UNIVERSITÀ TELEMATICA E-CAMPUS

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e
dell'Automazione

Studio ed implementazione di un'applicazione web per la gestione ed il collaudo di progetti infrastrutturali georeferenziati

Relatore: Prof. Antonino Longo Minnolo

Tesi di Laurea di:
Giordano Cetti
Matricola numero 001392213

"Nonostante il mito di Kevin Mitnick creato dalla stampa, io non sono un hacker malintenzionato"

KEVIN MITNICK (1963 - 2023)

Che la storia di Kevin Mitnick serva da potente promemoria per chiunque. La sua esperienza dimostra come, indipendentemente dalle scelte passate, giuste o sbagliate, esista sempre la possibilità di orientarsi verso una direzione positiva.

La resilienza e la capacità di trasformare le sfide in opportunità sono entrambe la forza motrice che riflette i valori che abbiamo cercato di incorporare in questa ricerca.

Dedicato a Kevin Mitnick, che con la sua storia ci ha insegnato che tutto può, in ogni momento, volgere nella giusta direzione, offrendo una lezione di vita sull'importanza dell'adattabilità e del riscatto personale. Riposa in pace Kevin.

Sommario

Capitolo 1: Specifica del progetto

1.1 Introduzione

StoraGIS è il sistema che studiamo ed implementiamo, con l'obiettivo di permettere agli utenti di caricare, visualizzare e validare visivamente progetti georeferenziati. Questi progetti sono composti da file in formato ESRI Shapefile[1] [1] e da file di tipo CSV[2]. Il sistema si avvale principalmente di tre componenti software:

- Django Rest Framework
- GeoServer
- Openlayers

1.2 Caricamento e visualizzazione dati

L'applicazione permette di caricare un set specifico di dati, raccolti in un unico file compresso e organizzati secondo regole precise, attraverso il browser. Questi dati vengono poi convertiti in formato JSON, sfruttando un servizio di conversione esterno, e poi importati e visualizzati grazie ad un processo automatico di elaborazione, il quale è integrato nel sistema e dettagliatamente documentato.

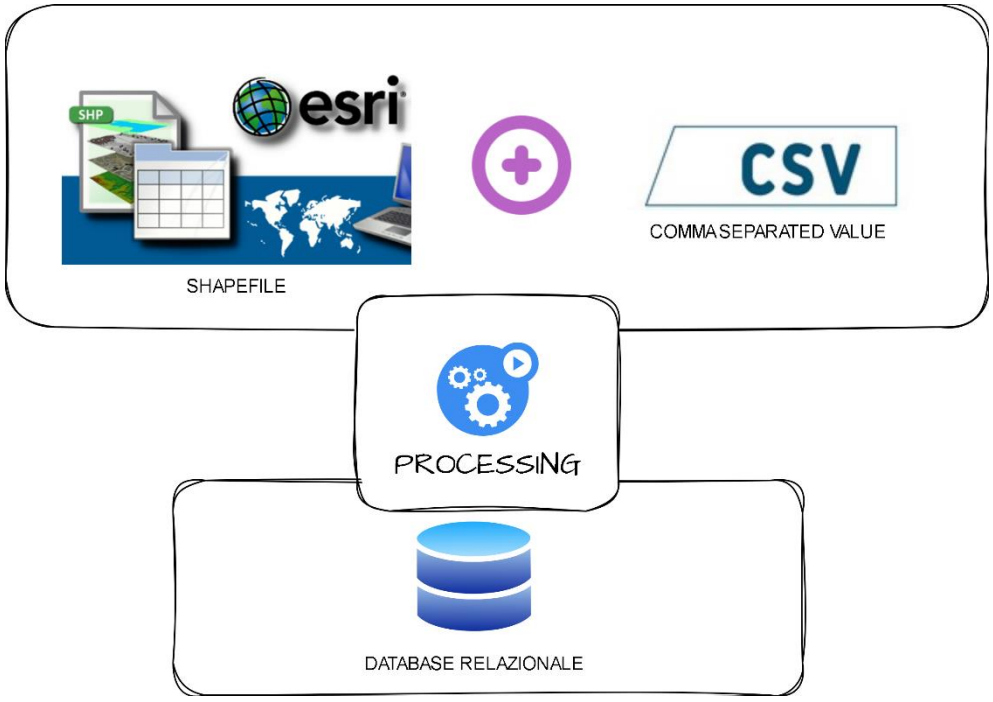
---

1 Uno shapefile è un formato file per memorizzare dati geospaziali in forma vettoriale, rappresentando forme geometriche quali punti, linee e poligoni. Questo formato consente anche di associare informazioni aggiuntive a tali forme, come nomi di luoghi o categorie.

2 Il comma-separated values (CSV), letteralmente "valori separati da virgola", è un formato di file testuale usato per importare ed esportare tabelle di dati, come da fogli elettronici o database. Sebbene non esista uno standard formale che lo definisca, esistono prassi consolidate. Una delle definizioni più accettate si trova nel documento RFC 4180 [2].

1.3 Casi d'uso

L'applicazione supporta principalmente il caso d'uso in cui gli utenti necessitano di scambiare informazioni georeferenziate e di archiviarle in un formato facilmente accessibile ed elaborabile. Questo è particolarmente rilevante per progetti come le reti in fibra ottica, dove le informazioni devono essere salvate in un database relazionale, permettendo così una consultazione ed una gestione efficiente dei dati.



1.4 User Stories

Nello sviluppo software, una User Story descrive informalmente una o più funzionalità di un sistema dalla prospettiva dell'utente finale. Questo approccio facilita i gruppi di sviluppo nella comprensione dei requisiti e delle caratteristiche da implementare [3].

Le User Stories seguenti hanno guidato la creazione di StoraGIS:

Visualizzazione

Come collaudatore di progetti di rete per conto del Cliente, vorrei poter aprire e visualizzare facilmente i progetti ricevuti dall'installatore utilizzando solamente il browser. Questo migliorerebbe significativamente l'efficienza nel processo di verifica e collaudo.

Funzioni principali

In qualità di responsabile del settore ingegneria e data integration per conto del Cliente, desidero un database unico che consolidi tutti i caricamenti effettuati dai collaudatori. Questo database dovrebbe permettere la conservazione dei risultati per l'approvazione o per future consultazioni dello stesso progetto.

Caricamento

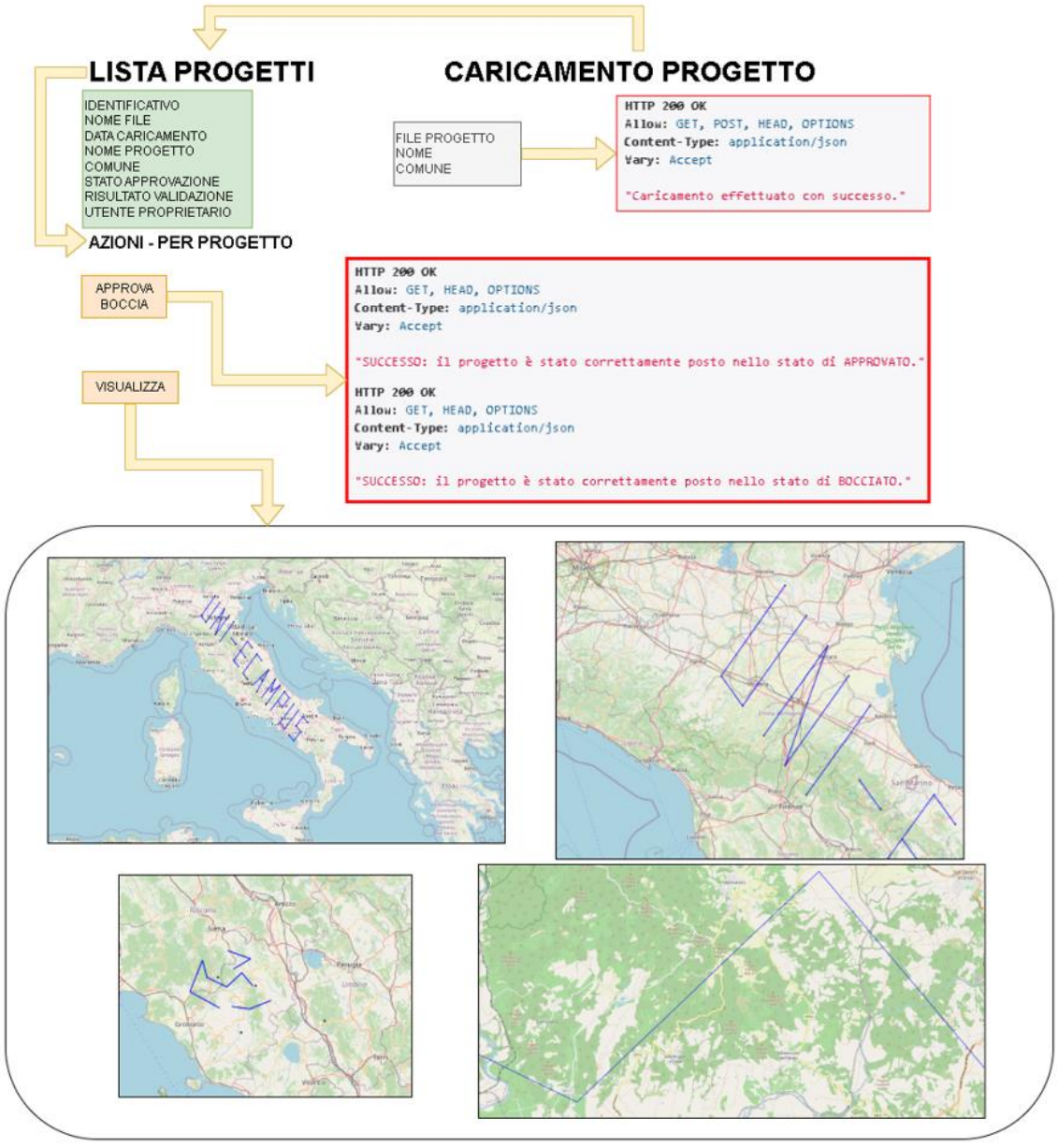
Come utente di StoraGIS, vorrei avere la possibilità di caricare rapidamente pacchetti SHP+CSV forniti al termine delle lavorazioni, con la facilità di aggiungere informazioni aggiuntive come etichette o comune di riferimento. Questa funzionalità sarebbe essenziale per semplificare il processo di gestione dei dati.

Gestione ed evoluzione

Come sistemista per conto del Cliente, è mio desiderio gestire completamente i parametri di configurazione della fase di "Validazione dei dati" prima del loro caricamento definitivo nel sistema. Questo controllo garantirebbe la qualità e l'affidabilità dei dati all'interno del nostro sistema.

1.5 Mockups

I mockups rappresentano una riproduzione in scala di un prodotto, utilizzati per insegnamento, dimostrazione, valutazione e promozione di un progetto. Nella creazione di software, l'impiego più frequente dei mockups consiste nel presentare le interfacce grafiche e le funzionalità agli utenti, evitando la necessità di scrivere codice. Questi possono spaziare da layout estremamente basilari, realizzati a mano, a interfacce utente quasi complete sviluppate attraverso strumenti professionali [4].

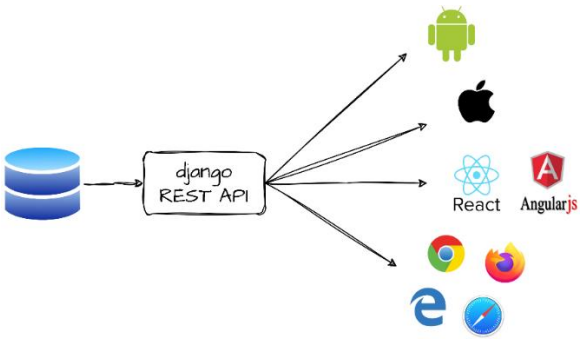


Capitolo 2: Tecnologie e metodologie utilizzate

Per lo sviluppo della ricerca, abbiamo seguito un percorso ben definito che illustreremo in dettaglio. Qui di seguito, una panoramica delle componenti principali che fornisce una visione chiara della struttura del progetto specifico.

2.1 Django Rest Framework

Django REST Framework (DRF) è un toolkit potentemente integrato nel web framework Django, progettato per minimizzare la quantità di codice necessaria per creare interfacce REST [5]. Questo strumento facilita lo sviluppo di API RESTful, ossia metodi per lo scambio di informazioni tra un'interfaccia utente ed un database in maniera efficiente [6].



Il framework distingue nettamente tra l'interfaccia utente e la memorizzazione dei dati, permettendo comunicazioni tra l'utente ed il database attraverso l'invio di file in formato ".json". Grazie alla sua potenza e flessibilità, DRF si afferma come uno strumento essenziale per lo sviluppo di Web API [7].

---

1 Un'API Web RESTful è un'interfaccia di programmazione per applicazioni che facilita l'interazione con servizi web aderenti allo stile architetturale REST [8]. REST, acronimo di REpresentational State Transfer, si basa su principi come l'identificazione delle risorse, la comunicazione senza stato e l'uniformità dell'interfaccia. Le API RESTful impiegano metodi HTTP quali GET, POST, PUT e DELETE per eseguire operazioni di lettura, creazione, aggiornamento e eliminazione di risorse.

2.2 Geoserver

GeoServer [9] è un server web progettato per distribuire mappe e dati geospaziali a client standard, inclusi browser web e software GIS desktop. Questa distribuzione avviene attraverso l'uso di interfacce conformi a standard[1] internazionali, quali WMS (Web Map Service), WFS (Web Feature Service), WCS (Web Coverage Service), WPS (Web Processing Service) e Tile Caching[2].

Service	Acronym	Purpose	Data Type Support	Example Use Cases
Web Mapping Service	WMS	Serve maps as images for visualization	Maps, Layers, Images	Displaying maps with layers and symbology
Web Feature Service	WFS	Serve geospatial features for querying	Vector Features	Retrieving, querying, and editing geographic features
Web Coverage Service	WCS	Serve multi-dimensional data (rasters)	Raster Data	Accessing and analyzing raster data
Web Processing Service	WPS	Execute geospatial processes remotely	Geoprocessing Tasks	Running geospatial analyses and algorithms
Web Map Tile Service	WMTS	Serve pre-rendered map tiles for speed	Maps, Layers	Efficiently displaying maps with cached tiles
Web Coverage Processing Service	WCPS	Execute complex operations on raster data	Raster Data	Advanced analysis and processing of raster coverages

GeoServer è dotato di un'interfaccia di gestione basata su web, che consente una facile configurazione ed amministrazione, supporta inoltre la connessione a diverse fonti di dati, rendendolo uno strumento estremamente versatile per la condivisione di dati geospaziali.

---

1 L'Open Geospatial Consortium (OGC) [10] svolge il ruolo di custode degli standard dei dati geospaziali, garantendo che questi siano liberi e accessibili senza alcun costo. Questa organizzazione promuove l'interoperabilità tra i sistemi che gestiscono informazioni geospaziali attraverso lo sviluppo e l'implementazione di standard aperti.

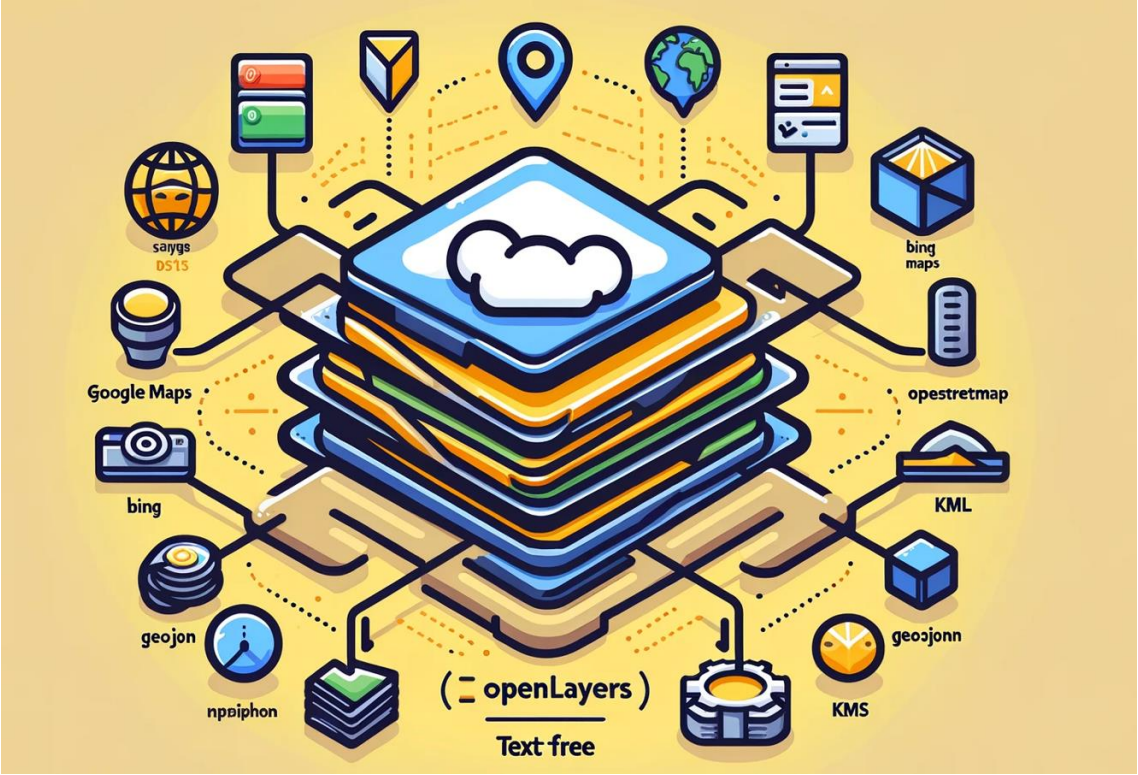
2 I "Web Mapping Services" rappresentano un insieme di protocolli progettati per fornire dati geospaziali a una vasta gamma di destinazioni. Questi servizi permettono agli utenti di richiedere mappe georeferenziate da un server, tramite una richiesta HTTP, che possono essere utilizzate in diverse applicazioni GIS o visualizzate in un browser web [11].

2.3 Openlayers

OpenLayers è una potente libreria JavaScript open-source per la visualizzazione di mappe interattive sul web. È progettato per essere flessibile ed estensibile, permettendo a sviluppatori, ricercatori ed analisti di visualizzare dati geospaziali di vario tipo, dalle mappe base a complessi dataset geografici, in modi personalizzati e interattivi. Openlayers offre tutte le funzionalità necessarie che la maggior parte degli sviluppatori richiede per implementare mappe online[1]

1 Le applicazioni web che integrano Openlayers permettono agli utenti di visualizzare, navigare e interagire con dati geografici. Servizi come Google Maps, ArcGIS Online e OpenStreetMap esemplificano il tipo di funzionalità offerte da queste piattaforme di mappe online.

OpenLayers supporta una vasta gamma di sorgenti di dati cartografici, inclusi quelli provenienti da sistemi di mappe online come Google Maps, Bing Maps, e OpenStreetMap, nonché formati di dati geospaziali standard come GeoJSON, KML, e WMS (Web Map Services). Progettata per essere al contempo semplice, performante e facile da usare [12], Openlayers funziona efficacemente su tutte le piattaforme, includendo dispositivi mobile e desktop. Grazie a queste caratteristiche, rappresenta una delle opzioni più valide per lo sviluppo di applicazioni web e mobile che necessitano di integrazione cartografica[1].



1 Sebbene ci siano numerose alternative ad OpenLayers, Leaflet JS rappresenta la sua principale concorrente. Un confronto dettagliato delle loro funzionalità è disponibile in un articolo su Geoapify: [https://www.geoapify.com/leaflet-vs-openlayers] [13].

Capitolo 3: Analisi concettuale del sistema

L'analisi concettuale del sistema si basa sulla "specifica dei dati"[1]. Nel quadro del progetto attuale, ci siamo prefissati l'obiettivo di gestire una determinata categoria di dati, che presentano una struttura specifica nel suo contesto. A partire da questa struttura, definiremo un Entity-Relationship Model[2] per la nostra applicazione.

Il processo prevede che lo scambio di dati tra l'entità responsabile dell'installazione delle infrastrutture di rete sul territorio, denominata "fornitore", e l'organismo incaricato di raccogliere e approvare i progetti, denominato "committente", avvenga attraverso il trasferimento di un "pacchetto .zip" per ciascun progetto. Ogni pacchetto verrà poi analizzato e, se ritenuto conforme, approvato.

1 La "specifica dei dati" stabilisce il formato dei dati forniti. Nel paragrafo successivo, verrà dettagliata la struttura dei dati che sono oggetto della ricerca, offrendo una chiara comprensione delle aspettative e dei requisiti per la loro gestione e analisi.

2 Un Modello Entity-Relationship (ER) di base include tipi di entità, che categorizzano gli oggetti di interesse, e definisce le relazioni possibili tra queste entità, ovvero le istanze dei tipi di entità menzionati. Questo modello fornisce una rappresentazione schematica delle informazioni gestite all'interno del sistema [14].

3.1 Specifica dei dati: composizione del “formato/specifica CT1”

Il "formato CT1", conosciuto anche come "specifica CT1", è un formato dati creato ad hoc per scopi accademici, con l'idea di poter essere sostituito in futuro da una diversa specifica in base ai casi d'uso e all'analisi concettuale di potenziali implementazioni in contesti professionali.

Come già menzionato nell'introduzione, uno dei modi per arrivare al formato JSON StoraGIS, è quello di partire da una serie di SHAPEFILE ed una serie di CSV. La struttura di ogni SHAPEFILE [15], a differenza dei file CSV (che sono autoconsistenti individualmente), si compone di più file con diverse estensioni, così definiti:

- File principale (.shp) – mandatorio
    - **SHP** contiene le informazioni sui dati vettoriali spaziali dello shapefile.
- File indice (.shx) – mandatorio
    - **SHX** archivia le informazioni relative all'indice posizionale, utilizzato per facilitare la ricerca dei dati all'interno del file.
- File dBase (.dbf) – mandatorio
    - **DBF** è un file di database standard che memorizza i dati attributivi e gli ID degli oggetti.
- File Projection (.prj) – opzionale[1]
    - **PRJ** generalmente non obbligatorio, contiene i metadati relativi al sistema di coordinate e di proiezione utilizzato.

1 Anche se il file PRJ è considerato opzionale secondo le specifiche ESRI, per le esigenze di questo progetto, verrà trattato come un elemento mandatorio, al pari degli altri file componenti dello shapefile.

Il tracciato completo dei file indispensabili per definire la coerenza di un progetto che possa esser convertito in formato StoraGIS include:

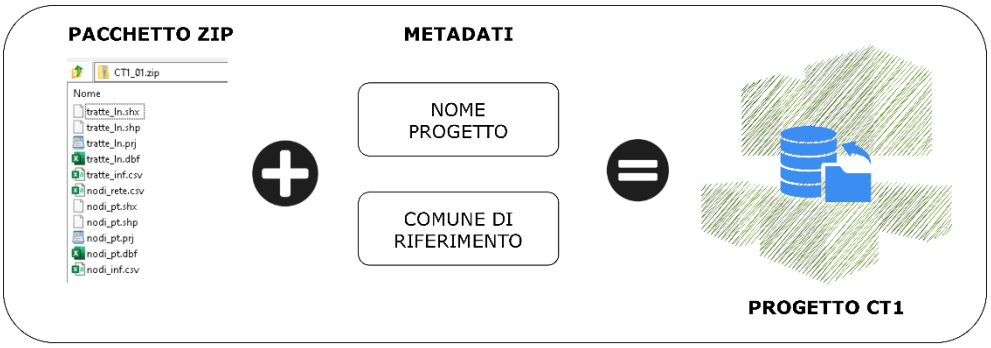
- nodi_pt.shp
  + nodi_pt.shx
  + nodi_pt.dbf
  + nodi_pt.prj
- nodi_inf.csv
- nodi_rete.csv
- tratte_ln.shp
  + tratte_ln.shx
  + tratte_ln.dbf
  + tratte_ln.prj
- tratte_inf.csv

Sia i file di tipo CSV sia i file di tipo SHP possono essere semplificati in una struttura tabellare[1]. Questi conterranno campi e attributi specificamente definiti per il progetto in questione:

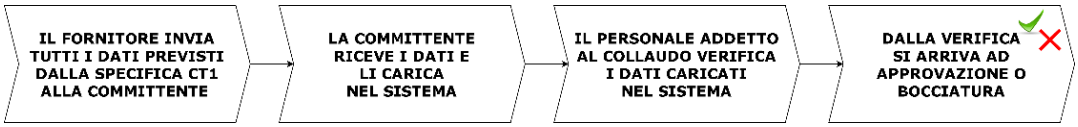
nome file	titolo campo	tipologia dato contenuto	mandatorio
nodi_pt.shp	idnodopt	intero a 8 bit	SI
	geom	geometrico	SI
nodi_inf.csv	idnodoinf	intero a 8 bit	SI
	idfeature	intero a 8 bit	SI
	etichetta	varchar 255	NO
	geom	geometrico	SI
nodi_rete.csv	idnodorete	intero a 8 bit	SI
	idfeature	intero a 8 bit	SI
	etichetta	varchar 255	NO
	geom	geometrico	SI
tratte_ln.shp	idtrattaln	intero a 8 bit	SI
	geom	geometrico	SI
tratte_inf.csv	idtrattainf	intero a 8 bit	SI
	idfeature	intero a 8 bit	SI
	idmateriale	intero a 4 bit	SI

1 Come indicato nel paragrafo 3.1, sebbene i file di tipo SHP siano compositi e contengano dati binari relativi agli elementi geospaziali vettoriali, è possibile rappresentare questi dati come una colonna all'interno di una tabella. Questo grazie alle funzionalità estese per il trattamento dei dati geospaziali offerte dai database Postgresql [16].

I metadati[1] fondamentali per l'identificazione del progetto includono: {nome progetto, comune di riferimento}. Queste informazioni vengono fornite dal "fornitore" al "committente" in formato testuale, tramite posta elettronica. In particolare, il "nome progetto" serve come identificativo univoco per ciascun incarico, mentre il "comune di riferimento" corrisponde ad un numero intero che fa riferimento all'ID univoco del comune (codice ISTAT[2]).



Il personale incaricato del collaudo dei progetti da parte del "committente" si occuperà quindi di inserire queste informazioni direttamente durante la fase di caricamento dei dati su StoraGIS.



Al termine della procedura di verifica, sarà compito del collaudatore comunicare al "fornitore" l'esito della verifica, chiudendo così il ciclo di revisione del progetto.

---

1 I metadati sono informazioni che descrivono caratteristiche specifiche di altri dati, quali il formato, il nome, l'autore, il contenuto o la posizione di un file, fornendo un contesto aggiuntivo e facilitando la gestione e l'interpretazione dei dati stessi [17].

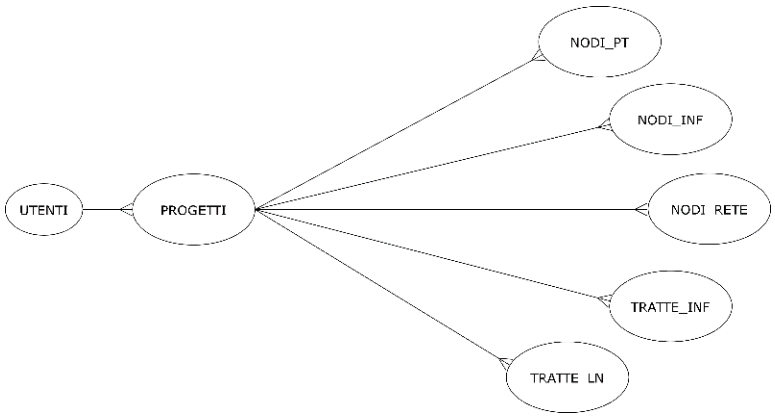
2 Il codice ISTAT è un identificativo unico assegnato dall'Istituto Nazionale di Statistica a ogni comune italiano, utilizzato per garantire una precisa identificazione geografica all'interno delle banche dati e nelle analisi statistiche [18].

3.2 EntityRelationship Model

Dopo aver definito in dettaglio la struttura del dato nel formato "CT1", possiamo procedere all'analisi delle relazioni tra le entità. Questo passaggio, come menzionato all'inizio del capitolo, è cruciale per avanzare verso la fase di realizzazione del sistema. Effettuando l'analisi ER, poniamo le basi per l'implementazione del modello[1] nel framework Django REST Framework (DRF).

Le prime osservazioni per una panoramica del modello ER includono:

- Ogni tabella di ogni file all'interno della specifica CT1 è correlata al progetto stesso, configurando una relazione di tipo 1-a-molti {1 progetto molti nodi_pt, nodi_inf, ecc. }
- Ogni utente può caricare molti progetti, ma ogni progetto può esser caricato da un singolo utente {1 **progetto** 1 **utente**, 1 **utente** molti **progetti**}.

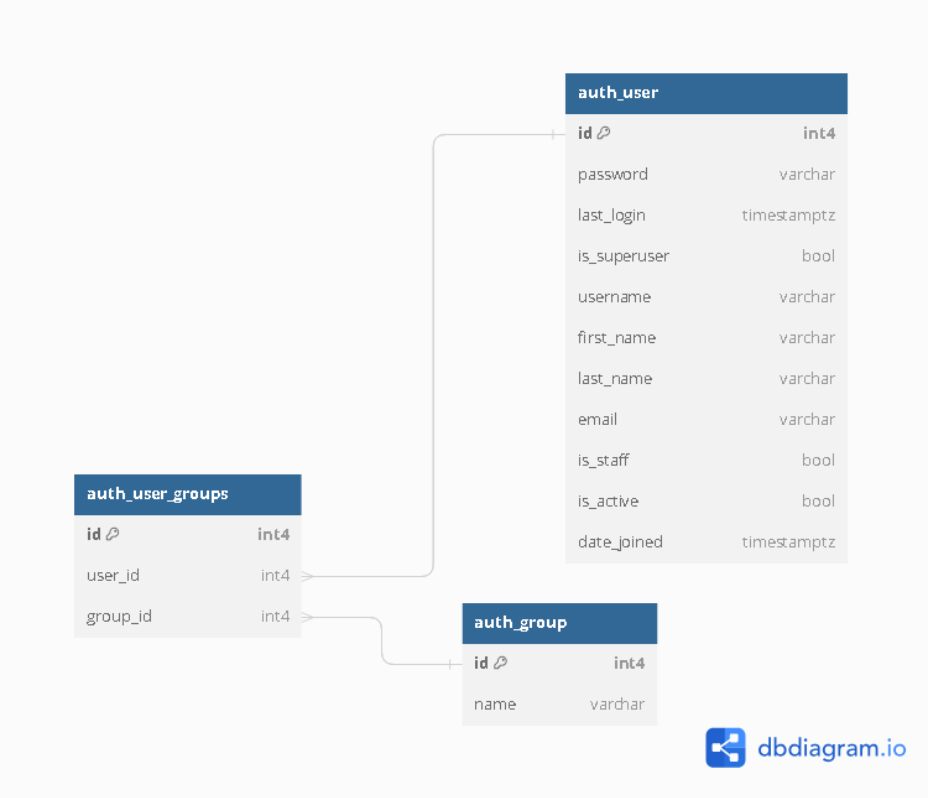


Le relazioni dirette tra le tabelle (file) del progetto saranno dettagliate nelle sezioni successive.

---

1 Un modello Django rappresenta la struttura dei dati all'interno dell'ambiente Django, fungendo da ponte tra il codice Python e il database. Questo modello, che è essenzialmente una classe Python, consente la definizione di tabelle, campi e vincoli nel database in maniera diretta e intuitiva. Ogni attributo di un modello corrisponde a un campo nel database, facilitando così la gestione dei dati [19].

Il modello ER dedicato alla fase di autenticazione si articola come segue, corrispondendo esattamente al modello di autenticazione standard [20] implementato in DRF.

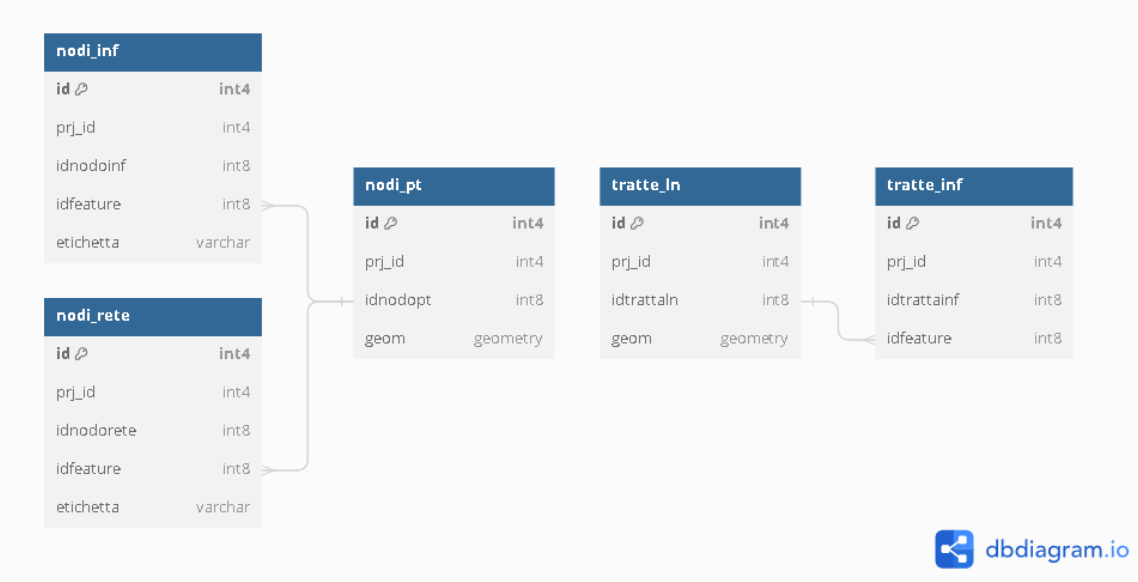


La tabella auth_user rappresenta l'entità UTENTI nel contesto macroscopico del diagramma ER.

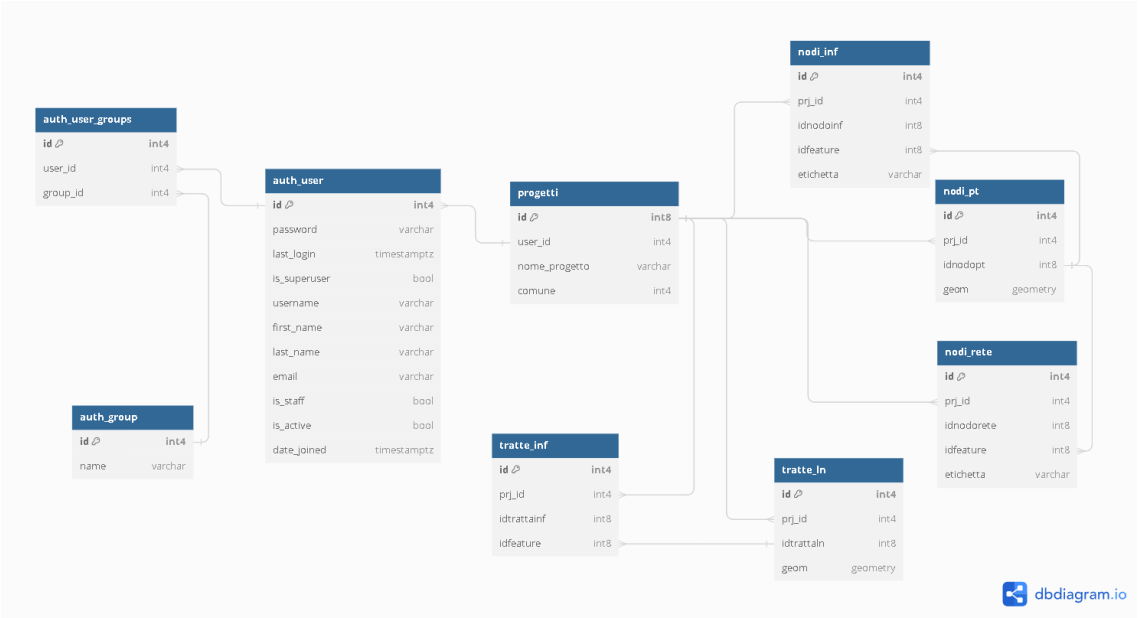
La tabella auth_user_groups, funge da tabella intermedia[1], anche nota come tabella di chiavi esterne, essenziale per trasformare una relazione molti-a-molti in un insieme di relazioni 1-a-molti. Questo permette di rappresentare le dinamiche in cui un utente può appartenere a più gruppi e viceversa.

1 Una tabella intermedia è utilizzata per implementare relazioni molti-a-molti tra due o più tabelle in un database. La presenza di questa tabella permette di gestire efficacemente le associazioni complesse tra le entità, rendendo possibile collegare ciascun elemento di una tabella con molti elementi di un'altra e viceversa [21].

Per quanto riguarda il dato specifico della "CT1", il modello ER si presenta come segue:



Qui, le due entità principali, nodi_pt e tratte_ln, sono collegate rispettivamente a {nodi_inf + nodi_rete} e a {tratte_inf} mediante la chiave idfeature. Il diagramma ER completo del sistema StoraGIS illustrerà quindi in modo esaustivo queste relazioni:



Capitolo 4: progettazione del sistema

4.1 Architettura tecnica

L'architettura tecnica del sistema che stiamo sviluppando si basa principalmente su una Virtual Machine, la quale opera sotto il sistema operativo Linux (Ubuntu 20.04). Su questa piattaforma Linux, installeremo le componenti chiave necessarie per il funzionamento del nostro sistema, includendo:

- PostgreSQL Database relazionale
- Supervisord[1] Orchestratore di processi
- DRF Framework
- Openlayers Libreria Javascript
- Nginx/Gunicorn WebServer[2]
- GeoServer WMS

Supervisord avrà il compito di avviare e mantenere attivo il processo DRF, che verrà eseguito come "socket unix"[3] tramite Gunicorn. Questo socket unix verrà poi convertito in un "socket di rete"[4] attraverso le funzionalità di web server integrate in Nginx.

---

1 Supervisord è un sistema client/server utilizzato per monitorare e controllare vari processi su sistemi operativi UNIX-like. Questo strumento si rivela fondamentale per gestire il ciclo di vita dei processi legati all'applicazione, assicurando che rimangano attivi e si riavviino automaticamente in caso di fallimento [22].

2 Un web server è una piattaforma che archivia e distribuisce contenuti web a un browser client, utilizzando il protocollo HTTP (Hypertext Transfer Protocol) per facilitare questa comunicazione. Questo sistema è essenziale per rendere accessibili i contenuti web agli utenti attraverso Internet [23].

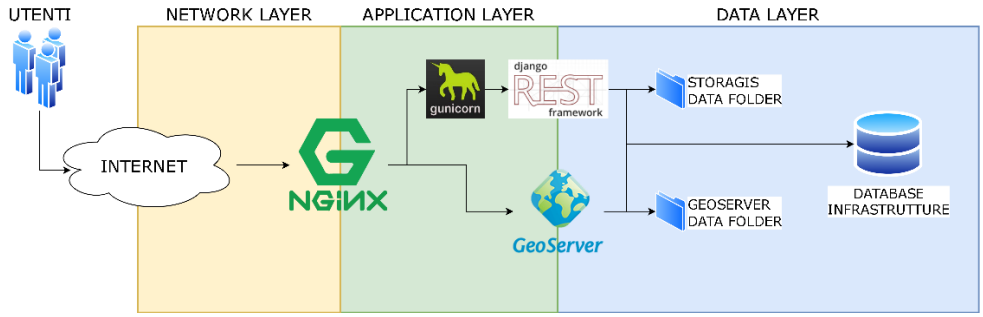
3 Un socket UNIX, noto anche come UDS (Unix Domain Socket), facilita la comunicazione inter-processo su un'unica macchina host. Questo tipo di socket è ottimizzato per la trasmissione di dati tra processi sullo stesso sistema, offrendo un metodo efficiente per la comunicazione locale [24].

4 Un socket di rete, o Socket IP, permette la comunicazione tra processi su macchine diverse attraverso la rete Internet. A differenza degli UDS, i socket di rete espandono le capacità di comunicazione dell'applicazione al di là dei confini della macchina locale, consentendo di gestire le richieste degli utenti finali da remoto.

Il framework DRF sarà principalmente impegnato nell'esecuzione di query SQL verso il database PostgreSQL e nella memorizzazione dei dati di lavoro in formato binario[1] all'interno di una "working directory" dedicata. Inoltre, DRF fornirà al client una pagina web HTML che incorpora le librerie Openlayers, consentendo così la connessione al GeoServer e la visualizzazione delle mappe con le infrastrutture precedentemente caricate.

4.2 Architettura logica

L'architettura logica del sistema StoraGIS può essere sintetizzata attraverso il diagramma funzionale seguente.



GeoServer e Django REST Framework (DRF) avranno accesso ad un database (DB) condiviso, oltre che ad una cartella di lavoro (working folder) locale. Quest'ultima sarà utilizzata per la memorizzazione di dati operativi, quali la cache applicativa[2], i file originali caricati dagli utenti[3], configurazioni del sistema e altri elementi che saranno esaminati più dettagliatamente nei capitoli successivi.

---

1 I file binari riducono lo spazio di memorizzazione e semplificano le operazioni, ma non sono facilmente editabili o portabili [25].

2 La cache applicativa accelera l'accesso ai dati frequentemente utilizzati, attraverso soluzioni come GeoWebCache [26] e cache personalizzate per StoraGIS.

3 I dati caricati dagli utenti su StoraGIS sono le informazioni inviate per l'elaborazione post-autenticazione.

4.3 API Endpoints

Per raggiungere l'obiettivo delineato nell'analisi concettuale del sistema, implementeremo i seguenti endpoints[1] essenziali per l'interazione utente:

- Caricamento: Permette agli utenti di caricare dati georeferenziati nel sistema
- Visualizzazione: Consente agli utenti di visualizzare i dati caricati
- Approvazione: Abilita i revisori ad esaminare e ad approvare i dati caricati

4.3.1 Definizione Caricamento

Nella progettazione del flusso di caricamento, è fondamentale considerare quali informazioni salvare per ciascuna operazione di caricamento effettuata dall'utente. L'interfaccia utente deve essere progettata per raccogliere in modo chiaro e organizzato i seguenti dati associati a ogni caricamento:

- UTENTE PROPRIETARIO
- DATA CARICAMENTO
- PACCHETTO DATI
- STATO CARICAMENTO
- NOME PROGETTO E COMUNE DI RIFERIMENTO

Questi parametri sono cruciali per tracciare e gestire efficacemente i dati all'interno del sistema StoraGIS, assicurando che l'interazione con l'utente sia gestita in modo efficiente e conforme alle aspettative del progetto.

---

1 Nel contesto delle API, un endpoint si riferisce specificamente a un URL (Uniform Resource Locator) che consente a dispositivi o software di connettersi e scambiare informazioni con un servizio web, agendo come un'interfaccia per le operazioni definite dall'API.

4.3.2 Definizione Visualizzazione

Il flusso di visualizzazione differisce significativamente da quello di caricamento poiché si concentra sulla capacità dell'utente di accedere e interagire solo con i propri dati caricati. La progettazione di questo flusso richiede la definizione di due principali aree di interazione:

- VISUALIZZAZIONE DI LISTA
    - Gli utenti devono poter visualizzare una lista ordinata dei propri caricamenti. Questa lista serve come punto di partenza per l'accesso a informazioni più dettagliate su ogni progetto caricato
- VISUALIZZAZIONE DI PROGETTO
    - Una volta selezionato un elemento dalla lista, l'utente deve poter accedere ad una visualizzazione su mappa del progetto selezionato

4.3.3 Definizione Approvazione

Il flusso di approvazione può essere concepito come una variazione del flusso di visualizzazione di progetto, con l'aggiunta cruciale di un'azione che permette di modificare lo stato di approvazione del progetto. Questo stato è rappresentato da un campo booleano che può anche assumere un valore nullo, riflettendo così lo stato iniziale del progetto.

- NULL – Indica che il progetto è in attesa di verifica.
- TRUE – Denota che il progetto è stato approvato.
- FALSE – Denota che il progetto è stato bocciato.

I parametri essenziali per questo processo includono l'ID del progetto ed il valore di approvazione, consentendo così la gestione dello stato di ciascun progetto all'interno del sistema StoraGIS.

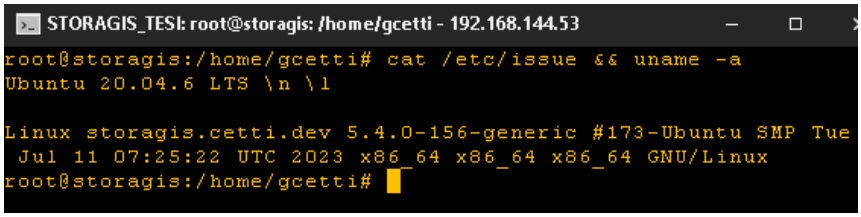
Questi flussi di visualizzazione e approvazione sono fondamentali per l'interazione degli utenti con il sistema StoraGIS, offrendo un'interfaccia chiara e funzionale per la gestione dei progetti georeferenziati.

Capitolo 5: Realizzazione del sistema

La fase di realizzazione del sistema StoraGIS viene documentata dettagliatamente per facilitare la riproduzione del procedimento adottato.

5.01 Contesto di laboratorio

La realizzazione avviene su un ambiente di virtualizzazione[1] di laboratorio, preconfigurato per offrire connettività IPv4, una macchina virtuale con sistema operativo aggiornato, accesso SSH[2] e credenziali amministrative, garantendo così un contesto ottimale per lo sviluppo e il testing.



5.02 Caratteristiche hardware

La macchina virtuale destinata ai test è configurata con le seguenti specifiche:

- 2 vCPU[3]
- 8 GB RAM
- 50 GB STORAGE

1 La virtualizzazione utilizza il software per simulare le funzionalità hardware e creare un'istanza di sistema virtuale [27].

2 Il protocollo SSH consente di stabilire una connessione sicura tra due computer [28].

3 Una CPU virtuale (vCPU) è una parte di una macchina virtuale (VM) che simula le funzioni di una CPU fisica [29].

5.03 Installazione pacchetti di sistema e preparazione

Per proseguire con la realizzazione del sistema StoraGIS, come descritto nei capitoli precedenti, la fase successiva riguarda l'installazione delle componenti software essenziali. Queste componenti costituiscono la base tecnologica su cui si fonderà l'intero sistema, permettendo il funzionamento delle funzionalità previste e l'interazione con gli utenti. Di seguito, è elencato il software necessario da installare sulla macchina virtuale preconfigurata:

- postgresql-12
- postgresql-12-postgis-3
- postgresql-client-12
- python3-distutils
- python3-venv
- python3-pip
- libpq-dev
- nginx
- supervisor
- default-jdk
- unzip
- libcurl4-openssl-dev
- libssl-dev libpq-dev

Per eseguire l'installazione dei pacchetti necessari e delle dipendenze, è necessario eseguire il seguente comando [30] nella shell di amministrazione:

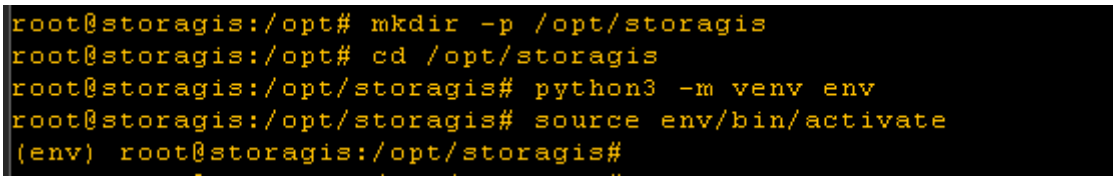
```
apt install -y postgresql-12 postgresql-12-postgis-3 postgresql-client-12
python3-distutils python3-venv python3-pip nginx default-jdk unzip libcurl4-
openssl-dev libssl-dev libpq-dev supervisor
```

5.04 Python Virtual Environment

Per semplificare la gestione delle dipendenze applicative, si raccomanda di utilizzare un "ambiente virtuale"[1] all'interno del server. In questo modo, i pacchetti Python, noti anche come wheels[2], vengono installati esclusivamente nella directory applicativa. Per creare e attivare l'ambiente virtuale, è necessario eseguire i seguenti comandi nella shell di amministrazione:

```
mkdir -p /opt/storagis
cd /opt/storagis
python3 -m venv env
source env/bin/activate
```

Una volta attivato l'ambiente virtuale, è possibile procedere con l'installazione delle wheels necessarie al progetto.



1 Uno strumento per la creazione di ambienti Python virtuali isolati, che consente una gestione separata delle dipendenze per progetto, riducendo i conflitti tra pacchetti e facilitando la configurazione del progetto [31].

2 Un formato di distribuzione dei pacchetti Python che semplifica il processo di installazione, garantendo installazioni più rapide e una maggiore stabilità durante la distribuzione del software [32].

5.05 Python DRF – Dispiegamento iniziale

La prima e fondamentale wheel da installare è quella del Django Rest Framework. Assicurandosi che l'ambiente virtuale sia attivo[1], si procede con l'inserimento del comando di installazione:

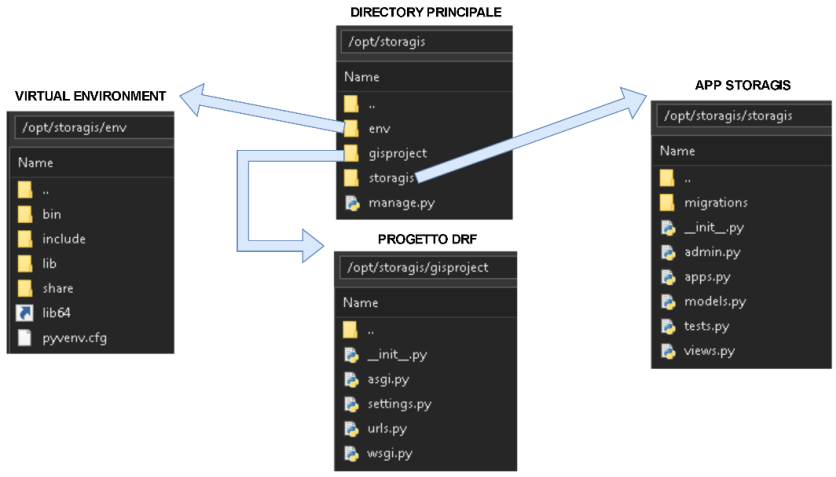
```
python3 -m pip install wheel
python3 -m pip install djangorestframework==3.13.1 Django==3.2.12 gunicorn
pycurl psycopg2
```

```
(storagis) root@storagis:/opt/storagis# python3 -m pip install djangorestfram
Collecting djangorestframework==3.13.1
  Downloading djangorestframework-3.13.1-py3-none-any.whl (958 kB)
     |                                | 958 kB 8.5 MB/s
```

Dopo aver installato con successo il framework, si avvia l'inizializzazione di un "django-project"[2] dedicato e la creazione dell'app principale con i comandi specifici:

```
django-admin startproject gisproject .
django-admin startapp storagis
```

Al termine di questo processo, si otterrà una configurazione pronta per lo sviluppo ulteriore dell'applicazione, composta come di seguito:



1 Comando per attivare l'ambiente virtuale: cd /opt/storagis && source env/bin/activate [31].

2 Procedura ufficiale per l'avvio di una web app Django, come descritto nella documentazione ufficiale [33].

5.06 Inizializzazione database Postgresql

Sulla macchina virtuale dedicata allo sviluppo del progetto StoraGIS, abbiamo installato i pacchetti necessari per il funzionamento di PostgreSQL. Tuttavia, al momento non abbiamo ancora effettuato le modifiche richieste alla configurazione iniziale del database.

Questo paragrafo si prefigge l'obiettivo di:

- creare una "**utenza** SQL"[1] dedicata all'applicazione
- creare una "**istanza** database" SQL dedicata all'applicazione
- assicurare che l'utenza applicativa disponga di tutti i permessi necessari per operare sul database applicativo

Per raggiungere questi obiettivi, sarà sufficiente eseguire i seguenti comandi:

```
su - postgres -c psql #ACCESSO AL CLIENT RDBMS
create user storagisusr with superuser password 'password';
create database storagisdb owner storagisusr;
\q #PER USCIRE DAL CLIENT PSQL
```

Dopo aver completato questi passaggi, l'utente storagisusr diventerà il proprietario dell'intera istanza database storagisdb. Di conseguenza, questo utente potrà operare senza restrizioni su tutti i dati creati o modificati all'interno del database, e ciò avverrà in modo esclusivo per questa destinazione.

È fondamentale salvare la password specificata durante l'esecuzione del comando. Questo parametro sarà necessario nei paragrafi successivi per configurare correttamente il Django Rest Framework (DRF).

---

1 In ambito di gestione dei database, un'utenza è un profilo utente che consente l'identificazione e l'autenticazione per l'accesso al database. Questo meccanismo garantisce che solo gli utenti autorizzati possano interagire con i dati e le risorse del database.

5.07 Python DRF – Inizializzazione modello dati e connessione DB

L'installazione predefinita del modulo DRF provvederà ad istanziare automaticamente un database di tipo SQLite[1] su filesystem locale. Con l'obiettivo di semplificare la gestione del DB, si procederà all'inizializzazione dei dati di origine specificandone come destinazione il database Postgresql precedentemente istanziato.

Per fare ciò, è necessario rimuovere i parametri predefiniti nella sezione DATABASES del file di configurazione settings.py, situato in /opt/storagis/gisproject/settings.py:

```
73 # Database
74 # https://docs.djangoproject.com/en/4.2/ref/settings/#databases
75
76 DATABASES = {
77     'default': {
78         'ENGINE': 'django.db.backends.sqlite3',
79         'NAME': BASE_DIR / 'db.sqlite3',
80     }
81 }
```

I valori da modificare, specificati nel paragrafo 5.06, includeranno dettagli come il nome del database, l'utente, la password e altre configurazioni essenziali per la connessione a PostgreSQL:

```
DATABASES = {
   'default': {
      'ENGINE': 'django.contrib.gis.db.backends.postgis',
      'NAME': 'storagisdb',
      'USER': 'storagisusr',
      'PASSWORD': 'password',
      'HOST': 'localhost',
      'PORT': '5432',
   }
}

STATIC_ROOT = '/opt/storagis/static'
```

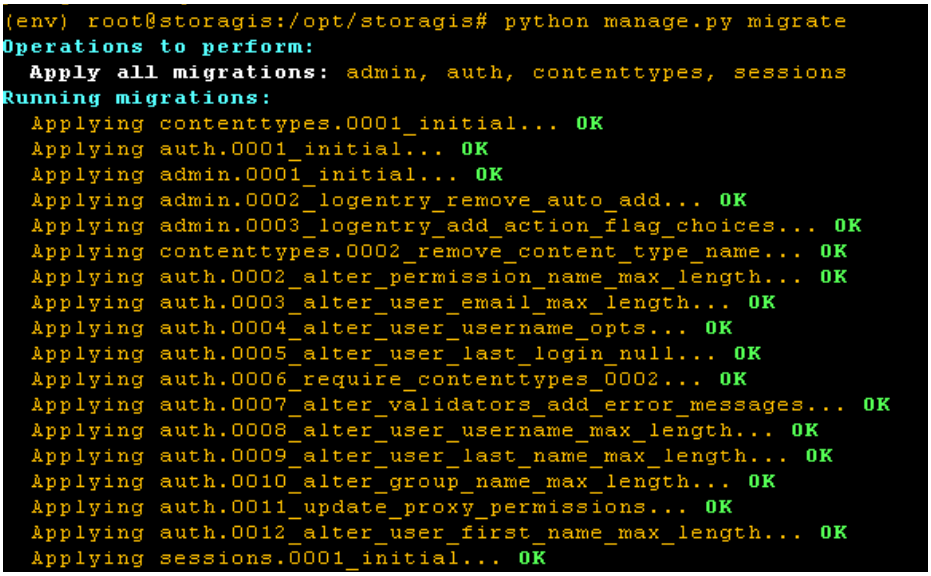
1 Un sistema di gestione di database SQL leggero, che non richiede un server e si basa su un file singolo. Questa soluzione è apprezzata per la sua affidabilità, semplicità di utilizzo, codice aperto e supporto a lungo termine [34].

È importante anche definire il parametro STATIC_ROOT nel file settings.py. Dopo aver apportato queste modifiche, salvare il file ed uscire dall'editor per tornare alla shell di sistema.

Con l'ambiente virtuale attivato, si procederà all'applicazione della prima migrazione[1] Django, un processo che adatta lo schema del database alle necessità dell'applicazione secondo le definizioni dei modelli dati. I comandi per effettuare questa operazione sono i seguenti:

```
cd /opt/storagis
source env/bin/activate
python manage.py migrate
```

Dopo aver completato con successo la migrazione, il sistema dovrebbe riflettere le modifiche apportate, pronte per essere utilizzate nell'ambito dell'applicazione StoraGIS.

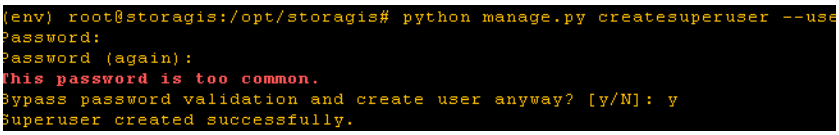


1 Le migrazioni permettono di modificare lo schema del database in modo controllato e versionato, adattando la struttura alle necessità dell'applicazione senza perdere dati o compromettere l'integrità del sistema [35].

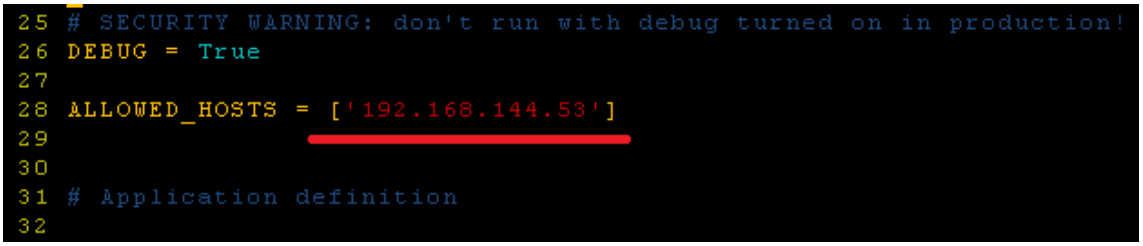
5.08 Python DRF – Superuser da interfaccia web

Per assicurare un accesso amministrativo iniziale a Django Rest Framework (DRF) tramite interfaccia web, sarà necessario creare un account superuser. Questo passaggio si effettua attraverso l'inserimento dei seguenti comandi nella shell di sistema del server su cui è ospitato StoraGIS.

```
cd /opt/storagis
source env/bin/activate
python manage.py createsuperuser --username admin --email
admin@example.com
python manage.py collectstatic
```

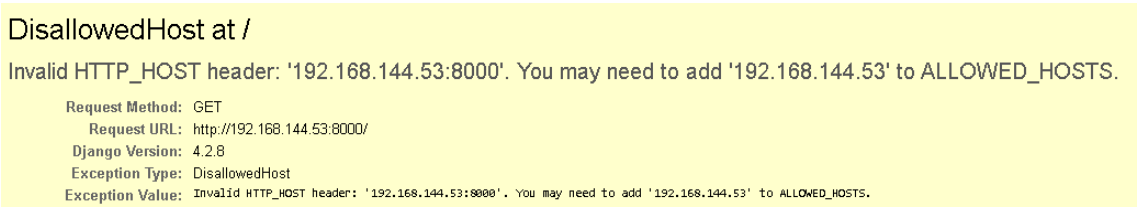


Dopo aver creato il superuser, per accedere al framework DRF via web è importante aggiungere l'indirizzo IP della macchina virtuale agli ALLOWED_HOSTS[1] nel file settings.py del progetto, situato in /opt/storagis/gisproject/settings.py. Questa modifica è cruciale per evitare problemi di sicurezza e configurazione che potrebbero impedire l'accesso al pannello di amministrazione DRF.



1 Django impiega la variabile ALLOWED_HOSTS per contrastare gli attacchi tramite l'intestazione HTTP dell'host. Tale variabile consiste in un elenco di stringhe che identificano i nomi degli host/domini ai quali il sito Django è autorizzato a rispondere [36].

Questa precauzione è fondamentale per la prevenzione di errori legati alla configurazione degli host consentiti, che potrebbero manifestarsi durante il tentativo di accesso al framework DRF attraverso il browser web.

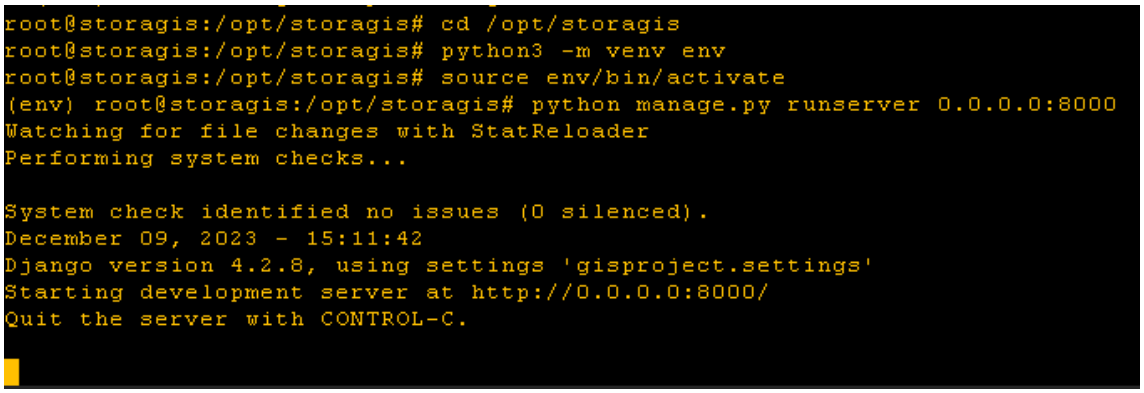


5.09 Python DRF – Avvio manuale del servizio

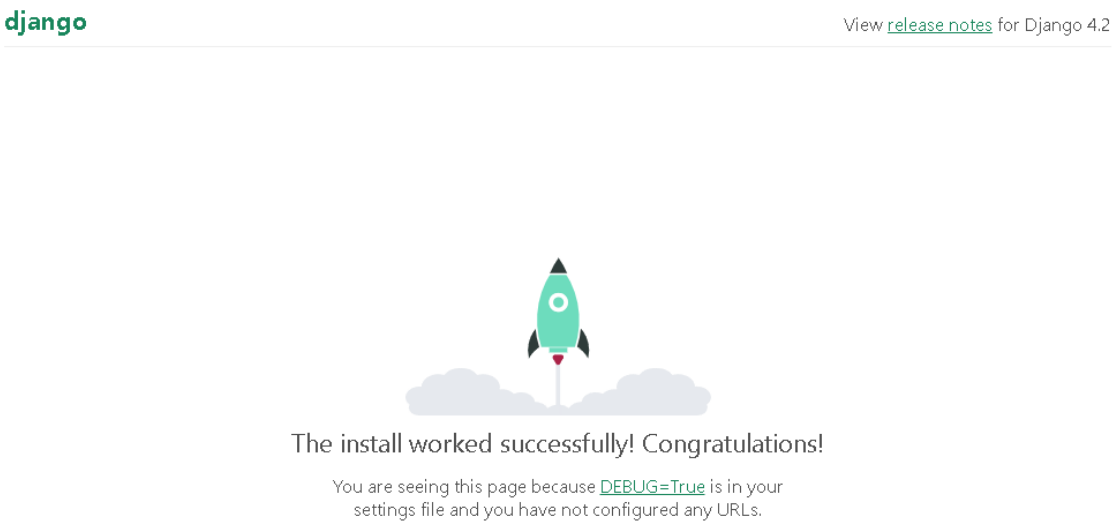
Per avviare manualmente il web server e rendere accessibile l'applicazione Django Rest Framework (DRF), si dovranno eseguire specifici comandi nella shell di sistema:

```
cd /opt/storagis
source env/bin/activate
python manage.py runserver 0.0.0.0:8000
```

Questi comandi permetteranno di avviare il server di sviluppo Django, che si porrà in stato di ascoltato su un'interfaccia e su una porta specifica (socket).



Una volta avviato il servizio, si potrà accedere al framework DRF utilizzando un browser web. Per fare ciò, sarà necessario inserire nella barra del browser l'indirizzo IP della macchina su cui è stato avviato il servizio, seguito dalla porta designata nei comandi di avvio.



Seguendo questi passaggi, si rende l'applicazione StoraGIS accessibile via web, permettendo agli sviluppatori e agli amministratori del sistema di interagire con l'interfaccia DRF per la gestione ed il test delle funzionalità implementate. Questo processo di avvio manuale del servizio è particolarmente utile durante le fasi di sviluppo e test, offrendo un modo semplice e diretto per verificare il corretto funzionamento dell'applicazione.

5.10 Python DRF – Finalizzazione del framework

A questo stadio, si dispone di un framework Django Rest Framework (DRF) completamente inizializzato e pronto per l'integrazione di moduli personalizzati. È importante interrompere qualsiasi processo avviato manualmente per procedere alla configurazione di un servizio di sistema dedicato che garantirà un avvio automatico e una gestione più affidabile del servizio.

5.10.1 – Creazione utenza applicativa framework

Per motivi di sicurezza e di buona gestione del sistema, si opterà per l'utilizzo di un'utenza dedicata[1] al processo applicativo DRF. Questa utenza sarà specificatamente creata per eseguire l'applicazione, assicurando che disponga dei soli permessi necessari per il suo funzionamento, senza sovraesporre dati o risorse del sistema:

```
useradd storagisusr -d /home/storagisusr -s /bin/false -m
```

1 L'adozione di un'utenza dedicata per i processi applicativi rappresenta una misura di sicurezza informatica essenziale. Configurando questa utenza con i soli permessi necessari per l'esecuzione dell'applicazione, si limita l'accesso al sistema a quanto strettamente indispensabile, riducendo il rischio di esposizioni non autorizzate o di abusi [37].

5.10.2 – Creazione SERVICE GUNICORN in SystemD

Utilizzando un editor di testo con privilegi amministrativi, sarà necessario creare un file di configurazione relativo al servizio Gunicorn. Questo file dovrà essere posizionato in /etc/systemd/system/gunicorn.service e contenere le specifiche per l'avvio e per la gestione del processo, che serve l'applicazione DRF:

```
[Unit]
Description=StoraGIS
Requires=gunicorn.socket
After=network.target

[Service]
User=storagisusr
Group=storagisusr
WorkingDirectory=/opt/storagis
ExecStart=/opt/storagis/env/bin/gunicorn --access-logfile - --workers 3 --bind
unix:/run/gunicorn.sock gisproject.wsgi:application

[Install]
WantedBy=multi-user.target
```

5.10.3 – Creazione SOCKET GUNICORN in SystemD

Per completare la configurazione di Gunicorn ed integrarlo efficacemente con SystemD, è necessario creare un file di configurazione per il socket. Questo file, collocato in /etc/systemd/system/gunicorn.socket, definisce come il socket si comporta e come interagisce con il servizio Gunicorn:

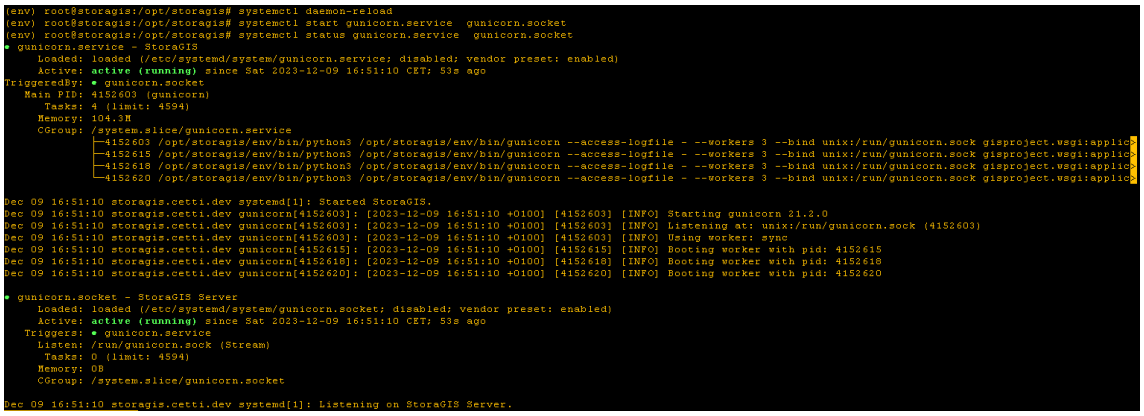
```
[Unit]
Description=StoraGIS Server

[Socket]
ListenStream=/run/gunicorn.sock

[Install]
WantedBy=sockets.target
```

5.10.4 – Abilitazione, avvio e verifica stato del framework

Dopo aver configurato i servizi Gunicorn ed il socket correlato, è essenziale far sì che systemd[1] riconosca e applichi le nuove configurazioni. Per avviare ed abilitare l'avvio automatico dei servizi, si utilizzano specifici comandi di seguito elencati:

```
systemctl daemon-reload
chown -R storagisusr:storagisusr /opt/storagis
systemctl start gunicorn.service  gunicorn.socket
systemctl status gunicorn.service  gunicorn.socket
```

L'ultimo dei comandi, in particolare, confermerà lo stato active (running) dei servizi, indicando che il framework è stato correttamente avviato e risulti funzionante.



Attraverso i prossimi passaggi, si assicurerà che l'applicazione StoraGIS sia non solo funzionante e sicura ma anche accessibile agli utenti finali tramite il web, sfruttando le capacità di Nginx come server proxy inverso per dirigere il traffico verso l'applicazione.

1 Un sistema di gestione servizi per Linux, fornisce funzionalità per l'avvio, lo stop e la gestione dei servizi, nonché capacità di parallelizzazione e controllo avanzate [38].

5.11 – Configurazione Nginx

Con il servizio DRF esposto tramite un socket di sistema, l'accesso diretto tramite browser non è possibile. La configurazione del socket viene generalmente utilizzata per comunicazioni interne al server. Per rendere l'applicazione accessibile dalla rete locale, è necessario configurare Nginx. Modificando il file di configurazione di default situato in /etc/nginx/sites-enabled/default, si può dirigere il traffico verso l'applicazione DRF con la seguente configurazione:

```
server {
  access_log /var/log/nginx/storagis_access.log;
  error_log /var/log/nginx/storagis_error.log warn;

  listen 80;
  client_max_body_size 20m;

  index index.jsp index.html index.html;

  location /static {
    autoindex on;
    alias /opt/storagis/static/;
  }

  location / {
    proxy_pass http://unix:/run/gunicorn.sock;
    proxy_set_header Host $http_host;
  }

}
```

```
systemctl restart nginx
```

Dopo aver riavviato Nginx, l'applicazione sarà accessibile tramite la porta 80, diversamente dall'avvio manuale visto precedentemente che utilizzava la porta 8000. Questo passaggio cambia la URL di accesso in http://192.168.X.X[1]

1 La porta standard per il traffico web HTTP. Secondo la RFC2616 [39], specificare questa porta nell'indirizzo non è mandatorio, poiché risulta implicita nel protocollo HTTP.

5.12 – Installazione geoserver

L'installazione di GeoServer segue un processo che presenta alcune similitudini con l'installazione del framework Django Rest Framework (DRF). Per raggiungere l'obiettivo, si dovranno completare i passaggi seguenti.

5.12.1 – Recupero pacchetto

I pacchetti necessari per l'installazione di GeoServer devono essere scaricati nel percorso /opt utilizzando i seguenti comandi specifici dalla shell di amministrazione.

```
cd /opt
wget
https://sourceforge.net/projects/geoserver/files/GeoServer/2.21.0/geoserver-2.21.0-bin.zip
```

```
HTTP request sent, awaiting response... 200 OK
Length: 110376595 (105M) [application/octet-stream]
Saving to: 'geoserver-2.21.0-bin.zip.1'
```

5.12.2 – Predisposizione filesystem

Per la prepararazione dell'ambiente in grado di servire l'applicazione GeoServer, si utilizzano i seguenti comandi[1]:

```
mkdir /opt/geoserver
unzip -d /opt/geoserver/ geoserver-2.21.0-bin.zip
chown -R storagisusr:storagisusr /opt/geoserver
```

Con l'attesa del seguente risultato.

```
(env) root@storagis:/opt# chown -R storagisusr:storagisusr /opt/geoserverr
(env) root@storagis:/opt# ls
geoserver  geoserver-2.21.0-bin.zip  storagis
(env) root@storagis:/opt# ls geoserver
bin       etc  license  modules     README.txt  RUNNING.txt  start.jar    webapps
data_dir  lib  logs     NOTICE.md  resources   start.ini    VERSION.txt
(env) root@storagis:/opt#
```

1 Per questioni di praticità useremo la stessa utenza applicativa creata nel paragrafo 5.10.1 per lanciare i processi di geoserver, ma nulla vieta di utilizzare due distinte utenze funzionali.

5.12.3 – Creazione SERVIZIO GEOSERVER in SystemD

In maniera analoga a quanto descritto per Gunicorn nei paragrafi 5.10.2 e 5.10.3, si procede alla creazione[1] del file di configurazione del servizio GeoServer in /etc/systemd/system/geoserver.service:

```
[Unit]
Description=GeoServer Service
After=network.target

[Service]
Type=simple
User=storagisusr
Group=storagisusr

Environment="GEOSERVER_HOME=/opt/geoserver"

ExecStart=/opt/geoserver/bin/startup.sh
ExecStop=/opt/geoserver/bin/shutdown.sh

[Install]
WantedBy=multi-user.target
```

5.12.4 – Abilitazione, avvio e verifica stato geoserver

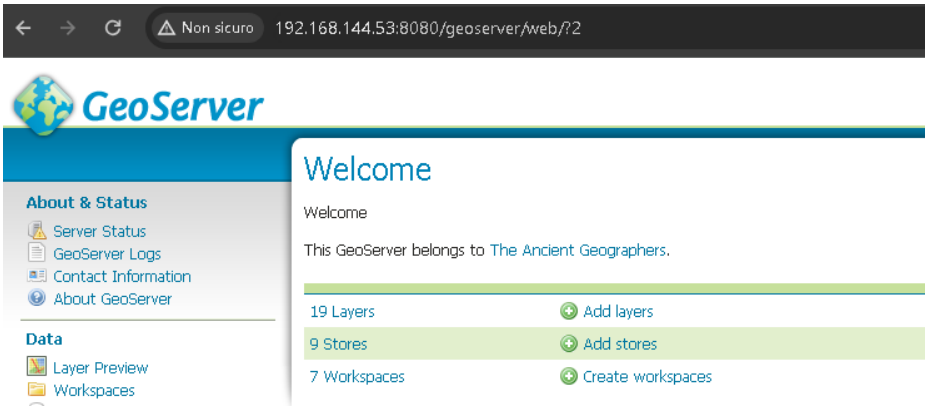
Dopo aver configurato il servizio GeoServer in SystemD, si registrano le modifiche e si avvia il servizio. I comandi utilizzati restituiranno un output che confermerà lo stato active (running) del servizio GeoServer, indicando che esso risulti correttamente avviato e funzionante:

```
systemctl daemon-reload
systemctl start geoserver.service && sleep 8
systemctl status geoserver.service
```

1 In modalità amministratore con l'editor di testo preferito.

5.13 – Accesso geoserver

L'accesso a Geoserver può esser effettuato attraverso l'URL http://192.168.X.X:8080/geoserver, utilizzando le credenziali predefinite (admin / geoserver). Questo passaggio consente di verificare la corretta installazione ed accessibilità del servizio.



Con l'installazione e l'avvio di Geoserver, il servizio è ora pronto per essere configurato ulteriormente in funzione delle specifiche esigenze del progetto. Tuttavia, prima di procedere con le fasi di sviluppo applicativo, è fondamentale adeguare la configurazione di Nginx. Questo passaggio mira ad ospitare il path webserver[1] che indirizzerà le chiamate verso Geoserver sulla stessa porta di servizio DRF. Facendo ciò, si elimina la necessità di specificare la porta 8080 durante la consultazione dei livelli georeferenziati o l'accesso amministrativo a Geoserver, semplificando l'interazione con il servizio attraverso un unico indirizzo standard del server.

---

1 Una directory o cartella all'interno di un server web che contiene i file e le risorse che il server web può fornire ai client richiedenti. Configurare correttamente questo path è essenziale per dirigere il traffico web verso i servizi specifici, come Geoserver, facilitando l'accesso degli utenti alle risorse senza dover navigare attraverso porte specifiche.

5.14 – Adeguamento configurazione Nginx

Per facilitare l'accesso a Geoserver senza dover specificare la porta 8080, si procede con l'adeguamento della configurazione Nginx. Modificando il file di configurazione di base situato in /etc/nginx/sites-enabled/default, si adeguerà, quindi, il sistema per far sì che il traffico sia correttamente reindirizzato verso il servizio Geoserver.

```
server {
  access_log /var/log/nginx/storagis_access.log;
  error_log /var/log/nginx/storagis_error.log warn;

  listen 80;
  server_name _;

  index index.jsp index.html index.html;

  location /static {
    autoindex on;
    alias /opt/storagis/static/;
  }

  location / {
    proxy_pass http://unix:/run/gunicorn.sock;
    proxy_set_header Host $http_host;
  }

  location /geoserver {
    proxy_pass http://127.0.0.1:8080/geoserver;
    proxy_set_header Host $http_host;
  }

}
```

```
systemctl restart nginx
```

Dopo aver riavviato Nginx, è possibile accedere a Geoserver direttamente dall'URL http://192.168.X.X/geoserver/ [1], senza la necessità di specificarne la porta, grazie alla configurazione Nginx che ora gestisce il traffico verso il servizio.

1 Il servizio geoserver risulterà "direttamente consultabile" (senza passare per NGINX) usando la porta 8080, mentre "indirettamente consultabile" attraverso la porta 80 implicita (NGINX).

Capitolo 6: dispiegamento e validazione

In questo capitolo, si finalizza la configurazione del sistema StoraGIS, seguendo le linee guida delineate nell'analisi concettuale. Gli obiettivi principali includono l'adeguamento del codice e della configurazione del framework Django Rest Framework (DRF) e la modifica delle impostazioni di Geoserver per integrarli pienamente nella struttura del progetto.

6.01 – DRF Configurazione Iniziale

Il primo passo per il dispiegamento dell'applicazione DRF riguarda la modifica dei parametri nel file /opt/storagis/gisproject/settings.py. Le modifiche necessarie includono:

- L'aggiunta di rest_framework alle INSTALLED_APPS, posizionandola alla fine dell'elenco.

- L'inserimento di configurazioni aggiuntive alla fine del file per integrare completamente DRF nell'applicazione:

```
REST_FRAMEWORK = {
   'DEFAULT_PAGINATION_CLASS':
'rest_framework.pagination.PageNumberPagination',
   'PAGE_SIZE': 10
}
```

6.02 – DRF Viste Iniziali

Le viste [40] in Django svolgono un ruolo cruciale nella gestione delle richieste web, elaborando i dati e restituendo le risposte appropriate. Le funzioni principali di una vista includono la ricezione delle richieste, l'elaborazione dei dati e la restituzione delle risposte, che possono variare da pagine HTML a file JSON o codici di errore.

6.02.1 – VISTA GRUPPI UTENTI

Per definire una vista dedicata ai gruppi utenti, è necessario modificare il file /opt/storagis/storagis/views.py, sostituendo il contenuto esistente con le nuove definizioni delle viste:

```
from django.contrib.auth.models import Group, User
from rest_framework import permissions, viewsets

from storagis.serializers import GroupSerializer, UserSerializer

class UserViewSet(viewsets.ModelViewSet):
  queryset = User.objects.all().order_by('-date_joined')
  serializer_class = UserSerializer
  permission_classes = [permissions.IsAuthenticated]

class GroupViewSet(viewsets.ModelViewSet):
  queryset = Group.objects.all()
  serializer_class = GroupSerializer
  permission_classes = [permissions.IsAuthenticated]
```

6.03 – DRF Routers Iniziali

I router in Django facilitano la mappatura tra le viste e gli URL, permettendo una configurazione automatica e coerente degli endpoint. Essi collegano la logica delle viste a specifici schemi di URL, basandosi sul tipo di metodo di richiesta (GET, POST, ecc.). Il router genera schemi di URL standardizzati [41].

6.03.1 – ROUTER PREDEFINITO

La configurazione iniziale dei router richiede la modifica del file /opt/storagis/gisproject/urls.py, eliminando il contenuto preesistente per introdurre direttamente solo i nuovi schemi di URL:

```
from django.urls import include, path
from rest_framework import routers

from storagis import views

router = routers.DefaultRouter()
router.register(r'users', views.UserViewSet)
router.register(r'groups', views.GroupViewSet)

urlpatterns = [
   path('', include(router.urls)),
   path('api-auth/', include('rest_framework.urls',
namespace='rest_framework'))
]

urlpatterns += router.urls
```

6.04 – DRF Serializzatori Iniziali

I serializzatori[1] in Django consentono la conversione dei dati tra formati complessi e tipi nativi di Python, facilitando la renderizzazione in formati come JSON o XML. Questi strumenti supportano anche la deserializzazione per la validazione e la riconversione dei dati in ingresso.

---

1 Un serializzatore Django è uno strumento che consente di convertire dati complessi come queryset e istanze di modelli in datatype nativi di Python, che possono poi essere facilmente renderizzati in JSON, XML o altri tipi di contenuto. I serializzatori forniscono anche la deserializzazione, consentendo di convertire i dati analizzati di nuovo in tipi complessi, dopo aver prima validato i dati in entrata [42].

6.04.1 – User and Group Serializer

Per definire i serializzatori per utenti e gruppi, creare un nuovo file /opt/storagis/storagis/serializers.py con le necessarie definizioni dei serializzatori:

```
from django.contrib.auth.models import Group, User
from rest_framework import serializers


class UserSerializer(serializers.HyperlinkedModelSerializer):
  class Meta:
    model = User
    fields = ['url', 'username', 'email', 'groups']


class GroupSerializer(serializers.HyperlinkedModelSerializer):
  class Meta:
    model = Group
    fields = ['url', 'name']
```

6.05 – DRF Modelli Iniziali

In Django, un Modello è una classe Python che deriva da django.db.models.Model, con ogni attributo che rappresenta un campo in una tabella del database. Generalmente, ciascun modello corrisponde ad una tabella. Django genera automaticamente un'API per l'accesso al database basata su questi modelli [43]. Per questa fase iniziale, non è richiesta la creazione di modelli specifici, poiché si opta per l'utilizzo dei modelli predefiniti e supportati da DRF.

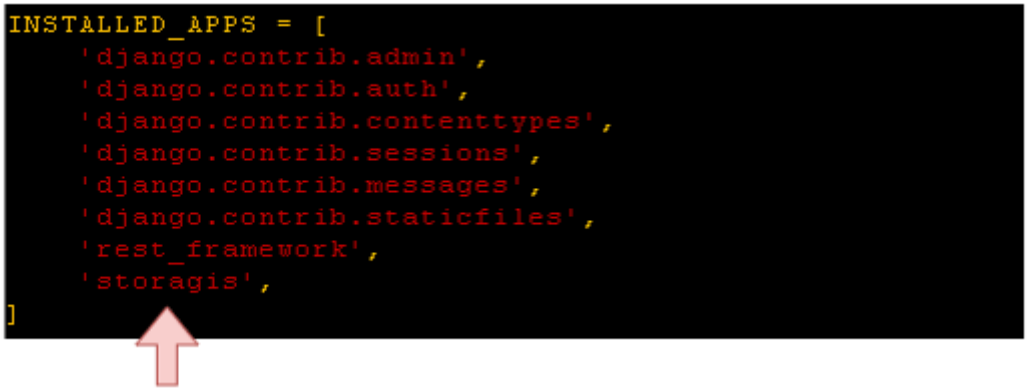
6.06 – Implementazione flusso di caricamento

Per implementare il flusso di caricamento, si inizia creando o sovrascrivendo il file situato al percorso /opt/storagis/storagis/models.py con il contenuto specificato:

```
from django.contrib.auth.models import User
from django.contrib.gis.db import models
import os
os.environ['DJANGO_SETTINGS_MODULE'] = 'gisproject.settings'
from django.conf import settings

def get_statocar_choices():
  statocar_choiches = [
    (-10,'N/D'),
    (-2,'Errore Procedura'),
    (0,'Ricevuto'),
    (1,'Validazione in corso'),
    (4,'Errori specifica'),
    (6,'Importato')
  ]
  return statocar_choiches

class progetti(models.Model):

  statocar_choices = get_statocar_choices()

  filename = models.CharField(max_length=255)
  data_caricamento = models.DateTimeField(auto_now_add=True)
  stato_caricamento = models.IntegerField(
    default=-10,
    choices=statocar_choices
  )
  owner = models.ForeignKey(User, related_name='progetti',
on_delete=models.CASCADE)
  nome_progetto = models.CharField(max_length=16)
  comune = models.IntegerField()
  approvazione = models.BooleanField(null=True)
  riferimento_remoto = models.CharField(max_length=255,null=True)
  risultato_validazione = models.CharField(max_length=16000,null=True)
```

Una volta definito il file storagis/models.py, si applicherà (descritto in seguito) una nuova migrazione. Sarà cruciale informare il framework che l'applicazione storagis è stata inizializzata, prima della migrazione, aggiungendo storagis alla lista delle INSTALLED_APPS, come illustrato al paragrafo 6.01.

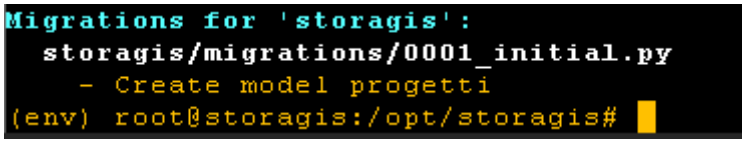
Si apre in modifica il file /opt/storagis/gisproject/settings.py e si aggiunge il contenuto necessario:



Successivamente, si attiva l'ambiente virtuale e si procede con l'applicazione della migrazione come indicato:

```
cd /opt/storagis
source env/bin/activate
python manage.py makemigrations
```

Questo processo genererà un set di modifiche SQL corrispondente alle modifiche apportate al modello dati per l'applicazione StoraGIS.



A questo punto, è possibile scegliere di applicare direttamente la modifica al framework DRF o di visualizzarne un'anteprima SQL.

È necessario ripetere la fase di collectstatic per aggiungere le dipendenze CSS:

```
python manage.py collectstatic
#Confermare con YES qualora richiesto
```

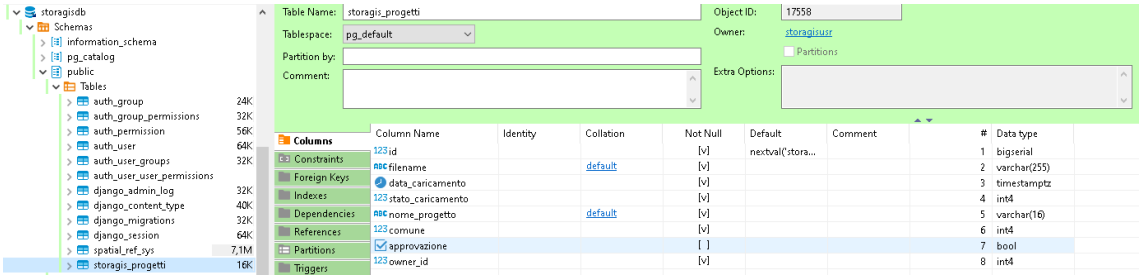
Per l'anteprima SQL, è sufficiente eseguire il comando python manage.py sqlmigrate storagis 0001, assicurandosi che l'ambiente virtuale sia attivo durante l'operazione.

```
(env) root@storagis:/opt/storagis# python manage.py sqlmigrate storagis 0001
BEGIN;
--
-- Create model progetti
--
CREATE TABLE "storagis_progetti" ("id" bigserial NOT NULL PRIMARY KEY, "filename" varchar(255)
NOT NULL, "data_caricamento" timestamp with time zone NOT NULL, "stato_caricamento" integer NOT
 NULL, "nome_progetto" varchar(16) NOT NULL, "comune" integer NOT NULL, "approvazione" boolean
NULL, "owner_id" integer NOT NULL);
ALTER TABLE "storagis_progetti" ADD CONSTRAINT "storagis_progetti_owner_id_2abb2eb3_fk_auth_use
r_id" FOREIGN KEY ("owner_id") REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED;
CREATE INDEX "storagis_progetti_owner_id_2abb2eb3" ON "storagis_progetti" ("owner_id");
COMMIT;
(env) root@storagis:/opt/storagis#
```

Per applicare la migrazione, si esegue il comando python manage.py migrate storagis 0001

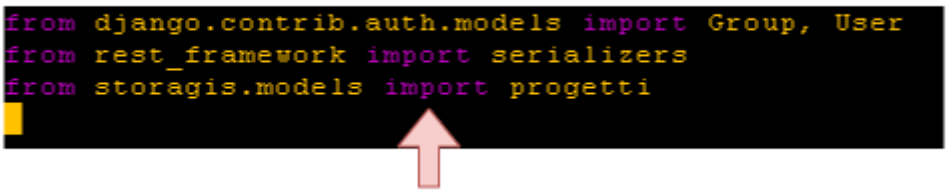
```
(env) root@storagis:/opt/storagis# python manage.py migrate storagis 0001
Operations to perform:
  Target specific migration: 0001_initial, from storagis
Running migrations:
  Applying storagis.0001_initial... OK
(env) root@storagis:/opt/storagis#
```

L'output risultante dovrebbe confermare che tutte le migrazioni richieste sono state applicate senza errori. Con questo passaggio, viene effettivamente creata la tabella SQL necessaria ad ospitare i progetti.



Il passo successivo è la creazione di un SERIALIZZATORE in grado di elaborare e correlare i dati inviati dall'utente al sistema.

Si riapre il file serializzatore /opt/storagis/storagis/serializers.py e si aggiunge l'importazione della risorsa Model appena creata con: from storagis.models import progetti



Dopo la definizione delle classi per Utenti e Gruppi, in fondo al file, si aggiunge una nuova classe per gestire la serializzazione dei dati progetto:

```
class ImportProjectSerializer(serializers.ModelSerializer):

 file_progetto = serializers.FileField()

 class Meta:
  model = progetti
  fields = [  'file_progetto','nome_progetto',
              'owner','data_caricamento',
              'comune','riferimento_remoto'  ]
  read_only_fields = ['owner','data_caricamento','riferimento_remoto']

 def create(self, validated_data):
  self.data.pop('file_progetto')
  validated_data.pop('file_progetto')
  return super(ImportProjectSerializer, self).create(validated_data)
```

Infine, si procede alla creazione di una VISTA che offra un'interfaccia web agli utenti. Gli utenti, una volta collegati ed autenticati, potranno iniziare a caricare i propri progetti.

Prima di tutto, è necessario creare una cartella sul sistema dove Storagis potrà scrivere i dati caricati dagli utenti, eseguendo il comando mkdir -p /opt/storagis/data/caricamenti dalla shell di amministrazione del sistema:

```
(env) root@storagis:/opt/storagis# mkdir -p /opt/storagis/data/caricamenti
(env) root@storagis:/opt/storagis# ls -d /opt/storagis/data/caricamenti
/opt/storagis/data/caricamenti
(env) root@storagis:/opt/storagis#
```

Dopo aver preparato il terreno, si apre in modifica il file situato al percorso /opt/storagis/storagis/views.py, al quale si aggiungono le importazioni delle risorse necessarie:

```
from storagis.serializers import ImportProjectSerializer
from storagis.models import progetti, get_statocar_choices
from rest_framework import response, status
import uuid
from django.core.files.base import ContentFile
from django.utils import timezone
```

```
from django.contrib.auth.models import Group, User
from rest_framework import permissions, viewsets

from storagis.serializers import GroupSerializer, UserSerializer

from storagis.serializers import ImportProjectSerializer
from storagis.models import progetti, get_statocar_choices
from rest_framework import response, status
import uuid
from django.core.files.base import ContentFile
from django.utils import timezone
```

Successivamente, in fondo al file, oltre alle classi UserViewSet e GroupViewSet già definite, si introduce la classe ImportProject come di seguito descritta.

```
class ImportProject(viewsets.ViewSet):
  serializer_class = ImportProjectSerializer
  permission_classes = [permissions.IsAuthenticated]
```

Questa nuova classe verrà arricchita con tre funzioni specifiche, le quali dovranno essere incollate all'interno dello stesso file, consecutivamente, rispettando l'indentazione prevista:

LIST

```
  def list(self, request):

    user_filter = str(request.user.id)
    user = User.objects.filter(id=request.user.id).values().first()
    uploads =
list(progetti.objects.filter(owner=user['id']).values().all().order_by('-id'))

    uploads = self.tuneup(uploads, request)

    if len(uploads) > 0:
      return response.Response(uploads)
    else:
      return response.Response("Nessun caricamento trovato.")
```

TUNEUP

```
  def tuneup(self,uploads, request):
    tuned_ups = [dict() for x in uploads]
    for i,ob in enumerate(uploads):
      tuned_ups[i]['id'] = ob['id']
      tuned_ups[i]['filename'] = ob['filename']
      tuned_ups[i]['data_caricamento'] =
ob['data_caricamento'].astimezone().strftime("%d/%m/%Y %H:%M:%S")
      tuned_ups[i]['stato_caricamento'] = next(x[1] for x in
get_statocar_choices() if x[0] == ob['stato_caricamento'])
      tuned_ups[i]['nome_progetto'] = ob['nome_progetto']
      tuned_ups[i]['comune'] = ob['comune']
      tuned_ups[i]['approvazione'] = ob['approvazione']
      tuned_ups[i]['risultato_validazione'] = ob['risultato_validazione']
      tuned_ups[i]['owner'] =
User.objects.filter(id=ob['owner_id']).values().first()['username']

    return tuned_ups
```

CREATE

```
  def create(self, request):
    serializer = ImportProjectSerializer(data=request.data)
    if not serializer.is_valid():
      return response.Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

    uploaded_filename=request.FILES['file_progetto'].name
    generated_uuid=str(uuid.uuid4())
    full_filename='/opt/storagis/data/caricamenti/%s.zip'%generated_uuid

    fout=open(full_filename, 'wb+')
    file_content=ContentFile(request.FILES['file_progetto'].read())
    stato_caricamento=-10
    data_caricamento = timezone.localtime(timezone.now())

    for chunk in file_content.chunks():
      fout.write(chunk)
    fout.close()

    if serializer.is_valid():
      serializer.save(
        owner=request.user,
        filename=generated_uuid,
        stato_caricamento=stato_caricamento,
        data_caricamento=data_caricamento,
      )
      return response.Response('Caricamento effettuato con successo.')

    return response.Response(serializer.data)
```

A seguire, è essenziale configurare DRF per esporre la route necessaria che permetta l'accesso e l'utilizzo della vista ImportProject. Per fare ciò, si modifica il file /opt/storagis/gisproject/urls.py, aggiungendovi la riga appropriata:

router.register(r'importproject', views.ImportProject, basename="Import Project")

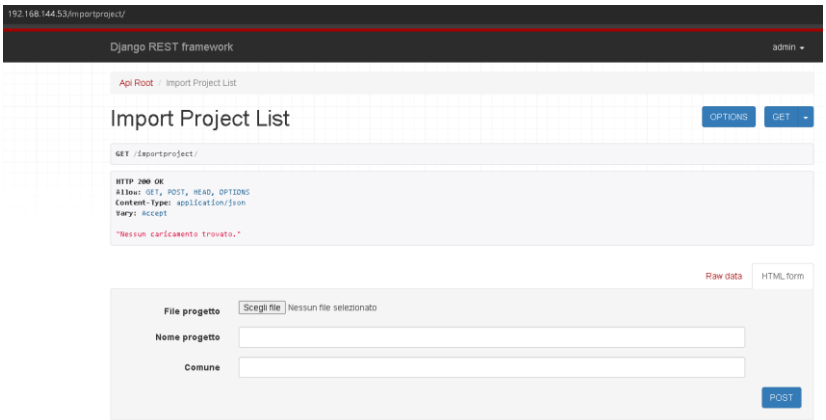
Dopo l'aggiunta, il contenuto complessivo di urls.py rifletterà le modifiche apportate:

```
from django.urls import include, path
from rest_framework import routers

from storagis import views

router = routers.DefaultRouter()
router.register(r'users', views.UserViewSet)
router.register(r'groups', views.GroupViewSet)
router.register(r'importproject', views.ImportProject, basename="Import Project")

# Wire up our API using automatic URL routing.
# Additionally, we include login URLs for the browsable API.
urlpatterns = [
    path('', include(router.urls)),
    path('api-auth/', include('rest_framework.urls', namespace='rest_framework'))
]

urlpatterns += router.urls
```

Conclusa questa fase, si procede ad assegnare la proprietà dell'utente applicativo a tutti i file all'interno della directory /opt/storagis e si effettua il riavvio dei servizi mediante l'inserimento dei comandi necessari nella shell di sistema:

chown -R storagisusr:storagisusr /opt/storagis
systemctl restart gunicorn.socket gunicorn.service nginx.service

Il sistema sarà ora in grado di ricevere dati e di salvarli localmente. Le funzioni appena realizzate saranno consultabili all'indirizzo IP della macchina di sviluppo, sotto il percorso /importproject/ (es: http://192.168.144.53/importproject/)



Sarà sufficiente popolare la maschera, includendo un file qualsiasi, per effettuare e verificare la possibilità di caricamento. Sarebbe opportuno utilizzare uno dei files campione forniti con la tesi in questo passaggio, in modo da agevolare la successiva configurazione dei livelli in geoserver (6.09.1)


```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

"Caricamento effettuato con successo."
```

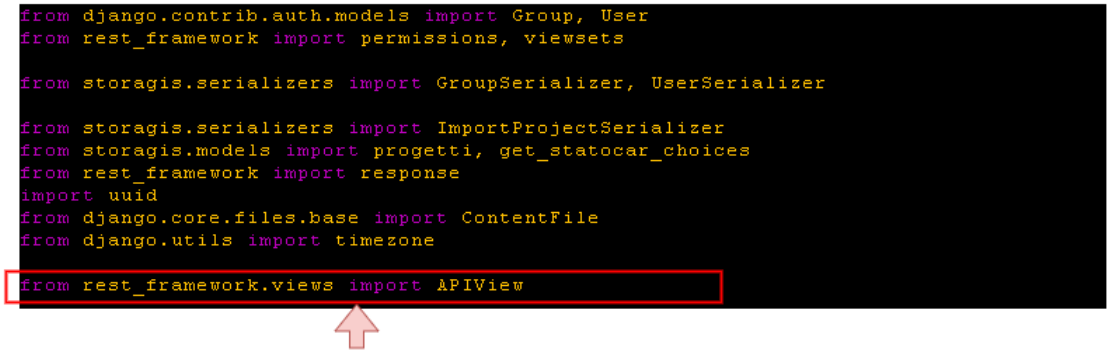

```
[
    {
        "id": 5,
        "filename": "b2c5a0a0-3e55-4b2d-9c6a-1a1f0eea1898",
        "data_caricamento": "10/01/2024 15:17:50",
        "stato_caricamento": "N/D",
        "nome_progetto": "PROGETTO TEST",
        "comune": 1001,
        "approvazione": null,
        "owner": "admin"
    }
]
```


6.07 – Implementazione flusso di approvazione

Questo paragrafo si concentra sull'aggiornamento della vista relativa alla funzione importproject, che permetterà la visualizzazione del riepilogo dei caricamenti pregressi e faciliterà il processo di approvazione dei progetti caricati. Le modifiche richieste comprendono:

- L’aggiunta di una nuova vista che permetterà il cambio dello stato di approvazione di un progetto caricato.
- L’integrazione, per ogni caricamento mostrato, di un endpoint URL che punti alla nuova vista creata, semplificando così l'operazione di approvazione.

Per iniziare, si modifica il file situato in /opt/storagis/storagis/views.py aggiungendo l'importazione della classe APIView da rest_framework.views, necessaria per creare la nuova vista: from rest_framework.views import APIView



Aggiungendo alla fine del file il seguente contenuto:

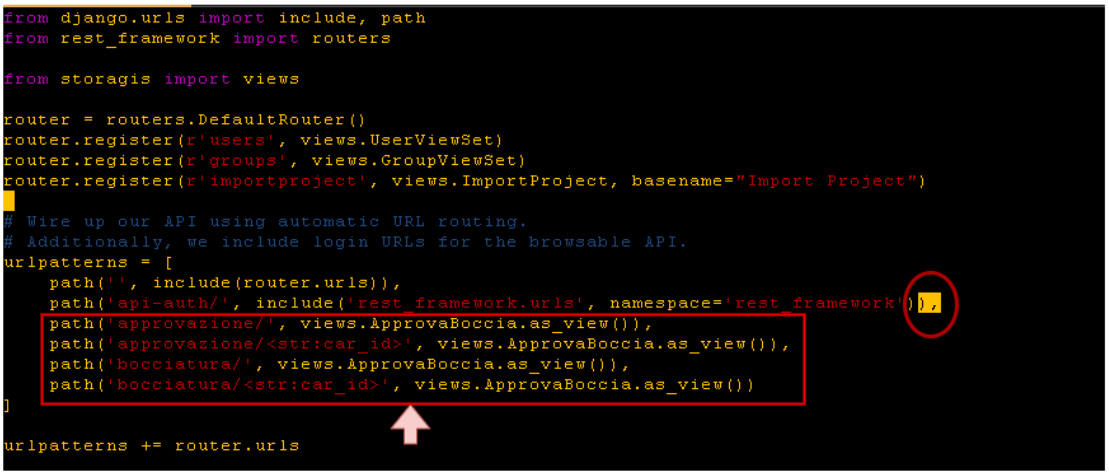
```
class ApprovaBoccia(APIView):
  permission_classes = [permissions.IsAuthenticated]

  def get(self, request, car_id=None):

    current_url = request.path.split('/')[1]
    approvazione=True
    approvazione_label='APPROVATO'
    if current_url == 'bocciatura':
      approvazione=False
      approvazione_label='BOCCIATO'

    if not car_id:
      return response.Response("ID CARICAMENTO NECESSARIO")

    prog = progetti.objects.filter(owner=request.user.id,id=car_id).first()
    if not prog:
      return response.Response("Non è possibile continuare, l'ID indicato è errato
oppure non presente tra i tuoi caricamenti. Riprova.")

    if prog.approvazione != approvazione:
      prog.approvazione = approvazione
      prog.save()
    else:
      return response.Response("ERRORE: il progetto si trova già in stato di:
%s"%(approvazione_label))

    return response.Response("SUCCESSO: il progetto è stato correttamente
posto nello stato di %s."%(approvazione_label))
```

Successivamente, è essenziale configurare Django Rest Framework (DRF) per esporre le routes necessarie che consentano l'accesso e l'utilizzo della nuova vista. Si procede con l'apertura in modifica del file /opt/storagis/gisproject/urls.py, aggiungendo le nuove righe nella sezione urlpatterns:

path('approvazione/', views.ApprovaBoccia.as_view()),
path('approvazione/<str:car_id>', views.ApprovaBoccia.as_view()),
path('bocciatura/', views.ApprovaBoccia.as_view()),
path('bocciatura/<str:car_id>', views.ApprovaBoccia.as_view())

È importante ricordarsi di aggiungere una virgola alla fine dell'ultima riga per mantenere la corretta sintassi di lista in Python.

Una volta completata questa fase, il file urls.py dovrebbe riflettere fedelmente le modifiche apportate ed apparire come illustrato nel contenuto fornito.



Per implementare la seconda operazione principale, necessaria a raggiungere l'obiettivo di questo paragrafo, si riapre il file /opt/storagis/storagis/views.py.

Qui, si modifica la funzione tuneup della classe ImportProject, arricchendola con le nuove righe di codice che aggiungeranno le informazioni necessarie per l'approvazione dei progetti:

```
def tuneup(self,uploads, request):
  tuned_ups = [dict() for x in uploads]
  for i,ob in enumerate(uploads):
    tuned_ups[i]['id'] = ob['id']
    tuned_ups[i]['filename'] = ob['filename']
    tuned_ups[i]['data_caricamento'] = ob['data_caricamento'].astimezone().strftime("%d/%m/%Y %H:%M:%S")
    tuned_ups[i]['stato_caricamento'] = next(x[1] for x in get_statocar_choices() if x[0] == ob['stato_caricamento'])
    tuned_ups[i]['nome_progetto'] = ob['nome_progetto']
    tuned_ups[i]['comune'] = ob['comune']
    tuned_ups[i]['approvazione'] = ob['approvazione']
    tuned_ups[i]['risultato_validazione'] = ob['risultato_validazione']
    tuned_ups[i]['owner'] = User.objects.filter(id=ob['owner_id']).values().first()['username']

  return tuned_ups
```

Queste modifiche permetteranno alla funzione tuneup di "pulire ed arricchire" i parametri restituiti durante la consultazione dell'elenco dei caricamenti, inserendo i link necessari all'approvazione dei progetti. Aggiungere, in fondo alla funzione tuneup, rispettando sempre l'indentazione, le seguenti due righe:

```
    tuned_ups[i]['ESECUZIONE APPROVAZIONE'] =
request.build_absolute_uri('/approvazione/%s'%ob['id'])
    tuned_ups[i]['ESECUZIONE BOCCIATURA'] =
request.build_absolute_uri('/bocciatura/%s'%ob['id'])
```

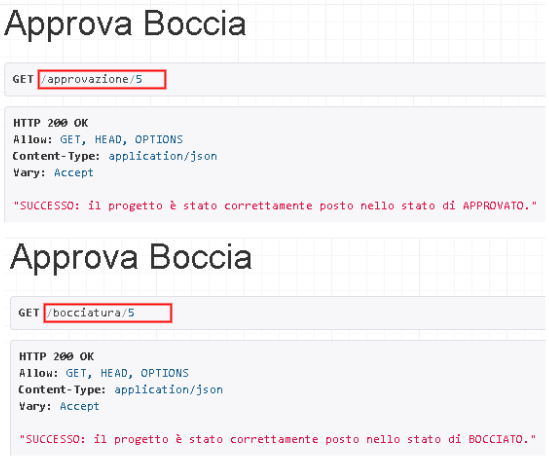
Dopo aver effettuato l'adeguamento, la funzione tuneup presenterà una nuova forma, come indicato di seguito:

```
def tuneup(self,uploads, request):
  tuned_ups = [dict() for x in uploads]
  for i,ob in enumerate(uploads):
    tuned_ups[i]['id'] = ob['id']
    tuned_ups[i]['filename'] = ob['filename']
    tuned_ups[i]['data_caricamento'] = ob['data_caricamento'].astimezone().strftime("%d/%m/%Y %H:%M:%S")
    tuned_ups[i]['stato_caricamento'] = next(x[1] for x in get_statocar_choices() if x[0] == ob['stato_caricamento'])
    tuned_ups[i]['nome_progetto'] = ob['nome_progetto']
    tuned_ups[i]['comune'] = ob['comune']
    tuned_ups[i]['approvazione'] = ob['approvazione']
    tuned_ups[i]['risultato_validazione'] = ob['risultato_validazione']
    tuned_ups[i]['owner'] = User.objects.filter(id=ob['owner_id']).values().first()['username']
    tuned_ups[i]['ESECUZIONE APPROVAZIONE'] = request.build_absolute_uri('/approvazione/%s'%ob['id'])
    tuned_ups[i]['ESECUZIONE BOCCIATURA'] = request.build_absolute_uri('/bocciatura/%s'%ob['id'])

  return tuned_ups
```

Infine, per garantire la corretta gestione dei permessi, si applica l'ownership dell'utente applicativo a tutti i file all'interno della cartella /opt/storagis. Si procede poi con un riavvio dei servizi attraverso i comandi specificati, da eseguire nella shell di sistema:

```
chown -R storagisusr:storagisusr /opt/storagis
systemctl restart gunicorn.socket gunicorn.service nginx.service
```

Questi passaggi completano l'implementazione del flusso di approvazione all'interno dell'applicazione StoraGIS, offrendo agli amministratori un metodo intuitivo e diretto per gestire lo stato dei progetti caricati sul sistema. Le funzioni appena realizzate saranno consultabili all'indirizzo della macchina di sviluppo, sotto il percorso /importproject/ (es: http://192.168.144.53/importproject/). Partendo da questa pagina, sarà ora possibile ritrovare, quindi, anche gli endpoint necessari per applicare la modifica allo stato di approvazione.



6.08 – Implementazione processo di elaborazione dati

Per visualizzare i progetti su mappa, è essenziale predisporre i dati in modo che Geoserver possa consultarli attraverso un connettore SQL.

6.08.1 – Adeguamento del modello dati

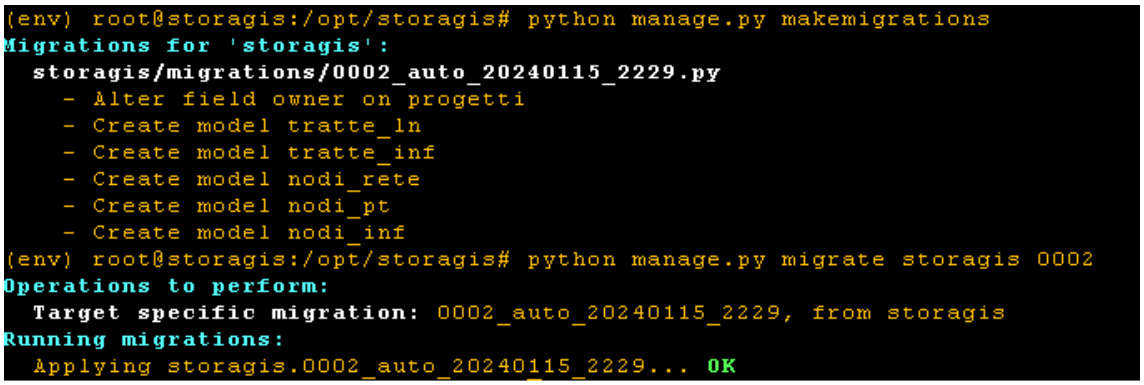
Come indicato al paragrafo 6.06, il modello dati attualmente definito in storagis/models.py comprende un'unica entità, la tabella progetti. Per raggiungere gli obiettivi di ricerca, è tuttavia necessario implementare ulteriori entità correlate alla specifica CT1. Si procede aprendo in modifica il file situato in /opt/storagis/storagis/models.py ed aggiungendo il contenuto richiesto:

```
class tratte_ln(models.Model):
  prj = models.ForeignKey(progetti, related_name='tratte_ln',
on_delete=models.CASCADE)
  idtrattaln = models.BigIntegerField(null=True)
  geom = models.GeometryField(null=True)

class nodi_pt(models.Model):
  prj = models.ForeignKey(progetti, related_name='nodi_pt',
on_delete=models.CASCADE)
  idnodopt = models.BigIntegerField(null=True)
  geom = models.GeometryField(null=True)

class tratte_inf(models.Model):
  prj = models.ForeignKey(progetti, related_name='tratte_inf',
on_delete=models.CASCADE)
  idtrattainf = models.BigIntegerField(null=True)
  idfeature = models.BigIntegerField(null=True)

class nodi_inf(models.Model):
  prj = models.ForeignKey(progetti, related_name='nodi_inf',
on_delete=models.CASCADE)
  idnodoinf = models.BigIntegerField(null=True)
  idfeature = models.BigIntegerField(null=True)
  etichetta = models.CharField(max_length=255,null=True)

class nodi_rete(models.Model):
  prj = models.ForeignKey(progetti, related_name='nodi_rete',
on_delete=models.CASCADE)
  idnodorete = models.BigIntegerField(null=True)
  idfeature = models.BigIntegerField(null=True)
  etichetta = models.CharField(max_length=255,null=True)
```

Dopo aver aggiunto le nuove definizioni al file storagis/models.py, si applica una nuova migrazione per rendere effettive le modifiche:

```
cd /opt/storagis
source env/bin/activate
python manage.py makemigrations
python manage.py migrate storagis 0002
```

Che, applicato correttamente, risulterà nel seguente output :



6.08.2 – Dispiegamento processo batch

L'implementazione di questo obiettivo richiede la creazione di:

- Un batch script denominato StoraGISProcessing
- Un servizio SystemD dedicato all'esecuzione ed al monitoraggio dello script

In modo simile a quanto svolto nel paragrafo 5.04, si attiva l'ambiente virtuale e si crea la cartella che ospiterà il processo batch:

```
cd /opt/storagis
source env/bin/activate
mkdir -p bin/SGP/
```

Successivamente, si crea il file StoraGISProcessing.py nel percorso /opt/storagis/bin/SGP/StoraGISProcessing.py, popolandolo con il contenuto seguente, diviso in sezioni (per semplicità) tutte parti dello stesso file:

IMPORTAZIONI E VARIABILI

```
import traceback
import time
import os, sys
import json
import django
import pycurl
from io import BytesIO
if not '/opt/storagis/' in sys.path:
  sys.path.append('/opt/storagis/')
os.environ['DJANGO_SETTINGS_MODULE'] = 'gisproject.settings'
django.setup()
from storagis.models import progetti
from django.apps import apps

sleeptime=1 #secondi
```

DEFINIZIONE SPECIFICA CT1 – PRESTARE LA MASSIMA ATTENZIONE IN FASE DI COPIA E INCOLLA. IL JSON DOVRA' RISULTARE CONSISTENTE


```
ct1_params={"mapping":{"nodi_pt.shp":{"idnodopt":{"titleoptions":["idnodopt
"],"datatype":"int8_notnull"},"geom":{"titleoptions":["geom"],"datatype":False
}},"nodi_inf.csv":{"idnodoinf":{"titleoptions":["idnodoinf"],"datatype":"int8_no
tnull"},"idfeature":{"titleoptions":["idfeature"],"datatype":"int8_notnull"},"etic
hetta":{"titleoptions":["etichetta"],"datatype":"varchar255"}},"nodi_rete.csv":
{"idnodorete":{"titleoptions":["idnodorete"],"datatype":"int8_notnull"},"idfeat
ure":{"titleoptions":["idfeature"],"datatype":"int8_notnull"},"etichetta":{"titleo
ptions":["etichetta"],"datatype":"varchar255"}},"tratte_ln.shp":{"idtrattaln":{"
titleoptions":["idtrattaln"],"datatype":"int8_notnull"},"geom":{"titleoptions":["
geom"],"datatype":False}},"tratte_inf.csv":{"idtrattainf":{"titleoptions":["idtrat
tainf"],"datatype":"int8_notnull"},"idfeature":{"titleoptions":["idfeature"],"data
type":"int8_notnull"}}},"constraints":{"nodi_pt.shp":{"constraint":[["idnodopt
"]]},"nodi_inf.csv":{"constraint":[["idnodoinf"]]},"nodi_rete.csv":{"constraint":
[["idnodorete"]]},"tratte_ln.shp":{"constraint":[["idtrattaln"]]},"tratte_inf.csv"
:{"constraint":[["idtrattainf"]]}},"relations":{"nodi_pt.shp":[],"nodi_inf.csv":[{
"f1":"idfeature","t2":"nodi_pt.shp","f2":"idnodopt"}],"nodi_rete.csv":[{"f1":"idf
eature","t2":"nodi_pt.shp","f2":"idnodopt"}],"tratte_ln.shp":[],"tratte_inf.csv":[
{"f1":"idfeature","t2":"tratte_ln.shp","f2":"idtrattaln"}]},"traduzione_tabelle":{
"nodi_pt.shp":"nodi_pt","nodi_inf.csv":"nodi_inf","nodi_rete.csv":"nodi_rete","t
ratte_ln.shp":"tratte_ln","tratte_inf.csv":"tratte_inf"}}
```


DATI NECESSARI ALLA FASE DI CONVERSIONE


```
ps_car_url = 'https://storagisprocessing.bycloud.eu/importproject/'
ps_get_url = 'https://storagisprocessing.bycloud.eu/getproject/'
ps_user = 'utenzagenerale'
ps_pass = '%+WCk6r03B"T:1;^>8p=8\?NE1CRjH-H'
```


Nota Bene: La password dell'utenza generale potrebbe cambiare nel tempo. È possibile richiedere una credenziale specifica contattando giordano.cetti@bycloud.eu. Non verranno poste domande e, se il servizio viene utilizzato con parsimonia, sarà garantito grazie al patrocinio di ByCloud SRL (https://bycloud.eu), di cui l'autore è amministratore.

FUNZIONE NECESSARIA ALL'INVIO E ALLA RICEZIONE DELLE RISPOSTE REMOTE

```
def restcall(url,post,timeout,auth,params=False,ffield=False):
  try:
    b_obj=BytesIO()
    h_obj=BytesIO()
    c=pycurl.Curl()
    c.setopt(c.URL, url)
    c.setopt(c.CONNECTTIMEOUT, timeout)
    c.setopt(c.USERPWD, auth)
    if post is True:
      postdata=[(ffield[0],(c.FORM_FILE, ffield[1]))]
      postdata.extend([('params',params)])
      c.setopt(c.HTTPPOST,postdata)
      c.setopt(c.REFERER, 'storagis-lover')

    c.setopt(c.WRITEDATA, b_obj)
    c.setopt(c.HEADERFUNCTION, h_obj.write)
    c.perform()
    statuscode = c.getinfo(c.RESPONSE_CODE)
    c.close()
    response=b_obj.getvalue()
    return (statuscode,response)

  except Exception:
    traceback.print_exc()
    return False
```

HEADER FUNZIONE PRIMARIA

```
primogiro = True
while True:
  if not primogiro:
    time.sleep(sleeptime)
  primogiro = False

  uploads = progetti.objects.all().order_by('id')
```

PRESA IN CARICO PROGETTI CARICATI

```
  for u in uploads.filter(stato_caricamento=-10):
    u.stato_caricamento = 0
    u.save()
```

INVIO DEI DATI AL SERVIZIO REMOTO DI VALIDAZIONE E CONVERSIONE

```
  for u in uploads.filter(stato_caricamento=0):
    try:
      ffield =
['file_progetto','/opt/storagis/data/caricamenti/%s.zip'%u.filename]
      params = json.dumps(ct1_params)
      auth = '%s:%s'%(ps_user, ps_pass)
      esito = restcall(ps_car_url,True,60,auth,params,ffield)
      if esito is False:
        raise Exception('Errore generale.')

      statuscode, response = esito

      if statuscode != 200:
        raise Exception('Errore di autenticazione, o di servizio. Verificare i dati di
accesso.')

      u.riferimento_remoto=response.decode('utf8').replace('"','')
      u.stato_caricamento=1

    except Exception:
      traceback.print_exc()
      u.stato_caricamento=-2
      continue
    finally:
      u.save()
```

RECUPERO DELLA RISPOSTA E DEI DATI CONVERTITI DAL SERVIZIO REMOTO

```
  for u in uploads.filter(stato_caricamento=1):
    try:
      url = ps_get_url+u.riferimento_remoto
      auth = '%s:%s'%(ps_user, ps_pass)

      esito = restcall(url,False,60,auth)
      if esito is False:
        raise Exception('Errore generale.')

      statuscode, response = esito
      if statuscode == 201:
        continue

      if statuscode == 204:
        u.stato_caricamento=4
        u.risultato_validazione=response.decode('utf8')
        continue

      if statuscode != 200:
        raise Exception('Errore di autenticazione, o di servizio. Verificare i dati di
accesso usando il browser.')

      response=json.loads(response.decode('utf8'))

      for k,v in response.items():
        thismodel = apps.get_model('storagis', k)

        for row in v[1:]:
          m = thismodel.objects.create(prj_id=u.id)

          for i,title in enumerate(v[0]):
            setattr(m,title,row[i])
            m.save()

      u.stato_caricamento=6

    except Exception:
      traceback.print_exc()
      u.stato_caricamento=-2
    finally:
      u.save()
```

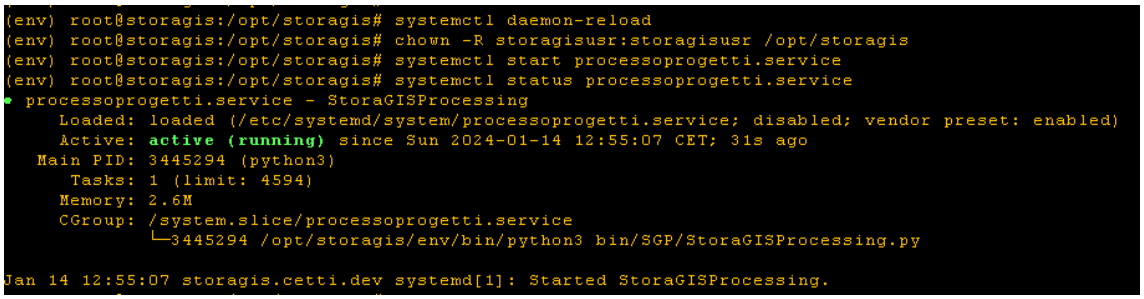
Infine, analogamente a quanto visto precedentemente al paragrafo 5.10, creare il file al seguente percorso /etc/systemd/system/processoprogetti.service con il seguente contenuto:

```
[Unit]
Description=StoraGISProcessing
After=network.target

[Service]
User=storagisusr
Group=storagisusr
WorkingDirectory=/opt/storagis
ExecStart=/opt/storagis/env/bin/python3 -u bin/SGP/StoraGISProcessing.py
StandardOutput=append:/var/log/storagis_processing.log
StandardError=append:/var/log/storagis_processing.err

[Install]
WantedBy=multi-user.target
```

Registriamo le modifiche in SystemD ed avviamo il processo con i seguenti comandi:

```
systemctl daemon-reload
chown -R storagisusr:storagisusr /opt/storagis
systemctl start processoprogetti.service
systemctl status processoprogetti.service
```

Di seguito un'illustrazione del risultato positivo.



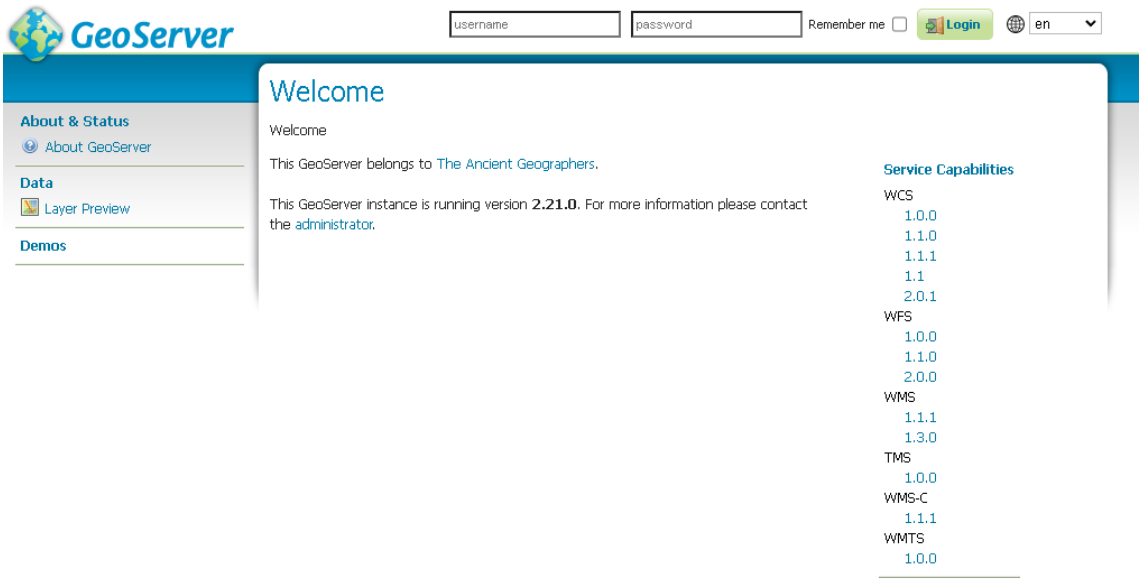
6.09 – Implementazione flusso di visualizzazione

Il flusso di visualizzazione richiede due macro-attività principali: l'adeguamento della configurazione di Geoserver e l'aggiustamento di DRF per la visualizzazione dei progetti CT1 caricati:

6.09.1 – Adeguamento geoserver

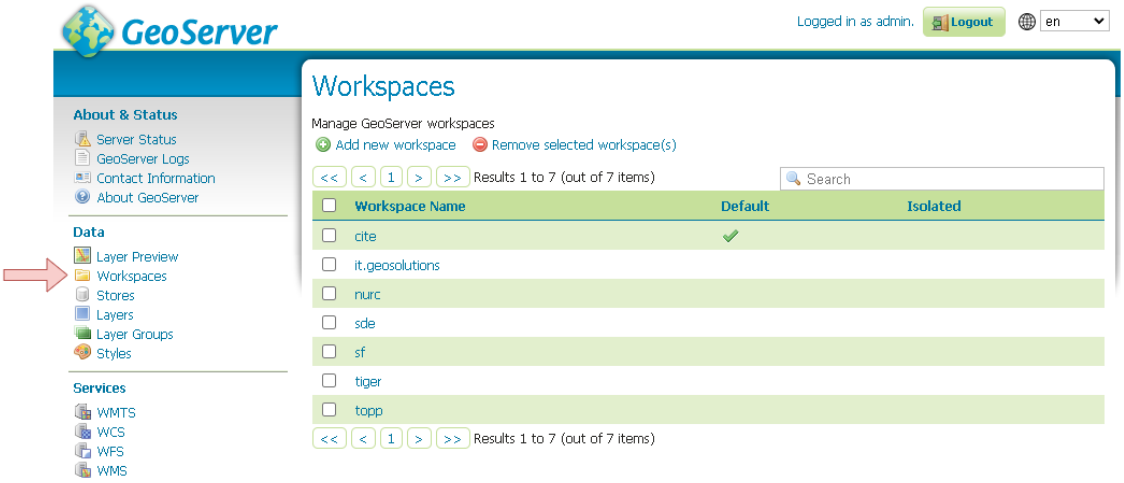
Il Geoserver installato presenterà dati campione non pertinenti alla ricerca. Si procede quindi all'eliminazione di questi dati:

Accedere al pannello di amministrazione di Geoserver tramite il browser: http://192.168.144.53/geoserver

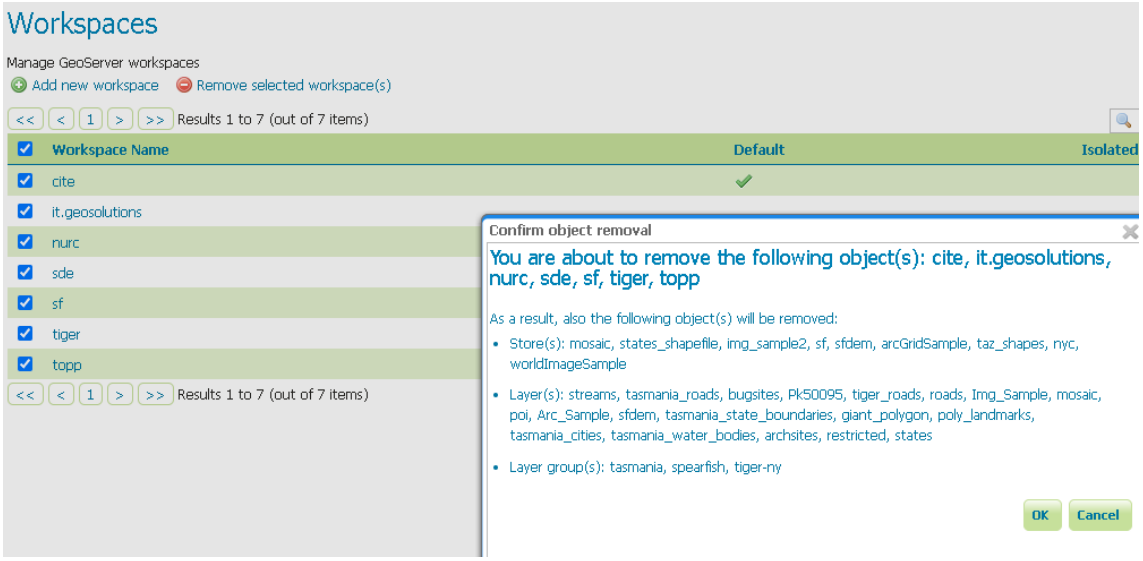


Le credenziali di accesso standard sono admin/geoserver, sarebbe preferibile cambiare questi dati di accesso qualora si preveda di esporre pubblicamente il servizio.

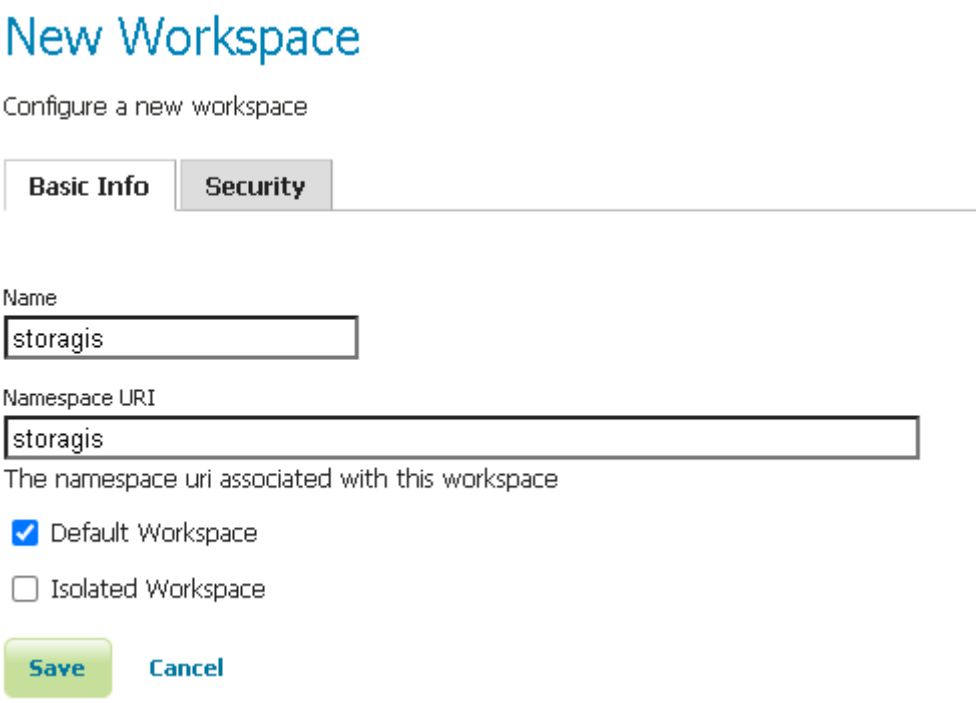
Iniziare la pulizia dei workspace esistenti ed aggiungere il workspace principale necessario al progetto:



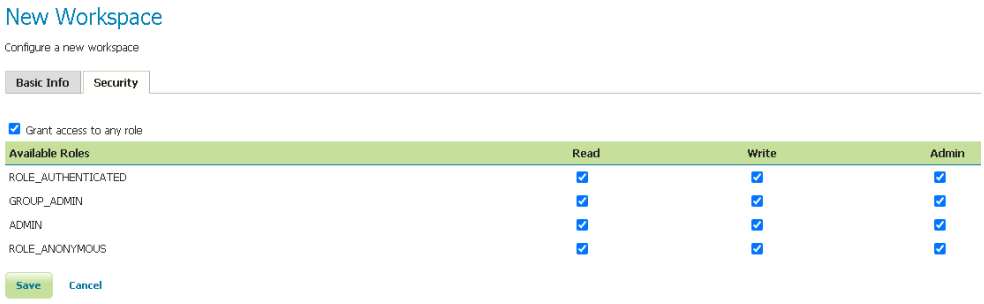
Selezionare tutti i workspaces, e cliccare su remove selected in alto.



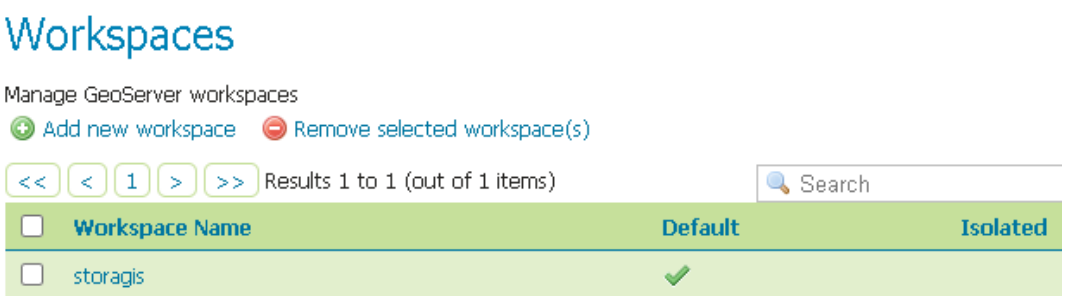
Giunti a questo punto, aggiungere il workspace principale facendo click sul tasto Add new workspace.



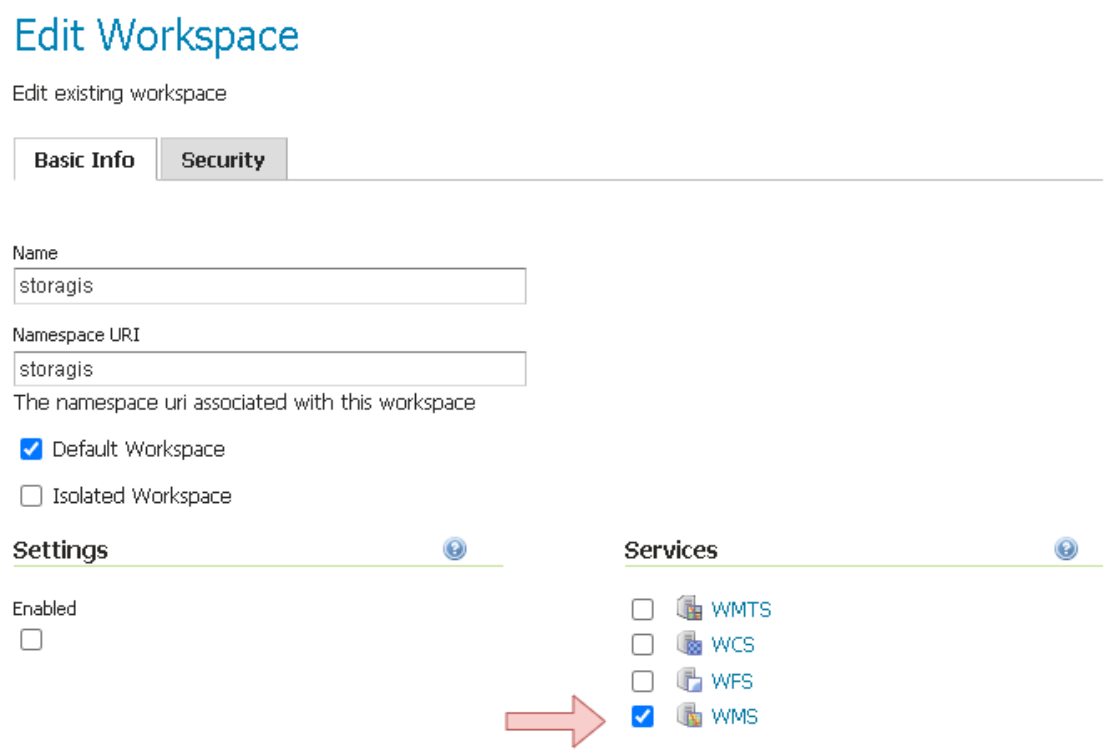
Prima di salvare: spostarsi nel tab Security.



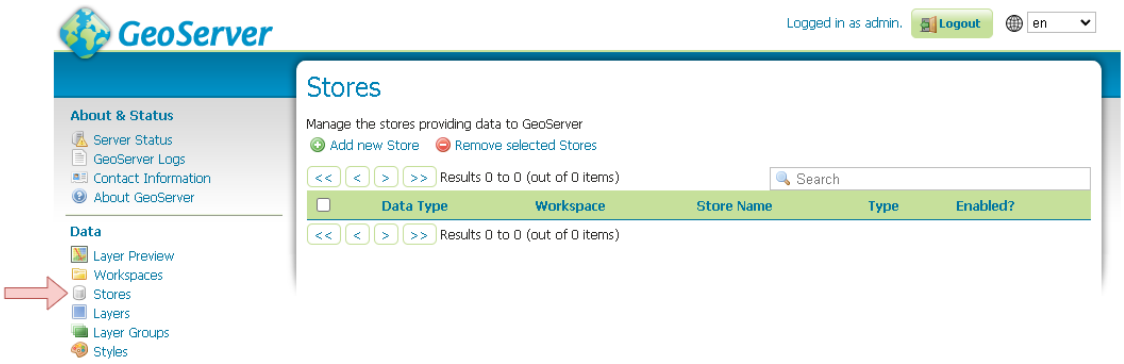
Assicurarsi l'accesso a tutti i ruoli prima di salvare.



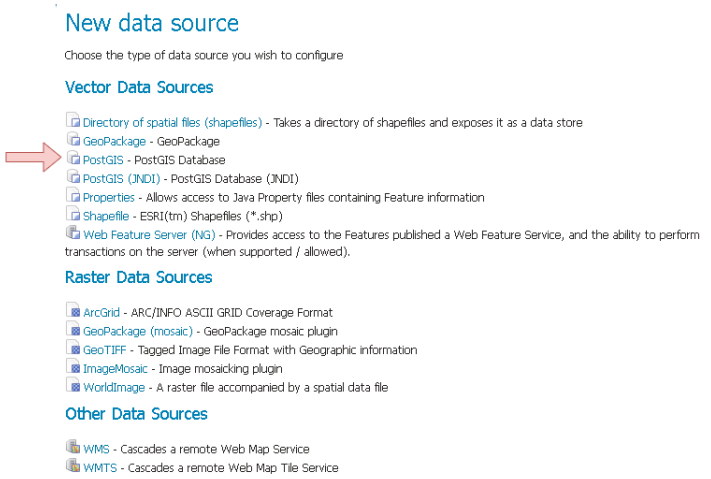
Aprire nuovamente il workspace in modifica, cliccandoci sopra. A differenza della fase di creazione, ora sarà possibile attivare il servizio WMS e cliccare su SAVE.



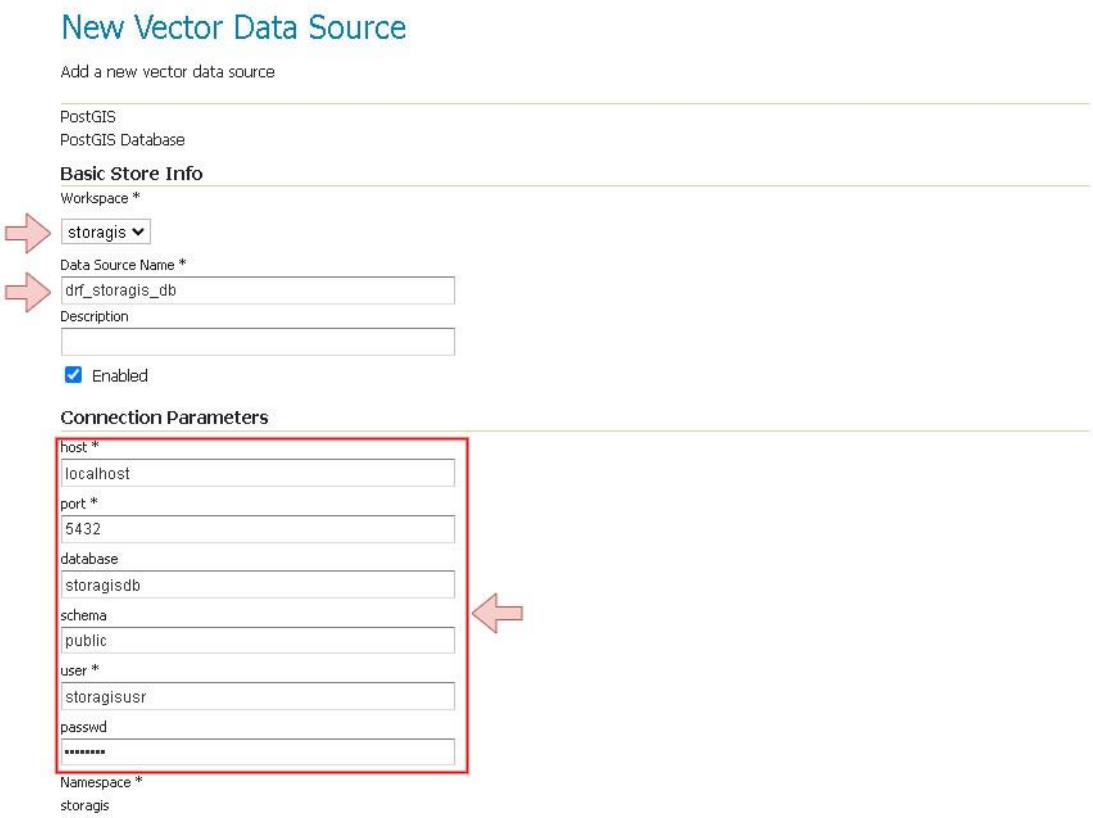
Spostarsi nel menù STORES.



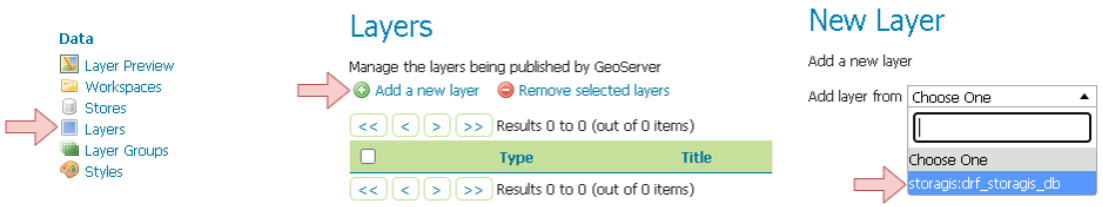
Dopo aver cliccato su Add new store, selezionare PostGIS come Vector Data Source.



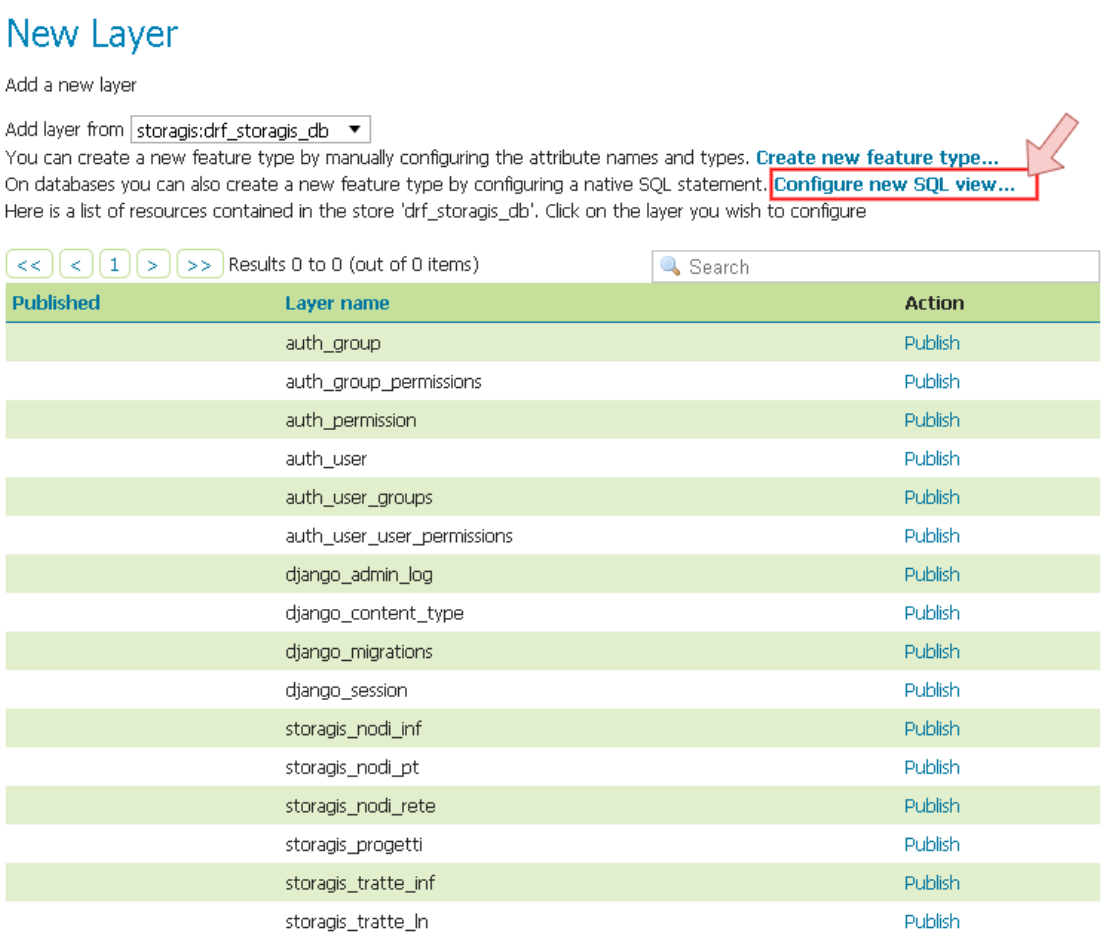
Di seguito, si dovranno inserire tutte le informazioni necessarie all'accesso al database, che saranno le stesse utilizzate da Storagis in DRF, create nel capitolo 5.



Una volta inserite le credenziali corrette, al click sul tasto di salvataggio si verrà trasportati nella vista necessaria alla creazione dei layers. Uscire dalla creazione guidata del primo layer, ed entrare direttamente nel menù Layers per vedere esattamente come ripetere il processo sul secondo livello.



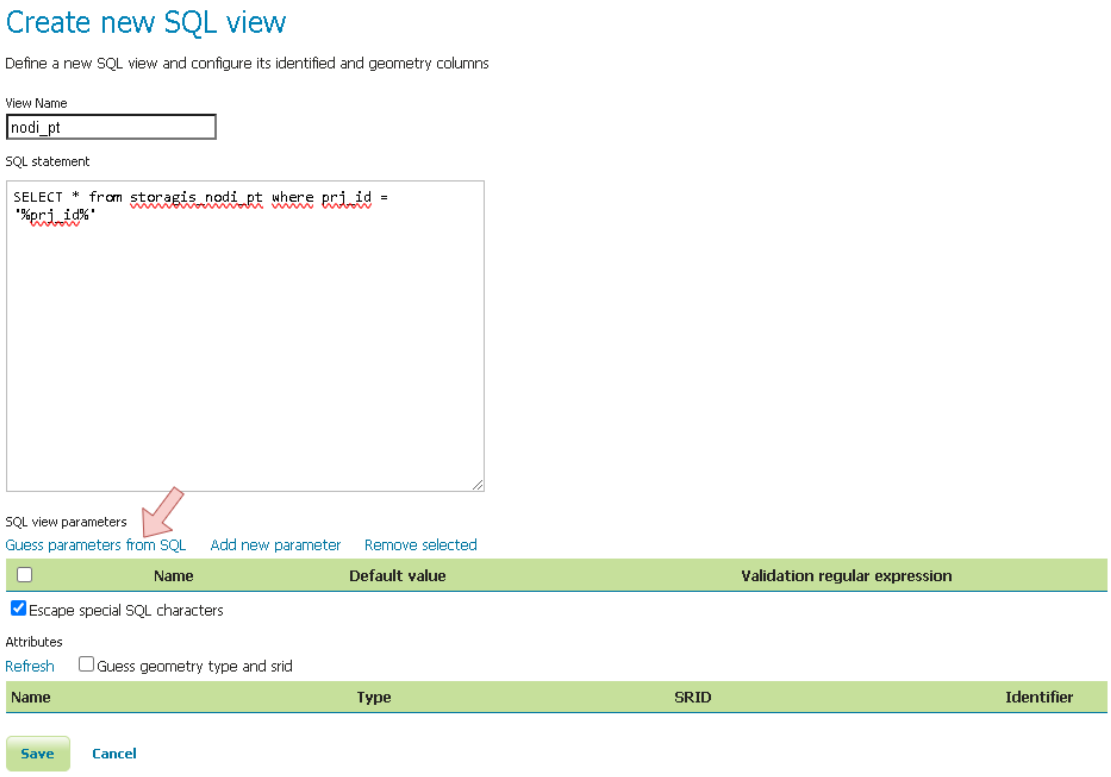
Si aprirà di nuovo il menù visto poco prima, clicchiamo su Create new SQL view.



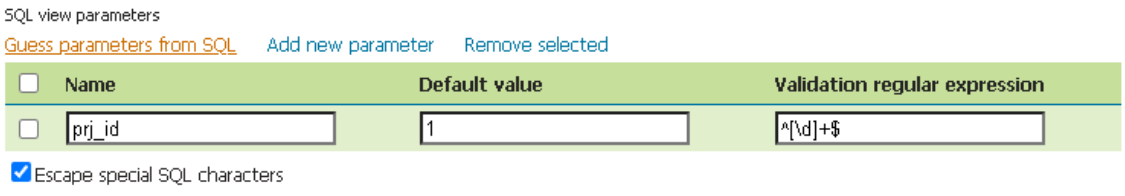
Nella seguente schermata, adeguare il nome della tabella (nodi_pt / nodi_ln) ed inserire nella textbox la seguente query, aggiornandola in base alla tabella che si sta attualmente configurando:

SELECT * from storagis_nodi_pt where prj_id = '%prj_id%'

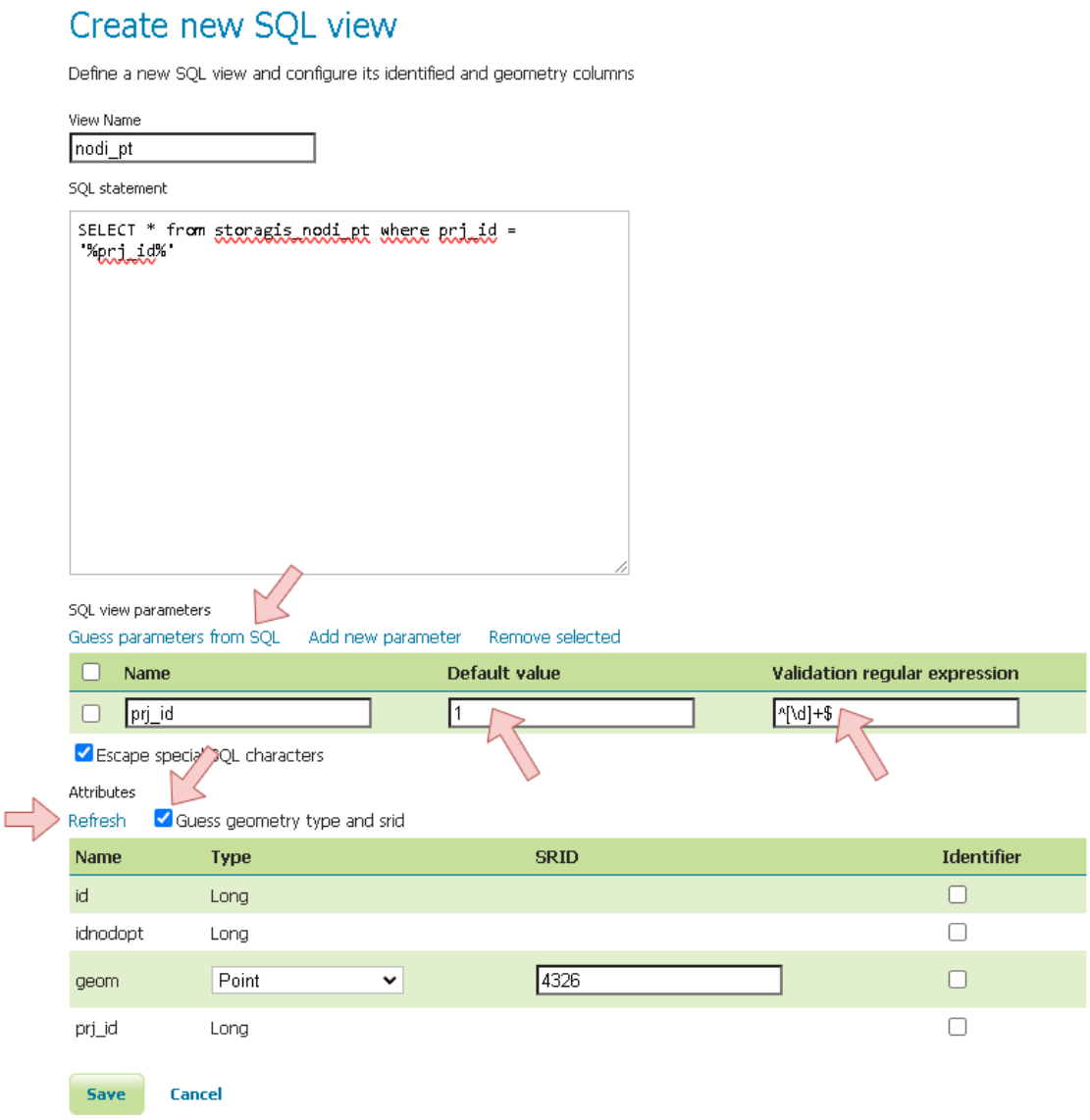
Dopodiché, cliccare su "Guess parameters from SQL" come di seguito illustrato:



Apparirà una nuova riga di valori, impostare 1 come default e rimpiazzare la stringa di validazione espressione regolare, con la seguente, che ammette solo interi positivi: ^[\d]+$ come di seguito illustrato

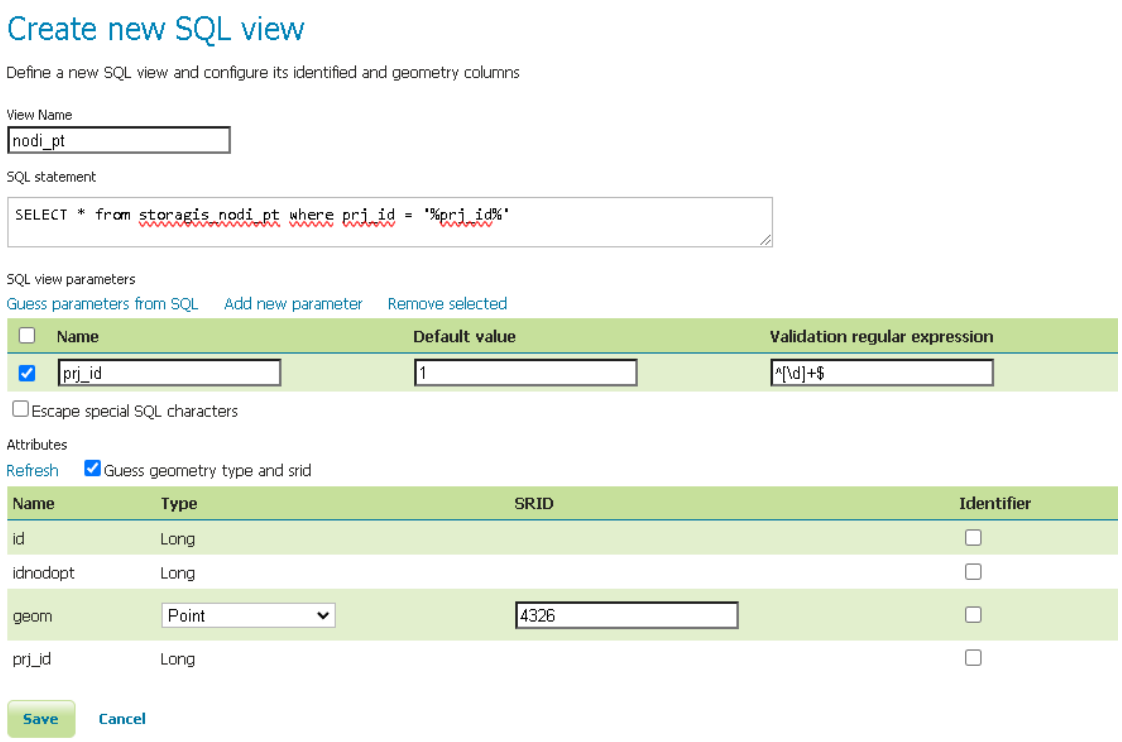


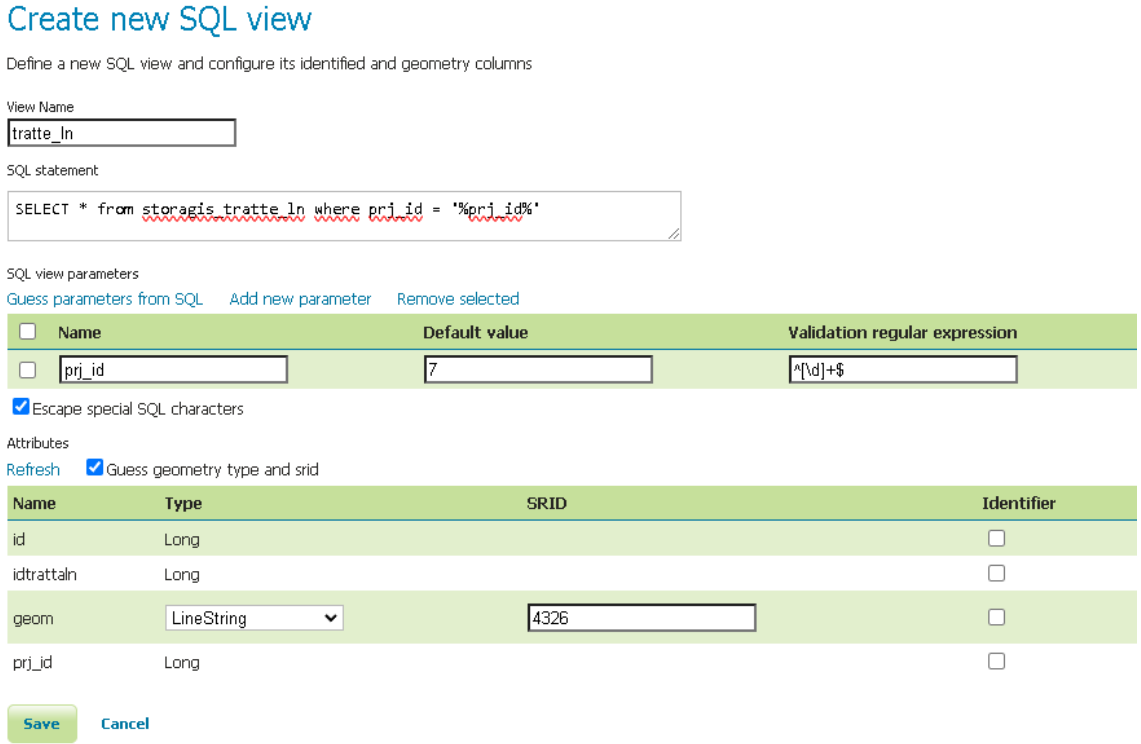
All'altezza della riga attributes, impostare la flag attiva nel campo "Guess geometry type and srid" e cliccare su refresh. Se tutto è stato svolto correttamente, la situazione, prima di cliccare su SALVA, sarà la seguente:



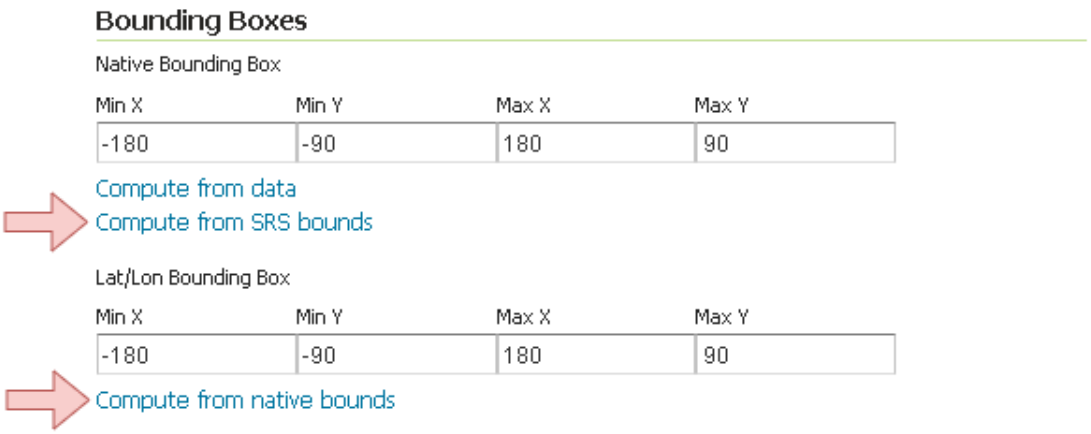
NB: Per far sì che "Guess geometry type and srid" risulti funzionante, devrà esserci almeno un caricamento in stato di Importato con lo stesso ID del default value, anche impostandolo momentaneamente ad un valore diverso da 1.

In ogni caso, nel dubbio, per quanto riguarda StoraGIS, sarà possibile utilizzare anche solo manualmente le seguenti configurazioni per nodi_pt e tratte_ln, evitando tutte le fasi di "Guessing" che sono legate alla PRESENZA DEI DATI:





Cliccando su salva, si aprirà, quindi, la vista di modifica. Abbiamo già configurato tutto il necessario, resta solo di popolare i campi di Bounding Boxes che si auto completeranno al click di "Compute from [SRS, native] bounds" come di seguito illustrato:



Dopo aver impostato gli attributi ed i bounds, si salva la configurazione e si ripete il processo per completare l'adeguamento di Geoserver, che non richiederà ulteriori modifiche a meno di specifiche necessità legate alla gestione degli stili[1] delle geometrie.

1 Gli stili in geoserver sono gestiti in un formato XML nativo che rispecchia una sintassi specifica dell'implementazione stessa. Un esempio utile su come comporre diverse tipologie di stili è disponibile sulla pagina di Krisna Lodha, che ha partecipato attivamente, tra altre cose, all'evoluzione del software QGIS [44].

6.09.2 – DRF Implementazione flusso di visualizzazione

Ora che il workspace geoserver è stato correttamente configurato, si procede alla creazione di un template in grado di mostrare una pagina HTML contenente la mappa, ed il progetto importato.

Si procede creando una nuova cartella al percorso /opt/storagis/storagis/templates:

```
mkdir -p /opt/storagis/storagis/templates
```

Per poi creare ed aprire in modifica il file index.html nello stesso percorso con il seguente contenuto (/opt/storagis/storagis/templates/index.html). Come svolto in precedenza, il contenuto del file verrà diviso in più sezioni che andranno incollate sequenzialmente all'interno del template:

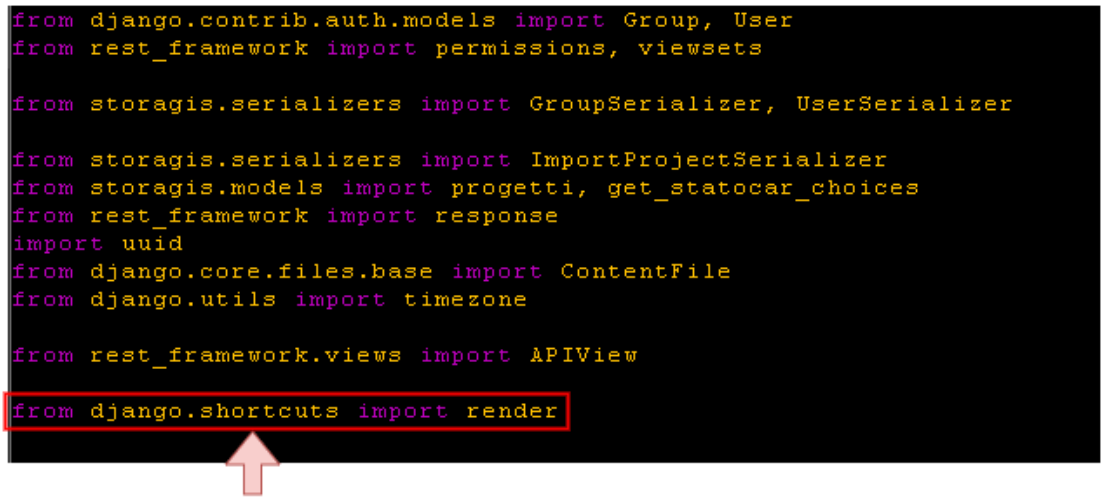
HEADER HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Tiled WMS</title>
    <script src="https://cdn.jsdelivr.net/npm/ol@v9.0.0/dist/ol.js"></script>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/ol@v9.0.0/ol.css">
    <style>
      .map {
        width: 100%;
        height: 800px;
      }
    </style>
  </head>
```

SCRIPT E CONTENUTO DEL CORPO

```
  <body>
    <div id="map" class="map"></div>
    <div id="nodelist"></div>
    <script type="module">
const layers = [
  new ol.layer.Tile({
    source: new ol.source.OSM(),
    projection: 'EPSG:4326'
  }),
  new ol.layer.Tile({
    source: new ol.source.TileWMS({
      url: '{{ geoserver_url }}/wms',
      params: {'LAYERS': 'storagis:nodi_pt', 'TILED': true, 'EPSG': '4326',
viewparams: 'prj_id:{{ prj_id }}'},
      serverType: 'geoserver',
      transition: 0,
    }),
  }),
  new ol.layer.Tile({
    source: new ol.source.TileWMS({
      url: '{{ geoserver_url }}/wms',
      params: {'LAYERS': 'storagis:tratte_ln', 'TILED': false, 'EPSG': '4326',
viewparams: 'prj_id:{{ prj_id }}'},
      serverType: 'geoserver',
      transition: 0,
    }),
  }),
];
const map = new ol.Map({
  layers: layers,
  target: 'map',
  view: new ol.View({
    center: ol.proj.transform([13.139648, 42.231445], 'EPSG:4326', 'EPSG:3857'),
    zoom: 6,
  }),
});
    </script>
  </body>
</html>
```

Salvare il file ed uscire dall'editor per tornare alla shell di sistema, per poi aprire in modifica il file situato in /opt/storagis/storagis/views.py aggiungendo l'importazione della classe render da django.shortcuts, necessaria per creare la nuova vista: from django.shortcuts import render

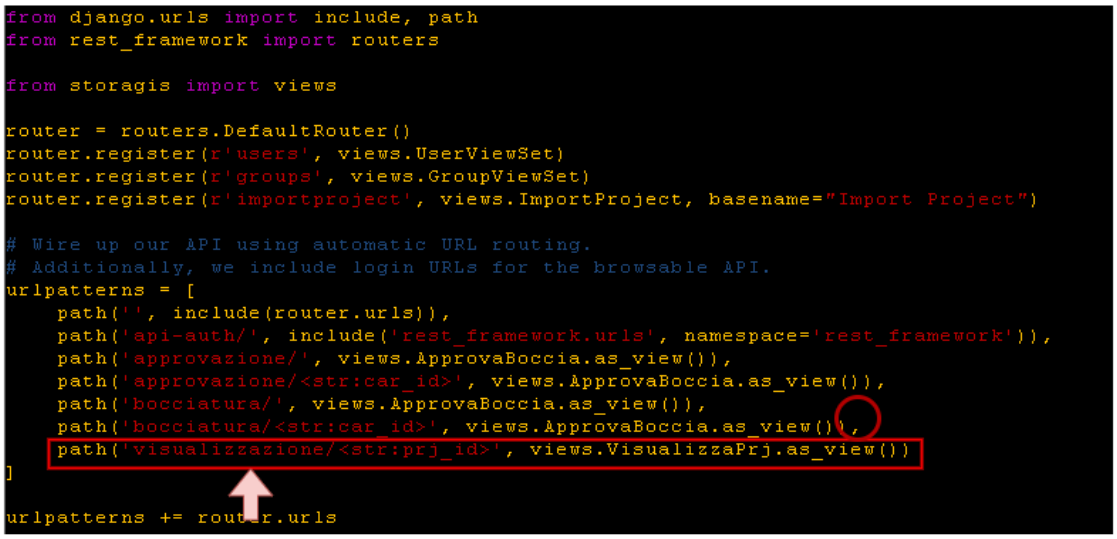


Aggiungendo in fondo al file il seguente contenuto:

```
class VisualizzaPrj(APIView):
  permission_classes = [permissions.IsAuthenticated]

  def get(self, request, prj_id=None):

    visualizzaprj_template = 'index.html'
    geoserver_url = request.build_absolute_uri().split(request.path)[0] +
'/geoserver'
    oldata = {'prj_id':prj_id,'geoserver_url':geoserver_url}

    if not prj_id:
      return response.Response("PROGETTO NON TROVATO.")

    prog = progetti.objects.filter(owner=request.user.id,id=prj_id).first()
    if not prog:
      return response.Response("PROGETTO NON TROVATO O NON
APPARTENENTE ALL'UTENTE AUTENTICATO.")

    return render(request, 'index.html', oldata)
```

Successivamente, sarà essenziale configurare Django Rest Framework (DRF) per esporre le routes necessarie che consentano l'accesso e l'utilizzo della nuova vista. Si proceda con l'apertura in modifica del file /opt/storagis/gisproject/urls.py, aggiungendo la seguente nuova riga nella sezione urlpatterns:

path('visualizzazione/<str:prj_id>', views.VisualizzaPrj.as_view())



L'attività di implementazione del flusso di visualizzazione è ora completata. Analogamente a quanto visto nell'implementazione del flusso di approvazione, si modifichi la funzione tuneup della classe ImportProject (/opt/storagis/storagis/views.py) arricchendola con la nuova riga di codice che aggiungerà l'indirizzo web necessario per la visualizzazione dei progetti selezionati:

```
tuned_ups[i]['VISUALIZZA PROGETTO'] =
request.build_absolute_uri('/visualizzazione/%s'%ob['id'])
```

Dopo aver effettuato l'adeguamento, la funzione tuneup presenterà una nuova forma, come indicato di seguito:

```
def tuneup(self,uploads, request):
  tuned_ups = [dict() for x in uploads]
  for i,ob in enumerate(uploads):
    tuned_ups[i]['id'] = ob['id']
    tuned_ups[i]['filename'] = ob['filename']
    tuned_ups[i]['data_caricamento'] = ob['data_caricamento'].astimezone().strftime("%d/%m/%Y %H:%M:%S")
    tuned_ups[i]['stato_caricamento'] = next(x[1] for x in get_statocar_choices() if x[0] == ob['stato_caricamento'])
    tuned_ups[i]['nome_progetto'] = ob['nome_progetto']
    tuned_ups[i]['comune'] = ob['comune']
    tuned_ups[i]['approvazione'] = ob['approvazione']
    tuned_ups[i]['risultato_validazione'] = ob['risultato_validazione']
    tuned_ups[i]['owner'] = User.objects.filter(id=ob['owner_id']).values().first()['username']
    tuned_ups[i]['ESECUZIONE APPROVAZIONE'] = request.build_absolute_uri('/approvazione/%s'%ob['id'])
    tuned_ups[i]['ESECUZIONE BOCCIATURA'] = request.build_absolute_uri('/bocciatura/%s'%ob['id'])
    tuned_ups[i]['VISUALIZZA PROGETTO'] = request.build_absolute_uri('/visualizzazione/%s'%ob['id'])
```

Salvare il file, per garantire la corretta gestione dei permessi, inoltre, si applichi l'ownership dell'utente applicativo a tutti i file all'interno della cartella /opt/storagis. Si proceda poi con un riavvio dei servizi attraverso i comandi specificati, da eseguire nella shell di sistema:

```
chown -R storagisusr:storagisusr /opt/storagis
systemctl restart gunicorn.socket gunicorn.service nginx.service
processoprogetti.service geoserver.service
systemctl enable gunicorn.socket gunicorn.service nginx.service
processoprogetti.service geoserver.service
```

Al termine del riavvio del servizio appena effettuato, tutto il sistema risulterà completamente funzionante. Sarà possibile utilizzare uno o più pacchetti campione tra quelli disponibili in allegato alla tesi (reperibili tramite la repository github dedicata)

Licenza del Progetto

In linea con lo spirito di apertura e condivisione che ha guidato lo sviluppo di StoraGIS, e con l'intento di supportare ed incoraggiare ulteriori ricerche e sviluppi nel campo del software geospaziale e oltre, si è scelto di rilasciare l'intero lavoro illustrato in questa tesi sotto la GNU General Public License (GPL). Questa decisione sottolinea l'impegno dell'autore verso la promozione di un software libero e aperto, che possa essere utilizzato, modificato e distribuito liberamente da chiunque per qualsiasi scopo.

La GPL è una licenza per il software libero che garantisce agli utenti finali la libertà di eseguire, studiare, condividere e modificare il software. Rilasciando StoraGIS sotto questa licenza, assicuriamo che ogni contributo futuro al progetto, così come eventuali derivati dello stesso, rimangano accessibili alla comunità sotto gli stessi termini, promuovendo così un circolo virtuoso di condivisione della conoscenza e dell'innovazione.

L'applicazione della licenza GPL al progetto StoraGIS implica che ogni utilizzo, distribuzione o modifica del software dovrà avvenire in conformità con i termini stabiliti dalla licenza stessa, inclusa la necessità di mantenere aperto ed accessibile il codice sorgente di eventuali versioni modificate del software. Per ulteriori dettagli sulla licenza GPL e sulle sue specifiche clausole, si rimanda al testo completo della licenza disponibile sul sito ufficiale della Free Software Foundation (https://www.gnu.org/licenses/gpl-3.0.html).

Adottando la licenza GPL, auspichiamo di contribuire alla crescita di un ecosistema tecnologico aperto e collaborativo, dove il libero scambio di idee e soluzioni possa accelerare il progresso scientifico e tecnologico a beneficio di tutti.

Patrocinio ByCloud SRL

Il progetto StoraGIS è stato patrocinato da ByCloud SRL (https://bycloud.eu), una società specializzata in tecnologie informatiche, consulenza e servizi cloud, fondata da Giordano Cetti. Il supporto di ByCloud SRL ha fornito risorse cruciali per lo sviluppo e la distribuzione di StoraGIS, contribuendo a rendere il software liberamente accessibile e pronto per l'uso da parte della comunità.

ByCloud SRL ha condiviso la visione del progetto StoraGIS, incentrata sull'importanza di sviluppare soluzioni software aperte e accessibili, per favorire l'avanzamento tecnologico e sostenere la ricerca e l'innovazione nel campo geospaziale. Questo patrocinio riflette l'impegno della società nel promuovere iniziative che stimolino la collaborazione tra il mondo accademico ed il settore tecnologico.

L'associazione con ByCloud SRL ha rappresentato un elemento chiave nel successo di StoraGIS, evidenziando come la collaborazione tra enti accademici e aziende possa produrre soluzioni innovative che rispondono a reali esigenze del settore.

Codice Sorgente e Repository del Progetto

Per facilitare l'accesso al lavoro svolto ed incoraggiare la collaborazione oltre al contributo da parte della comunità scientifica e tecnologica, il codice sorgente completo di StoraGIS è disponibile pubblicamente nel suo repository GitHub. Questo

consente a ricercatori, sviluppatori e appassionati del settore di esaminare, utilizzare e contribuire al progetto in maniera diretta.

Il repository del progetto si trova all'indirizzo web: https://github.com/giordanocetti/StoraGIS. All'interno del repository sono presenti non solo il codice sorgente ma anche documentazione dettagliata sullo sviluppo, l'installazione e l'utilizzo di StoraGIS, oltre ad esempi pratici e linee guida per gli sviluppatori interessati a contribuire al progetto o a crearne derivati.

Invito tutti gli interessati a visitare il repository per esplorare il progetto, scaricare il codice, ed eventualmente, contribuire al miglioramento di StoraGIS attraverso suggerimenti, modifiche o estensioni. La collaborazione aperta è un pilastro fondamentale su cui si basa la filosofia di questo progetto, e crediamo fermamente che la condivisione delle conoscenze e delle risorse possa portare ad innovazioni significative nel campo del GIS e dello sviluppo software.

L'obiettivo di rendere StoraGIS accessibile su una piattaforma come GitHub è anche quello di stimolare una comunità attiva attorno al progetto, in grado di supportare sia gli utenti che gli sviluppatori attraverso scambio di idee, risoluzione di problemi e sviluppo di nuove funzionalità. Incoraggiamo quindi tutti gli interessati a partecipare attivamente, sia utilizzando il software per i propri progetti sia contribuendo al suo sviluppo futuro.

Note integrative

Ai ricercatori, agli studenti, o agli sviluppatori più attenti, non sarà sfuggita la scelta di implementazione di un servizio REST API esterno per la conversione e la validazione dei dati. Nel contesto attuale, abbiamo deciso di non includere tale componente, ma di lasciarne solo "intravedere" la flessibilità e la dinamicità che esso rappresenta, appunto, grazie all'astrazione di una variabile JSON con "parametri di conversione e validazione". Questa scelta mira a promuovere un'implementazione più diretta mediante il formato JSON nativo, creando, in compenso, un esempio pratico di integrazione tra sistemi. Confidiamo che tale assenza non limiti l'esperienza utente, evidenziando invece la flessibilità del progetto. La scelta di posticipare la divulgazione di tale componente serve anche come termometro per misurare l'effettivo interesse oltre al livello di partecipazione della comunità verso il progetto.

Il futuro rilascio di questa componente sotto licenza GPL sarà valutato proprio in base all'interesse dimostrato ed al coinvolgimento attivo nello sviluppo di StoraGIS. Incoraggiamo ricercatori, studenti e sviluppatori ad esplorare il progetto, contribuendo ed influenzando così le direzioni future, compresa la potenziale divulgazione di suddetta componente, che ad oggi potrebbe richiede un enorme lavoro di rifattorizzazione.

Conclusioni

Concludendo il viaggio intrapreso nella realizzazione di StoraGIS, questa tesi non solo testimonia l'efficacia dell'integrazione di tecnologie avanzate come Django Rest Framework, Geoserver, e Openlayers per rispondere a complesse esigenze nel campo geospaziale, ma sottolinea anche un approccio metodologico volto all'ingegnerizzazione essenziale del software.

Il lavoro svolto su StoraGIS è stato deliberatamente ingegnerizzato ai minimi termini, con l'obiettivo di eliminare tutte quelle sovrastrutture tipicamente presenti nei software in produzione. Questa scelta è stata guidata dalla volontà di offrire anche agli studenti più giovani un punto di partenza solido e snello, una base da cui poter esplorare non solo il vasto mondo del GIS, ma anche lo sviluppo di software in generale. In quest'ottica, StoraGIS si propone come un modello di riferimento per l'elaborazione di soluzioni software che, pur radicate in specifiche esigenze di natura geospaziale, possano estendere la loro applicabilità a contesti diversi.

L'implementazione dei flussi di caricamento, approvazione, e visualizzazione dei progetti georeferenziati ha rivelato il potenziale di StoraGIS come supporto decisionale nelle mani degli operatori del settore infrastrutturale, sottolineando l'importanza di un accesso facilitato a dati geospaziali accurati e in tempo reale per la pianificazione e l'ottimizzazione dei progetti.

Guardando al futuro, il percorso di sviluppo di StoraGIS non si ferma qui. Il dinamismo del settore GIS e delle tecnologie web apre a un panorama di sviluppi futuri entusiasmanti, dall'integrazione di nuove fonti di dati all'espansione delle

capacità di analisi spaziale, fino all'esplorazione di applicazioni di intelligenza artificiale per la previsione e l'ottimizzazione infrastrutturale.

Un'ulteriore evoluzione prevedibile riguarda l'adozione di standard aperti e la promozione della collaborazione tra diversi attori del settore, passi che potrebbero arricchire l'ecosistema di StoraGIS e incoraggiare la formazione di una comunità attiva di utenti e sviluppatori.

In sintesi, il lavoro svolto su StoraGIS rappresenta un contributo significativo verso l'obiettivo di dotare il settore delle infrastrutture georeferenziate di strumenti tecnologici avanzati e facilmente accessibili. StoraGIS emerge come un caso esemplare di come la tecnologia, ingegnerizzata con un approccio essenziale e inclusivo, possa non solo migliorare la gestione del territorio ma anche ispirare nuovi percorsi di ricerca e sviluppo nel vasto universo del software, oltre i confini del GIS.

Nota sul trattamento dei dati personali

Nel contesto della presente ricerca e nell'impiego di servizi esterni per il processamento dei dati, adottiamo un approccio rispettoso della privacy e della sicurezza dei dati personali, in ottemperanza al Regolamento Generale sulla Protezione dei Dati (GDPR). Ci impegniamo a trattare i dati personali con la massima attenzione, seguendo i principi di legalità, correttezza e trasparenza.

Per ulteriori dettagli sulle nostre politiche di privacy, vi invitiamo a consultare la Dichiarazione sulla Privacy completa al seguente indirizzo: https://bycloud.eu/dichiarazione-sulla-privacy-ue/ .

Specificatamente, nel contesto del servizio di processamento dati legato a StoraGIS, vengono trattati in modo sicuro e anonimizzato i seguenti dati:

- Indirizzo IP dell'istanza StoraGIS remota
- Dati dei pacchetti caricati secondo la specifica
- Parametri della specifica (es: CT1)
- Data e ora di caricamento

L'utilizzo dell' "utenza generale" per la fruizione del servizio di conversione, potrebbe comportare la condivisione involontaria di informazioni come l'orario di caricamento e i parametri di mapping, nulla di più. Tali informazioni, pur essendo trattate con la massima cura per garantirne la sicurezza, rientrano in un contesto di utilizzo pubblico che ne consente l'inevitabile visibilità.

Garantiamo l'adozione di tutte le misure necessarie per proteggere i dati raccolti, limitando l'accesso ai soli soggetti autorizzati e procedendo alla loro anonimizzazione quando possibile. Il trattamento dei dati ha l'unico scopo di fornire e migliorare il servizio, escludendo qualsiasi finalità di profilazione o marketing diretto.

Ringraziamenti

Desidero esprimere la mia più profonda gratitudine all'Università degli Studi eCampus per avermi fornito le conoscenze, le opportunità ed il sostegno necessari per perseguire e realizzare i miei obiettivi accademici e professionali. Questo percorso formativo, arricchito da un ambiente stimolante e da docenti eccezionali, ha rappresentato un pilastro fondamentale per il mio sviluppo personale.

Un ringraziamento speciale va a tutte le persone a me care: famiglia, amici e mentori, che mi hanno accompagnato, sostenuto e creduto in me lungo tutto il cammino. Il loro supporto incrollabile, i loro consigli ed il loro incoraggiamento sono stati essenziali per superare le sfide e raggiungere i traguardi che mi sono prefissato.

Desidero inoltre riconoscere l'importante ruolo che Infratel Italia S.p.A. ed Invitalia S.p.A. hanno avuto nel mio percorso professionale. Lavorare a fianco di queste istituzioni nel settore della pubblica amministrazione mi ha permesso di contribuire concretamente al progresso tecnologico e alla digitalizzazione del Paese, ottenendo risultati significativi che hanno arricchito ulteriormente la mia esperienza ed il mio impegno nel mondo tecnologico.

Questo viaggio non sarebbe stato lo stesso senza il contributo e il sostegno di ognuno di voi. A tutti voi, il mio più sincero ringraziamento per aver reso possibile questo percorso e per avermi accompagnato verso il raggiungimento di questo importante traguardo.

Bibliografia e sitografia

[1] Environmental Systems Research Institute, Inc. <<*ESRI Shapefile Technical Description*>> Available: https://support.esri.com/en-us/technical-paper/esri-shapefile-technical-description-279

[2] Y. Shafranovich, 2005 <<*Common Format and MIME Type for Comma-Separated Values (CSV) Files*>> Available: https://www.rfc-editor.org/rfc/rfc4180

[3] P. Ralph, «*The Sensemaking-coevolution-implementation theory of software design*» Science of Computer Programming, n. 101, pp. 21-41, 17 February 2013.

[4] D. Patterson e A. Fox, Engineering Software as a Service: An Agile Approach Using Cloud Computing., Berkeley: Strawberry Canyon LLC, 2013.

[5] C. Trudeau, Real Python << *Building With Django REST Framework*>> Available: https://realpython.com/lessons/building-drf-overview/ 2022.

[6] DavidMM, Github at https://github.com/david1707 Available: https://letslearnabout.net/blog/what-is-django-rest-framework-and-why-you-should-learn-it/ 2019.

[7] Tom Christie, PyPy (Python Software Fundation) Available: https://pypi.org/project/djangorestframework/ 2022.

[8] Red Hat, Inc. << Cos'è un'API REST? >> Available: https://www.redhat.com/it/topics/api/what-is-a-rest-api 2020.

[9] @osgeo, The Open Source Geospatial Foundation http://live.osgeo.org/it/overview/geoserver_overview.html

[10] Open Geospatial Consortium *https://www.ogc.org/about-ogc/* https://www.ogc.org/standards/ 1999-2023.

[11] GISGeography, citato in più di 600 articoli di ricerca (Fonte: Google Scholar) *https://gisgeography.com/about-us/ <<An Introduction to Web Mapping Services (WMS)>>* https://gisgeography.com/web-mapping-services-wms/ 2023.

[12] @osgeo, The Open Source Geospatial Foundation https://www.osgeo.org/projects/openlayers/

[13] Geoapify GmbH, P.&A. Tarasenko *https://www.geoapify.com/about-us <<Leaflet vs OpenLayers. What to choose?>>* Available: https://www.geoapify.com/leaflet-vs-openlayers 2019.

[14] Peter P. Chen, Computer Science Department at Louisiana State University, *<<Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned>>* Available: https://csc.lsu.edu/~chen/pdf/Chen_Pioneers.pdf

[15] GISGeography, citato in più di 600 articoli di ricerca (Fonte: Google Scholar) *https://gisgeography.com/about-us/ <<ArcGIS Shapefile Files Types & Extensions>>* Available: https://gisgeography.com/arcgis-shapefile-files-types-extensions/

[16] PostGIS PSC & OSGeo, *<<PostGIS extends the capabilities of the PostgreSQL>>* Available: https://postgis.net/

[17] J. Azimjonov, Cornell University, <<*Rule Based Metadata Extraction Framework from Academic Articles*>> Available: https://arxiv.org/abs/1807.09009 2018.

[18] Pagine Bianche, Italiaonline S.p.A. Available: https://www.paginebianche.it/codice-istat 2023.

[19] Django Software Foundation, <<*Documentation, Models*>> Available: https://docs.djangoproject.com/en/4.2/topics/db/models/ 2023.

[20] Django Software Foundation, <<*Documentation, Using the Django authentication system*>> Available: https://docs.djangoproject.com/en/4.2/topics/auth/default/ 2023.

[21] Burak Guzel, Computer Science and Engineering from The Ohio State University, EnvatoTuts <<*SQL per Principianti: Parte 3 - Relazioni del Database*>> Available: https://code.tutsplus.com/it/sql-for-beginners-part-3-database-relationships--net-8561a

[22] P.Kute, DEV Community, <<*What is supervisord?*>> Available: https://dev.to/pratapkute/what-is-supervisord-1omf 2021.

[23] Mozilla Developer Network, <<*What is a web server?*>> Available: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server 2023.

[24] RV Sharma, Red Hat Developer <<*What is a Socket?*>> Available: https://developers.redhat.com/blog/2017/10/17/what-is-a-socket 2017.

[25] Università di Roma La Sapienza <<*File binari*>> Available: https://www.diag.uniroma1.it/~liberato/tecniche/binari/index.shtml

[26] Jody Garnett, Geoserver Fundation, <<*GeoWebCache*>> Available: https://docs.geoserver.org/stable/en/user/geowebcache/index.html

[27] VMware, Inc. <<*Cos'è la virtualizzazione?*>> Available: https://www.vmware.com/it/solutions/virtualization.html 2023.

[28] IONOS SE, <<*Che cos'è l'SSH? La secure shell spiegata in modo semplice*>> Available: https://www.ionos.it/digitalguide/server/tools-o-strumenti/ssh/ 2020.

[29] VMware, Inc. <<*Configurazione della topologia della CPU di una macchina virtuale*>> Available: https://docs.vmware.com/it/VMware-vSphere/8.0/vsphere-vm-administration/GUID-19925544-A04A-4488-9095-40E40EFA93B4.html

[30] Python Software Fundation, <<*Creation of virtual environments*>> Available: https://docs.python.org/3/library/venv.html 2023.

[31] S. Hudaib, DEV Community, <<*Why Virtualenv is Important for Django Development*>> Available: https://dev.to/sarahhudaib/why-virtualenv-is-important-for-django-development-58fp 2023.

[32] B. Solomon, DEV Community, Real Python Tutorial Team <<*What Are Python Wheels and Why Should You Care?*>> Available: https://realpython.com/python-wheels/ 2020.

[33] Django Software Foundation, <<*Writing your first Django app, part 1*>> Available: https://docs.djangoproject.com/en/4.2/intro/tutorial01/ 2023.

[34] The SQLite Consortium, <<*About SQLite*>> Available: https://www.sqlite.org/about.html 2023.

[35] Django Software Foundation, << *Migrations* >> Available: https://docs.djangoproject.com/en/5.0/topics/migrations/ 2023.

[36] Django Software Foundation, << *Settings* >> Available: https://docs.djangoproject.com/en/4.2/ref/settings/ 2023.

[37] Ubuntu Community, << *Official Ubuntu Documentation* >> Available: https://help.ubuntu.com/ 2023.

[38] SystemD << *System and Service Manager* >> Available: https://systemd.io/ 2023.

[39] RFC, << *Hypertext Transfer Protocol -- HTTP/1.1* >> Available: https://www.rfc-editor.org/rfc/rfc2616 2023.

[40] Django Software Foundation, << *Writing views* >> Available: https://docs.djangoproject.com/en/4.2/topics/http/views/ 2023.

[41] Django Software Foundation, << *ViewSets & Routers* >> Available: https://www.django-rest-framework.org/tutorial/6-viewsets-and-routers/ 2023.

[42] Django Software Foundation, << *Serializers* >> Available: https://www.django-rest-framework.org/api-guide/serializers/ 2023.

[43] Django Software Foundation, << *Models* >> Available: https://docs.djangoproject.com/en/5.0/topics/db/models/ 2023.

[44] Krishna Lodha << Style Points in Geoserver >> Available: https://www.linkedin.com/pulse/style-points-geoserver-krishna-lodha 2022.

Appendice A – QUICKSTART AWS AMI IMAGE

Con l'obiettivo di semplificare la fase di inizializzazione di un proprio sistema storagis definibile come self hosted, è stata messa a disposizione un'immagine AMI AWS pubblica che aiuterà ricercatori e studenti a compiere con semplicità i primi passi verso il rilascio della propria versione di StoraGIS da personalizzare.

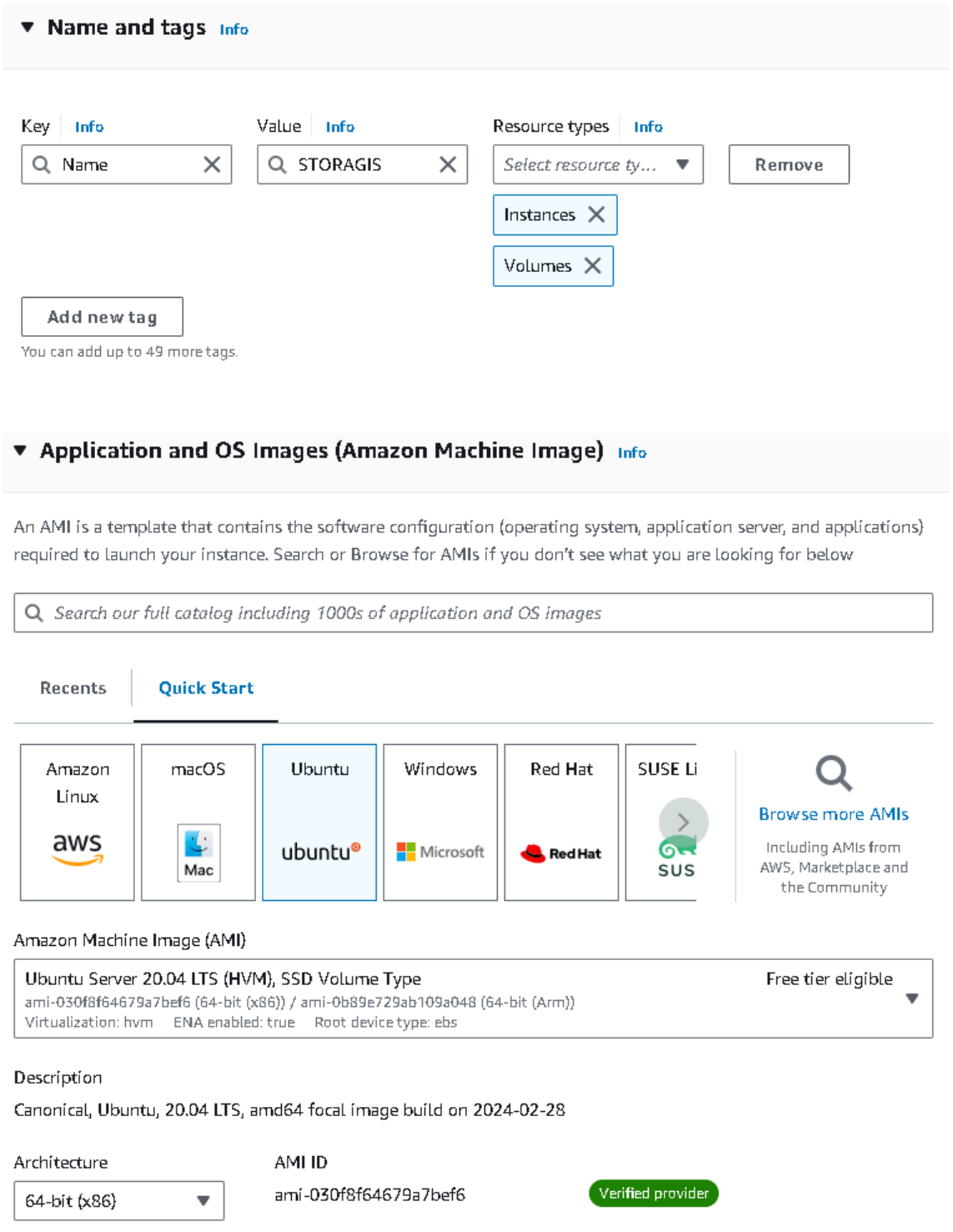
In sostanza, per la creazione del template, è stata avviata un'istanza Ubuntu 20.04 con le caratteristiche esposte di seguito, ridotte con l'obiettivo di mantenere il servizio GRATUITO ovvero, sfruttando il Free Tier AWS.

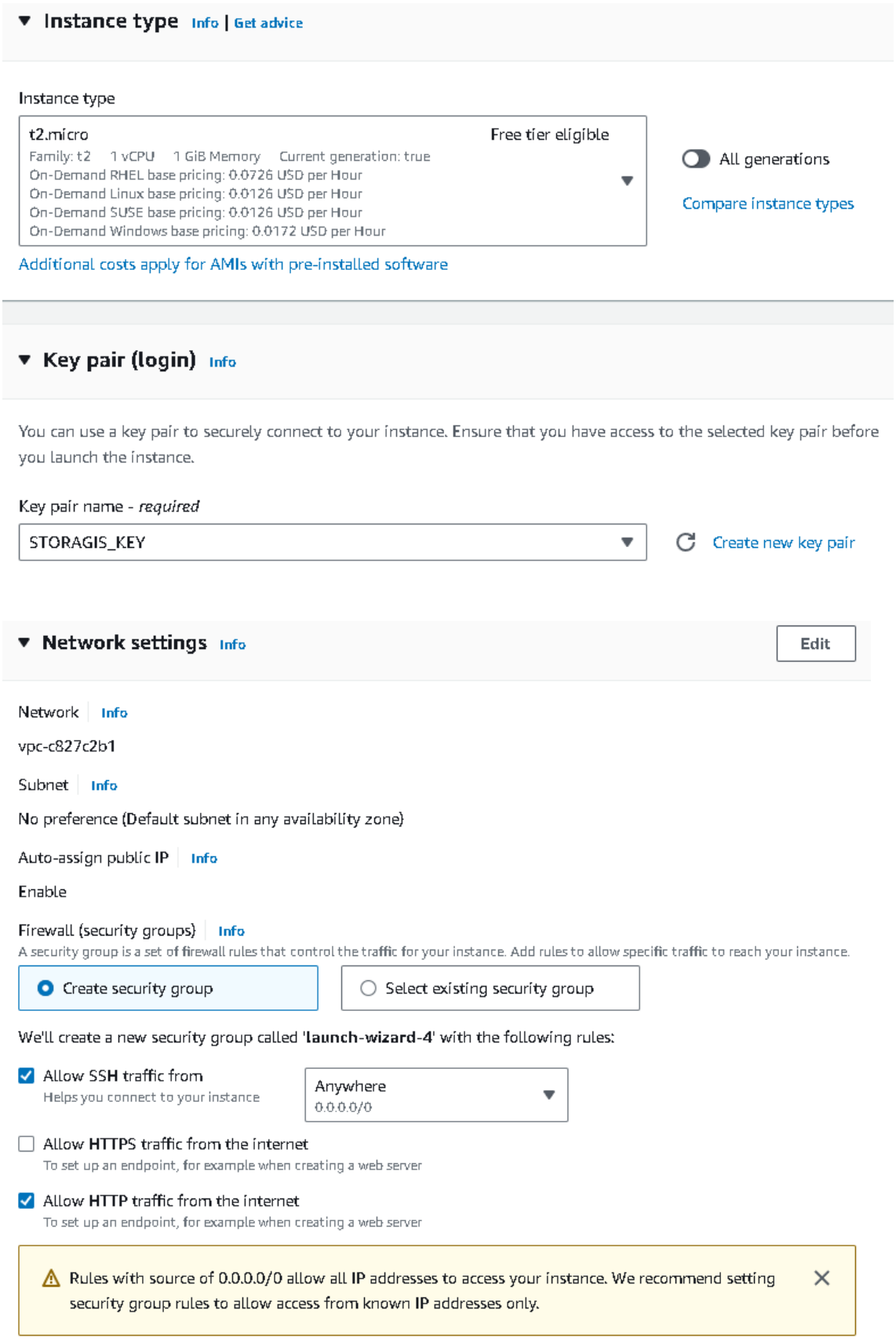
- Famiglia: t2.micro
- Connettività: pubblica (IP PUBBLICO)
- Storage: base 20GB
- Regole di sicurezza: SSH, HTTP
- Parametri di accesso ssh: CHIAVE PRIVATA

Il tipo di istanza gratuita (t2.micro), per un breve test, è possibile.

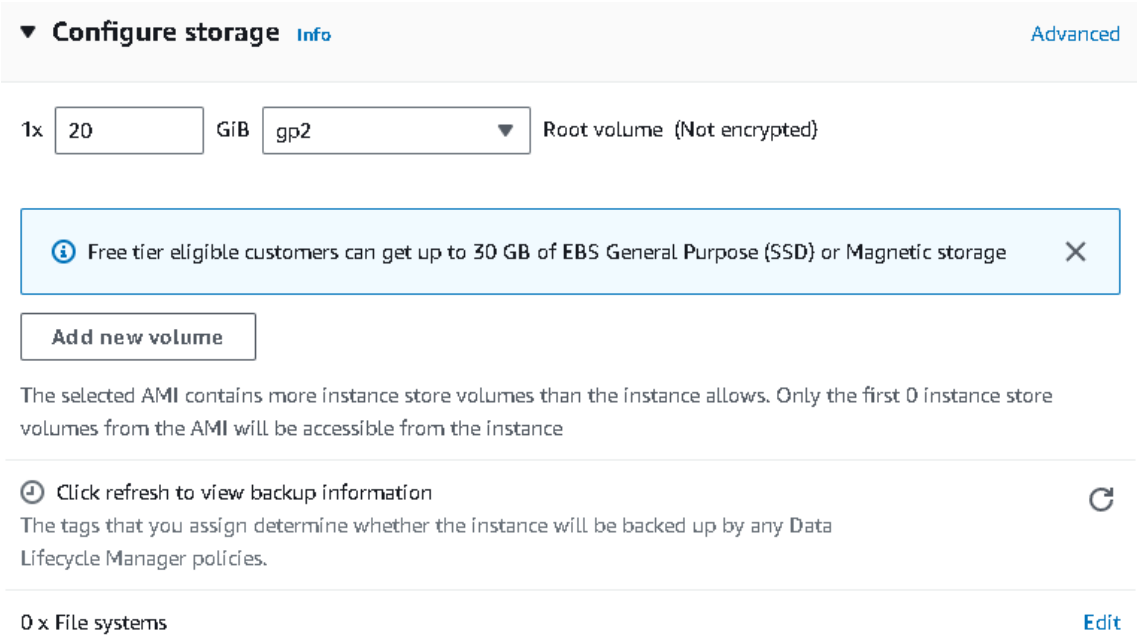
Tuttavia, potrebbero presentarsi problemi in fase di visualizzazione dei dati. Per un test più consistente, si consiglia di utilizzare almeno un t2.medium

PARTE 1 – Modalità di realizzazione TEMPLATE





NB: aprire anche la porta 8000 per eseguire un test di avvio manuale DRF.

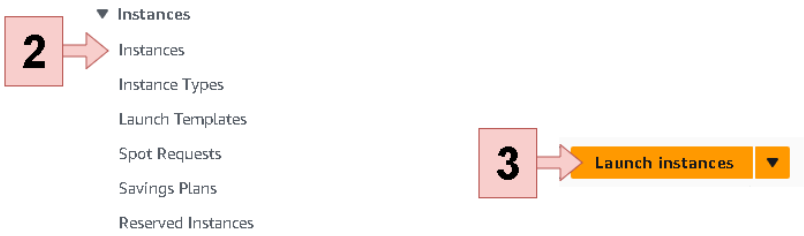


Una volta avviata l'istanza, è sufficiente entrare usando il protocollo SSH ed eseguire l'operazione documentata nella ricerca sperimentale, che, se correttamente eseguita, condurrà al risultato atteso.

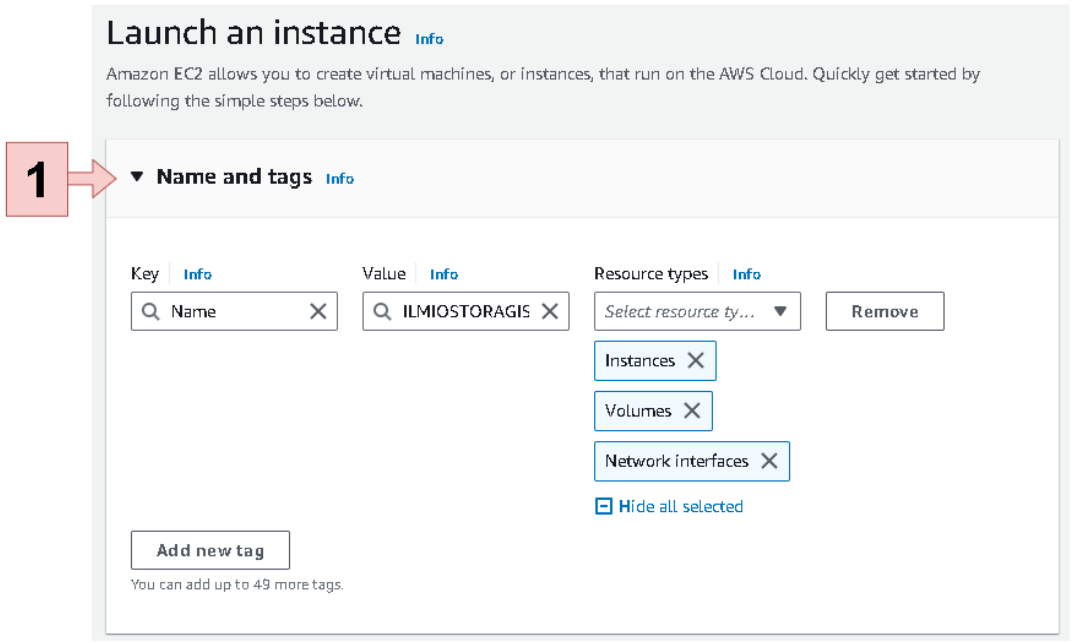
PARTE 2 – Modalità di utilizzo TEMPLATE

Per il completamento della seguente procedura, si presume che si abbia già a disposizione un account root AWS o comunque un account in grado di operare sulle risorse di virtualizzazione EC2.

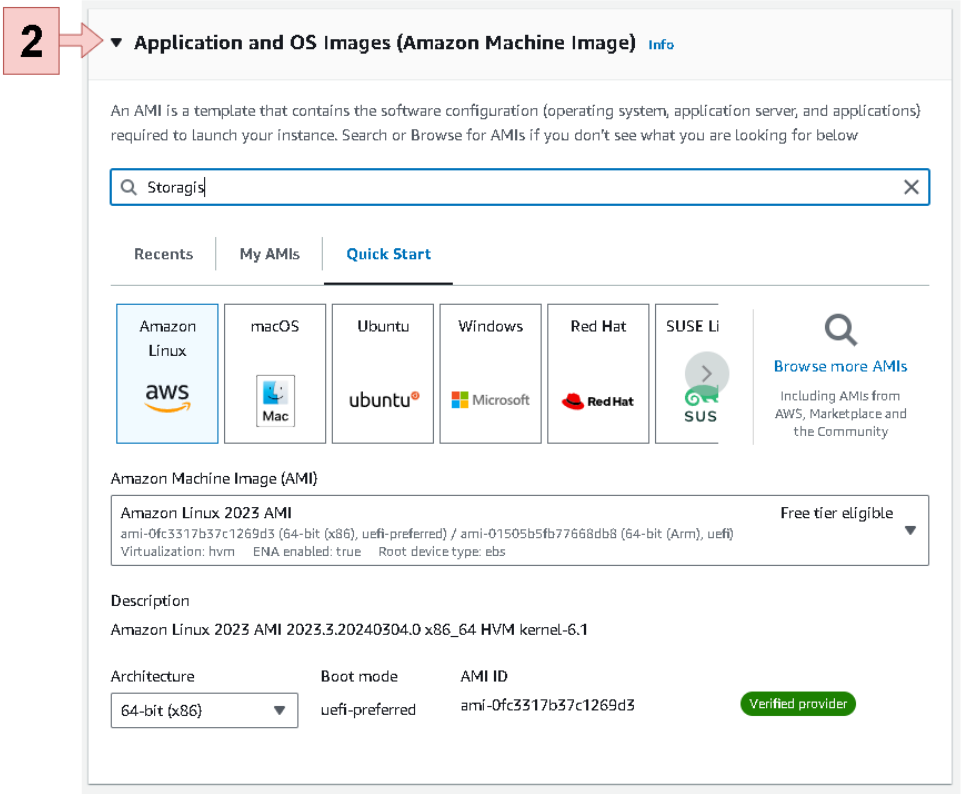
Una volta autenticati su AWS, assicurarsi di aver selezionato in alto la regione Irlanda (eu-west-1) e procedere come di seguito:



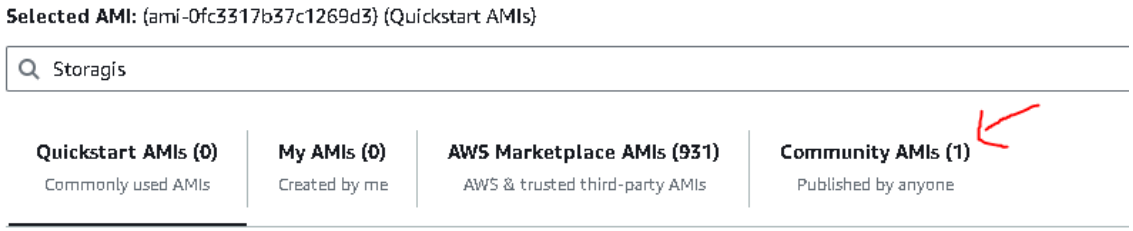
Impostare, a piacimento, i parametri di identificazione dell'istanza.



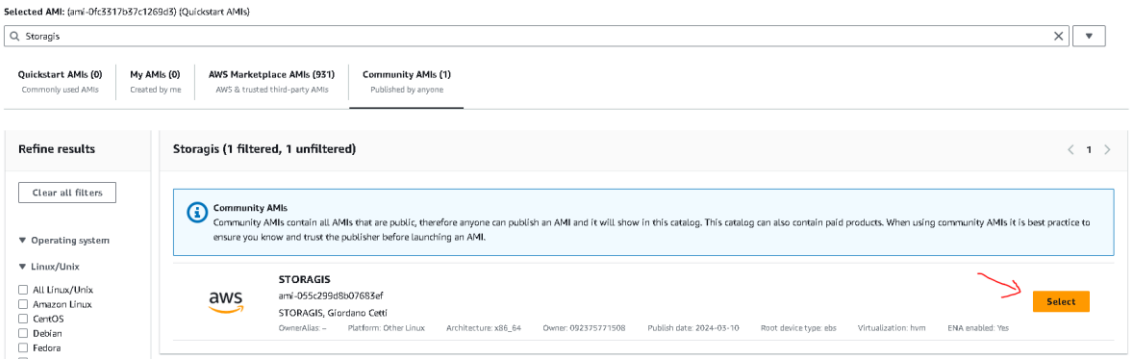
Cercare quindi, nella barra di ricerca, Storagis.



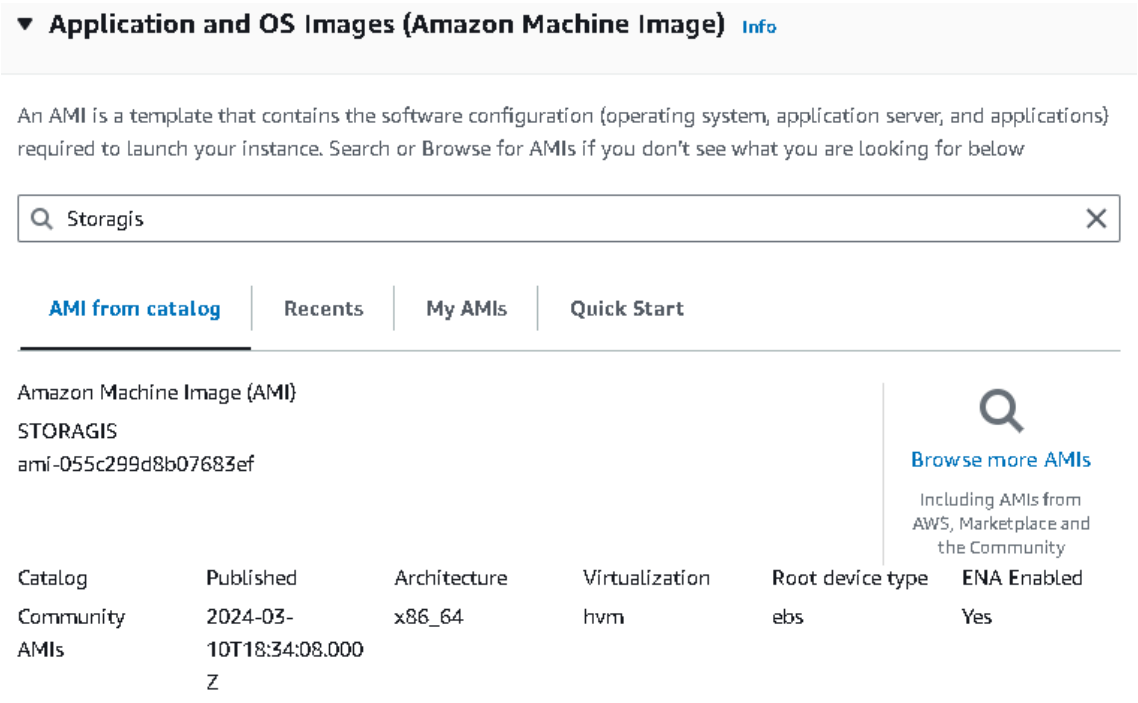
Selezionare “Community AMIs”



Selezionare Select all’altezza dell’ami con il seguente ID: ami-055c299d8b07683ef

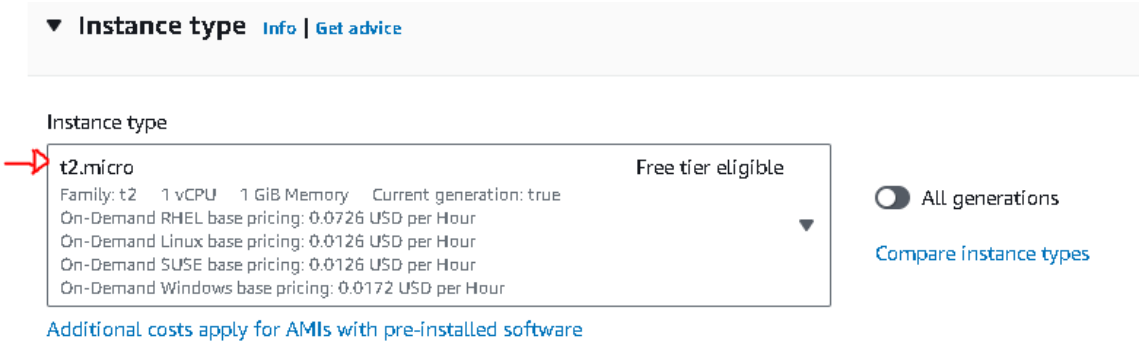


Di seguito il risultato, una volta selezionata la AMI corretta:

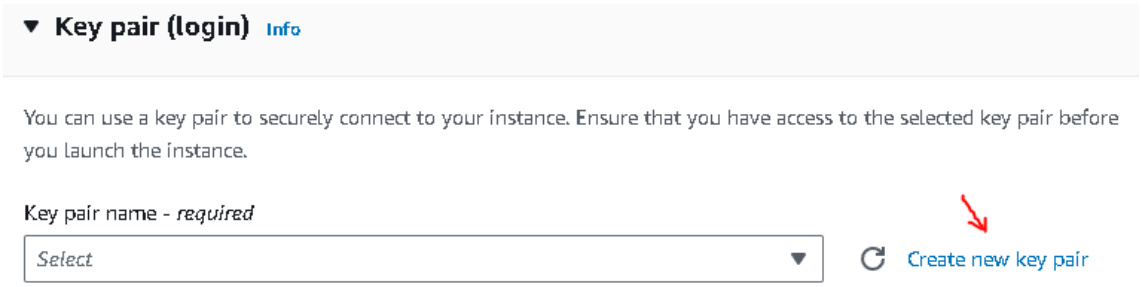


Selezionare il tipo di istanza, per un breve test, è possibile usare la famiglia t2.micro

Tuttavia, potrebbero presentarsi problemi in visualizzazione dei dati. Per un test più consistente, si consiglia di utilizzare almeno un t2.medium

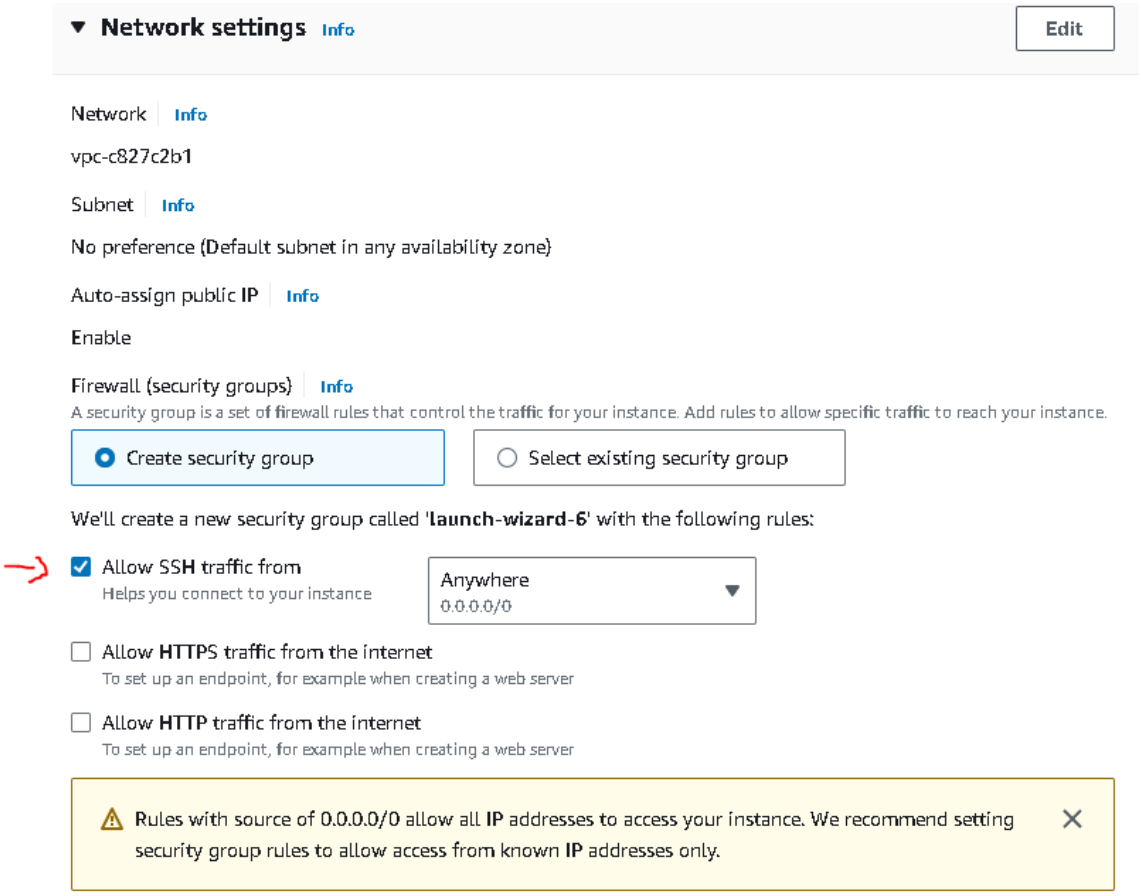


Creare la propria chiave, o selezionarne una preesistente dalla lista. Questa sarà necessaria per l'accesso SSH e la configurazione finale.

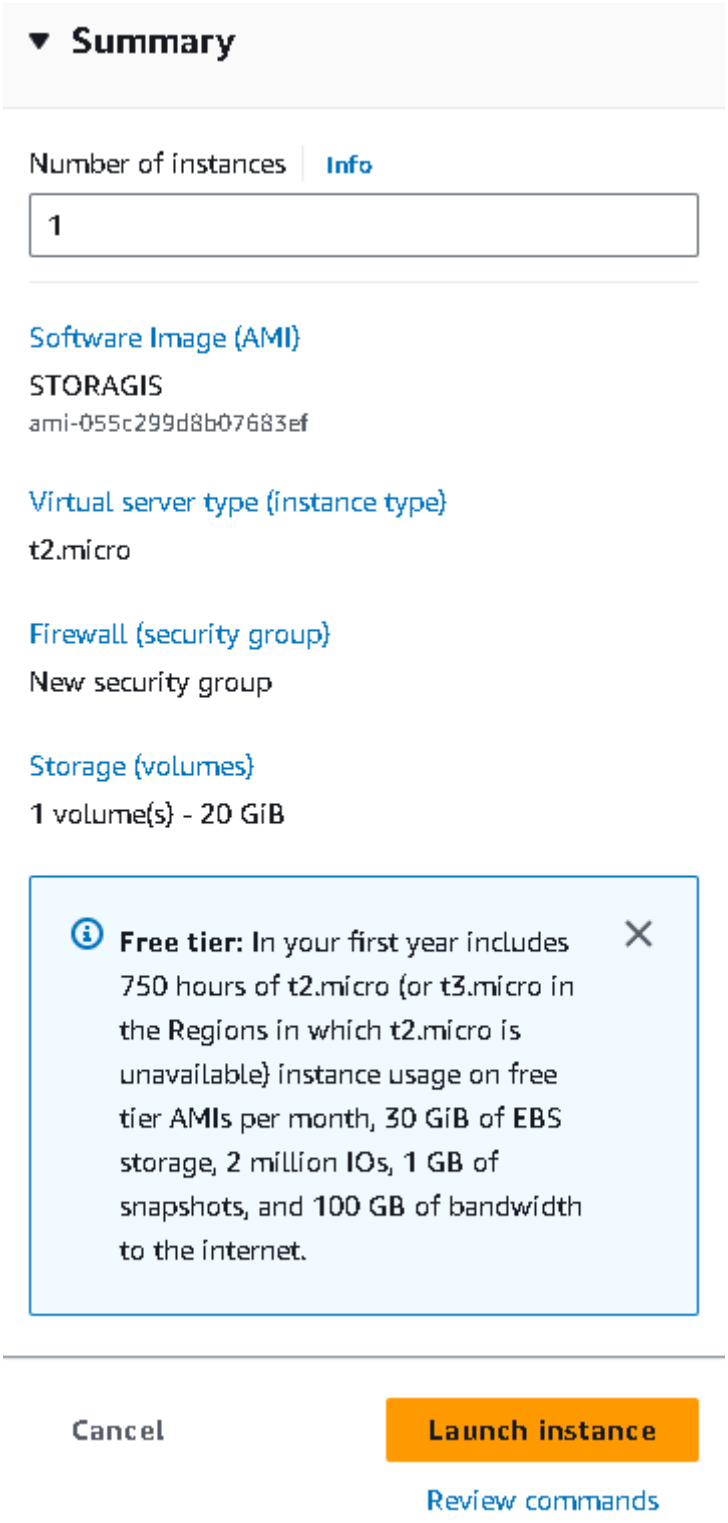


Nel passaggio successivo si configura l'accessibilità di rete.

E’ preferibile non selezionare altro che Allow SSH traffic from Anywhere e lasciar chiuso, fino a cambio credenziali, l’accesso http.



Per il resto, sarà sufficiente lasciare tutto invariato, e procedere con il lancio dell'istanza.



Appena l'istanza risulterà avviata correttamente, si annoti l'indirizzo della macchina, si potrà usare sia l'IP che il public IPv4 DNS. Si proceda, quindi, ad effettuare l'accesso al terminale usando il protocollo SSH con il proprio client preferito usando il nome di rete, la porta SSH standard, il nome utente predefinito ubuntu, e la chiave privata selezionata o creata in fase di lancio istanza.

Una volta effettuata la connessione, procedere con i seguenti passaggi:

```
sudo su –
cd /opt/storagis/
source env/bin/activate
python manage.py changepassword admin
```

Impostare, quando richiesto, la nuova password dell'utente admin: unico amministratore nel template in grado di creare nuovi utenti applicativi.

Adeguare le impostazioni DRF, includendo il proprio nome di rete (IP/FQDN) usato per l'accesso SSH, tra i consentiti (ALLOWED_HOSTS) senza includere http:// nè altri caratteri speciali, ed eventualmente disattivare la modalità DEBUG:

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = False

ALLOWED_HOSTS = ['ec2-34-241-193-114.eu-west-1.compute.amazonaws.com']
```

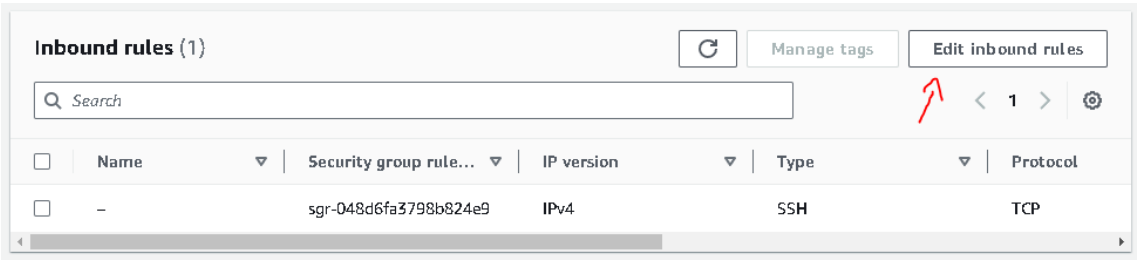
Avviare o riavviare le componenti StoraGIS con i seguenti comandi (sempre da root):

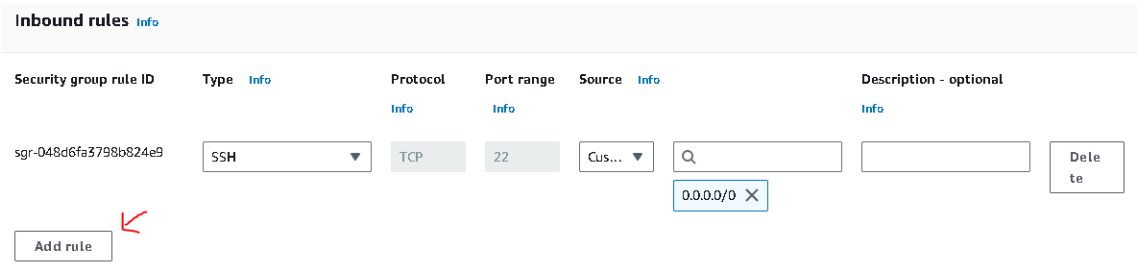
```
systemctl restart gunicorn.socket gunicorn.service nginx.service
processoprogetti.service geoserver.service
```

Rendere automatico l'avvio delle componenti Storagis:

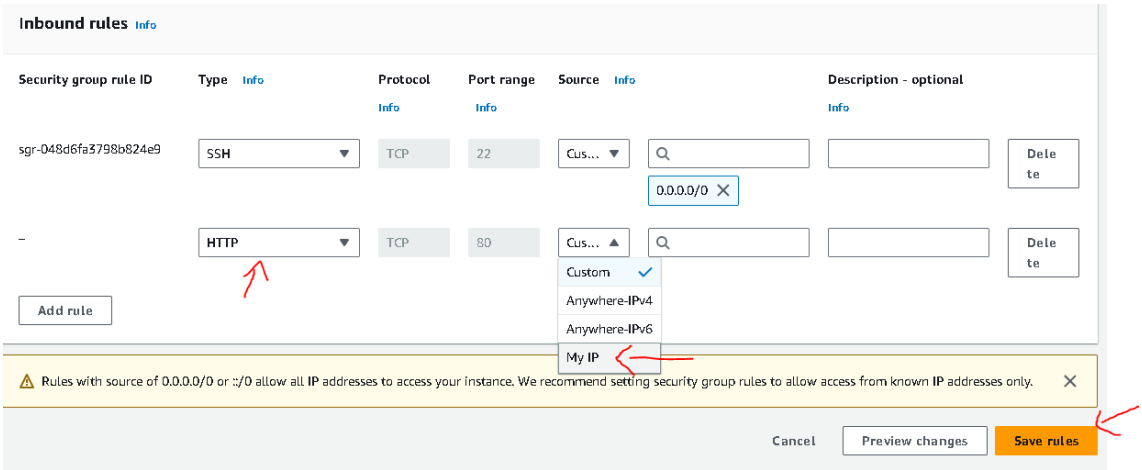
```
systemctl enable gunicorn.socket gunicorn.service nginx.service
processoprogetti.service geoserver.service
```

Rientrare in AWS, e modificare il security group associato all'istanza. Consentire il traffico sulla porta HTTP ma specificare il proprio indirizzo come unica sorgente consentita, come di seguito illustrato.





Selezionare HTTP nella tipologia, My IP nella sorgente, e lasciar completare il campo automaticamente ad AWS. Dopodichè salvare la regola.

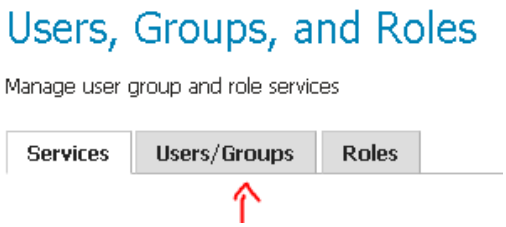


Procedere quindi con il cambio password dell'utente amministratore geoserver. Utilizzando lo stesso nome di rete (IP/FQDN) usato per l'accesso SSH, aprire un browser e visitare la seguente URL: http://<NOME_DI_RETE>/geoserver

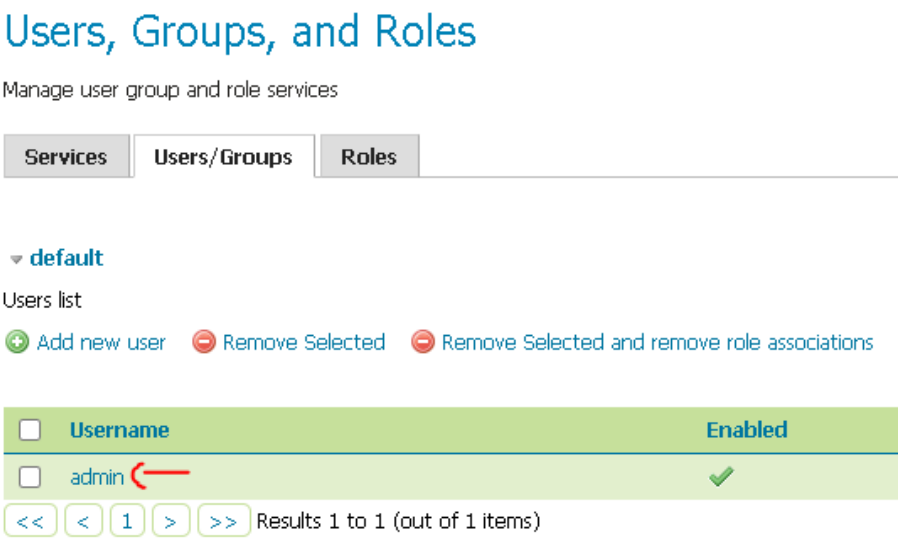
Le credenziali predefinite per l'accesso sono: admin / geoserver

Nel menu laterale, selezionare Users, Groups, Roles

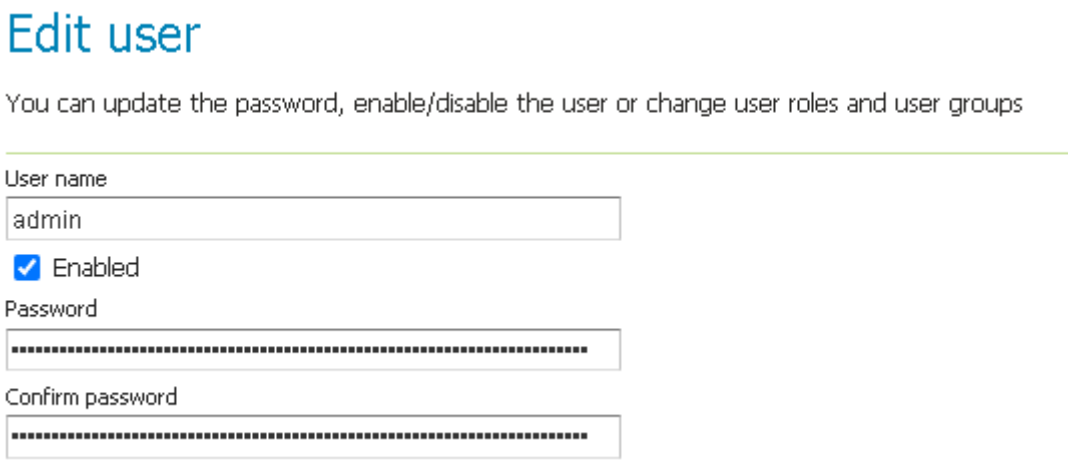
Quindi selezionare il tab in alto User/Groups



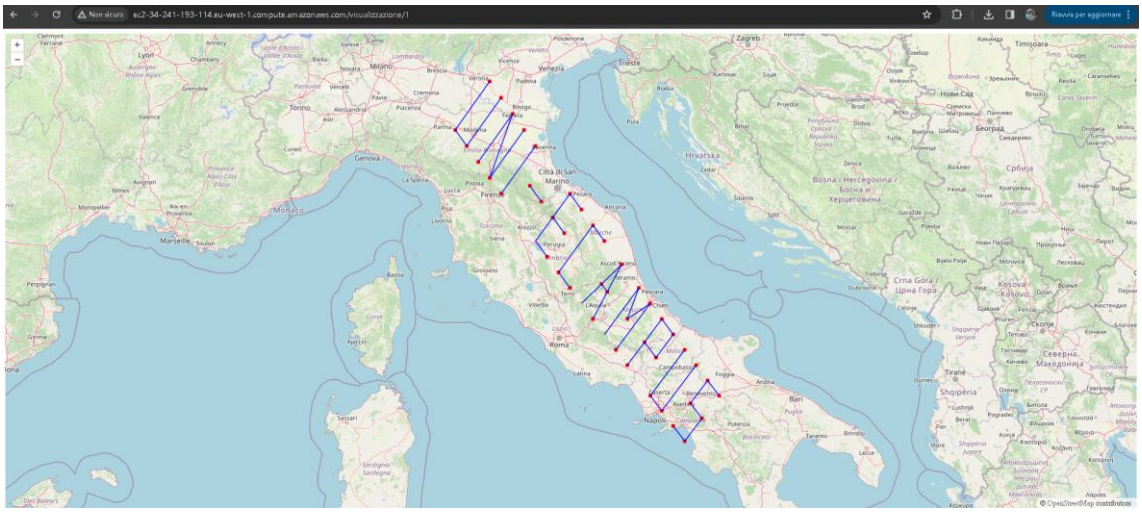
Cliccare sull'utenza admin



Senza modificare altro, impostare una propria password complessa e cliccare SAVE



Ora è possibile caricare nuovi progetti, o visualizzare il progetto base incluso nel template.



Appendice B – Accesso al progetto SAAS

Grazie al patrocinio ByCloud SRL, è possibile accedere direttamente al risultato della sperimentazione al seguente link: https://storagis.bycloud.eu usando l'utenza pubblica user/pass:

utenzagenerale
%+WCk6r03B"T:1;^>8p=8\?NE1CRjH-H

Si ricorda che è sempre possibile richiedere, senza alcun costo e senza dover rispondere ad alcun quesito, un'utenza personalizzata. In tal caso si rammenta che l'eventuale uso improprio potrebbe portare a cessazione del servizio, anche dell'utenza generale stessa.

Si ricorda inoltre, che i servizi SAAS sono due distinti, ovvero:

- PROCESSING – proprietario.
- STORAGIS – GPL.

E che l'utenza personale verrebbe erogata per entrambe le piattaforme. Le credenziali della piattaforma di Processing si inseriscono all'interno del batch processing della versione StoraGIS GPL (sostituendo utenzagenerale), mentre le credenziali della piattaforma GPL si possono utilizzare direttamente nel browser.

Appendice C – Pacchetti CT1 sperimentali, pronti da usare

I pacchetti CT1 usati per i TEST iniziali, sono 2 al momento della pubblicazione:

- ct1_test_01.zip
- ct1_uniecampus.zip

E sono reperibili al seguente LINK:

https://drive.google.com/drive/folders/1w6McEqhmXUKkfX7uH8pl-wj_S2qJbBcb?usp=sharing