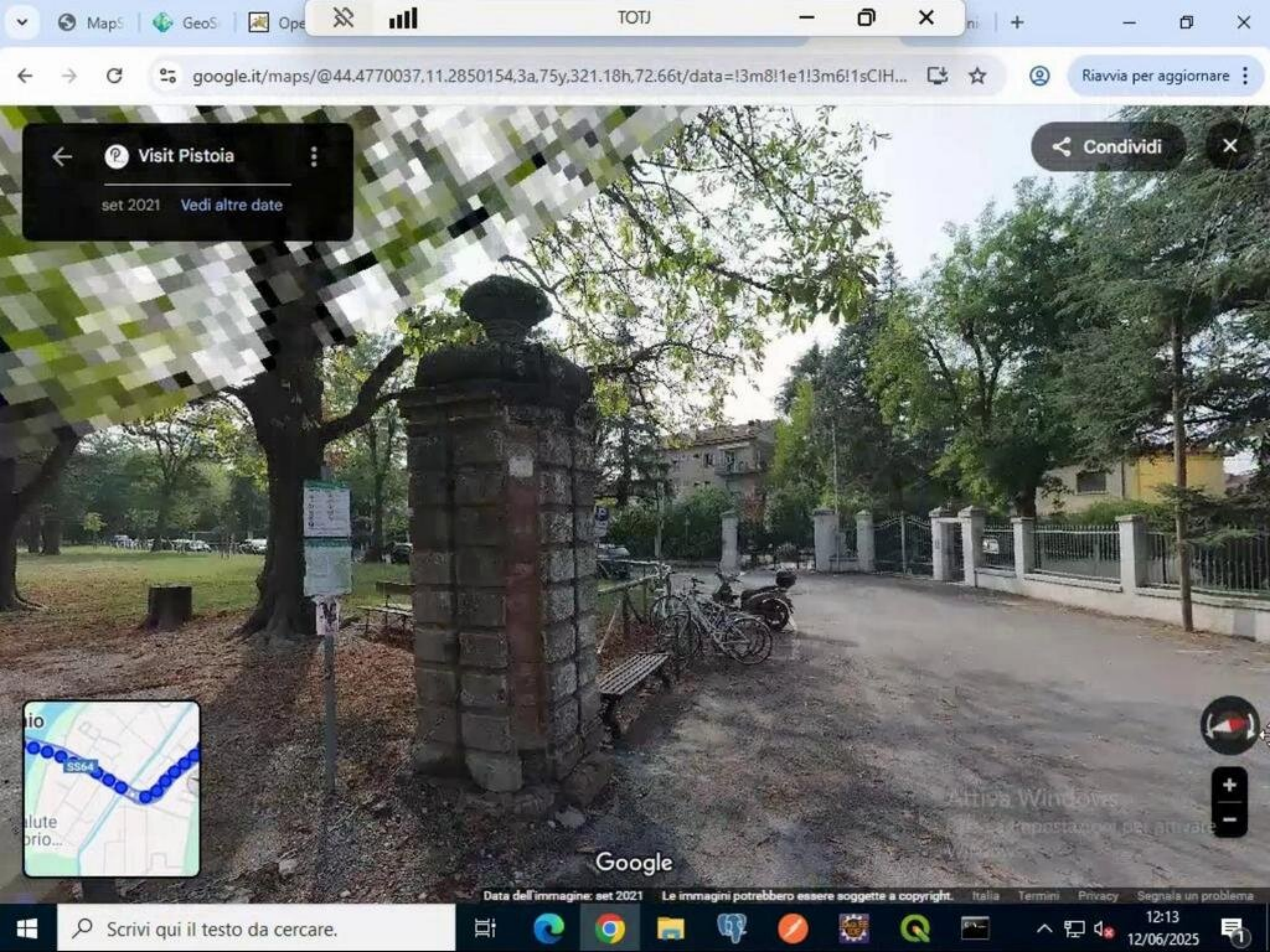
Integrazione delle informazioni disponibili

Nonostante l'obiettivo sia quello di procedere con un approccio bottom-up partendo da fonti ufficiali, la sovrapposizione con layer estratti da OpenStreetMap e da Copernicus Urban Atlas Land Cover/Land Use 2018 (UA) diventa utile per integrare il contenuto informativo dei layer resi disponibili dalle Amministrazioni locali.

Inoltre, se i dati dei punti di accesso non fossero disponibili, si potrebbero identificare attraverso l'analisi spaziale nei seguenti modi:

- Considerare solo i nodi estratti da OpenStreetMap con la parola chiave "barrier" ed i tag "gate" e "entrance", localizzati all'interno o sul perimetro dell'area verde considerata
- Considerare l'intersezione tra il layer "strade" e "aree verdi urbane", scaricati da OpenStreetMap.

Laddove le condizioni precedenti non possono essere soddisfatte, una soluzione potrebbe essere rappresentata dalla individuazione di punti ogni 100 metri lungo il perimetro.



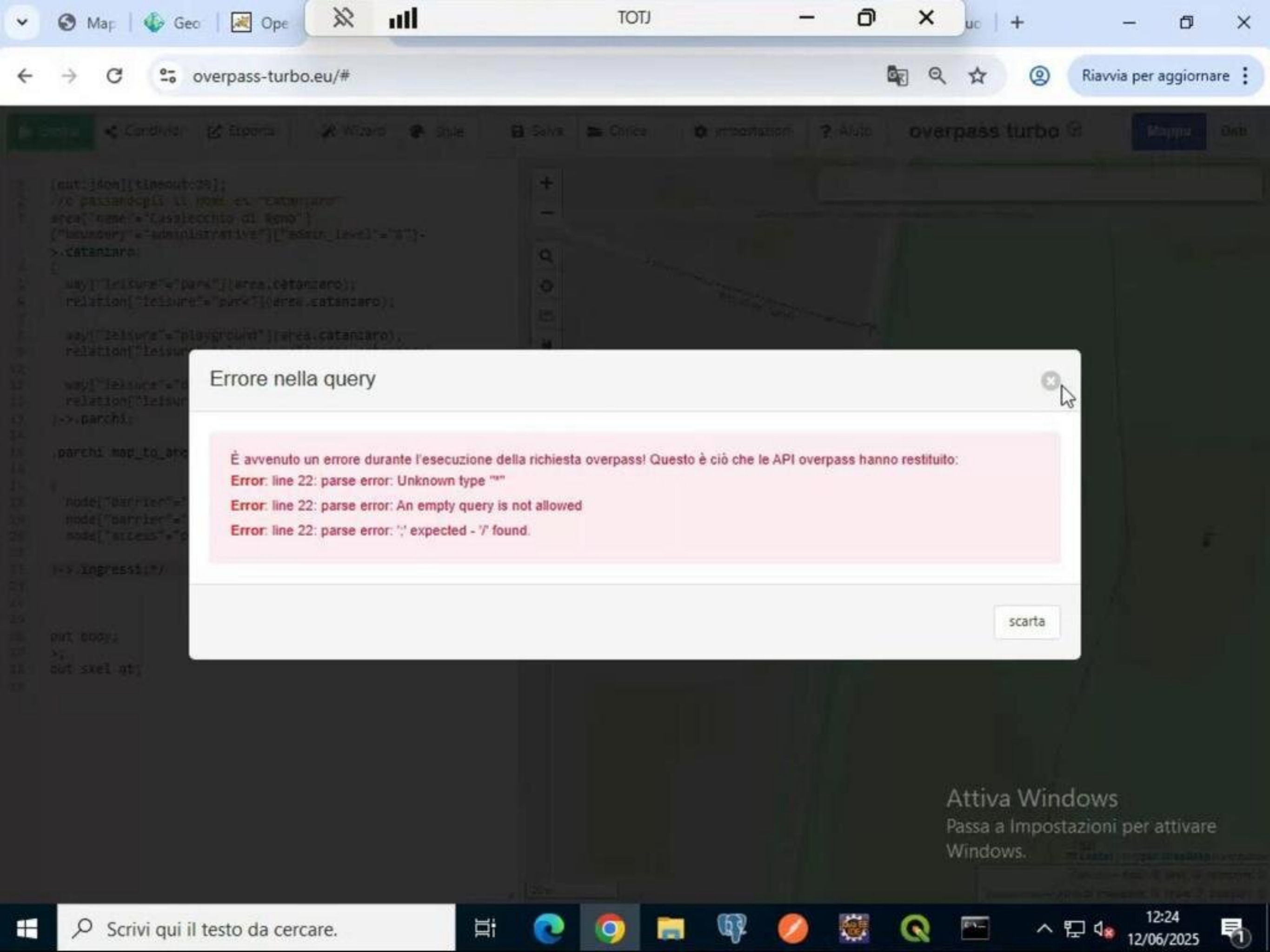
← Visit Pistoia

set 2021 Vedi altre date

Condividi



Google



Errore nella query

È avvenuto un errore durante l'esecuzione della richiesta overpass! Questo è ciò che le API overpass hanno restituito:

Error: line 22: parse error: Unknown type ""

Error: line 22: parse error: An empty query is not allowed

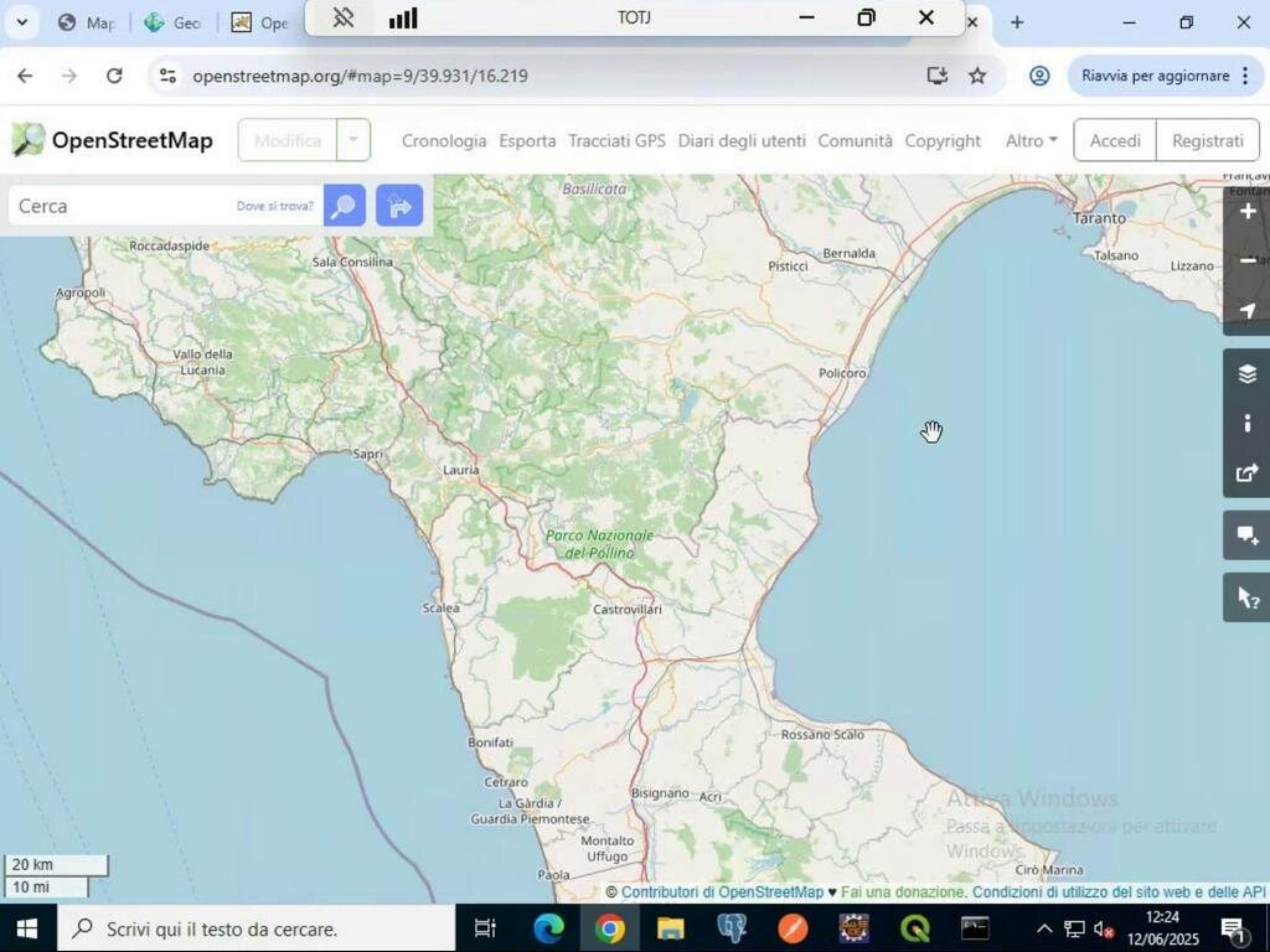
Error: line 22: parse error: ';' expected - '/' found.

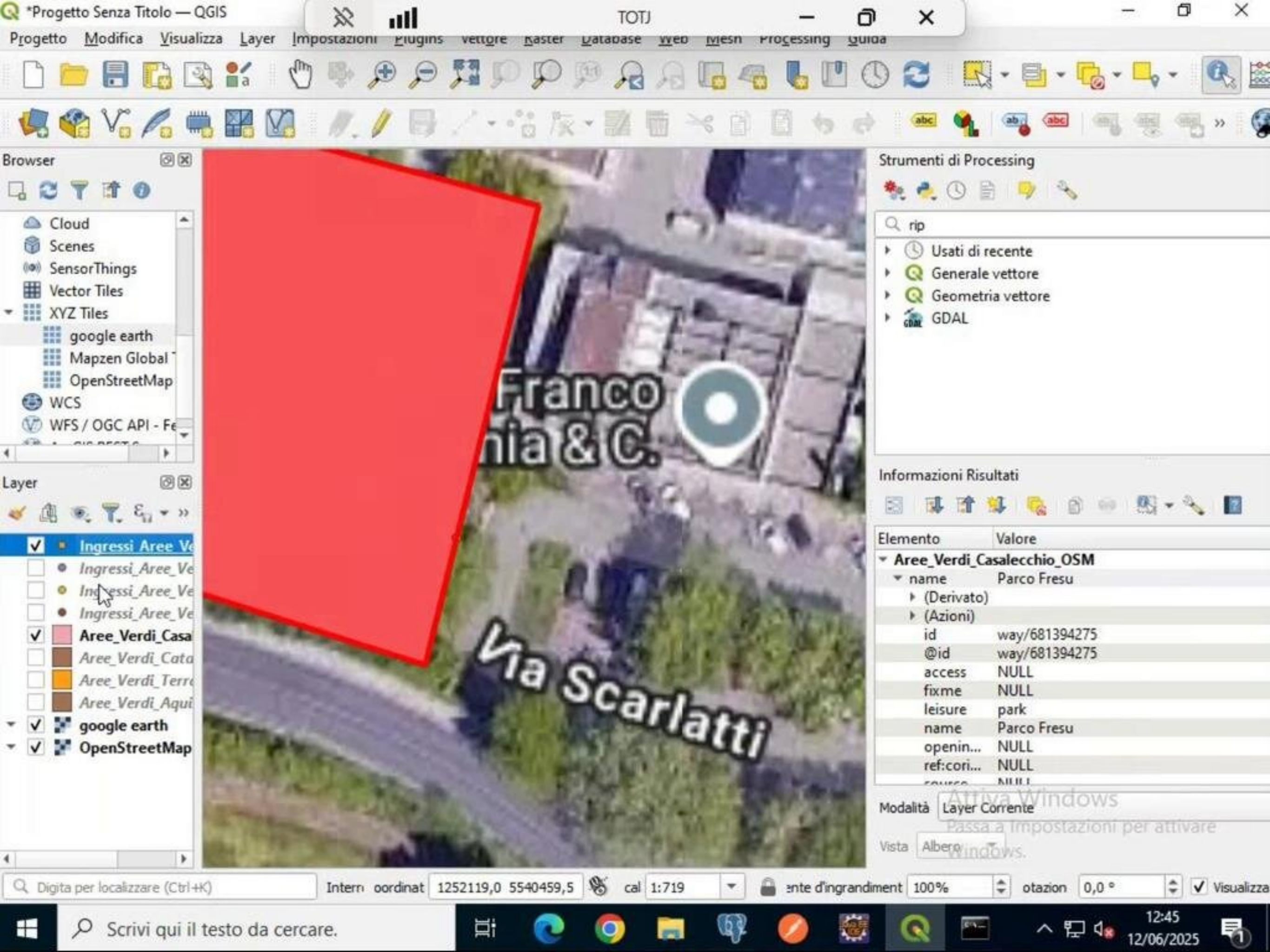
scarta

Attiva Windows
Passa a Impostazioni per attivare
Windows.

Scrivi qui il testo da cercare.

12:24
12/06/2025



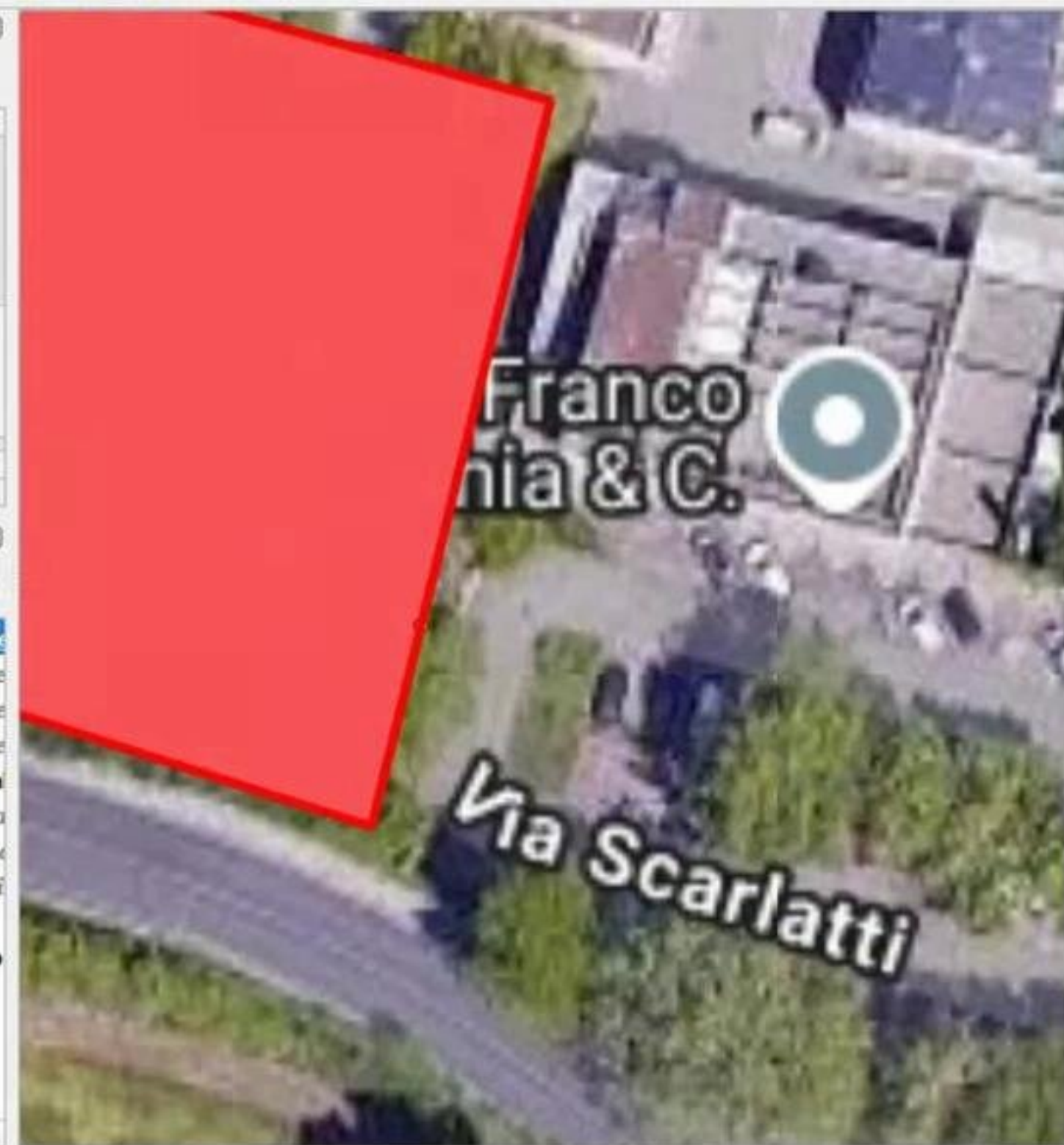


Browser

- Cloud
- Scenes
- SensorThings
- Vector Tiles
- XYZ Tiles
 - google earth
 - Mapzen Global
 - OpenStreetMap
- WCS
- WFS / OGC API - Fe

Layer

- ☒ Ingressi Aree Ve
- ☐ Ingressi_Aree_Ve
- ☐ Ingressi_Aree_Ve
- ☐ Ingressi_Aree_Ve
- ☒ Aree_Verdi_Casa
- ☐ Aree_Verdi_Cata
- ☐ Aree_Verdi_Terre
- ☐ Aree_Verdi_Aqui
- ☒ google earth
- ☒ OpenStreetMap



Strumenti di Processing



rip

- Usati di recente
- Generale vettore
- Geometria vettore
- GDAL

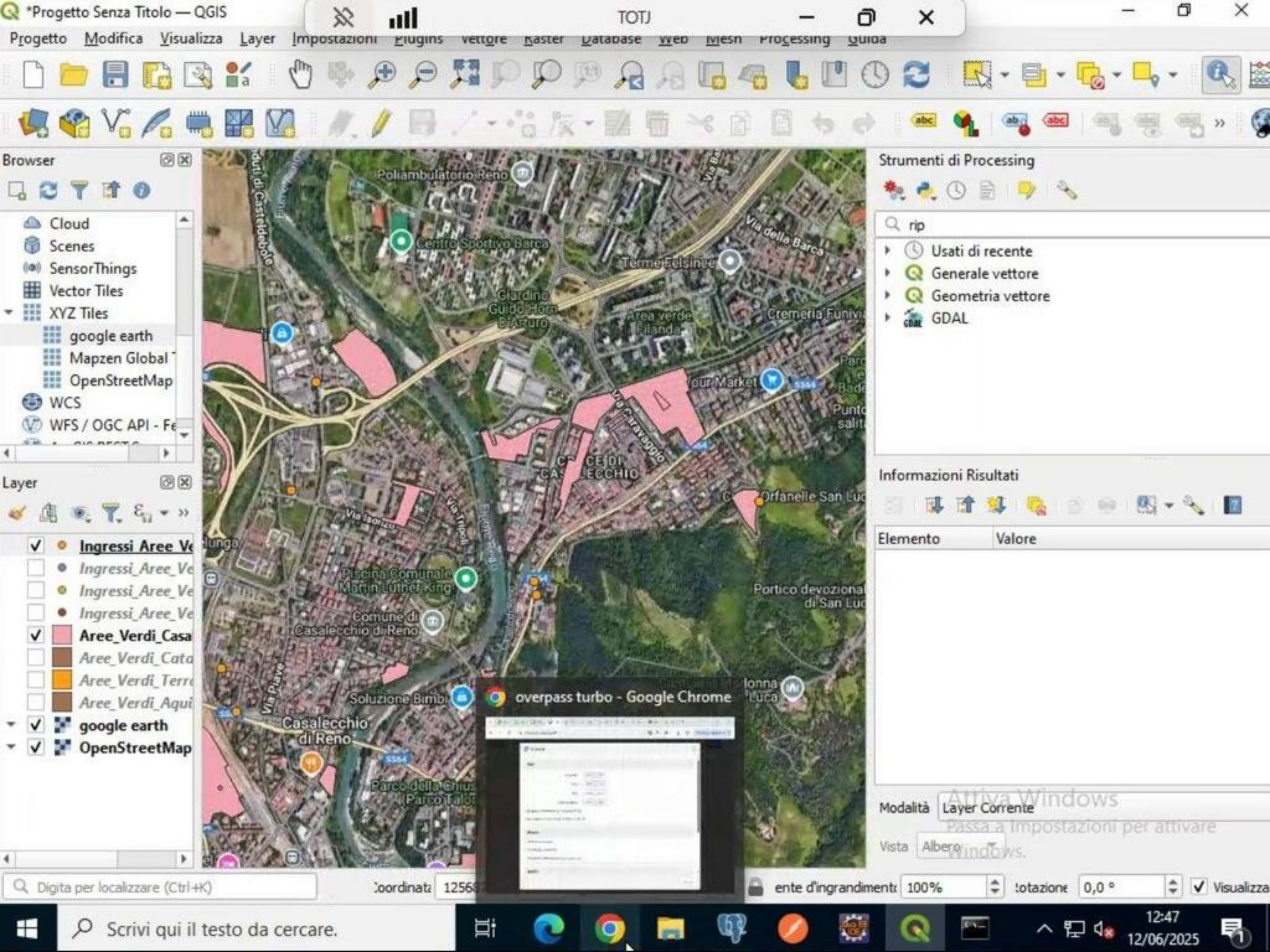
Informazioni Risultati



Elemento	Valore
Aree_Verdi_Casalecchio_OSM	
name	Parco Fresu
(Derivato)	
(Azioni)	
id	way/681394275
@id	way/681394275
access	NULL
fixme	NULL
leisure	park
name	Parco Fresu
openin...	NULL
ref:cori...	NULL
source	NULL

Modalità Layer Corrente

Passa a impostazioni per attivare windows.



Continuazione sviluppo delle classi per le API REST e dei relativi servizi


In questa fase del training si proseguirà lo sviluppo dell'architettura a strati del progetto, con particolare attenzione alla costruzione delle API REST e dei corrispondenti servizi di business logic nel backend Java/Spring.

Obiettivi principali:


- Consolidare e completare le operazioni CRUD (Create, Read, Update, Delete) per le principali entità del progetto, in particolare quelle legate alla gestione delle aree verdi, dei poligoni associati e di altri elementi territoriali.
- Sviluppare nuove API REST personalizzate per supportare operazioni specifiche richieste dal flusso applicativo, ad esempio:
 - Inserimento di poligoni selezionati su mappa
 - Aggiornamento di metadati ambientali associati a un'area
 - Recupero filtrato di aree sulla base di criteri spaziali o descrittivi
- Implementare la comunicazione tra controller, service e repository, garantendo la separazione delle responsabilità.

Attività tecniche previste:


- Progettazione e stesura di nuovi controller REST, annotati con @RequestMapping.
- Definizione di endpoint REST con annotazioni come @GetMapping, @PostMapping, @PutMapping e @DeleteMapping.
- Implementazione dei metodi nei service, che conterranno la logica di business.

 >


Bluetooth

 >


Modalità aereo

 >


Accessibilità




Risparmio energia




Sottotitoli in tempo reale





Luce notturna





 >





Continuazione sviluppo delle classi per le API REST e dei relativi servizi

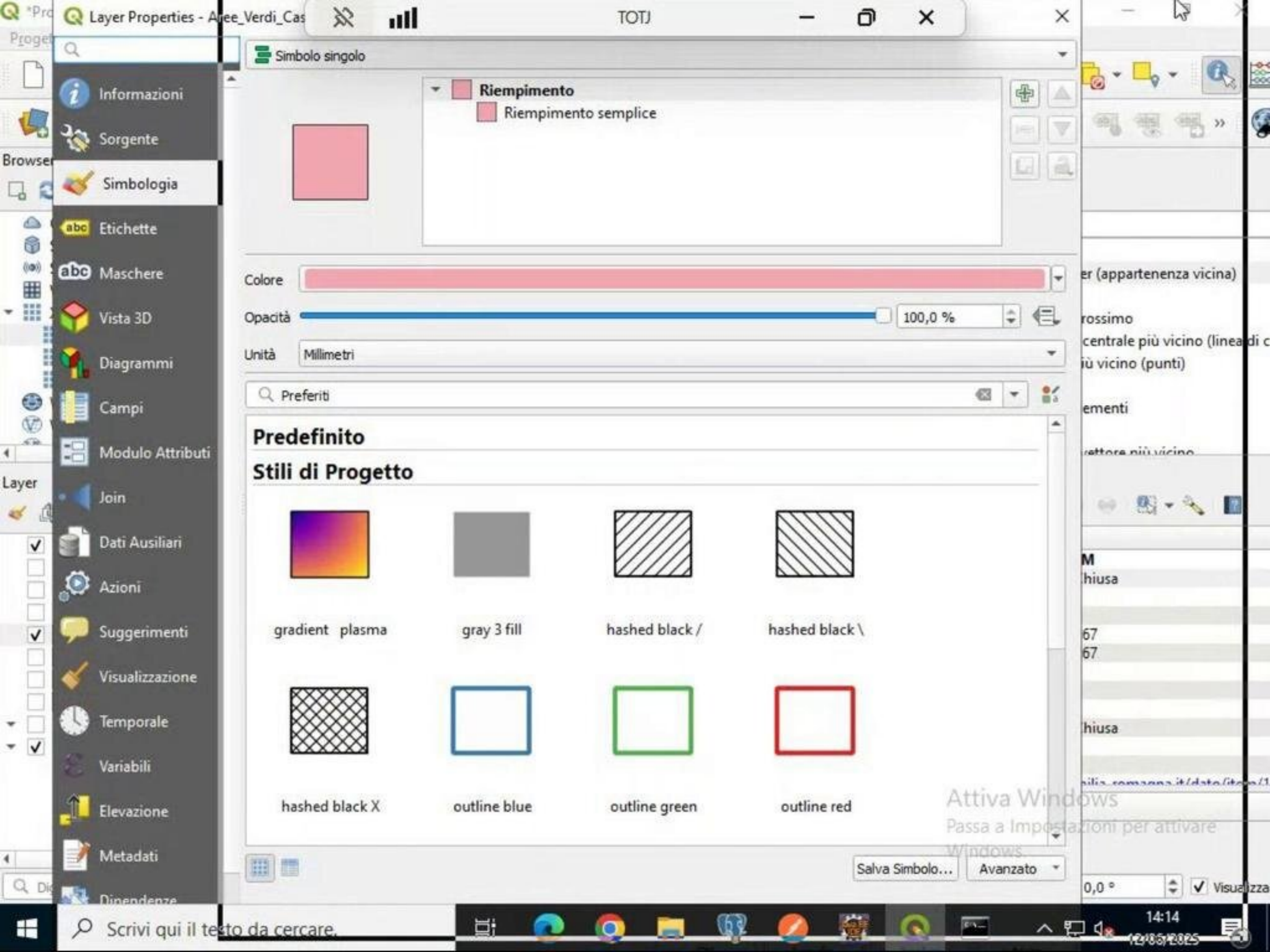
In questa fase del training si proseguirà lo sviluppo dell'architettura a strati del progetto, con particolare attenzione alla costruzione delle API REST e dei corrispondenti servizi di business logic nel backend Java/Spring.

Obiettivi principali:

- Consolidare e completare le operazioni CRUD (Create, Read, Update, Delete) per le principali entità del progetto, in particolare quelle legate alla gestione delle aree verdi, dei poligoni associati e di altri elementi territoriali.
- Sviluppare nuove API REST personalizzate per supportare operazioni specifiche richieste dal flusso applicativo, ad esempio:
 - Inserimento di poligoni selezionati su mappa
 - Aggiornamento di metadati ambientali associati a un'area
 - Recupero filtrato di aree sulla base di criteri spaziali o descrittivi
- Implementare la comunicazione tra controller, service e repository, secondo i principi della separazione delle responsabilità.

Attività tecniche previste:

- Progettazione e stesura di nuovi controller REST, annotati con `@RestController` e `@RequestMapping`.
- Definizione di endpoint REST con annotazioni come `@GetMapping`, `@PostMapping`, `@PutMapping` e `@DeleteMapping`.
- Implementazione dei metodi nei service, che conterranno la logica di validazione e



Progetto Senza Titolo - QGIS

Progetto Modifica Visualizza

Browser

Cloud

Scenes

SensorThings

Vector Tiles

XYZ Tiles

google earth

Mapzen Global

OpenStreetMap

WCS

WFS / OGC API - Fe

Layer

☒ Ingressi_Aree_Ve

☐ Ingressi_Aree_Ve

☐ Ingressi_Aree_Ve

☐ Ingressi_Aree_Ve

☒ Aree Verdi Casa

☐ Aree Verdi Cata

☐ Aree Verdi Terr

☐ Aree Verdi Aqu

☐ google earth

☒ OpenStreetMap

Layer P

Informazioni

Sorgente

Simbologia

Etichette

Maschere

Vista 3D

Diagrammi

Campi

Modulo Attributi

Join

Dati Ausiliari

Azioni

Suggerimenti

Visualizzazione

Temporale

Variabili

Elevazione

Metadati

Dipendenze

TOTJ

Simbolo singolo

Riempimento

Riempimento semplice

Colore

Opacità

Unità

Preferiti

Predefinito

Stili di Progetto

gradient plasma

gray 3 fill

hashed black /

hashed black \

hashed black X

outline blue

outline green

outline red

Salva Simbolo...

Avanzato

Attiva Windows

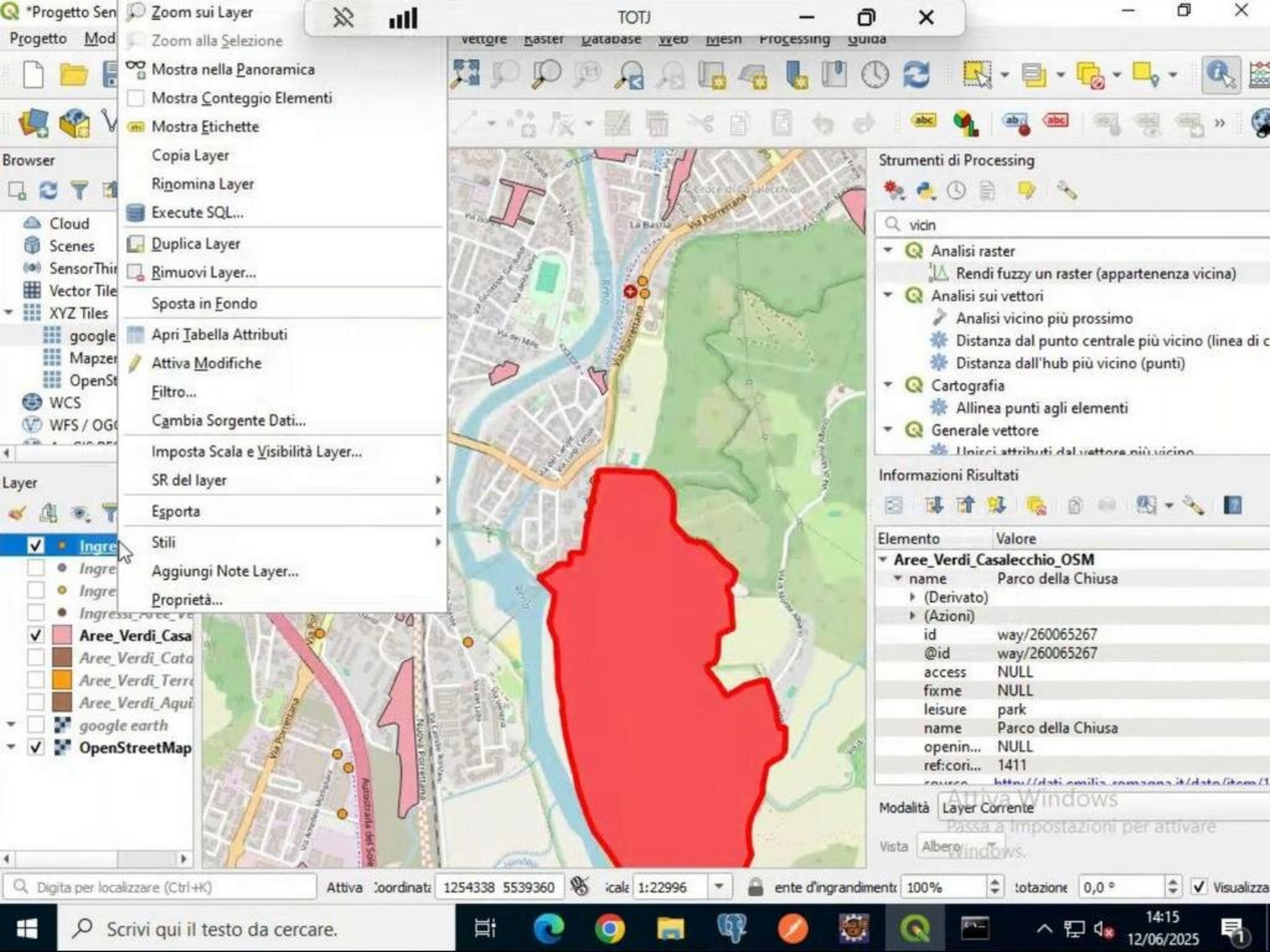
Passa a Impostazioni per attivare

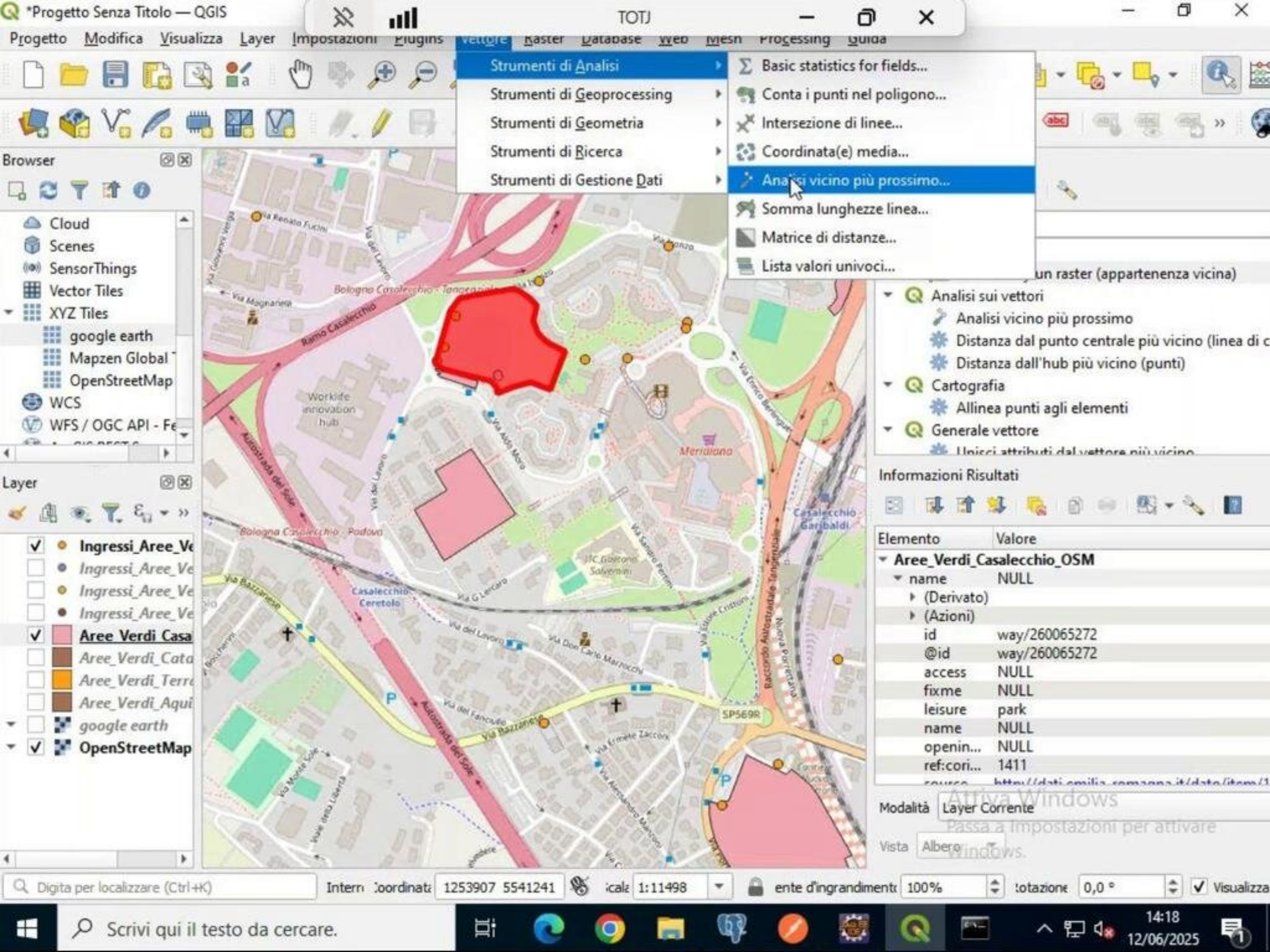
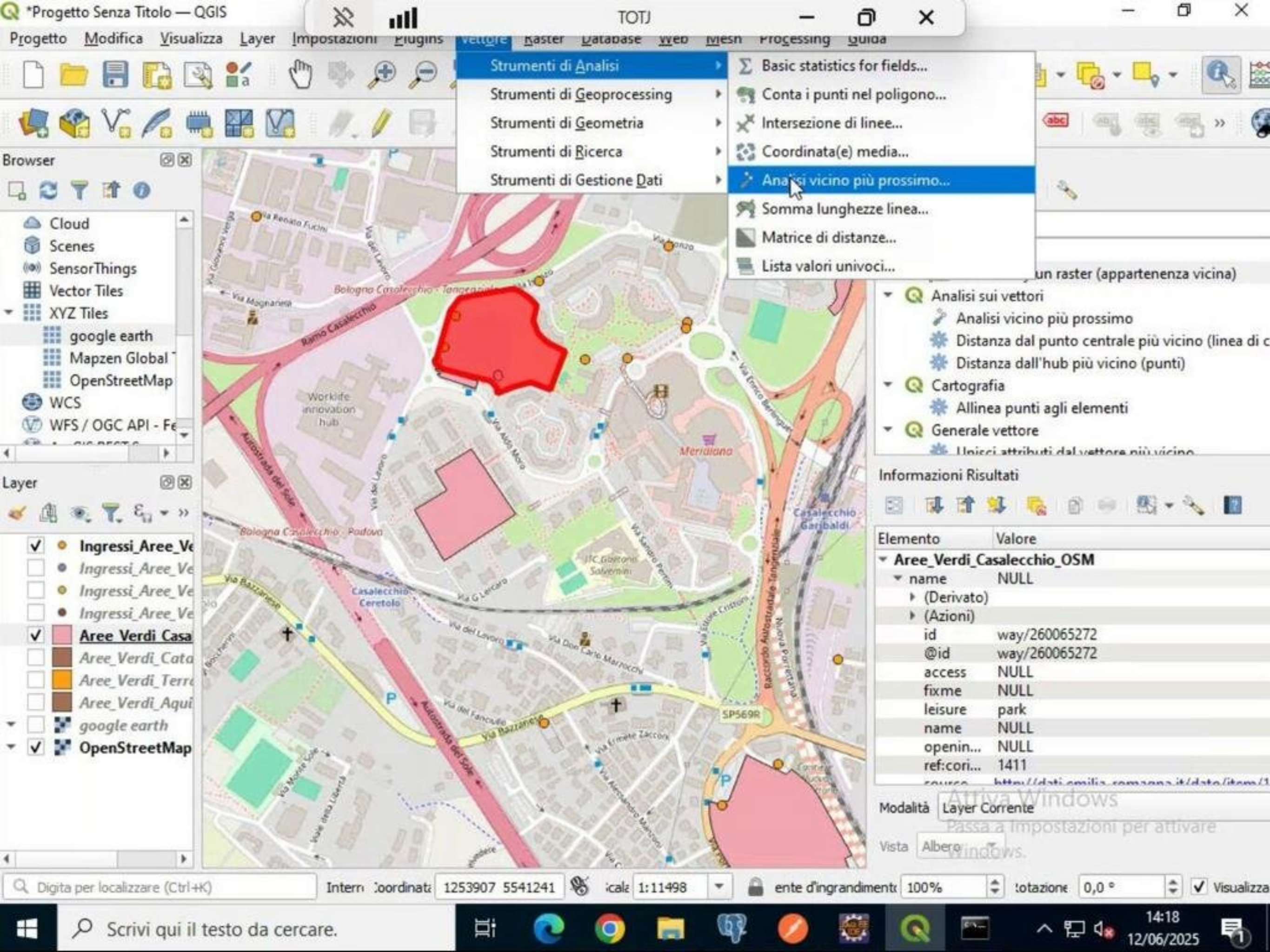
Windows

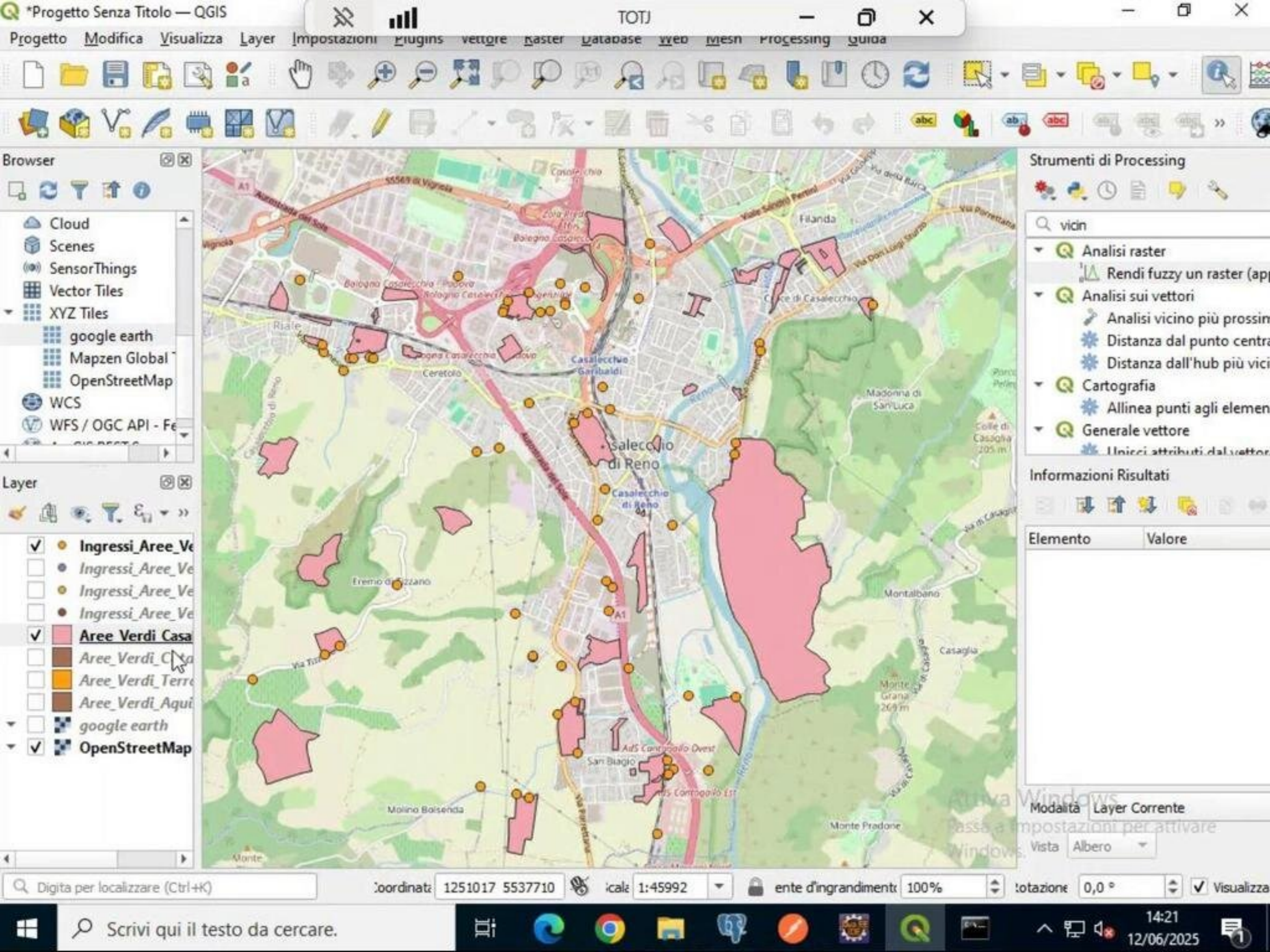
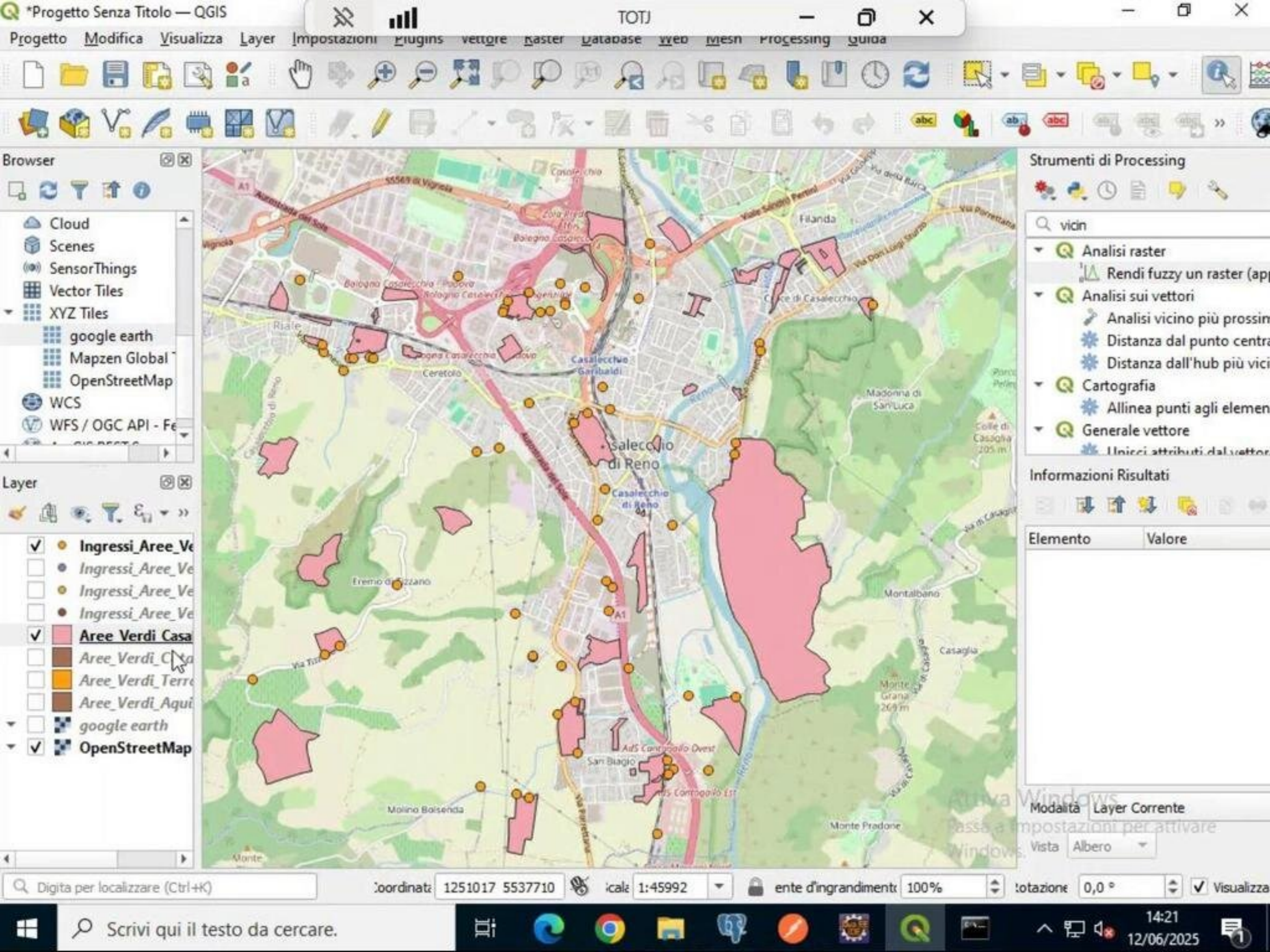
Scrivi qui il testo da cercare.

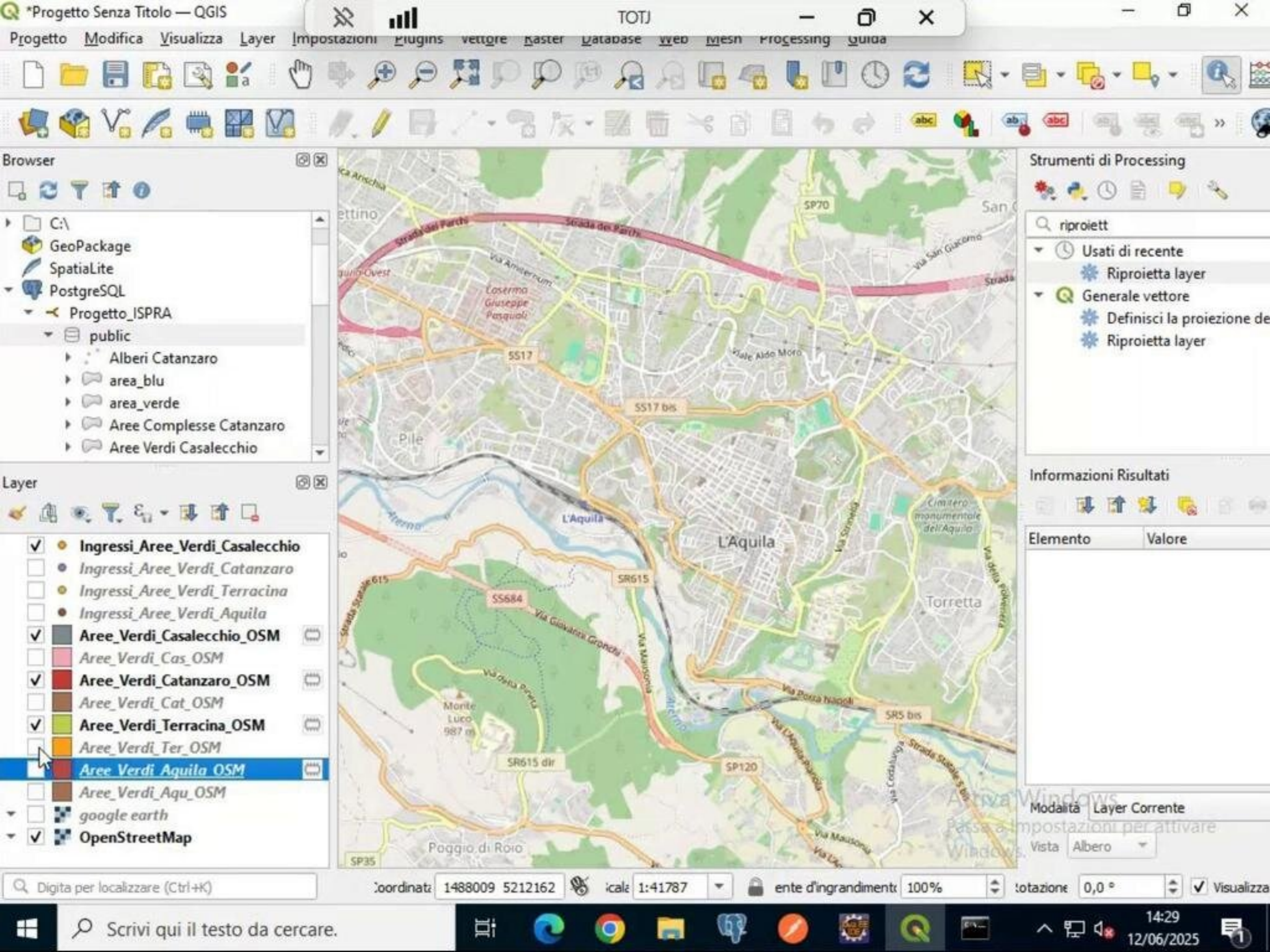
14:14

12/06/2025









Browser



- C:\
- GeoPackage
- Spatialite
- PostgreSQL
 - Progetto_ISPRA
 - public
 - Alberi Catanzaro
 - area_blu
 - area_verde
 - Aree Complesse Catanzaro
 - Aree Verdi Casalecchio

Layer



- ☒ Ingressi_Aree_Verdi_Casalecchio
- ☐ Ingressi_Aree_Verdi_Catanzaro
- ☐ Ingressi_Aree_Verdi_Terracina
- ☐ Ingressi_Aree_Verdi_Aquila
- ☒ Aree_Verdi_Casalecchio_OSM
- ☐ Aree_Verdi_Cas_OSM
- ☒ Aree_Verdi_Catanzaro_OSM
- ☐ Aree_Verdi_Cat_OSM
- ☒ Aree_Verdi_Terracina_OSM
- ☐ Aree_Verdi_Ter_OSM
- ☒ Aree_Verdi_Aquila_OSM
- ☐ Aree_Verdi_Aqu_OSM
- ☐ google earth
- ☒ OpenStreetMap

Strumenti di Processing



- riproiett
- Usati di recente
 - Riproietta layer
 - Generale vettore
 - Definisci la proiezione de
 - Riproietta layer

Informazioni Risultati



Elemento	Valore
----------	--------

Modalità Layer Corrente

Vista Albero

Digita per localizzare (Ctrl+K)

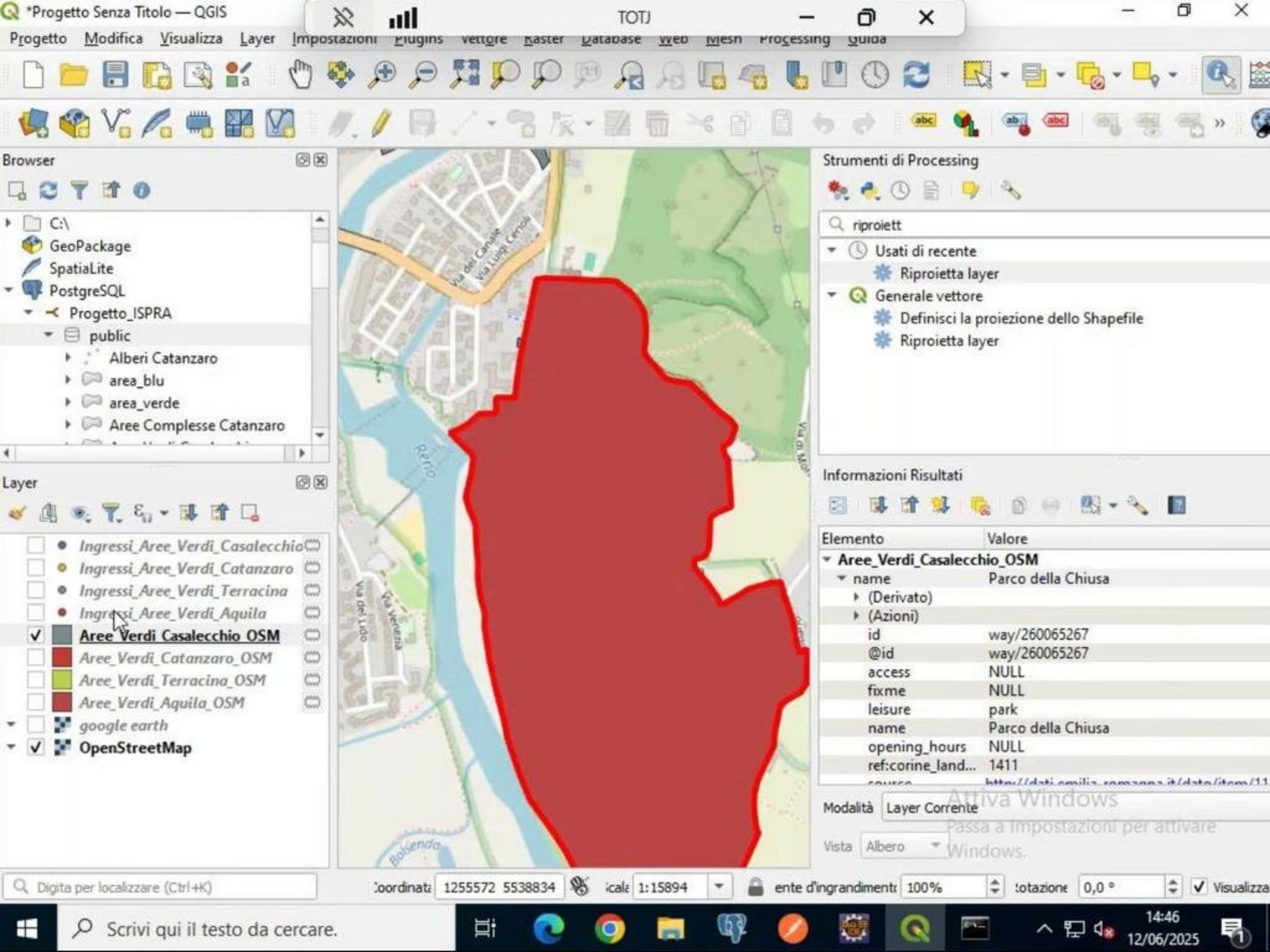
Coordinate 1488009 5212162

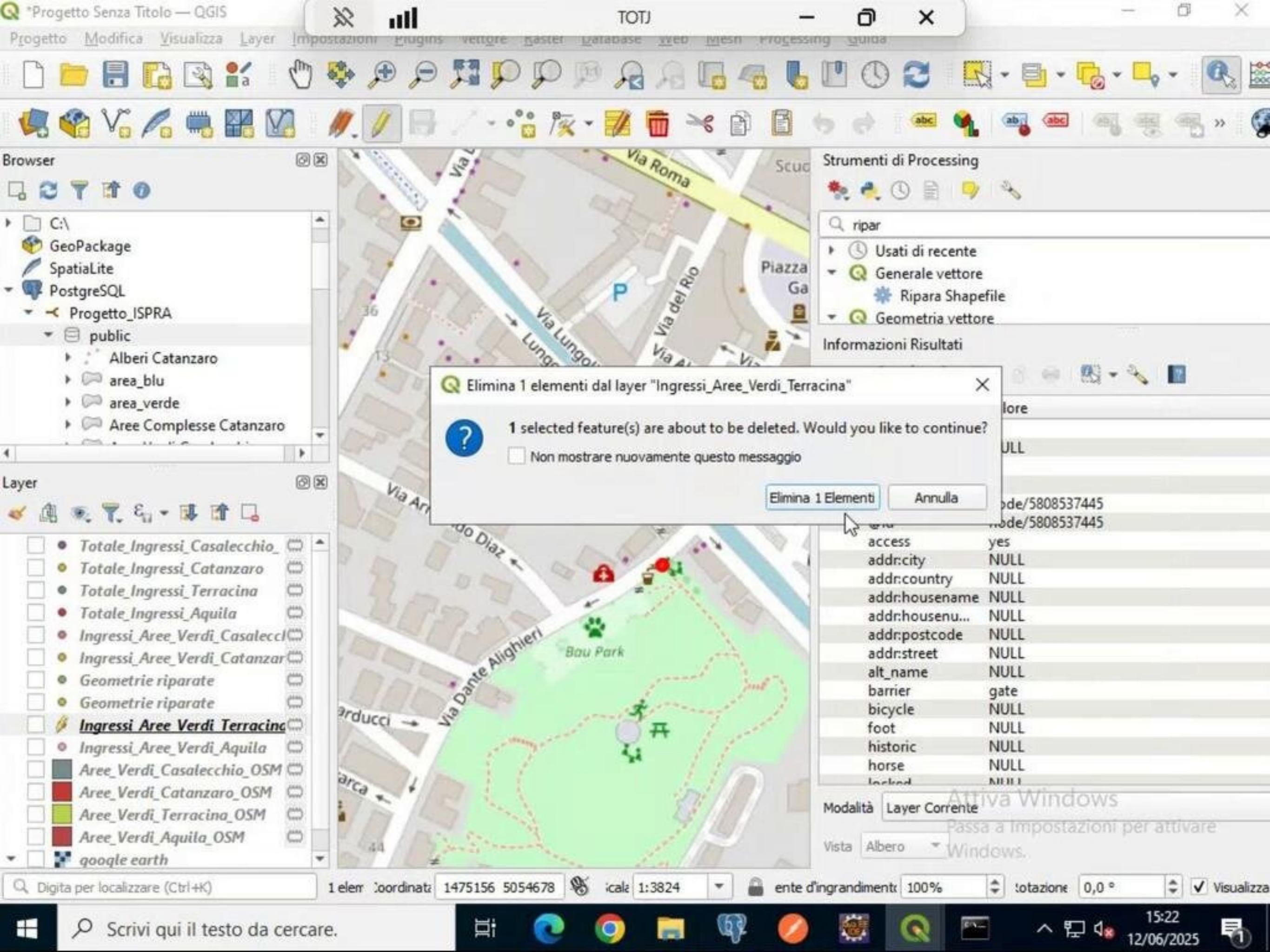
Scale 1:41787

ente d'ingrandimento 100%

Rotazione 0,0°

Visualizza





Continuazione sviluppo delle classi per le API REST e dei relativi servizi

In questa fase del training si proseguirà lo sviluppo dell'architettura a strati del progetto, con particolare attenzione alla costruzione delle API REST e dei corrispondenti servizi di business logic nel backend Java/Spring.

Obiettivi principali:

- Consolidare e completare le operazioni CRUD (Create, Read, Update, Delete) per le principali entità del progetto, in particolare quelle legate alla gestione delle aree verdi, dei poligoni associati e di altri elementi territoriali.
- Sviluppare nuove API REST personalizzate per supportare operazioni specifiche richieste dal flusso applicativo, ad esempio:
 - Inserimento di poligoni selezionati su mappa
 - Aggiornamento di metadati ambientali associati a un'area
 - Recupero filtrato di aree sulla base di criteri spaziali o descrittivi
- Implementare la comunicazione tra controller, service e repository, secondo i principi della separazione delle responsabilità.

Attività tecniche previste:

- Progettazione e stesura di nuovi controller REST, annotati con `@RestController` e `@RequestMapping`.
- Definizione di endpoint REST con annotazioni come `@GetMapping`, `@PostMapping`, `@PutMapping` e `@DeleteMapping`.
- Implementazione dei metodi nei service, che conterranno la logica di validazione e

cronologia modifiche, utenti, ecc.)

2. Modificare dati

Si può contribuire a OSM (da utente registrato) con:

- **iD editor**: editor online facile da usare, direttamente dal sito OSM
- **JOSM**: editor avanzato desktop (Java) per modifiche massive
- **Field Papers** o app mobile (Vespucci, StreetComplete)

3. Scaricare dati

Come ottenere dati OSM:

- In formato **.osm (XML)** tramite Geofabrik
- In formato **GeoJSON, KML** usando [Overpass Turbo](#)
- Con strumenti GIS come QGIS (tramite plugin QuickOSM)

4. Sviluppo software

È possibile integrare OSM in app/web usando:

- **Leaflet.js + Tile Server**: per visualizzare mappe
- **OpenLayers**: libreria GIS frontend potente
- **osmnx (Python)**: per analisi di reti stradali
- **Nominatim API**: per geocoding/reverse geocoding
- **Overpass API**: per query personalizzate sui dati
- **PostGIS**: puoi importare i dati OSM nel tuo DB spaziale per analisi avanzate

Approfondimento dell'uso dell'Overpass API

Overpass API è un'API che consente di interrogare i dati di **OpenStreetMap (OSM)** in modo avanzato e personalizzato.

È progettata per estrarre *solo i dati che servono*, senza dover scaricare l'intero database OSM.

Non è un'API REST "standard" tipo GET /endpoint, ma un motore di query geospaziali su un database OSM.

Funzionalità di Overpass Turbo:

- - Creazione ed esecuzione di query Overpass QL
- - Visualizzazione su mappa degli oggetti ottenuti
- - Esportazione in vari formati (GeoJSON, KML, GPX, ecc.)

Overpass API lavora su una copia aggiornata del database OSM e consente di interrogarlo tramite due linguaggi:

1. Overpass QL (query language compatto)
2. Overpass XML (più verboso, in formato XML)

Concetti base

- Gli oggetti OSM sono: node, way, relation.
- Ogni oggetto ha **tag** del tipo chiave=valore (es: amenity=school).
- Si può filtrare per:
 - **Tag** (es: tutti i parchi)
 - **Area geografica** (box lat/lon)
 - **Tipo di oggetto** (nodo/poligono)
 - **Tempo** (modifiche recenti)

- - Esportazione in vari formati (GeoJSON, KML, GPX, ecc.)

Overpass API lavora su una copia aggiornata del database OSM e consente di interrogarlo tramite due linguaggi:

1. Overpass QL (query language compatto)

2. Overpass XML (più verboso, in formato XML)

Concetti base

- Gli oggetti OSM sono: node, way, relation.
- Ogni oggetto ha **tag** del tipo chiave=valore (es: amenity=school).
- Si può filtrare per:
 - **Tag** (es: tutti i parchi)
 - **Area geografica** (box lat/lon)
 - **Tipo di oggetto** (nodo/poligono)
 - **Tempo** (modifiche recenti)

Esempio di Query in Overpass QL

```
[out:json][timeout:25];  
// seleziona tutti i parchi in un'area o specificando le coordinate dell'area
```



```
//es:" (45.42,10.95,45.45,10.98)"
(
  node["leisure"="park"](45.42,10.95,45.45,10.98);
  way["leisure"="park"](45.42,10.95,45.45,10.98);
  relation["leisure"="park"](45.42,10.95,45.45,10.98);
);

[out:json][timeout:25];
//o passandogli il nome es:"Catanzaro"
area["name"="Catanzaro"]["boundary"="administrative"]["admin_level"="8"]->.catanzaro;
(
  node["leisure"="park"](area.catanzaro);
  way["leisure"="park"](area.catanzaro);
  relation["leisure"="park"](area.catanzaro);
) ->.parchi;

// trova i nodi con barrier=gate o entrance nelle aree verdi (via map_to_area)
.parchi map_to_area -> .aree_parco;

(
  node["barrier"="gate"](area.aree_parco);
  node["barrier"="entrance"](area.aree_parco);
);

out body;
>;
out skel qt;
// oppure out geom;
```

Spiegazione:

- out:json: formato di output (può essere anche XML)
- timeout:25: tempo massimo di esecuzione in secondi

Integrazione delle informazioni disponibili

Nonostante l'obiettivo **sia quello di procedere con un approccio bottom-up partendo da fonti ufficiali**, la sovrapposizione con layer estratti da OpenStreetMap e da Copernicus Urban Atlas Land Cover/Land Use 2018 (UA) diventa utile per integrare il contenuto informativo dei layer resi disponibili dalle Amministrazioni locali.

Inoltre, se i dati dei punti di accesso non fossero disponibili, si potrebbero identificare attraverso l'analisi spaziale nei seguenti modi:

- Considerare solo i nodi estratti da OpenStreetMap con la parola chiave "barrier" ed i tag "gate" e "entrance", localizzati all'interno o sul perimetro dell'area verde considerata
- Considerare l'intersezione tra il layer "strade" e "aree verdi urbane", scaricati da OpenStreetMap.

Laddove le condizioni precedenti non possono essere soddisfatte, una soluzione potrebbe essere rappresentata dalla individuazione di punti ogni 100 metri lungo il perimetro.

Fare clic per inserire le note