

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

**Sviluppo di un'applicazione web per la
valutazione dei risultati ottenuti
da un sistema di visione**

Relatore:
Chiar.mo Prof.
GABBRIELLI MAURIZIO

Presentata da:
DI CESARE ERIKA

Correlatore:
CONVERTINO EDOARDO

**III Sessione: Marzo 2023
Anno Accademico 2021/22**

Alla mia famiglia e ai miei amici...

Indice

Introduzione	1
Progetto	2
Tecnologie utilizzate	2
1 Analisi generale	5
1.1 Contesto di utilizzo	5
1.2 Architettura software	6
2 Strumenti di sviluppo	9
2.1 CVAT	9
2.1.1 API REST CVAT	11
2.2 FiftyOne	13
2.2.1 Plugin	14
2.3 MySQL	14
2.4 Flask	14
2.4.1 Lato Server	16
2.4.2 Lato Client	19
2.4.3 Jinja	20
3 Funzionalità richieste	23
3.1 Software MVE	24
3.2 Plugin FiftyOne	27

4	Implementazione	29
4.1	Configurazione CVAT	29
4.2	Configurazione FiftyOne	30
4.3	Database	33
4.4	Software MVE	37
4.5	Plugin FiftyOne	59
5	Guida operativa	63
5.1	Creazione progetto MVE	63
5.2	Creazione progetto CVAT	65
5.3	Caricamento immagini	66
5.4	Caricamento verità	70
5.5	Caricamento predizioni	71
5.6	Confronto	73
5.7	FiftyOne	75
	Conclusioni	81
	Bibliografia	83

Elenco delle figure

1.1	Architettura progetto	7
2.1	Schema API REST CVAT	10
4.1	Modello relazionale database	34
4.2	Esempio di file con verità salvati su database	35
4.3	Esempio di file con predizioni salvati su database	36
4.4	Esempio rappresentazione dei dati per lo scaricamento della verità	52
5.1	Homepage	63
5.2	Nuovo progetto MVE	64
5.3	Homepage	65
5.4	Nuovo progetto CVAT	65
5.5	Homepage	66
5.6	Progetto CVAT	67
5.7	Caricamento immagini task	68
5.8	Progetto CVAT con task completato	69
5.9	Scelta azione per caricamento verità	70
5.10	Caricamento file di verità	71
5.11	Scelta azione per caricamento predizioni	71
5.12	Caricamento primo file di predizioni	72
5.13	Scelta azione per confronto	73
5.14	Confronto tra verità e predizione	74

5.15 Homepage FiftyOne con dataset appena creato	75
5.16 Selezione di un campione	76
5.17 Selezione di un campione - zoom	77
5.18 Selezione delle proprietà da visualizzare nel grafico	78
5.19 Visualizzazione grafica di una proprietà verità	78
5.20 Confronto grafico tra i valori di una proprietà	79
5.21 Click su un punto e selezione del relativo campione	79

Elenco delle tabelle

3.1	Operazioni CRUD per entità.	25
4.1	API per eseguire il login e restituire il token REST se le credenziali sono valide e autenticate	44
4.2	API per creare un nuovo progetto	45
4.3	API per creare un nuovo task vuoto senza immagini	45
4.4	API per allegare in modo permanente immagini a un task	46
4.5	API per restituire un elenco impaginato di progetti	46
4.6	API per restituire informazioni sui task del progetto con l'id selezionato	47
4.7	API per restituire i dettagli di un progetto specifico	47
4.8	API per eliminare un progetto specifico	47
4.9	API per eliminare un task specifico	48
4.10	API per esportare il task come dataset in un formato specifico	48

Introduzione

L'azienda CNI Informatica S.R.L.¹ ad Alfonsine (RA) opera dal 1976 nell'ambito dell'automazione industriale di macchine e impianti, sviluppando soluzioni software e hardware su misura per il cliente in differenti settori industriali (legno, vetro, marmo, plastica, calzaturiero, metallurgico, agroalimentare, packaging). CNI Group è organizzato in **4 Business Unit** che collaborano tra loro nella progettazione e realizzazione dell'automazione di macchine e di impianti:

- **Automazione:** software di Supervisione, CAD/CAM, Ottimizzatori, PLC per la gestione di macchine e linee di lavorazione automatiche (anche robotizzate).
- **Controlli numerici:** Sistemi a controllo numerico per applicazioni su centri di lavoro e linee.
- **Elettronica:** Apparecchiature elettroniche “embedded” dedicate e personalizzate.
- **Visione:** Sistemi di misura dimensionale e di verifica della qualità (difetti) su prodotti/lavorazioni.

Nell'ambito dell'unità di Visione, il progetto presentato nell'elaborato ha come obiettivo quello di creare un ambiente unificato per la validazione e la valutazione degli algoritmi di visione, permettendo di definire una procedura standard basata su di esso.

Progetto

La prima fase dello sviluppo del progetto è stata l'analisi dei requisiti che il software deve soddisfare, seguita dalla scelta delle tecnologie da utilizzare per l'implementazione. Il principale requisito preliminare riguarda gli strumenti ed è quello di poter utilizzare CVAT per l'archiviazione e l'annotazione di immagini e FiftyOne per l'esplorazione e il confronto dei risultati, da cui è seguito uno studio delle interfacce fornite dai due strumenti per l'integrazione con un applicativo esterno: CVAT offre una suite di API REST, mentre FiftyOne mette a disposizione una libreria Python e la possibilità di sviluppare e integrare plugin custom. La successiva fase è consistita nell'effettiva progettazione e implementazione del plugin per FiftyOne e del software, chiamato *Machine Vision Evaluator* (MVE); quest'ultimo è stato realizzato sotto forma di applicazione web strutturata e modellata in modo da poter essere integrata al meglio con CVAT e FiftyOne. In questo modo l'azienda potrà verificare velocemente il risultato dell'elaborazione immagini applicata durante il processo di controllo qualità delle linee di produzione, e creare e integrare i dataset interni utilizzati per la validazione degli algoritmi di visione.

Tecnologie utilizzate

Per la realizzazione del progetto sono stati utilizzate le seguenti tecnologie e software:

- CVAT (API REST)
- FiftyOne (React)
- Flask (Python, SQL, JavaScript, HTML, CSS, Bootstrap, Jinja)
- MySQL

Nel Capitolo 2 verrà fatta un'analisi più approfondita delle loro caratteristiche e, in generale, del ruolo che questi hanno avuto nello sviluppo del software e nell'architettura dell'applicativo definitivo.

Capitolo 1

Analisi generale

In questo capitolo verranno spiegati il contesto di utilizzo del progetto sviluppato e le sua architettura software.

1.1 Contesto di utilizzo

Come anticipato nel capitolo introduttivo, l'azienda per cui è stato realizzato il software si occupa del controllo della qualità delle linee di produzione. Durante il processo produttivo il prodotto subisce una serie di trattamenti/lavorazioni e, in base alla disponibilità della linea, è possibile inserire un sistema di visione dopo ognuno di questi trattamenti/lavorazioni. In altri casi, invece, il sistema di visione può essere utilizzato *offline* (fuori linea) per controlli a campione. I risultati delle “ispezioni” effettuate dal sistema di visione possono essere qualitativi o quantitativi: i dati qualitativi descrivono le caratteristiche, gli attributi, le proprietà, le qualità di un fenomeno o di un oggetto, mentre i dati quantitativi possono essere contati o espressi numericamente; nel primo caso si può ad esempio verificare la presenza di un difetto, mentre nel secondo caso si può misurare una determinata quantità. Gli algoritmi di visione e i loro risultati devono essere validati, cioè deve essere definito un requisito analitico e di conferma che lo strumento ha capacità prestazionali coerenti con ciò che l'applicazione richiede. Pertanto,

per poter effettuare la validazione di un algoritmo di visione è necessaria la *ground truth* (verità), cioè un'informazione nota per essere reale o vera, fornita dall'osservazione e dalla misurazione diretta. Quindi vengono definiti dei requisiti di performance (ad esempio di $\pm 0.5mm$) su cui viene verificata la misura effettuata dal sistema di visione rispetto a quanto riportato nella verità.

L'esigenza di implementare MVE nasce principalmente dalla mancanza di una procedura standard da eseguire sui dati ottenuti dal sistema di visione e dall'assenza di un ambiente unificato per l'archiviazione delle immagini e della verità e per il confronto di questa con il risultato degli algoritmi di elaborazione. Ad oggi la scelta delle immagini e il metodo/strumento per effettuare la validazione di un algoritmo di visione sono di competenza del tecnico specializzato, rendendo il processo di validazione parziale a seconda del dataset utilizzato e la valutazione degli effetti di nuovi algoritmi qualitativo e non organizzato, non avendo uno strumento sistematico e strutturato per valutare l'impatto delle variazioni introdotte. Infatti, possono nascere nuove richieste per un sistema di visione già in essere, il che comporta una nuova validazione degli algoritmi e delle performance in generale: ad esempio possono essere richieste una maggiore precisione su una misura, una nuova ispezione per un prodotto (verifica presenza di macchie), etc.

Lo strumento di visione attualmente utilizzato si chiama MVS, una piattaforma software sviluppata internamente che mette a disposizione la maggior parte delle elaborazioni e dei controlli standard, fornendo anche la possibilità di integrare dei plugin per l'applicazione di algoritmi su casi specifici.

1.2 Architettura software

L'architettura software è l'organizzazione di base di un sistema, espressa dalle sue componenti, dalle relazioni tra di loro e con l'ambiente, e i principi che ne guidano il progetto e l'evoluzione. In questo caso, si è scelto di sviluppare un'applicazione web, un'architettura tipica di tipo client-server.

Per semplificarne la realizzazione è stato scelto il micro-framework chiamato Flask che facilita anche la comunicazione tra l'app e il database MySQL. Per la gestione dei dataset di immagini è stato scelto CVAT, che oltre all'archiviazione permette di annotare le immagini e di interagire con l'applicazione web attraverso l'utilizzo delle sue API REST. Infine, per valutare in modo formale il miglioramento delle prestazioni la scelta è ricaduta su FiftyOne, che consente sia la visualizzazione di un dataset di immagini con le relative annotazioni, che l'integrazione di plugin implementati *ad hoc*, in questo caso, per ottenere un confronto tra dati numerici.

Nella figura 1.1 è possibile vedere lo schema con le varie componenti coinvolte e la loro interazione. È bene specificare che non per forza tutte devono essere eseguite sullo stesso computer: ad esempio, durante il corso dello sviluppo è stato utilizzato un server CVAT in esecuzione su una seconda macchina.

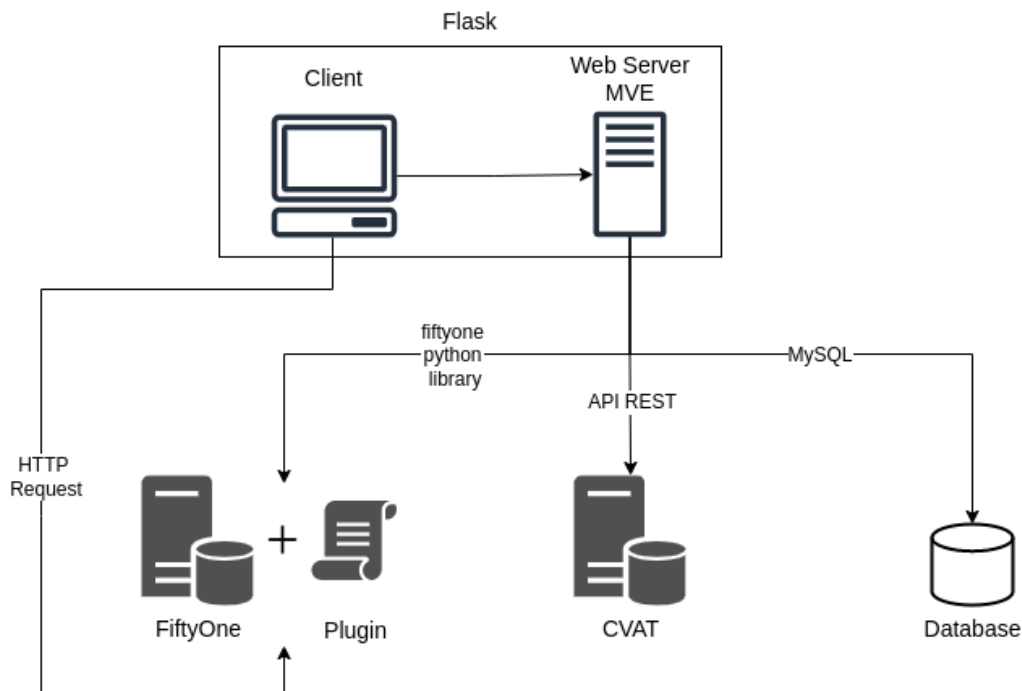


Figura 1.1: Architettura progetto

Capitolo 2

Strumenti di sviluppo

In questo capitolo verranno spiegati gli strumenti utilizzati durante lo sviluppo del progetto, che comprendono software esistenti, framework, linguaggi di programmazione e librerie.

2.1 CVAT

Computer Vision Annotation Tool (CVAT) è uno strumento di annotazione di immagini e video gratuito, open source, basato sul Web che viene utilizzato per etichettare i dati per gli algoritmi di visione artificiale. Originariamente sviluppato da Intel, CVAT è progettato per essere utilizzato da un team di annotazioni di dati professionale, con un'interfaccia utente ottimizzata per attività di annotazione di visione artificiale.² Supporta le attività principali dell'apprendimento automatico supervisionato: rilevamento di oggetti, classificazione delle immagini e segmentazione delle immagini, consentendo agli utenti di annotare i dati per ciascuno di questi casi. CVAT è organizzato in *progetti* a cui possono essere associati più *task* di annotazione, che a loro volta vengono suddivisi in *job* contenenti un numero arbitrario di immagini, utili per tenere traccia dello stato delle annotazioni. In questo progetto, non si parlerà di *job* poiché ad ogni *task* corrisponderà un solo *job*.

È scritto principalmente in TypeScript, React, Ant Design, CSS, Python e Django ed è distribuito con licenza MIT e il suo codice sorgente è disponibile su GitHub. In aggiunta, il server CVAT fornisce le API REST HTTP per l'interazione,³ cioè delle interfacce di programmazione delle applicazioni (API o API web) conformi ai vincoli dello stile architetturale REST.⁴ Una API è un insieme di procedure (in genere raggruppate per strumenti specifici) atte a risolvere uno specifico problema di comunicazione tra diversi computer o tra diversi software o tra diversi componenti di software. L'architettura REST si basa su HTTP. Il funzionamento prevede una struttura degli URL ben definita che identifica univocamente una risorsa o un insieme di risorse e l'utilizzo dei metodi HTTP specifici per il recupero di informazioni (*GET*), per la modifica (*POST*, *PUT*, *PATCH*, *DELETE*) e per altri scopi (*OPTIONS*, ecc.). Pertanto, ogni applicazione client, sia essa uno strumento a riga di comando, un browser o uno script, interagisce con CVAT tramite richieste e risposte HTTP, come mostrato nella Figura 2.1:

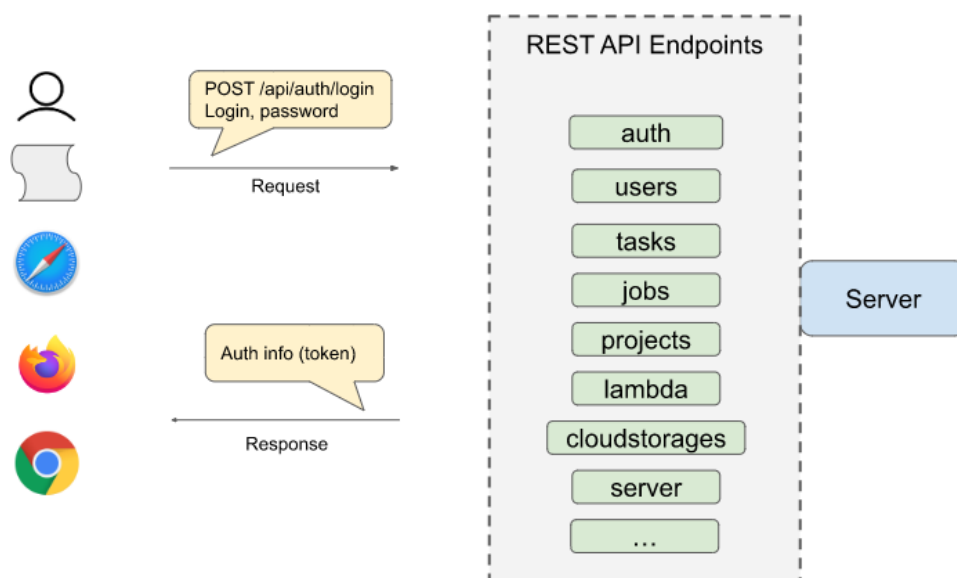


Figura 2.1: Schema API REST CVAT

2.1.1 API REST CVAT

CVAT provvede una vasta gamma di API REST⁵ che permettono di interagire con il server. Per questo progetto ne sono state utilizzate solamente una piccola parte, in particolare alcune inerenti all'autenticazione, ai progetti e ai task, come verrà illustrato in seguito. In generale, è possibile eseguire operazioni come:

- Autenticazione tramite terze parti (Google, GitHub, Amazon Cognito) o credenziali e la gestione di esse.
 - <https://app.cvat.ai/api/auth/amazon-cognito/login>
 - <https://app.cvat.ai/api/auth/github/login>
 - <https://app.cvat.ai/api/auth/google/login>
 - <https://app.cvat.ai/api/auth/login>
- Creazione, visualizzazione e gestione di archiviazioni cloud.
 - <https://app.cvat.ai/api/cloudstorages>
- Scrittura, lettura e gestione di commenti.
 - <https://app.cvat.ai/api/comments>
- Presentazione, lettura e gestione di problemi, con la possibilità di leggere anche un commento relativo ad un problema specifico.
 - <https://app.cvat.ai/api/issues>
- Creazione, visualizzazione e gestione di inviti, ad esempio per chiedere ad un altro utente di acquisire un ruolo all'interno del proprio server (supervisore, proprietario, manutentore, lavoratore).
 - <https://app.cvat.ai/api/invitations>
- Creazione, visualizzazione, aggiornamento parziale e cancellazione di una organizzazione.

- `https://app.cvat.ai/api/organizations`
- Applicazione di limitazioni agli utenti interni al server o ad una organizzazione.
 - `https://app.cvat.ai/api/limitations`
- visualizzazione e cancellazione degli utenti registrati nel server; è possibile restituire le informazioni (limitazioni comprese) e aggiornare alcuni campi specifici.
 - `https://app.cvat.ai/api/users`
- Visualizzazione, aggiornamento parziale di alcuni campi e cancellazione di un'iscrizione (membership).
 - `https://app.cvat.ai/api/membership`
- Creazione, visualizzazione, aggiornamento parziale e cancellazione di un progetto; in particolare è possibile scaricare le annotazioni esistenti, eseguire un backup, importare/esportare un progetto sotto forma di dataset in uno specifico formato, visualizzare l'immagine di anteprima di un progetto, restituire le informazioni relative ai task e creare un progetto partendo da un backup.
 - `https://app.cvat.ai/api/projects`
 - `https://app.cvat.ai/api/projects/{id}/tasks`: per ottenere informazioni sui task di un progetto specifico tramite il metodo GET.
- Creazione, visualizzazione, aggiornamento parziale e cancellazione di un task; tutto ciò che può essere eseguito sui progetti può essere applicato sui task. Inoltre, per ogni task è possibile eliminare le annotazioni, allegare immagini e ottenere una lista di job relativi ad esso.
 - `https://app.cvat.ai/api/tasks`

- <https://app.cvat.ai/api/tasks/{id}/dataset>: per esportare un task come dataset in un formato specifico, attraverso il metodo GET.
- <https://app.cvat.ai/api/tasks/{id}/data>: per acquisire i dati di un task tramite il metodo GET o per allegare le immagini a un task con il metodo POST.
- Visualizzazione e aggiornamento parziale di un job; per ogni job è possibile ottenere le annotazioni in un documento JSON, caricare, modificare e cancellare le annotazioni, ricevere una lista con la traccia dei cambiamenti avvenuti, esportare come dataset in un formato specifico, ottenere una lista di problemi relativi e ricavare l'immagine di anteprima.
 - <https://app.cvat.ai/api/jobs>
- Visualizzazione delle impostazioni e le informazioni del server.
 - <https://app.cvat.ai/api/server/about>
- Creazione, visualizzazione, modifica e cancellazione di un webhook;
 - <https://app.cvat.ai/api/webhooks>

2.2 FiftyOne

FiftyOne^{6,7} è uno strumento open source per la creazione di dataset e modelli di visione artificiale di alta qualità. Offre un'app, un'interfaccia utente grafica che semplifica l'esplorazione e consente di ottenere rapidamente intuizioni sui dataset. In aggiunta, permette la visualizzazione di etichette come riquadri di delimitazione e segmentazioni sovrapposte ai campioni e l'ordinamento, l'interrogazione e la suddivisione dei propri dataset in qualsiasi sottoinsieme di interesse.

2.2.1 Plugin

FiftyOne fornisce un sistema di plugin⁸ per personalizzare ed estendere il suo comportamento, attraverso l'implementazione di script TypeScript (estensione di JavaScript) e l'utilizzo di npm e Yarn (gestori di pacchetti per JavaScript). In particolare, è previsto l'uso di React,^{9,10} una libreria open-source di JavaScript, che rende semplice la creazione di UI interattive e progetta interfacce per ogni stato dell'applicazione: ad ogni cambio di stato React aggiornerà efficientemente solamente le parti della UI che dipendono da tali dati. Per la realizzazione del plugin sono stati utilizzati `react-chartjs-2`, una componente React per *Chart.js*, la libreria di grafici più popolare e `react-select`, un controllo *Select Input* flessibile per React con supporto multiselezione, completamento automatico e asincrono.

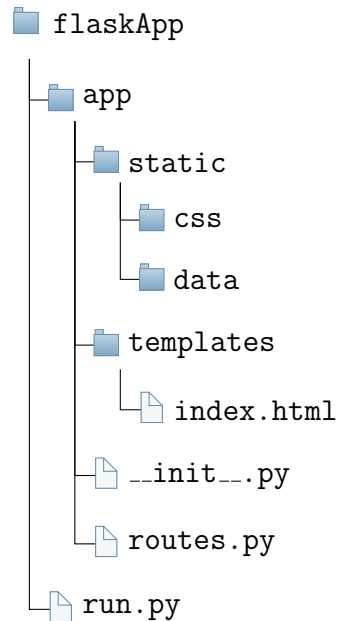
2.3 MySQL

Alla base dell'applicazione web c'è un database MySQL¹¹ in cui vengono memorizzate le informazioni necessarie. MySQL è un sistema di gestione di database relazionali open-source, con le caratteristiche di essere veloce, affidabile, scalabile, facile da usare e di funzionare in sistemi client/server. Inoltre, è stato utilizzato DBeaver, un'applicazione software client SQL e uno strumento di amministrazione del database.

2.4 Flask

Flask¹² è un *micro-framework Web* scritto in Python, basato sullo strumento Werkzeug WSGI e su Jinja. Il primo è un protocollo di trasmissione che stabilisce e descrive comunicazioni ed interazioni tra server ed applicazioni web scritte nel linguaggio Python, mentre il secondo è un motore di template. Inoltre, ha un server di sviluppo e debug, supporta le richieste RESTful e i cookie di sicurezza (sessioni lato client), è basato su Unicode e permette l'estensione delle sue caratteristiche. Flask fornisce quindi gli

strumenti e le funzionalità utili che semplificano la creazione di applicazioni Web. Un'applicazione Flask corretta utilizzerà più file, alcuni dei quali saranno file *template*. L'organizzazione di questi file deve seguire delle regole affinché l'app funzioni. Ecco uno schema della struttura tipica:



Tutto ciò di cui l'app ha bisogno si trova in una cartella, qui denominata `flaskApp`, composta da una cartella `app` e un file `run.py`. Quest'ultimo si occupa di lanciare l'app, mentre la cartella `app` contiene al suo interno due cartelle, specificamente denominate `static` e `templates`. La cartella `static` contiene le risorse utilizzate dai template, inclusi file CSS, file JavaScript e immagini. La cartella dei `templates` contiene solo template con una estensione `.html`. Oltre alle due cartelle, `app` contiene anche file `.py`. In particolare, il file `__init__.py` è necessario per fare in modo che Python tratti le directory contenenti il file come pacchetti, per impedire alle directory con un nome comune, ad esempio `string`, di nascondere involontariamente moduli validi che si verificano successivamente nel percorso di ricerca del modulo. Il file `routes.py` gestisce il sistema di *routing* fornito da Flask, ovvero tutto ciò che fa riferimento ai pattern URL di un'app. Oltre a questi

possono essere presenti altri file `.py` a cui si può accedere ad esempio tramite `routes.py` includendoli come se fossero librerie. Si noti che questi devono trovarsi all'esterno delle due cartelle denominate `static` e `templates`.

Di seguito verranno spiegati i linguaggi di programmazione utilizzati, che per maggiore chiarezza sono stati divisi in base al modo in cui operano, ovvero in “Lato Server” e “Lato Client”. Infine, verrà data una breve descrizione di Jinja.

2.4.1 Lato Server

I linguaggi lato server vengono elaborati dal server per eseguire operazioni che richiedono l'accesso a informazioni o funzionalità non disponibili sul client oppure che necessitano misure di sicurezza che sarebbero inaffidabili se eseguite lato client. Nel primo caso, un esempio può essere il trattamento e l'immagazzinamento di dati da client a server in un database, in cui possono essere memorizzati, modificati ed estratti all'occorrenza. In questo software i linguaggi lato server sono Python e SQL, che vengono interpretati ed elaborati dal server il quale, successivamente, invia i risultati al client.

Python

Python¹³ è nato all'inizio degli anni novanta ed è un linguaggio di programmazione di “alto livello”, orientato a oggetti, adatto a sviluppare applicazioni distribuite, scripting, computazione numerica e system testing e i suoi principali obiettivi sono dinamicità, semplicità e flessibilità.

Una delle caratteristiche principali di Python è quella di offrire un numero notevole di librerie e moduli; di seguito quelli utilizzati per il progetto:¹⁴⁻¹⁶

- `os`: fornisce un modo portatile per utilizzare le funzionalità dipendenti dal sistema operativo.
- `shutil`: offre una serie di operazioni di alto livello su file e raccolte di file. In particolare, sono fornite funzioni che supportano la copia e la rimozione dei file.

- **flask**: fornisce le API e gli oggetti di Flask, tra cui:
 - *render_template* per renderizzare un template dalla cartella dei template con il contesto dato.
 - *redirect* per restituire un oggetto risposta che, se chiamato, reindirizza il client alla posizione di destinazione.
 - *make_response* per impostare intestazioni aggiuntive in una vista quando necessario.
 - *url_for* per generare un URL per l'endpoint specificato con il metodo fornito.
 - *send_file* per inviare il contenuto di un file al client.
 - *request* per accedere ai dati delle richieste in entrata.
 - *session*, un oggetto che funziona praticamente come un normale dict, con la differenza che tiene traccia delle modifiche. Una sessione consente di ricordare le informazioni da una richiesta all'altra.
- **threading**: per costruire interfacce di threading di livello superiore sopra il modulo *_thread* di livello inferiore (che fornisce primitive di basso livello per lavorare con thread multipli).
- **queue**: implementa code multi-produttore e multi-consumatore. È particolarmente utile quando le informazioni devono essere scambiate in modo sicuro tra più thread.
- **pandas**: per la manipolazione e l'analisi dei dati.
- **dotenv**: legge coppie chiave-valore da un file `.env` e può impostarle come variabili di ambiente.
- **mysql.connector**: consente ai programmi Python di accedere ai database MySQL.

- `zipfile`: fornisce strumenti per creare, leggere, scrivere, aggiungere ed elencare un file ZIP.
- `requests`: consente di inviare richieste HTTP/1.1 in modo estremamente semplice e viene utilizzato per interagire con CVAT attraverso le API REST.
- `json`: utilizzato per lavorare con i dati nel formato JSON.
- `pathlib`: offre classi che rappresentano percorsi di filesystem con semantica appropriata per diversi sistemi operativi.
- `re`: per lavorare con le espressioni regolari.
- `xml.etree.ElementTree`: implementa un'API semplice ed efficiente per l'analisi e la creazione di dati XML.
- `PIL`: aggiunge funzionalità di elaborazione delle immagini all'interprete Python.
- `time`: offre varie funzioni relative al tempo.
- `fiftyone`: fornisce una rappresentazione strutturata ma dinamica per creare ed esplorare i dataset. È possibile interrogare e manipolare in modo efficiente i dataset aggiungendo tag personalizzati, previsioni del modello e altro ancora.

SQL

SQL (Structured Query Language) è un linguaggio standardizzato per database basati sul modello relazionale (RDBMS), progettato per creare e modificare schemi di database, interrogare i dati memorizzati e creare e gestire strumenti di controllo e accesso ai dati.

Il linguaggio SQL e in particolare il server MySQL vengono integrati con Python grazie all'utilizzo del driver `mysql.connector` riportato poco fa.

2.4.2 Lato Client

Il termine lato client indica le operazioni di elaborazione effettuate da un client in un'architettura client-server, contrapponendosi a quelle etichettate invece come lato server, rappresentando dunque il front-end di un sistema informatico e di un'applicazione web. Un'operazione tipica client-side ad esempio è quella del caricamento di un'interfaccia grafica utente: il browser client riceve i dati HTML e JavaScript dal server ed il motore di rendering li elabora visualizzandoli sul monitor dell'utente.

HTML

HTML (Hyper Text Markup Language)¹⁷ è un linguaggio di specifica delle informazioni che deriva da SGML (Standard Generalized Markup Language). È un linguaggio di markup, cioè definisce delle aree di testo attraverso l'uso di tag, che determinano come verrà trattato il testo incluso. Attualmente è disponibile la versione HTML5, che rispetto alla versione precedente, introduce nuove funzioni per la realizzazione di applicazioni ricche e interattive.

CSS

CSS (Cascading Style Sheet)¹⁸ è un linguaggio per la formattazione di documenti HTML. La formattazione viene separata dalla presentazione per velocizzare lo sviluppo e facilitare il riuso del codice. La presentazione è diventata talmente importante nel web moderno che sono nati diversi framework per facilitare lo sviluppo del codice CSS e la realizzazione di pagine Web “responsive”, ovvero con una formattazione in grado di adattarsi naturalmente a schermi di diversa orientazione e dimensioni, tramite CSS (per esempio Bootstrap).

JavaScript

JavaScript¹⁹ è un linguaggio di programmazione orientato agli oggetti utilizzato nella programmazione web lato client per rendere dinamiche le

pagine web. Per semplificare la scrittura del codice JavaScript sono state realizzate varie librerie, tra cui jQuery,²⁰ che rende possibile la scrittura di semplici script all'interno di pagine HTML e offre un set di funzioni per poter usare AJAX con facilità. AJAX²¹ è una tecnologia fondamentale per rendere le pagine interattive; infatti, utilizzata nelle pagine HTML, rende possibile lo scambio di dati in background fra web browser e server, nonché l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente.

Bootstrap

Per rendere il sito web completamente responsive, coerente e funzionante, viene utilizzato Bootstrap,²² un insieme di strumenti open source per lo sviluppo con HTML, CSS e JavaScript. Esso mette a disposizione elementi di stile che permettono alla pagina di adattarsi al dispositivo utilizzato e, al contempo, elementi di interfaccia comuni ai siti moderni, quelli cioè che l'utente si aspetta e di cui conosce comportamento e significato.

2.4.3 Jinja

Come anticipato poco fa, Flask utilizza Jinja²³ come motore di template, ovvero un software che permette la creazione di file HTML, XML o in altri formati di markup, che vengono restituiti all'utente mediante una risposta HTTP. Presenta una sintassi consistente e, in aggiunta rispetto agli altri motori di template, permette da un lato di inserire il codice all'interno dei template, dall'altro di essere sviluppato. La parte più potente di Jinja è l'ereditarietà del template che permette di costruire uno scheletro di base contenente tutti gli elementi comuni ai vari template e di definire blocchi che i template figli possono ereditare. Le variabili del template sono definite nel dizionario del contesto passato al template (attraverso l'API *render_template* di `flask` vista in precedenza) e riconosciute attraverso i nomi assegnati come chiavi nel dizionario. Inoltre, Jinja fornisce delle strutture di controllo, cioè

tutto ciò che controlla il flusso di un programma: istruzione if, ciclo for, macro e blocchi.

Capitolo 3

Funzionalità richieste

Lo sviluppo, il mantenimento e l'aggiornamento di un progetto di visione sono parte di un processo ciclico; dal requisito proveniente dall'esterno si arriva alla validazione dell'algoritmo di visione, per ritornare al requisito e ad una nuova validazione. Questo processo è strutturato in diversi punti:

- Analisi del requisito.
- Acquisizione delle immagini dei campioni: con l'utilizzo di diverse illuminazioni e inquadrature a seconda dell'analisi effettuata precedentemente.
- Osservazione dei campioni e sviluppo di un algoritmo di visione.
- Raccolta dei risultati ottenuti dall'algoritmo.
- Validazione dei risultati rispetto ai requisiti.
- Feedback esterno.

Tutti i dati che partecipato alle varie fasi di questo processo fanno parte del "Progetto di visione". Il progetto di visione è stato modellato nel progetto MVE, cioè un contenitore di informazioni e dati necessari per la realizzazione del processo appena descritto. All'interno del progetto MVE è stato scelto di utilizzare CVAT come repository per l'archiviazione delle immagini. Si è

deciso di utilizzare i progetti di CVAT all'interno di un progetto MVE per creare un collegamento tra CVAT e MVE.

3.1 Software MVE

Le entità modellate all'interno di MVE sono:

- Progetto MVE
- Progetto CVAT
- Task CVAT
- Verità
- Predizione
- Dataset FiftyOne

Un progetto MVE quindi fa riferimento ad un progetto di visione esistente all'interno dell'azienda e contiene campioni, verità e predizioni. Infatti, ad un progetto MVE vengono associati uno o più progetti CVAT, i quali a loro volta possono avere più task CVAT, che contengono le immagini acquisite durante il controllo di qualità. Il prodotto esaminato dallo strumento di visione ha delle proprietà reali che vengono chiamate “verità”, mentre ciò che viene restituito da MVS è denominato “predizione”. Entrambe queste entità possono avere dati di tipo qualitativo o quantitativo. Un dataset FiftyOne è l'entità che rappresenta l'insieme dei dati che verranno visualizzati a front-end.

Sulle entità appena descritte deve essere possibile eseguire alcune operazioni, che in Informatica sono considerate le azioni basilari della gestione persistente dei dati e che vengono chiamate CRUD: creazione (*Create*), lettura (*Read*), aggiornamento (*Update*) e rimozione (*Delete*). Nella Tabella 3.1 è mostrato un resoconto su ciò che è possibile effettuare attraverso l'applicazione.

Entità	Creazione	Lettura	Aggiornamento	Rimozione
Progetto MVE	Si	Si	Si	Si
Progetto CVAT	Si	Si	No	Si
Task CVAT	Si	Si	No	Si
Verità	Si	Si	Si	No
Predizione	Si	Si	No	Si
Dataset FO	Si	No	No	No

Tabella 3.1: Operazioni CRUD per entità.

L'applicazione web deve quindi permettere all'utilizzatore di creare uno o più progetti MVE, attraverso una pagina apposita, associandogli un nome, una descrizione e un'immagine significativa; questi devono essere visualizzabili nella *homepage* e modificabili attraverso l'apertura di una *view* opportuna che ne permetta anche la cancellazione. Una volta creato un progetto MVE deve essere possibile generare uno o più progetti CVAT associati ad esso, anche in questo caso tramite una pagina apposita che consenta all'utente di dare un nome al progetto, creando un'istanza sia sul database che è alla base dell'applicazione che sul server CVAT. Questi progetti devono essere visualizzati nella *homepage* raggruppati a seconda del progetto MVE a cui appartengono. In aggiunta, tramite un'ulteriore *view* correlata ad un determinato progetto CVAT devono essere possibili le seguenti azioni:

- Creazione di task CVAT attraverso il caricamento di immagini.
- Visualizzazione del progetto corrente sulla *dashboard* di CVAT
- Visualizzazione dei task CVAT esistenti associati a quel progetto. Ogni task deve poter essere eliminato, sia dal database che da CVAT, oppure visualizzato sulla *dashboard* di CVAT, per permettere ai tecnici di annotare le immagini aggiungendo così i dati di verità qualitativi.
- Eliminazione del progetto, sia dal database che da CVAT.

- Selezione del progetto CVAT da visualizzare tramite una lista.

I task CVAT quindi devono essere creati accedendo ad una pagina in cui vengono scelti il nome e la cartella in locale contenente le immagini che si vogliono caricare, con la possibilità di filtrare le foto attraverso l'utilizzo di una espressione regolare applicata sul nome e di mostrare all'utente dieci immagini in anteprima.

Inoltre, dalla *homepage* deve essere possibile per ognuno dei progetti MVE scegliere una tra le seguenti azioni:

- Caricamento della verità
- Scaricamento della verità
- Caricamento di una predizione
- Scaricamento di una predizione
- Confronto

Il caricamento della verità deve avvenire tramite un'apposita pagina che consenta all'utente di caricare un file in formato *.csv*, *.xls* o *.xlsx*, contenente N prodotti esaminati, con le relative proprietà e valori (dati quantitativi). Deve essere possibile caricare più file verità per ogni progetto MVE e nel caso in cui venga reinserito lo stesso prodotto, le proprietà devono essere modificate con i nuovi valori inseriti, restituendo un *feedback* all'utente sull'avvenuto aggiornamento. L'insieme totale delle verità di un determinato progetto MVE deve essere scaricabile sempre sotto forma di file in formato *.csv*.

Anche il caricamento di una predizione deve avvenire attraverso una *view* dedicata che permetta all'utente di caricare un file in formato *.csv*, *.xls* o *.xlsx* contenente N prodotti e i dati predetti delle proprietà (dati quantitativi e qualitativi). Una predizione deve essere scaricabile tramite l'apertura di un *modal* che consenta all'utente di scegliere la predizione da scaricare e deve anche essere eliminabile accedendo alla pagina di modifica di un progetto MVE.

Infine, deve essere possibile visualizzare su FiftyOne una predizione, il confronto tra due predizioni o il confronto tra una predizione e la verità. Una volta che l'utente sceglie l'azione "Confronto" deve essere reindirizzato in una pagina in cui può selezionare ciò che desidera rappresentare su FiftyOne. Inoltre, deve essere possibile visualizzare la cronologia dei confronti effettuati in precedenza, permettendo così all'utente di rivederli su FiftyOne o di eliminarli.

3.2 Plugin FiftyOne

Il plugin da integrare con FiftyOne deve permettere all'utente di osservare i valori di una o più proprietà del dataset tramite un grafico a linea, avente come asse orizzontale gli identificatori delle immagini del dataset correntemente selezionato e come asse verticale i valori delle proprietà. Pertanto, tramite un menù a tendina, deve essere possibile selezionare e deselegionare un numero arbitrario di proprietà e visualizzarle tramite una linea che unisce i punti relativi ad ogni immagine del dataset. Le proprietà in questione sono quelle salvate sul dataset, a cui FiftyOne associa in automatico un colore, perciò il colore della linea di una proprietà deve essere uguale a quello scelto dal software. Inoltre, devono essere possibili altre operazioni come: selezionare un'immagine del dataset cliccando sul punto con l'identificativo relativo all'immagine che si vuole visualizzare, deselegionare quest'ultima visualizzando nuovamente l'intero dataset e, infine, zoomare e scorrere il grafico solamente lungo l'asse x, l'asse y oppure lungo entrambi gli assi.

Capitolo 4

Implementazione

In questo capitolo verrà prima illustrata la configurazione delle componenti software esterne CVAT e FiftyOne (e del suo plugin), successivamente verrà descritto il database insieme alle tabelle e alle relazioni tra esse ed infine verrà spiegata l'implementazione dell'applicazione web e del plugin FiftyOne.

4.1 Configurazione CVAT

Innanzitutto, CVAT è stato scelto perché è uno strumento *open-source* che offre le funzionalità necessarie per l'archiviazione di immagini e della parte di verità che comprende i dati qualitativi sotto forma di annotazioni. Attraverso le API REST fornite è semplice interagire con il web server MVE per permettere la creazione, la visualizzazione e l'eliminazione di progetti e task.

CVAT permette di essere utilizzato *online* oppure di essere installato su un dispositivo. In questo caso si è scelto di installare un server CVAT su una macchina con sistema operativo Windows 10, in un ambiente virtuale, seguendo la seguente procedura:²⁴

1. Scaricamento e installazione dei seguenti software: WSL2 con una distribuzione Linux a scelta, Docker Desktop, Git e Google Chrome (unico browser supportato da CVAT)

2. Esecuzione della distribuzione di Linux per ottenere un terminale in cui eseguire la clonazione del codice sorgente CVAT dal repository GitHub:

```
$ git clone https://github.com/opencv/cvat
$ cd cvat
```

3. Configurazione dell'indirizzo di CVAT tramite il seguente comando:

```
$ export CVAT_HOST=<YOUR_DOMAIN>
```

4. Esecuzione di un comando fornito da Docker per creare e gestire più servizi nei container Docker, in modo che l'applicazione venga eseguita in modo rapido e affidabile:

```
$ docker compose up -d
```

5. Creazione di un *superuser* con tutti i permessi:

```
$ docker exec -it cvat_server /bin/bash
$ python3 /manage.py createsuperuser
```

Scegliendo un username e una password.

Una volta seguite queste istruzioni è possibile accedere alla *dashboard* di CVAT tramite Chrome all'indirizzo impostato al punto 3 alla porta `:8080` (di default).

4.2 Configurazione FiftyOne

Anche FiftyOne è stato scelto perché è un software *open-source* gratuito, che offre un'insieme di API da integrare con Python utilizzando la libreria *fiftyone* e che permette l'integrazione di dataset generati da CVAT contenenti immagini e annotazioni e di plugin per aggiungere funzionalità.

FiftyOne è stato installato da sorgente su una macchina con sistema operativo Ubuntu, in un ambiente virtuale, nel seguente modo:²⁵

1. Installazione di alcuni software quali Python, Node.js, Yarn, libcurl4, openssl e Git per soddisfare i prerequisiti.
2. Clonazione della repository GitHub:

```
$ git clone https://github.com/voxel51/fiftyone
$ cd fiftyone
```

3. Esecuzione dello *script* di installazione con il *flag* *-d* per effettuare una installazione per sviluppatori, in modo da poter integrare in seguito il plugin:

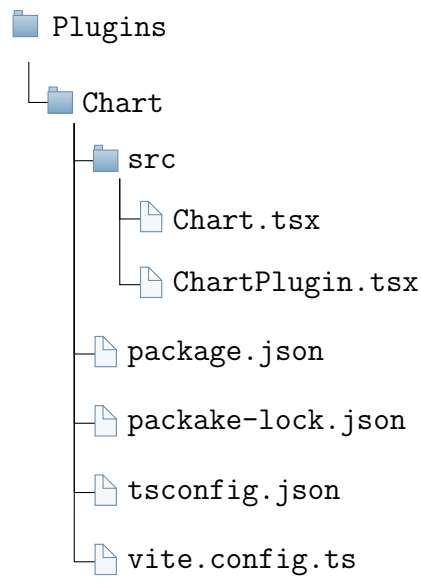
```
$ bash install.bash -d
```

4. Esecuzione dei seguenti comandi per lanciare il server:

```
$ cd fiftyone/fiftyone/server
$ python main.py
```

Riguardo all'integrazione del plugin, come suggerito dalla guida⁸ fornita da FiftyOne, il procedimento è stato il seguente:

1. Utilizzo di un plugin di esempio (*Hello World*) fornito da FiftyOne da usare come template. Questo consiste nello scaricamento di una repository GitHub e di una sua copia all'interno di una cartella. Nel file di configurazione di FiftyOne (*.fiftyone/config.json*) è necessario impostare la variabile *FIFTYONE_PLUGINS_DIR* uguale al percorso della cartella selezionata. La cartella scaricata è stata chiamata *Chart*, mentre la cartella contenente tutti i plugin *Plugins*; inoltre, i file all'interno della cartella *src* sono stati rinominati da *HelloWorld.tsx* e *HelloWorldPlugin.tsx* a *Chart.tsx* e *ChartPlugin.tsx* e questa modifica è stata riportata anche all'interno degli altri file in cui si faceva riferimento a *HelloWorld.tsx* e *HelloWorldPlugin.tsx*. Pertanto la struttura della cartella dei plugin è la seguente:



2. Apertura di un terminale all'interno della cartella *Chart* per installare npm e Yarn:

```
$ yarn install
```

```
$ npm install
```

3. Spostamento all'interno della cartella di FiftyOne con i pacchetti, per creare i collegamenti simbolici (*symlink*) globali che puntano ad essi. Si noti che *MY_FIFTYONE_SRC_DIR* è la cartella di FiftyOne con il codice sorgente:

```
$ cd $MY_FIFTYONE_SRC_DIR/app/packages;
```

```
$ cd aggregations;
```

```
$ npm link;
```

```
$ cd ../plugins;
```

```
$ npm link;
```

```
$ cd ../state;
```

```
$ npm link;
```

4. Tornare nella cartella con il plugin e specificare di utilizzare i *symlink* globali appena creati:

```
$ cd $Chart;  
$ npm link @fiftyone/aggregations;  
$ npm link @fiftyone/plugins;  
$ npm link @fiftyone/state;
```

5. Eseguire il seguente comando per distribuire il plugin:

```
$ yarn build
```

6. A questo punto è tutto pronto per l'implementazione del plugin modificando i file *Chart.tsx* e *ChartPlugin.tsx*. Si noti che dopo ogni modifica sarà necessario eseguire nuovamente il comando `yarn build`.

4.3 Database

Come anticipato nel capitolo 2, alla base dell'applicazione web MVE sviluppata con Flask vi è un database MySQL, il cui modello relazionale è raffigurato in Figura 4.1. In esso vengono rappresentate tutte le tabelle e le relazioni tra esse: ogni tabella ha una chiave privata composta da una o più colonne (campi in grassetto) ed in alcuni casi una chiave esterna (denominata con *FK*). Il fulcro del database, come si può evincere dall'immagine, è la tabella `ProjectsMVE` che tiene traccia dei progetti MVE creati dall'utente. Nella tabella `ProjectMVExProjectCVAT` vengono salvati gli id restituiti da CVAT dopo la creazione di un progetto CVAT e i relativi id dei progetti MVE a cui sono associati. La tabella `MVSxCVAT`, ogni volta che una immagine viene allegata ad un task CVAT, salva nel campo `IdCVAT` il nome dell'immagine, insieme all'id del progetto MVE corrente e l'id del task restituito da CVAT. Quando viene caricato un file contenente la verità, questa viene divisa in due tabelle; ogni file ha una colonna *Name* e N colonne relative alle proprietà,

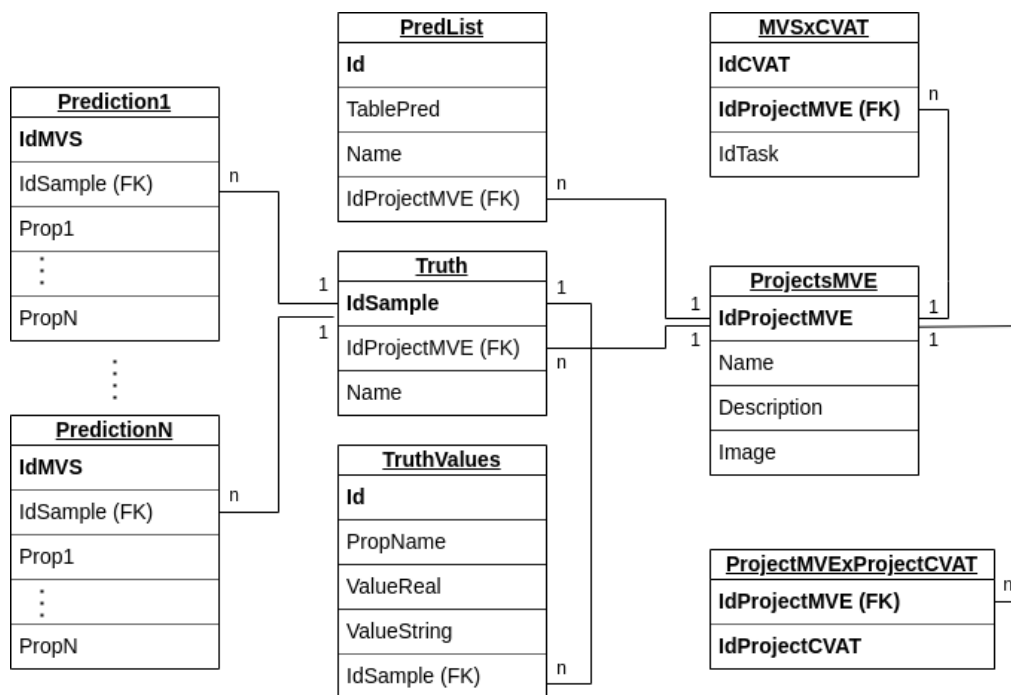


Figura 4.1: Modello relazionale database

pertanto, per ogni riga viene salvato il nome (*Name*) del campione nella tabella **Truth** insieme all'id del progetto MVE a cui fa riferimento, mentre le proprietà e i relativi valori vengono salvati nella tabella **TruthValues** sia in formato numerico che stringa, insieme all'id del campione appena creato nella tabella **Truth**. La Figura 4.2 mostra un esempio di come due file contenenti dati relativi a delle verità vengono salvati sul database.

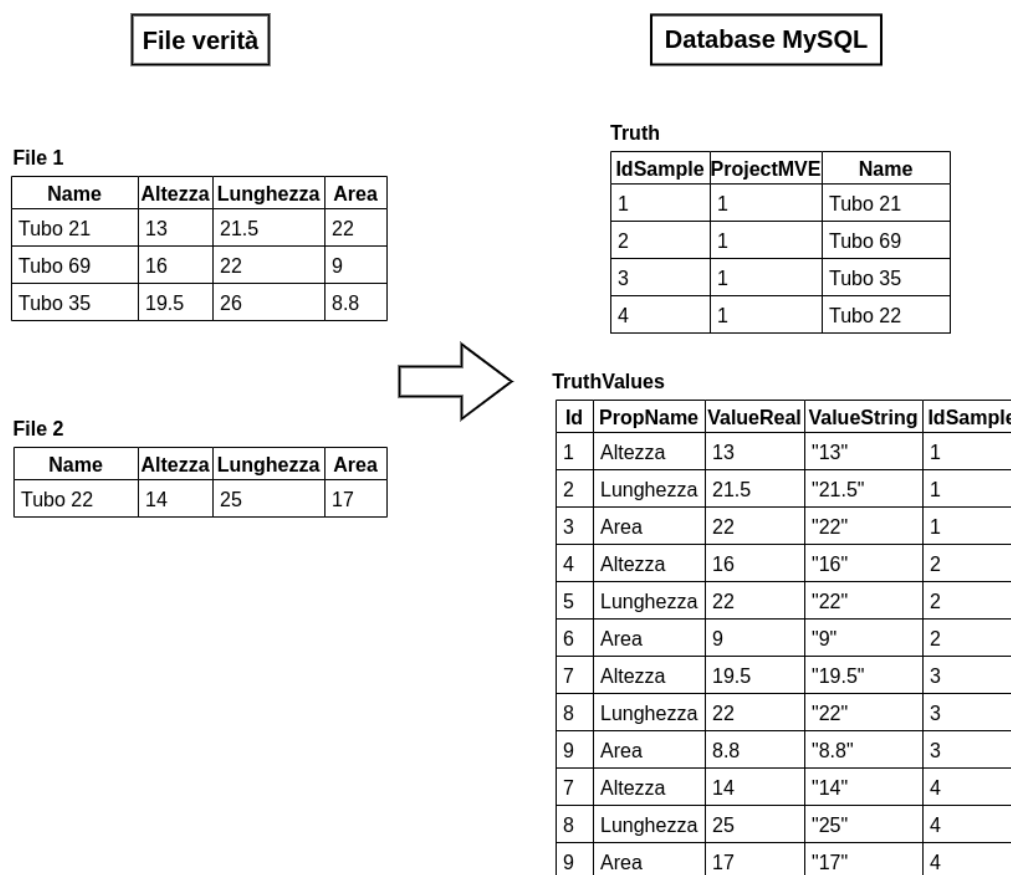


Figura 4.2: Esempio di file con verità salvati su database

Inoltre, durante il controllo qualità, quando vengono acquisiti i campioni, il sistema di visione genera in automatico un id (*IdMVS*) e l'*IdCVAT* visto nella tabella *MVSxCVAT*, che come detto poco fa corrisponde al nome dell'immagine, univoco ad essa e formato da *IdMVS* e una stringa poco rilevante per il caso. Pertanto, ogni volta che viene caricato un file contenente delle predizioni,

viene creata una tabella nuova denominata **PredictionX**, dove *X* è il numero di tabelle *Prediction* già esistenti +1. La tabella avrà una colonna con un id (**IdMVS**), *N* colonne con i nomi delle proprietà e una con l'id del campione verità a cui fa riferimento. Un **IdMVS** può essere elaborato dal sistema *N* volte, cioè un campione verità con lo stesso **IdMVS** può essere riproposto nelle varie *Prediction*. Dopodiché, nella tabella **PredList** viene salvato il nome della tabella *Prediction* appena creata, insieme al nome del file e il progetto MVE a cui è legata la predizione. Anche in questo caso, è possibile vedere un esempio tramite la Figura 4.3 per capire come vengono salvati due file contenenti delle predizioni. Si noti che le proprietà e i dati utilizzati nell'esempio sono solo esemplificativi poiché nella realtà oltre a proprietà con dati quantitativi (ad esempio *Altezza*) nei file con le predizioni saranno presenti anche proprietà con dati qualitativi, ad esempio inerenti a segmentazione e classificazione di immagini.

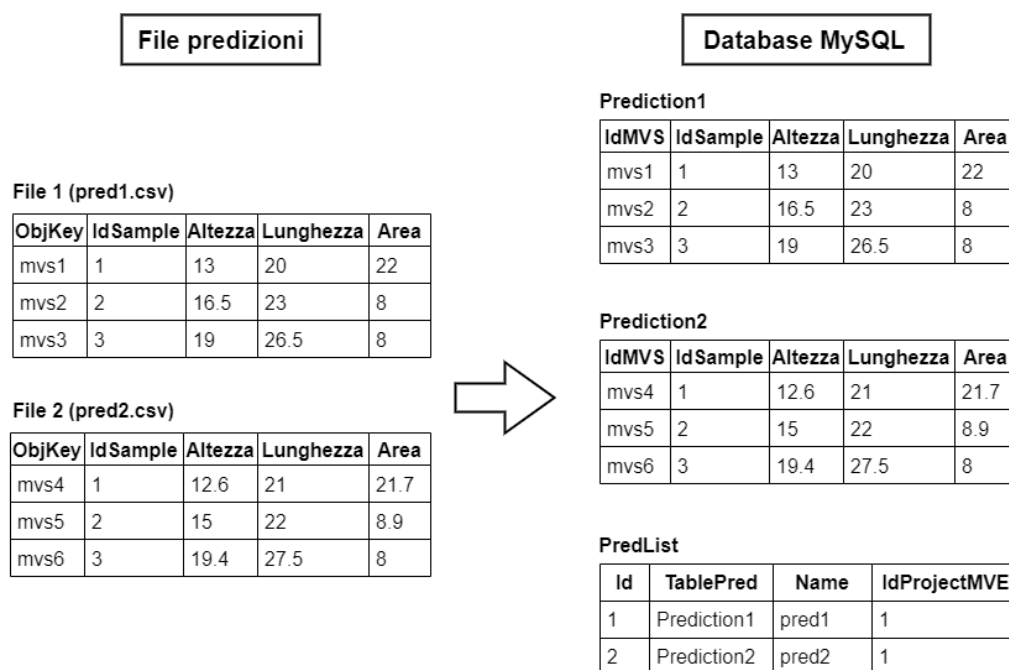


Figura 4.3: Esempio di file con predizioni salvati su database

Infine, è bene specificare che avere l'**IdMVS** nel nome delle immagini sal-

vate su CVAT serve per ottenere un collegamento tra di esse e il database. Non esiste una relazione nel database, pertanto questa informazione verrà utilizzata dal server durante il processamento dei dati.

4.4 Software MVE

Come indicato nei capitoli precedenti, l'applicazione web MVE è stata sviluppata utilizzando il *micro-framework* Flask, scelto principalmente per la sua semplicità e per la conoscenza dello strumento dovuta al suo utilizzo precedente in altri progetti. Per motivi di comodità e disponibilità delle risorse, l'applicazione è stata realizzata sulla stessa macchina Ubuntu su cui è stato installato FiftyOne.

Nel capitolo 2, nella parte inerente a Flask, è stata descritta la struttura generale di un'applicazione, specificando che oltre ai file `routes.py` e `__init__.py` possono esserci anche altri file *python*. Di seguito verranno descritti gli script che sono stati implementati e aggiunti a quelli appena citati nella cartella `app`.

dbquery.py

Questo script è fondamentale poiché si occupa della comunicazione tra il server e il database MySQL. Infatti, vi sono raccolte tutte le funzioni e le relative istruzioni SQL utilizzate per inserire, selezionare, aggiornare ed eliminare dati oppure creare e modificare tabelle.

In generale, per eseguire un'istruzione SQL, è necessario prima stabilire una connessione (sessione) con il server MySQL attraverso il metodo `mysql.connector.connect()` del modulo `mysql.connector`, avente come argomenti il nome dell'utente utilizzato per l'autenticazione con il server MySQL e il nome del database da utilizzare durante la connessione; dopodiché, tramite l'utilizzo del metodo `cursor()` dell'oggetto connessione appena creato, bisogna istanziare un oggetto in grado di eseguire le operazioni SQL. Successivamente, usufruendo del metodo `execute()` del cursore si compie

l'istruzione SQL desiderata. Inoltre, in base al tipo di istruzione, è necessario eseguire una operazione tra `fetchall()`, `fetchone()` o `commit()`; le prime due vengono utilizzate per le query come *SELECT* e *Show tables*, mentre la terza per le istruzioni quali *DELETE*, *INSERT INTO* e *UPDATE*. Per le istruzioni *CREATE TABLE* e *ALTER TABLE* non è necessaria nessuna ulteriore operazione. Infine, è opportuno chiudere il cursore e la connessione tramite il metodo `close()`. Di seguito è possibile vedere un esempio di codice per l'esecuzione di una query *SELECT*.

```
params = {
    'user': user_db,
    'database': name_db,
}
mydb = mysql.connector.connect(**params)
mycursor = mydb.cursor()
sql = "SELECT * FROM table_name;"
mycursor.execute(sql)
results = mycursor.fetchall()
mycursor.close()
mydb.close()
```

routes.py

Come anticipato nel Capitolo 2, questo script gestisce il sistema di *routing* fornito da Flask, cioè tutto ciò che fa riferimento ai pattern URL di un'applicazione. Il funzionamento è il seguente:²⁶ i client inviano richieste al web server, che a sua volta le invia all'istanza dell'applicazione Flask. L'istanza deve sapere quale codice deve essere eseguito per ogni URL richiesto e mappare gli URL alle funzioni Python. L'associazione tra un URL e la funzione che lo gestisce è chiamata *route*. Il modo più conveniente per definire un percorso in un'applicazione Flask è tramite `app.route()`, un decoratore esposto dall'istanza dell'applicazione, che registra la "funzione decorata". Si noti che i

decoratori sono funzionalità di Python che modificano il comportamento di una funzione, aggiungendo nuove funzionalità.

Ad esempio, nel seguente codice, è possibile vedere la *route* che si occupa dell'URL per la *homepage* dell'applicazione.

```
@app.route("/")
def index():
    projectsMVE = dbquery.get_projectsMVE()
    projectsCVAT = dbquery.get_projectMVEProjectCVAT()
    predictions = dbquery.get_predList(id)

    return render_template('index.html', projectsMVE=projectsMVE,
        ↪ projectsCVAT=projectsCVAT, predictions=predictions)
```

In questo caso, la funzione *index()* prepara alcuni dati, in particolare i progetti MVE, i progetti CVAT e la lista delle predizioni, che potranno essere utilizzati nel template *index.html* grazie al motore di template Jinja spiegato nel Capitolo 2. Jinja prevede una serie di delimitatori predefiniti per l'utilizzo delle variabili all'interno del template, configurati come segue:

- `{% ... %}` per le istruzioni (*if-else*, ciclo *for*, etc)
- `{{ ... }}` per le espressioni da stampare nell'output del modello
- `{# ... #}` per i commenti non inclusi nell'output del modello

Ad esempio, per iterare i progetti MVE attraverso un ciclo *for* basta il seguente codice:

```
{% for projectMVE in projectsMVE %}
...
{% endfor %}
```

In cui all'interno può essere inserito del codice HTML.

CVATapi.py

Anche questo file è indispensabile, dato che è composto dalle funzioni che utilizzano le API REST di CVAT. Per ognuna di esse è necessario essere in possesso di un *Token* per l'autorizzazione, ricavabile tramite il seguente codice contenente l'API per il login:

```
# Credenziali CVAT
credentials = {
    "username": username,
    "password": password
}
login = requests.post('{}/api/auth/login'.format(cvat_host_url),
    ↪ json=credentials)
jsonObj = json.loads(login.text)
tokenAuth = jsonObj['key']
```

Di seguito vengono spiegate le funzioni implementate nello script:

- `create_project`: permette la creazione di un progetto CVAT tramite la seguente API REST con parametro il nome che si vuole dare al progetto:

```
dataProject = {
    "name": name_project
}
createProject =
    ↪ requests.post('{}/api/projects'.format(cvat_host_url),
    ↪ data=dataProject, headers={"Authorization": f'Token
    ↪ {tokenAuth}'})
```

- `create_empty_task`: serve per creare un task vuoto specificando alcuni parametri quali il nome, il progetto a cui è associato e altri dati più tecnici, come è possibile vedere nel seguente codice:

```
dataTask = {
    "name": name_task,
```

```

    "project_id": id,
    "owner": 0,
    "assignee": 0,
    "overlap": 0,
    "segment_size": 150,
    "z_order": False,
    "image_quality": 100,
}
createEmptyTask =
↳ requests.post('{} /api/tasks'.format(cvat_host_url),
↳ data=dataTask, headers={"Authorization": f'Token
↳ {tokenAuth}'})

```

- `get_files`: si occupa del salvataggio temporaneo lato server delle immagini caricate dall'utente via browser:

```

pathFolder = 'temp{}'.format(uuid)
os.makedirs(pathFolder)
for file in files:
    headfp, tailfp = os.path.split(file.filename)
    pathFile = pathFolder + "/" + tailfp
    file.save(pathFile)

```

Per ogni file viene preso solamente il nome (*tailfp*) e viene creato un nuovo percorso per indicare che il file deve essere salvato nella cartella temporanea appena creata.

- `select_images`: una volta salvate le immagini, prima di allegarle ad un task, ne viene selezionata solamente una certa quantità alla volta dalla cartella temporanea, dato che il limite di quantità di dati caricabile in un singolo task è di un gigabyte. Il percorso delle immagini selezionate viene salvato in una lista *fs* utilizzata in seguito.

```

size = 0
for filename in os.listdir(directory):

```

```

pathFile = pathFolder + "/" + filename
size_file = Path(pathFile).stat().st_size
if (size+size_file<int(max_size_load_task)):
    size = size + size_file
    fs.append(pathFile)
else: # se la quantita' e' stata raggiunta
    return fs
return fs

```

- `upload_images`: per allegare le immagini ad un task vuoto è stata utilizzata l'apposita API, passando come dato `image_quality` in quanto obbligatorio e come file le immagini sotto forma di array di dati in formato binario:

```

images = {f'client_files[{i}]': open(f, 'rb') for i, f in
    ↪ enumerate(fs)}
uploadImgs =
    ↪ requests.post('{}/api/tasks/{}/data'.format(cvat_host_url,
    ↪ taskId), data={'image_quality':100}, files=images,
    ↪ headers={"Authorization": f'Token {tokenAuth}'})

```

In questo caso `fs` è la lista di percorsi file appena vista, che, una volta caricate le immagini nel task, verrà utilizzata anche per eliminare i file dalla cartella temporanea:

```

for f in fs:
    os.remove(f)

```

- `save_image_cover`: salva lato server (nella cartella `static`) una immagine del task a cui sono appena state allegate le immagini. Questa verrà utilizzata come `cover` nell'interfaccia web per il progetto CVAT associato al task.
- `get_projects_id`: permette di ottenere tutti gli id dei progetti esistenti su CVAT attraverso la seguente API:

```
projects =  
→ requests.get('{} /api/projects'.format(cvat_host_url),  
→ headers={"Authorization": f'Token {tokenAuth}'})
```

- `get_tasks`: utile per ottenere tutti i task di un determinato progetto CVAT, restituendoli sotto forma di lista di dizionari (tipo di dato in cui ad ogni elemento è associata una chiave) ognuno contenente l'id del task, il nome, la data e il tempo di creazione e il numero di immagini. Ciò è possibile utilizzando prima la seguente API passando come parametro il numero massimo di task che si vuole restituire e poi rielaborando i dati ottenuti:

```
tasks =  
→ requests.get('{} /api/projects/{}/tasks'.format(cvat_host_url,  
→ id), params={"page_size" : max_num_of_task},  
→ headers={"Authorization": f'Token {tokenAuth}'})
```

- `get_project`: consente di ottenere i dati di uno specifico progetto CVAT indicandone l'id, restituendo un dizionario contenente il suo id, l'id del progetto MVE a cui è associato, il nome, il numero di task, la somma delle immagini dei vari task e il percorso dell'immagine relativa all'ultimo task (salvata lato server tramite `save_image_cover`). Utilizza le ultime due API viste per ricavare i dati da rielaborare per ottenere le informazioni appena elencate:

```
project =  
→ requests.get('{} /api/projects/{}'.format(cvat_host_url,  
→ id), headers={"Authorization": f'Token {tokenAuth}'})  
tasks =  
→ requests.get('{} /api/projects/{}/tasks'.format(cvat_host_url,  
→ id), params={"page_size" : max_num_of_task},  
→ headers={"Authorization": f'Token {tokenAuth}'})
```

- `delete_project`: permette la cancellazione di un progetto CVAT tramite la specifica del suo id con questa API:

```
project =
  ↪ requests.delete('{}api/projects/{}'.format(cvat_host_url,
  ↪ id), headers={"Authorization": f'Token {tokenAuth}'})
```

- `delete_task`: gestisce l'eliminazione di uno specifico task attraverso la seguente API:

```
task = requests.delete('{}api/tasks/{}'.format(cvat_host_url,
  ↪ id), headers={"Authorization": f'Token {tokenAuth}'})
```

- `get_task_dataset`: consente di scaricare un intero dataset da CVAT (immagini e annotazioni relative), passando come parametri nell'API l'azione desiderata, il formato del dataset e la posizione in cui salvare i file:

```
task =
  ↪ requests.get('{}api/tasks/{}/dataset'.format(cvat_host_url,
  ↪ id), params={"action": "download", "format": "CVAT for
  ↪ images 1.1", "location": "local"},
  ↪ headers={"Authorization": f'Token {tokenAuth}'})
```

Dettaglio API REST utilizzate

Le API REST prevedono la specifica di alcuni parametri all'interno della richiesta per poter includere informazioni aggiuntive. Di seguito una lista di tabelle contenenti i principali parametri forniti da CVAT per le API REST utilizzate.

POST {}api/auth/login ²⁷	
Parametro	Tipo
username	stringa
email	stringa
password (richiesto)	stringa (non vuota)

Tabella 4.1: API per eseguire il login e restituire il token REST se le credenziali sono valide e autenticate

POST <code>{}/api/projects</code> ²⁸		
Parametro	Tipo	Descrizione
name (richiesto)	stringa (da 1 a 256 caratteri)	Nome assegnato al progetto
labels	array di oggetti	Etichette da utilizzare per le annotazioni
owner_id	intero o nullo	Proprietario del progetto
assignee_id	intero o nullo	Utente a cui è assegnato il progetto

Tabella 4.2: API per creare un nuovo progetto

POST <code>{}/api/tasks</code> ²⁹		
Parametro	Tipo	Descrizione
name (richiesto)	stringa (da 1 a 256 caratteri)	Nome assegnato al task
project_id	intero o nullo	Progetto a cui assegnare il task
owner_id	intero o nullo	Proprietario del task
assignee_id	intero o nullo	Utente a cui è assegnato il task
segment_size	intero	Numero di immagini per job
labels	array di oggetti	Etichette da utilizzare per le annotazioni

Tabella 4.3: API per creare un nuovo task vuoto senza immagini

POST <code>{}/api/tasks/{id}/data</code> ³⁰		
Parametro	Tipo	Descrizione
<code>id</code> (richiesto)	intero	Id del task a cui si vogliono aggiungere le immagini
<code>image_quality</code> (richiesto)	intero	Qualità dell'immagine
<code>client_files</code>	array di stringhe (formato binario)	Lista di immagini caricate da locale
<code>server_files</code>	array di stringhe	Lista di immagini caricate da uno spazio condiviso
<code>remote_files</code>	array di stringhe	Lista di immagini caricate da remoto

Tabella 4.4: API per allegare in modo permanente immagini a un task

GET <code>{}/api/projects</code> ³¹		
Parametro	Tipo	Descrizione
<code>filter</code>	stringa	Filtro in base al nome, al proprietario, all'id, ecc.
<code>page</code>	intero	Numero della pagina da restituire
<code>page_size</code>	intero	Numero di risultati da restituire per pagina
<code>search</code>	stringa	Un termine di ricerca, come il 'nome', il 'proprietario', ecc.
<code>sort</code>	stringa	Indica quale campo utilizzare quando si ordinano i risultati

Tabella 4.5: API per restituire un elenco impaginato di progetti

GET <code>{} /api/projects/{id}/tasks</code> ³²		
Parametro	Tipo	Descrizione
id (richiesto)	intero	Id del progetto di cui si vogliono restituire informazioni sui task
filter	stringa	Filtro in base al nome, al proprietario, all'id, ecc.
page	intero	Numero della pagina da restituire
page_size	intero	Numero di risultati da restituire per pagina
search	stringa	Un termine di ricerca, come il 'nome', il 'proprietario', ecc.
sort	stringa	Indica quale campo utilizzare quando si ordinano i risultati

Tabella 4.6: API per restituire informazioni sui task del progetto con l'id selezionato

GET <code>{} /api/projects/{id}</code> ³³		
Parametro	Tipo	Descrizione
id (richiesto)	intero	Id del progetto che si vuole restituire

Tabella 4.7: API per restituire i dettagli di un progetto specifico

DELETE <code>{} /api/projects/{id}</code> ³⁴		
Parametro	Tipo	Descrizione
id (richiesto)	intero	Id del progetto che si vuole eliminare

Tabella 4.8: API per eliminare un progetto specifico

DELETE <code>{} /api/tasks/{id}</code> ³⁵		
Parametro	Tipo	Descrizione
id (richiesto)	intero	Id del task che si vuole eliminare

Tabella 4.9: API per eliminare un task specifico

GET <code>{} /api/tasks/{id}/dataset</code> ³⁶		
Parametro	Tipo	Descrizione
id (richiesto)	intero	Id del task che si vuole utilizzare
action	stringa (“download”)	Utilizzato per avviare il processo di download
filename	stringa	Nome del file di output desiderato
format (richiesto)	stringa	Nome del formato di output desiderato
location	stringa (“local”, “cloud_storage”)	Dove salvare il dataset scaricato

Tabella 4.10: API per esportare il task come dataset in un formato specifico

truthdb.py

In questo script vengono gestiti il caricamento e lo scaricamento di un file verità. Per quanto riguarda il caricamento, le due fasi da seguire sono le seguenti:

1. Conversione del file in una struttura dati chiamata *DataFrame* fornita dalla libreria **Pandas** tramite il metodo `read_csv()` nel caso in cui il file sia in formato `.csv` oppure con il metodo `read_excel()` se il formato è `.xls` o `.xlsx`.

```

if (file_extension == ".csv"):
    data = pd.read_csv(filepath_or_buffer=pathFile)
else:
    data=pd.read_excel(io=pathFile)

```

2. Ricerca della colonna *Name* nel DataFrame: per ogni valore presente in essa viene prima controllato se esiste già nel database una verità con il nome corrente; in caso affermativo, vengono eliminate le proprietà relative a quella verità dalla tabella *TruthValues* per poi inserire quelle nuove. Nel caso in cui si tratti di una nuova verità, viene inserita nella tabella *Truth* una riga con l'IdMVE e il nome.

```

# vengono prese tutte le verita' presenti nella tabella Truth
samples = dbquery.get_sample_truth_MVE(idMVE)
if samples == []:
    updated = False
for sample in samples:
    # viene controllato ogni nome restituito precedentemente
    # col = 'Name'
    # i e' l'indice della riga del dataframe che si sta
    ↪ analizzando
    if data[col][i] == sample['name']:
        updated = True
        dbquery.delete_truth_value(sample['id'])
        break
    else:
        updated = False
if updated == False:
    idSample = dbquery.insert_truth(idMVE, data[col][i])

```

Dopodichè vengono create tre liste: una con i nomi delle proprietà, una con i valori numerici di esse e una con questi valori in formato stringa. Per farlo vengono prese le colonne del DataFrame (proprietà), eccetto quella relativa al nome, e i rispettivi valori. Successivamente,

per ognuna di esse viene inserita una riga nella tabella TruthValues con la corrente proprietà e i suoi valori (numerici e formato stringa), insieme all'idSample generato poco fa.

```

propsName = []
valuesReal = []
valuesString = []
for column in columns:
    if (column != col): # col e' la colonna relativa al nome
        # i e' l'indice della riga del dataframe che si sta
        ↪ analizzando
    if (pd.isnull(data[column][i]) != True): # se il
        ↪ valore e' diverso da null
        propsName.append(column)
        # se il valore e' effettivamente numerico lo salvo
        ↪ nella lista
        if ((isinstance(data[column][i], float)) or
            ↪ (isinstance(data[column][i], int))):
            valuesReal.append(data[column][i])
        else:
            valuesReal.append(None)
            valuesString.append(str(data[column][i]))
if (len(propsName) == len(valuesReal) and len(valuesReal) ==
    ↪ len(valuesString)):
    for propName, valueReal, valueString in zip(propsName,
        ↪ valuesReal, valuesString):
        dbquery.insert_truth_values(idSample, propName,
            ↪ valueReal, valueString)

```

Passando allo scaricamento della verità gli step sono i seguenti:

1. Selezione delle righe della tabella Truth corrispondenti ad un determinato progetto MVE: ogni riga restituita da questa operazione avrà un IdSample, un IdMVE e un nome.

```
truths = dbquery.get_truth_mve(idMVE)
```

2. Per ogni riga restituita, vengono presi i nomi delle proprietà presenti nella tabella `TruthValues` che fanno riferimento allo stesso `IdSample` e viene creata una lista con tutti i nomi delle proprietà esistenti in quel progetto MVE.

```
columns = ['Name']
for truth in truths:
    propNames =
        ↪ dbquery.get_truth_prop_names(truth['IdSample'])
    for propName in propNames:
        if propName['PropName'] not in columns:
            columns.append(propName['PropName'])
```

3. Per ogni riga restituita dalla prima query sulla tabella `Truth` (per ogni verità), viene creata una lista (*row*) inizializzata a `None` per ogni colonna presente nella lista `columns` vista al punto precedente, ottenendo così una lista di `N None`, dove `N` è appunto il numero di colonne. In questo modo, per ogni verità, viene creata una lista che fa riferimento a tutte le proprietà presenti nella tabella `TruthValues` inerenti agli `IdSample` considerati; alcuni valori rimarranno `None` (quelli che fanno riferimento a proprietà che non appartengono alla verità corrente), altri verranno cambiati con il valore effettivo della proprietà. Dopodiché, vengono prese le righe dalla tabella `TruthValues` in cui l'`IdSample` è uguale a quello corrente. Per ognuna di queste righe (*properties*), viene cercata la posizione della proprietà corrente nella lista `columns`, per poi andare a cambiare il valore nella lista `row` in quella posizione da `None` al valore effettivo di quella proprietà. Ogni *row* viene salvata in un'ulteriore lista chiamata *rows*.

```
rows = []
for truth in truths:
    row = []
    # columns contiene i nomi di tutte le proprietà
    for col in columns:
```

```

        row.append(None)
    # La prima posizione di row e' quella relativa al nome
    # → della verita'
    row[0] = truth['Name']
    properties = dbquery.get_truth_values(truth['IdSample'])
    for property in properties:
        if property['PropName'] in columns:
            index = columns.index(property['PropName'])
            row[index] = property['ValueReal']
    rows.append(row)

```

4. Infine, viene creato prima un DataFrame e poi convertito in un file *.csv*.

```

df = pd.DataFrame(data=rows, columns=columns)
df.to_csv(namefile, index=False, header=True)

```

Si noti che nel punto 3 è necessario ottenere una lista di liste per avere i dati nuovamente nel formato iniziale (pre-caricamento file). L'unica differenza è che ora alcune verità potrebbero avere delle proprietà con valore *None*: prendendo in considerazione tutte le verità caricate (più file) relative ad un progetto MVE è possibile che ad un sample siano associate alcune verità che non sono associate ad un altro. Pertanto la situazione potrebbe essere come quella in Figura 4.4.

	Name	Altezza	Lunghezza	Area	Diametro
rows {	Tubo 21	13	21.5	22	<i>None</i>
	Tubo 69	16	22	9	<i>None</i>
	Tubo 35	19.5	26	8.8	<i>None</i>
	Tubo 39	20	<i>None</i>	10	26

Figura 4.4: Esempio rappresentazione dei dati per lo scaricamento della verità

predictdb.py

In questo script vengono gestiti il caricamento e lo scaricamento di un file predizione. Per quanto riguarda il caricamento, il procedimento da seguire è stato il seguente:

1. Conversione del file in un *DataFrame* tramite il metodo *read_csv()* nel caso in cui il file sia in formato *.csv* oppure con il metodo *read_excel()* se il formato è *.xls* o *.xlsx*.

```
if (file_extension == ".csv"):
    data = pd.read_csv(filepath_or_buffer=pathFile)
else:
    data=pd.read_excel(io=pathFile)
```

2. Controllo dell'esistenza della colonna contenente l'*IdSample* (denominata *idSample* oppure *sampleId*) all'interno del *DataFrame*: in caso negativo viene restituito un errore.
3. Ricerca della colonna *ObjKey* contenente gli *IdMVS*, in caso non esistesse viene preso di default la prima colonna:

```
# columns sono le colonne del DataFrame
if 'ObjKey' in columns:
    idMVS = 'ObjKey'
else:
    idMVS = data.iloc[:, 0]
```

4. Conteggio delle tabelle *Prediction* all'interno del database, impostazione del nome della tabella, creazione di essa e successivo inserimento nella tabella *PredList* di una riga contenente il nome della tabella, il nome del file e l'*idMVE* a cui fa riferimento la predizione.

```
count = dbquery.count_table()
# idPred incrementa di uno rispetto all'ultima tabella
↪ Prediction inserita
```

```

idPred = 'Prediction{}'.format(count+1)
dbquery.create_table_prediction(idPred)
dbquery.insert_pred_list(idPred, file_name, idMVE)

```

5. Per ogni colonna del DataFrame:

- Se corrisponde alla colonna relativa all'IdSample, viene creata nella tabella *Prediction* una colonna apposita con chiave esterna che punta alla chiave primaria (colonna IdSample) della tabella Truth.

```
dbquery.add_column_prediction_fk(idPred)
```

- Altrimenti, se non corrisponde alla colonna *ObjKey*, viene preso e controllato il primo valore non nullo relativo alla colonna (proprietà) corrente. Se questo è di tipo *float* o *int*, viene salvata in una variabile (*dataType*) la stringa “DOUBLE”; oppure, se il valore sotto forma di stringa soddisfa un'espressione regolare che riconosce i *tag* del linguaggio XML, allora nella variabile *dataType* viene salvata la stringa “TEXT”; infine, se nessuna delle precedenti condizione è soddisfatta nella variabile *dataType* viene salvata la stringa “VARCHAR(100)”;

```

if ((isinstance(value, float)) or (isinstance(value,
↪ int))):
    dataType = "DOUBLE"
elif re.match(r"(<.[^(><)]+>)", str(value)):
    dataType = "TEXT"
else:
    dataType = "VARCHAR(100)"

```

Dopodiché, viene creata nella tabella *Prediction* una colonna apposita per la proprietà corrente, tramite la seguente query che prevede un *alter table* della tabella:

```

# idPred e' la tabella, column la proprieta' da inserire,
↪ dataType e' il tipo
dbquery.add_column_prediction(idPred, column, dataType)

```

6. A questo punto si ha la tabella *Prediction* strutturata, pronta all’inserimento dei dati. Per farlo, è necessario leggere ogni riga del DataFrame: per ognuna di esse, vengono presi tutti i valori corrispondenti alle varie colonne (proprietà) e salvati in una lista (*currentRow*), utilizzata poi in una query per inserire i valori nella tabella.

```

for index, row in data.iterrows():
    currentRow = []
    for column in columns:
        if (pd.isnull(row[column]) == False):
            currentRow.append(row[column])
        # se il valore e' nullo
    else:
        currentRow.append(None)
    dbquery.insert_prediction_row(idPred, currentRow)

```

Passando allo scaricamento di una predizione gli step sono i seguenti:

1. Selezione dei nomi delle colonne della tabella *Prediction* (*pred*) selezionata dall’utente e il loro salvataggio in una lista: la colonna con nome “*idMVS*” viene sostituita dalla stringa “*ObjKey*”.

```

dbccolumns = dbquery.get_columns_name_table(pred)
columns = []
for col in dbccolumns:
    if col['column_name'] == 'idMVS':
        columns.append('ObjKey')
    else:
        columns.append(col['column_name'])

```

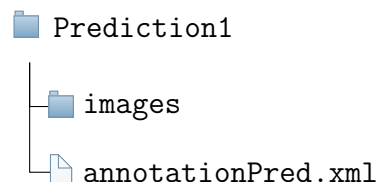
2. Creazione del DataFrame una volta recuperate le righe della tabella *Prediction* selezionata dall’utente e successiva conversione in file *.csv*.


```
rows = dbquery.get_prediction(pred)
df = pd.DataFrame(rows, columns=columns)
df.to_csv(namefile, index=False, header=True)
```

comparedb.py

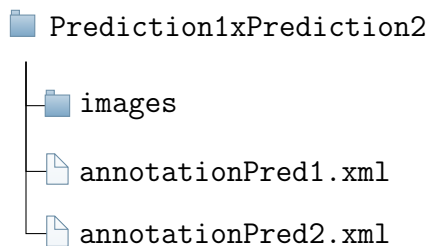
In questo script vengono preparati i dati da utilizzare per la creazione del dataset da visualizzare su FiftyOne. In questo caso non verrà spiegato il codice nel dettaglio ma solamente la logica utilizzata per ognuna delle tre scelte che l'utente può effettuare:

- **Visualizzazione di una predizione:** in base agli IdMVS presenti nella tabella *Prediction* scelta, vengono selezionate dalla tabella *MVSxCVAT* le righe con gli IdTask e gli IdCVAT (nome file) che hanno gli IdMVS all'interno del loro nome. Dopodiché, vengono scaricati da CVAT i task appena ricavati sotto forma di dataset, cioè una cartella con le immagini e un file *.xml* con le annotazioni, che in questo caso non verrà utilizzato dato che le annotazioni presenti su CVAT sono quelle relative alla verità. Tra le immagini scaricate vengono salvate in un'ulteriore cartella denominata con il nome della *Prediction* solamente quelle che fanno riferimento agli IdMVS presenti nella predizione. Una volta ottenute le immagini è necessario ricavare le annotazioni: esse sono presenti nella tabella *Prediction* sotto forma di *tag* XML, in cui negli attributi vi sono salvate tutte le informazioni; queste vengono estratte dalla tabella e viene creato un nuovo file *.xml* strutturato appositamente per essere utilizzato da FiftyOne.

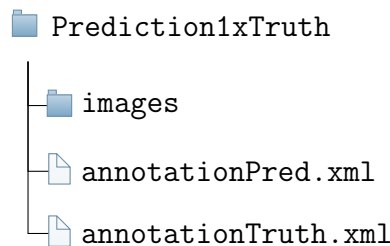


- **Confronto tra due predizioni:** in questo caso vengono prima selezionati solamente gli IdMVS in comune tra le due *Prediction*, poi le relative

righe con gli `IdTask` e gli `IdCVAT` della tabella `MVSxCVAT`. Dopodiché, il procedimento è lo stesso di quello visto poco fa per la visualizzazione di una predizione, con la differenza che la cartella contenente le immagini e le annotazioni verrà denominata *PredictionXxPredictionY* e le annotazioni verranno prese da entrambe le tabelle *Prediction*, creando in questo modo due file *.xml*.



- **Confronto tra verità e predizione:** il procedimento è lo stesso visto per la visualizzazione di una predizione, con la differenza che in questo caso la cartella contenente le immagini e le annotazioni verrà chiamata *PredictionXxTruth* e che verrà considerato il file di annotazioni contenente la verità scaricato da CVAT. Esso sarà esaminato e verranno copiati in un altro file *.xml* solamente i dati relativi alle immagini considerate dalla predizione.



foIntegration.py

Questo script si occupa di una parte fondamentale per il progetto, cioè della creazione e della gestione del dataset da utilizzare su `FiftyOne`, partendo dai dati creati nel file precedente. Per la creazione del dataset è stata utilizzata la libreria `fiftyone` e le sue classi; in particolare:

- `from_dir`: per generare un dataset partendo da una cartella di immagini (`data_path`) e da un file contenente le annotazioni (`labels_path`, specificando il formato (`fo.types.CVATImageDataset`); `label_field` indica il nome con cui verranno visualizzate le annotazioni su FiftyOne, mentre `name` è il nome del dataset stesso.

```
dataset = fiftyone.Dataset.from_dir(  
    dataset_type=fo.types.CVATImageDataset,  
    label_field=label_field,  
    data_path=data_path,  
    labels_path=labels_path,  
    name=name  
)
```

Nel caso della visualizzazione di una predizione questo è sufficiente per la creazione del dataset FiftyOne, poiché si ha il caricamento di una cartella di immagini e di un file con le relative annotazioni. Negli altri due casi invece è necessario caricare anche il secondo file di annotazioni relativo alla seconda predizione oppure alla verità a seconda del caso.

- `merge_dir`: dato un dataset, unisce il contenuto di un secondo file con annotazioni. Anche in questo caso viene specificato il percorso della cartella con le immagini (che non verranno duplicate), insieme al percorso del secondo file con le annotazioni, indicando il formato e il nome con cui verranno visualizzate le annotazioni su FiftyOne.

```
dataset.merge_dir(  
    data_path=data_path,  
    labels_path=labels_path2,  
    dataset_type=fo.types.CVATImageDataset,  
    label_field=label_field2,  
)
```

- `add_sample_field`: per aggiungere un nuovo campo al dataset. In questo modo vengono aggiunte come campi le proprietà quantitativi-

ve delle predizioni o delle verità a seconda dei casi, specificando per ogni proprietà il nome (*field_name*) e il tipo del dato che verrà inserito (*fo.FloatField*, *fo.IntField* o *fo.StringField*).

```
dataset.add_sample_field(field_name, ftype)
```

Questi campi, per ogni campione presente nel dataset, verranno riempiti con i valori delle proprietà delle predizioni e/o delle verità in base al caso scelto dall'utente.

4.5 Plugin FiftyOne

Come anticipato nel Paragrafo 4.2, per implementare il plugin è necessario modificare i file *ChartPlugin.tsx* e *Chart.tsx*: il primo si occupa della registrazione del plugin su FiftyOne, mentre il secondo dell'implementazione di ciò che è stato richiesto.

ChartPlugin.tsx

Tramite le due funzioni *registerComponent* e *PluginComponentType* fornite da FiftyOne è possibile registrare il plugin.

```
import { registerComponent, PluginComponentType } from '@fiftyone/plugins'
import Chart from './Chart';
export { default as Chart } from './Chart';

registerComponent({
  name: 'Chart',
  label: 'Chart',
  component: Chart,
  type: PluginComponentType.Plot,
  activator: true
})
```

Chart.tsx

Questo file è composto da una componente React (funzione) che al suo interno gestisce ed elabora i dati e che ritorna un elemento React. Grazie alle funzioni fornite da FiftyOne è possibile ricavare i dati del dataset che verrà visualizzato nell'app, in modo da poter accedere ai nomi dei campi e ai relativi valori. Infatti, vengono prima ottenuti gli id dei campioni, che saranno utilizzati come etichette per l'asse x del grafico, poi vengono ricavati i nomi di tutte le proprietà, che verranno usati come opzioni nel menu a tendina (*Select*) e infine vengono presi i valori di tutte le proprietà, che saranno visualizzati nel grafico. Dopodiché, con un'altra funzione fornita da FiftyOne, si ottengono i colori assegnati ai campi del dataset, così da poterli associare alle variabili relative alle proprietà.

Una volta ottenuti ed elaborati questi dati, si passa al loro utilizzo nell'elemento React ritornato dalla componente. All'interno di questo elemento vi sono altri elementi, in particolare:

- **Select**: per la gestione e la visualizzazione del menu a tendina; permette la selezione multipla delle opzioni e ogni volta che queste cambiano gestisce i dati da visualizzare nel grafico.

```
<Select
  closeMenuOnSelect={true}
  isMulti
  onChange={handleSelectChange}
  options={optionsSelect}
  styles={colourStyles}
/>
```

- **Line**: per rappresentare e gestire il grafico a linea; tra le *options* vengono gestiti in particolare lo zoom e lo scorrimento, il primo impostato di default solo sull'asse y, mentre il secondo su entrambi gli assi. I dati, come appena detto, vengono gestiti nella *Select* e poi passati al grafico. Tramite l'attributo *onClick* viene gestito l'evento relativo al click di

un punto del grafico: in questo caso viene selezionato il campione del dataset corrispondente all'id scelto dall'utente.

```
<Line
  ref={chartRef}
  options={options}
  data={data}
  onClick={(handlePointClick)}
/>
```

- **ButtonZoom**: per impostare uno zoom diverso da quello di default; pertanto, può essere impostato solamente lungo l'asse x, nuovamente sull'asse y oppure su entrambi gli assi, fornendo anche la possibilità di resettarlo.
- **ButtonRefresh**: per deselezionare il campione scelto cliccando sul grafico; in questo modo vengono visualizzati di nuovo tutti i campioni.

Capitolo 5

Guida operativa

In questo capitolo verrà illustrata una guida sul corretto utilizzo dell'applicazione MVE, in particolare verranno spiegati i passi da seguire con i relativi *screenshot*. Per farlo, saranno utilizzate dei dati e delle immagini acquisite durante il controllo qualità di alcuni puntalini elettrici.

5.1 Creazione progetto MVE

Il primo step da eseguire prevede la creazione di un progetto MVE, tramite il pulsante “Nuovo progetto MVE” situato in alto a destra nella *homepage*, come mostrato nella Figura 5.1.

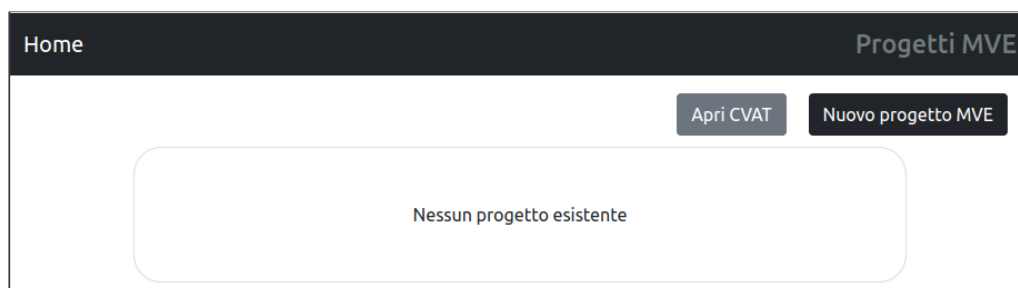
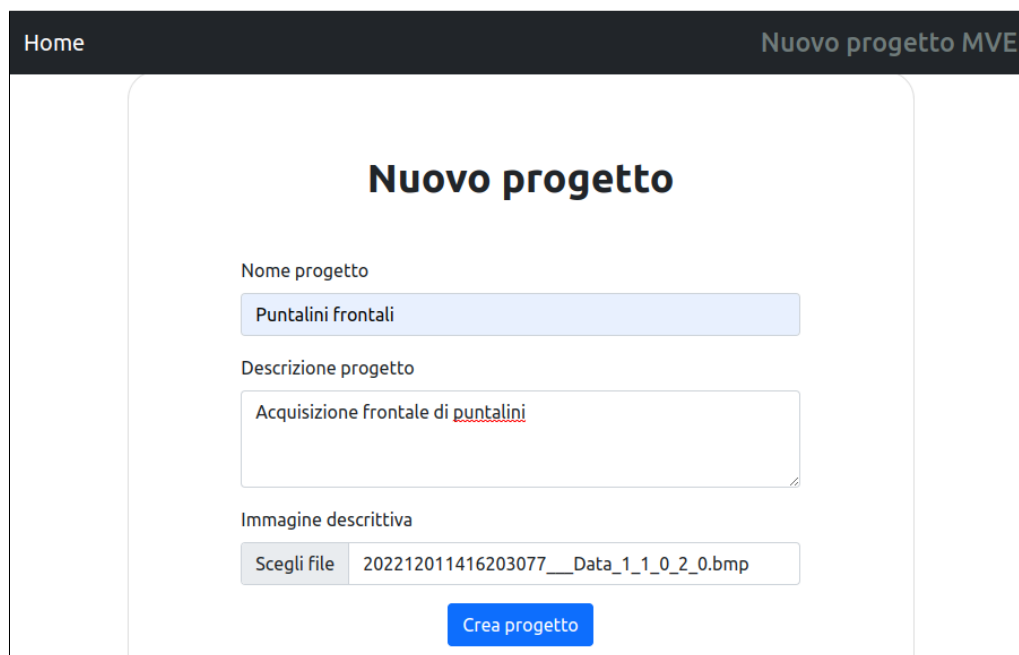


Figura 5.1: Homepage

Una volta aperta la *view*, è necessario compilare tutti i campi del *form*, che comprendono il nome del progetto, una sua descrizione e un'immagine che lo descriva (Figura 5.2).



The screenshot shows a web application interface for creating a new project. At the top, there is a dark navigation bar with 'Home' on the left and 'Nuovo progetto MVE' on the right. The main content area has a white background with a central heading 'Nuovo progetto'. Below the heading, there are three input fields: 'Nome progetto' with the value 'Puntalini frontali', 'Descrizione progetto' with the value 'Acquisizione frontale di puntalini', and 'Immagine descrittiva' with a file selection button 'Scegli file' and the filename '202212011416203077__Data_1_1_0_2_0.bmp'. At the bottom center, there is a blue button labeled 'Crea progetto'.

Figura 5.2: Nuovo progetto MVE

Dopo aver cliccato “Crea progetto” si verrà reindirizzati nuovamente alla *homepage* (Figura 5.3); lì sarà possibile vedere il progetto MVE appena creato, modificarlo e creare molteplici progetti CVAT associati ad esso.

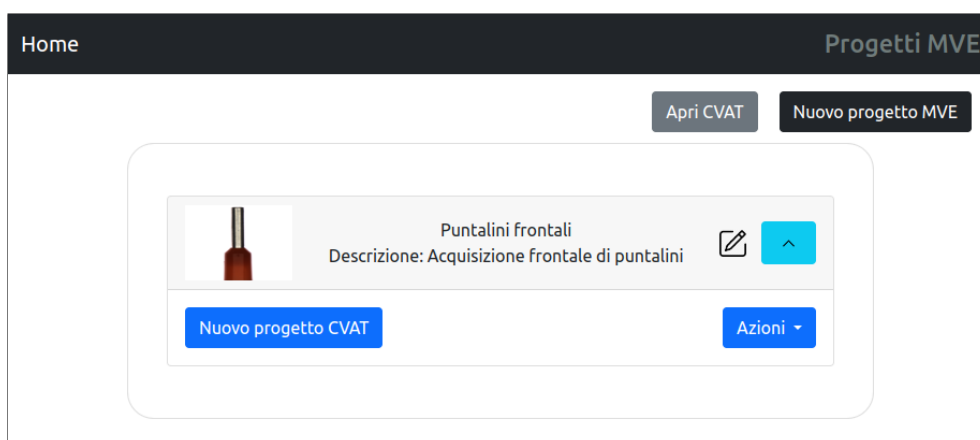


Figura 5.3: Homepage

5.2 Creazione progetto CVAT

Il passo successivo prevede la creazione di un progetto CVAT; cliccando il bottone “Nuovo progetto CVAT” si aprirà una pagina (Figura 5.4) contenente un *form* con un’unica voce, cioè il nome da assegnare al progetto.

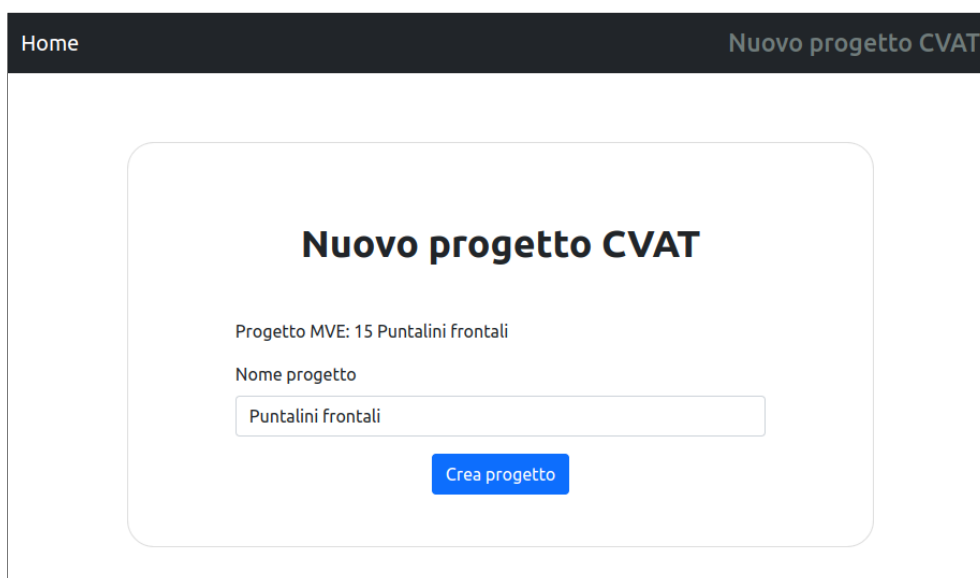


Figura 5.4: Nuovo progetto CVAT

Cliccando “Crea progetto” si verrà reindirizzati alla *homepage*, in cui si potrà vedere il progetto CVAT appena creato, come mostrato in Figura 5.5: dato che non ha ancora nessun task CVAT associato, esso avrà come immagine una “X” e un numero di task e un numero di immagini uguali a 0.

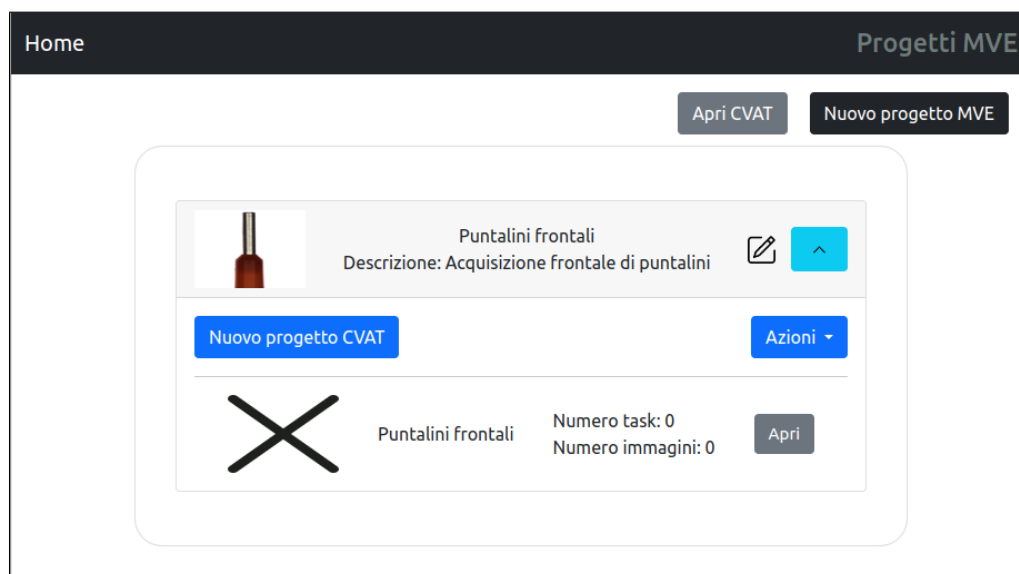


Figura 5.5: Homepage

5.3 Caricamento immagini

Il prossimo step prevede il caricamento delle immagini da archiviare su CVAT per poi annotarle in seguito. Per farlo, è necessario cliccare sul pulsante “Apri” relativo al progetto CVAT in cui si vogliono caricare le immagini (Figura 5.5), che aprirà la *view* visibile alla Figura 5.6. In questa pagina è possibile vedere i task relativi al progetto corrente (nella figura di esempio non vi è ancora nessun task), aprire il progetto sulla *dashboard* di CVAT, eliminare il progetto oppure caricare le immagini, cliccando sul bottone “Carica immagini”.

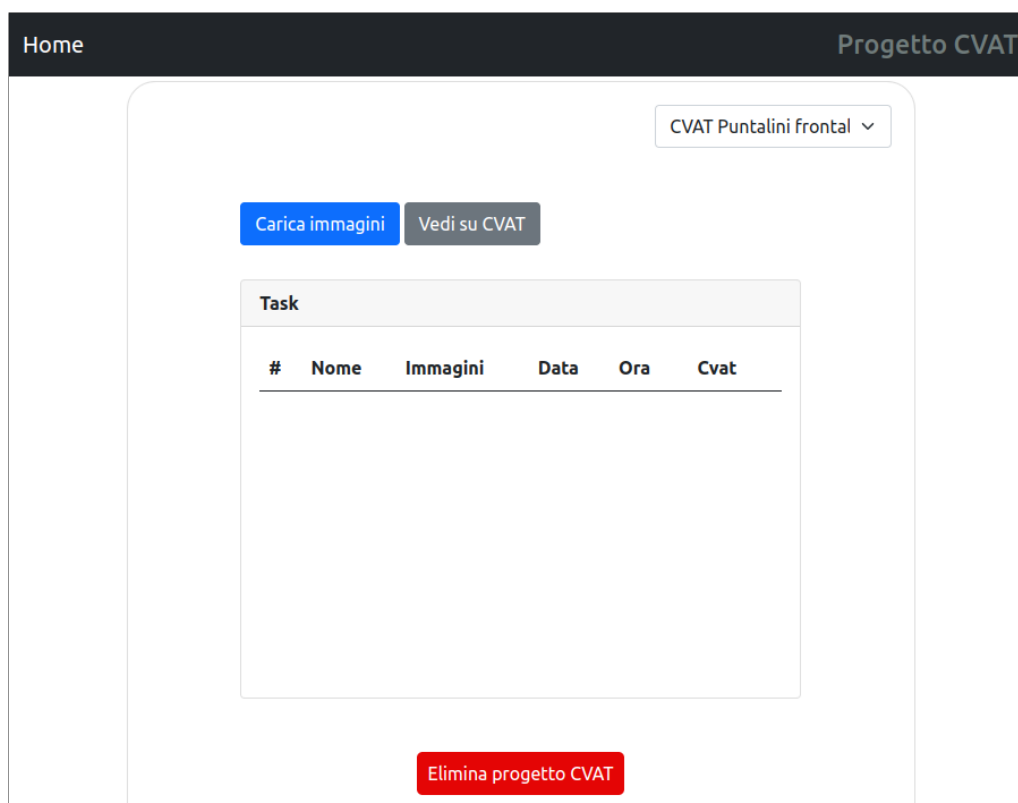


Figura 5.6: Progetto CVAT

Una volta aperta la pagina (Figura 5.7), è necessario compilare il *form* inserendo il nome che si vuole dare al task, scegliendo le immagini da caricare (selezionando una cartella) e opzionalmente filtrando le immagini per nome tramite il campo “filtro”. Prima di procedere al caricamento delle immagini cliccando su “Carica file” è possibile osservare l’anteprima di 10 immagini tra quelle che verranno caricate.

Home Caricamento immagini

Nome Task
Prima acquisizione puntalini

Filtro

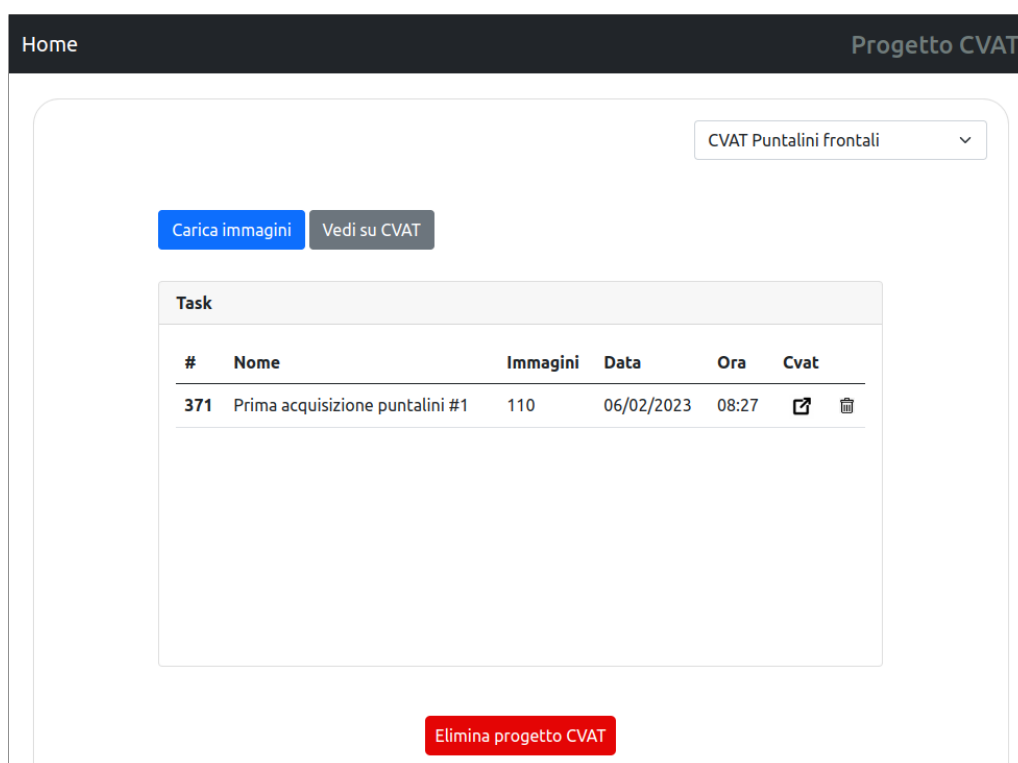
Scegli file 110 file

Carica file

Numero di immagini che soddisfano il filtro: 110 su 110

Figura 5.7: Caricamento immagini task

Dopo la creazione del task, si verrà reindirizzati di nuovo alla pagina relativa al progetto CVAT del task (Figura 5.8). Quest’ultimo è visualizzabile su CVAT, eliminabile ed è possibile avere alcune informazioni, quali id, nome, numero di immagini e data e ora di creazione.



The screenshot displays the CVAT web interface. At the top, there is a dark header with "Home" on the left and "Progetto CVAT" on the right. Below the header, a dropdown menu is set to "CVAT Puntalini frontali". Two buttons are visible: a blue "Carica immagini" button and a grey "Vedi su CVAT" button. A table titled "Task" contains one row of data. At the bottom of the interface, there is a red "Elimina progetto CVAT" button.



#	Nome	Immagini	Data	Ora	Cvat
371	Prima acquisizione puntalini #1	110	06/02/2023	08:27	 

Figura 5.8: Progetto CVAT con task completato

5.4 Caricamento verità

A questo punto si può caricare un file di verità. Per farlo, bisogna selezionare il progetto MVE a cui si è interessati, cliccare su “Azioni” e selezionare “Carica verità”, come visibile nella Figura 5.9.

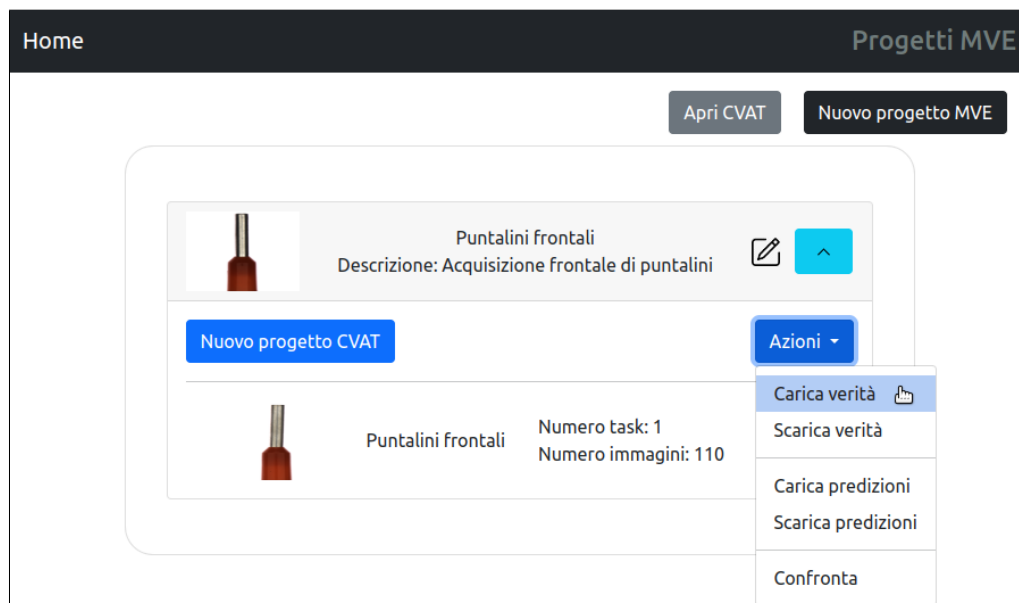


Figura 5.9: Scelta azione per caricamento verità

Si aprirà una *view* (Figura 5.10) in cui vi è un form con un unico campo per la selezione del file da caricare.

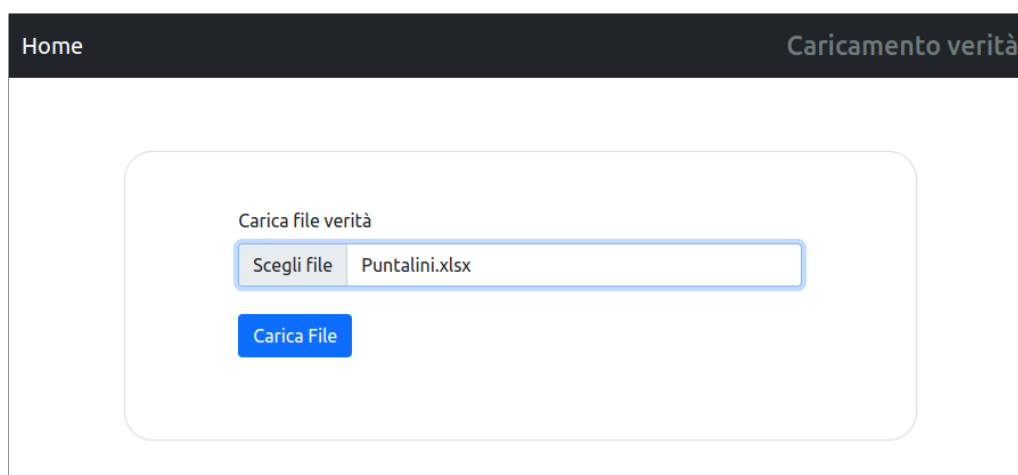


Figura 5.10: Caricamento file di verità

5.5 Caricamento predizioni

Una volta caricata la verità, si può passare al caricamento di un file con le predizioni, cliccando “Carica predizioni” tra le azioni disponibili del progetto MVE (Figura 5.11).

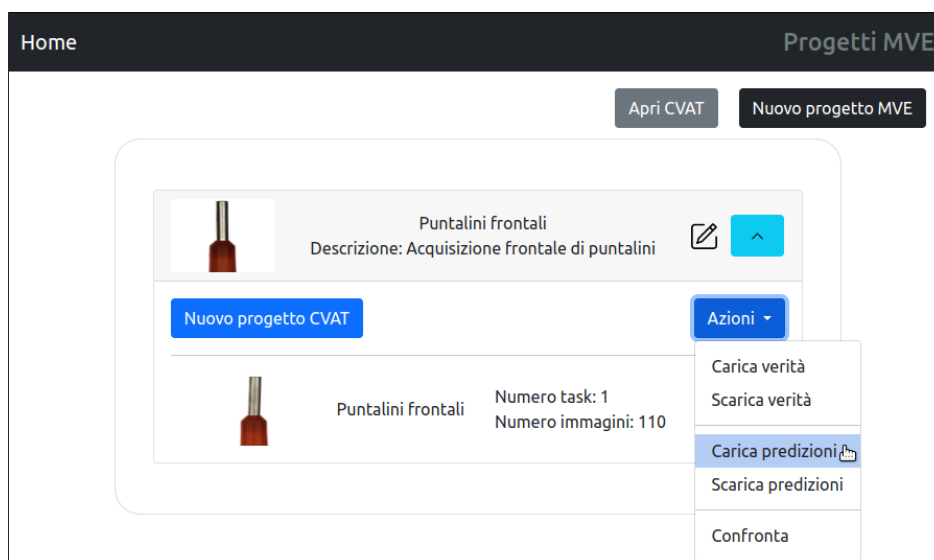
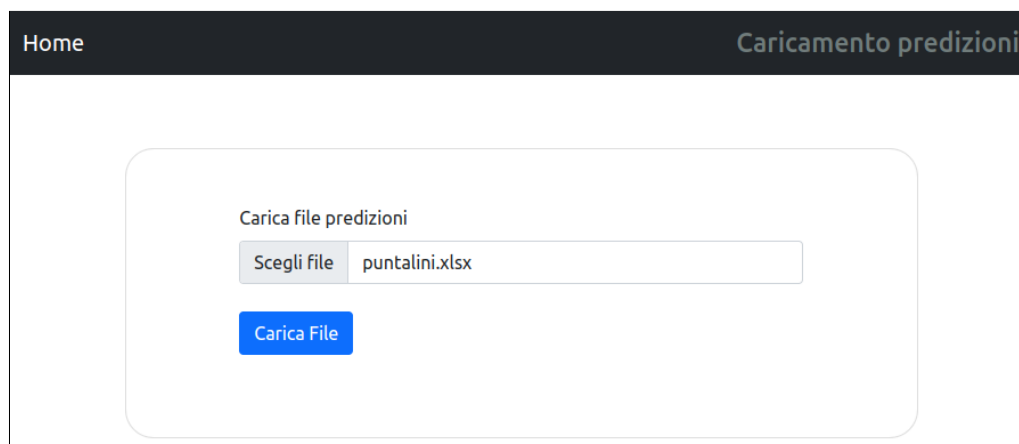


Figura 5.11: Scelta azione per caricamento predizioni

Verrà aperta una pagina (Figura 5.12) con un *form* costituito da un solo campo, anche in questo caso per la selezione del file da caricare.



The screenshot shows a web interface with a dark header bar. On the left, the word "Home" is visible. On the right, the text "Caricamento predizioni" is displayed. Below the header, there is a white rounded rectangular box containing the following elements: the text "Carica file predizioni", a file selection input field with a "Scegli file" button and the filename "puntalini.xlsx", and a blue "Carica File" button.

Figura 5.12: Caricamento primo file di predizioni

5.6 Confronto

A questo punto si può passare alla pagina di confronto selezionando la voce “Confronta” nel menu a tendina “Azioni”.

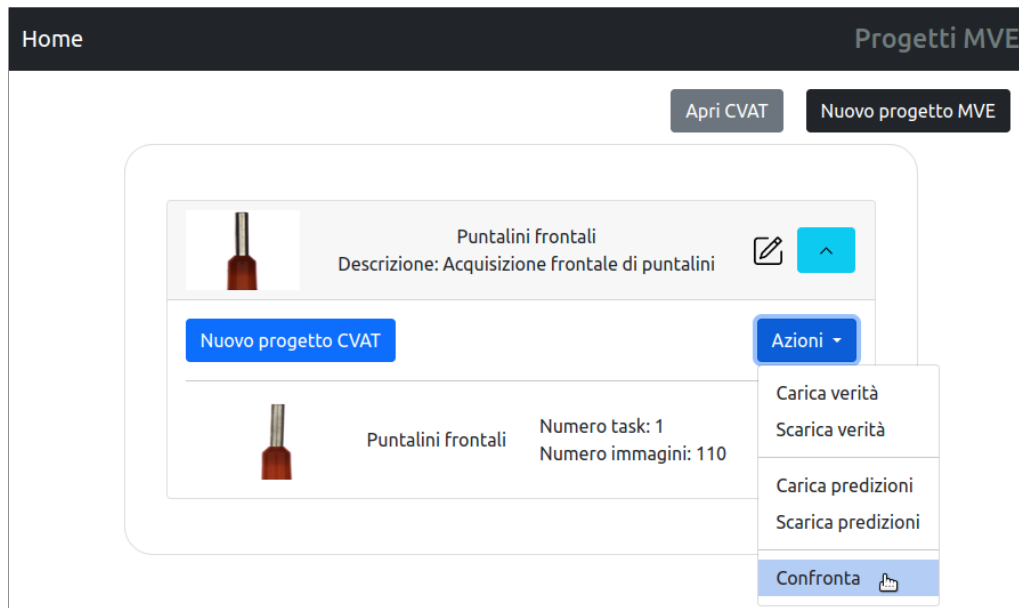


Figura 5.13: Scelta azione per confronto

Nella *view* che si apre è possibile scegliere se visualizzare una predizione, confrontare due predizioni oppure confrontare una predizione con la verità. Per esempio, nella Figura 5.14, è stato scelto di visualizzare il confronto tra una verità e una predizione, selezionando la predizione caricata poco fa. Una volta cliccato “Confronta” verrà generato un link a cui si accederà cliccando “Clicca qui per aprire su FiftyOne” che, come suggerisce la frase, aprirà una nuova finestra del browser con l’app di FiftyOne e il dataset appena creato. Inoltre, nella pagina è possibile visualizzare la cronologia dei confronti fatti in passato, con la possibilità di aprirli nuovamente su FiftyOne oppure cancellarli.

Home Confronto

Confronto

Predizione Predizione vs Predizione **Predizione vs Verità**

Prediction1, puntalini

Confronta

[Clicca qui per aprire su FiftyOne](#)

Cronologia confronti

#	Confronto		
3	Prediction1 vs Truth		
2	Prediction1 vs Prediction2		
1	Prediction1		

Figura 5.14: Confronto tra verità e predizione

5.7 FiftyOne

La pagina con l'app di FiftyOne si presenta come nella Figura 5.15: da una parte vi è la griglia con tutte le immagini del dataset, dall'altra l'elenco delle etichette e dei campi presenti nei campioni (ad esempio l'id e il *filepath*), tra cui le proprietà (ad esempio “*Lunghezza (mm)_Prediction1*”).

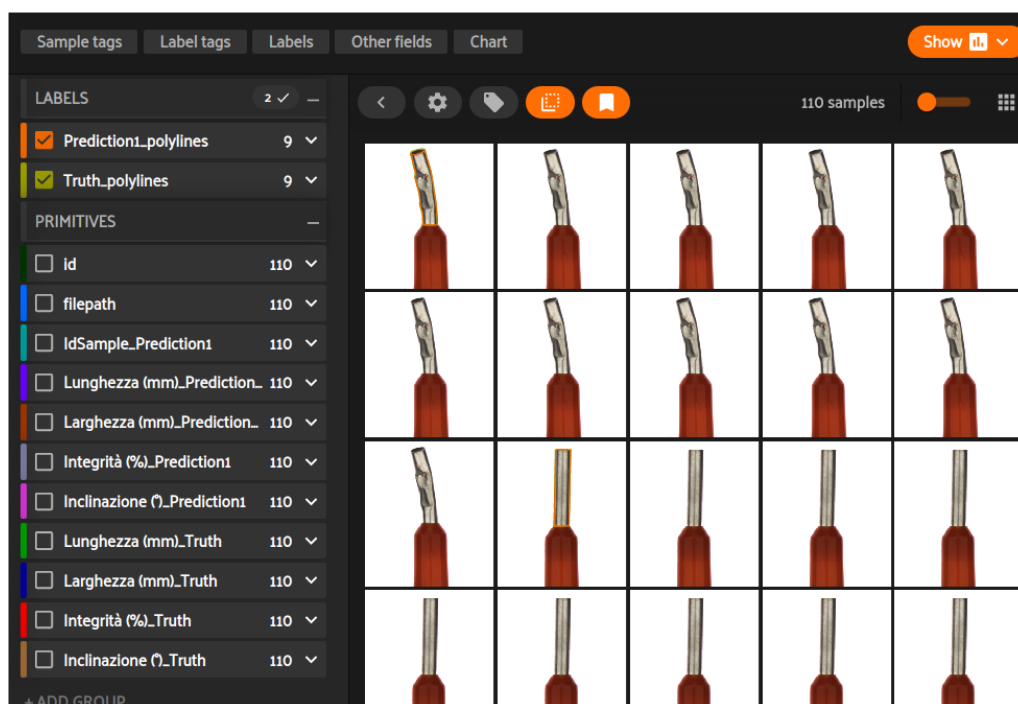


Figura 5.15: Homepage FiftyOne con dataset appena creato

Scegliendo un'immagine dalla griglia, questa verrà aperta a tutto schermo, come mostrato nella Figura 5.16. In questo modo è possibile vedere le annotazioni presenti sull'immagine e i valori dei campi/proprietà del campione, sia di verità (*Truth*) che di predizione (*Prediction1*).

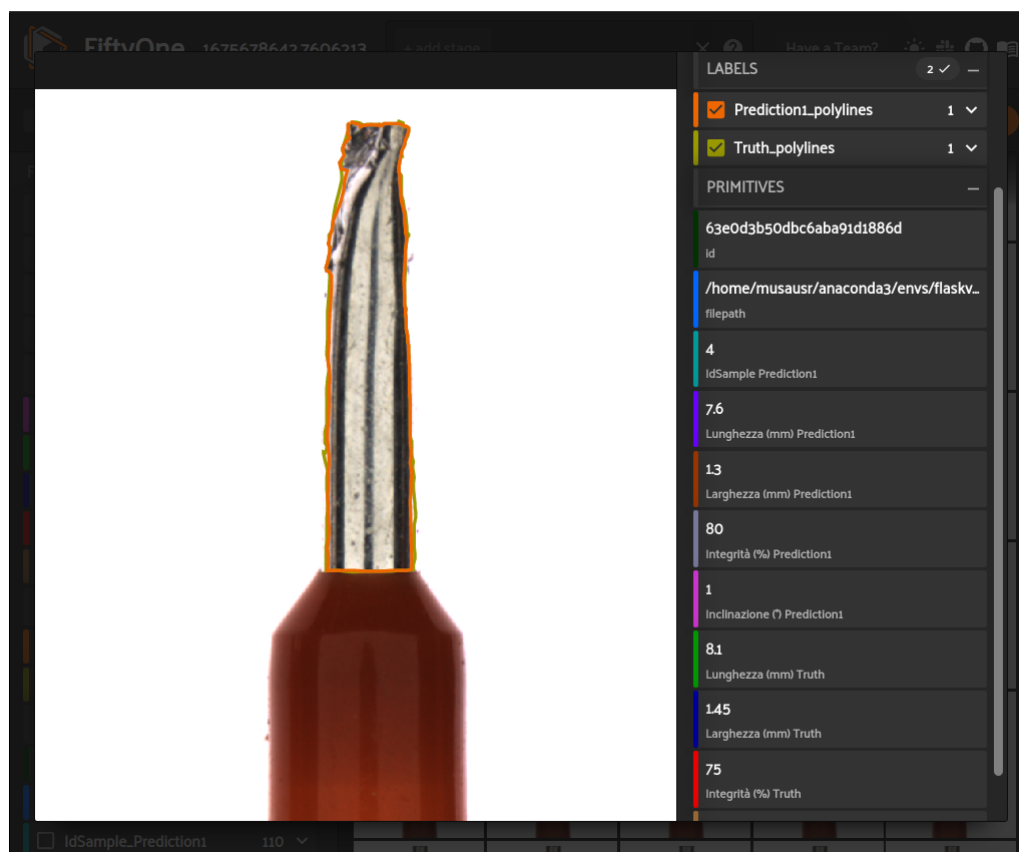


Figura 5.16: Selezione di un campione

Zoomando l'immagine come nella Figura 5.17, è possibile confrontare al meglio le segmentazioni presenti in essa.

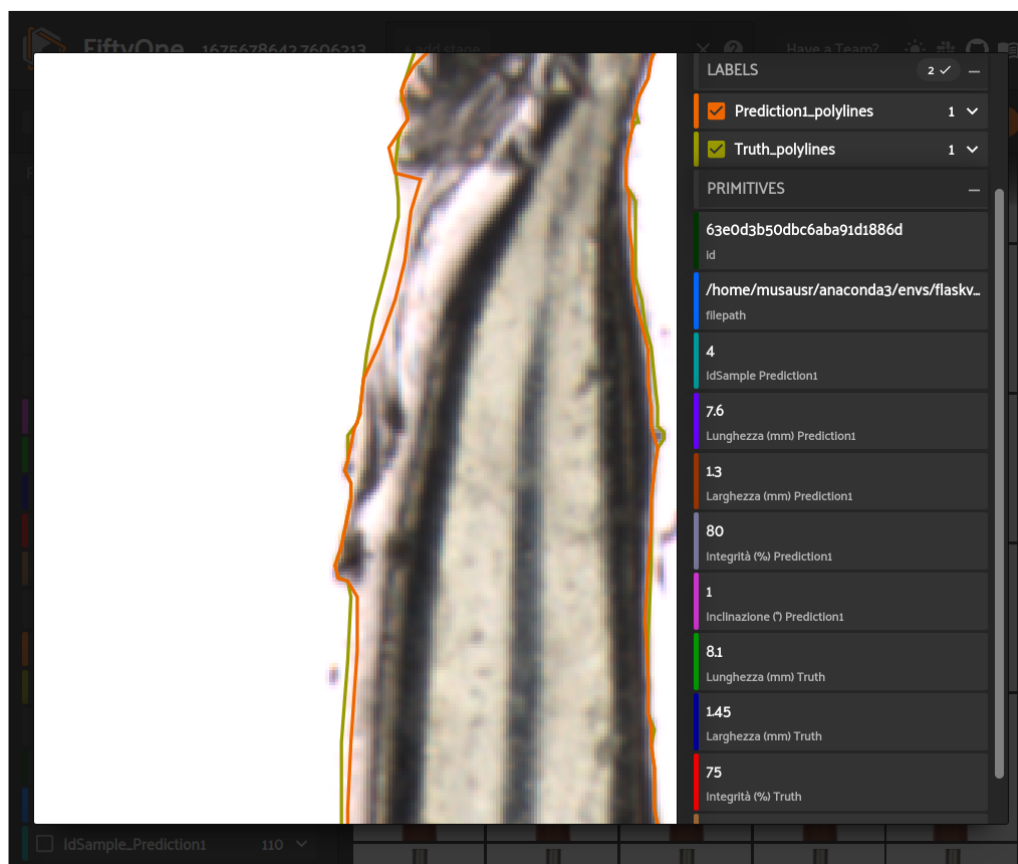


Figura 5.17: Selezione di un campione - zoom

Nel lato superiore dell'app FiftyOne vi è un bottone "Chart" che una volta cliccato eseguirà il plugin implementato (Figura 5.18). Dal menu a tendina è possibile scegliere le proprietà che si vogliono osservare e confrontare.

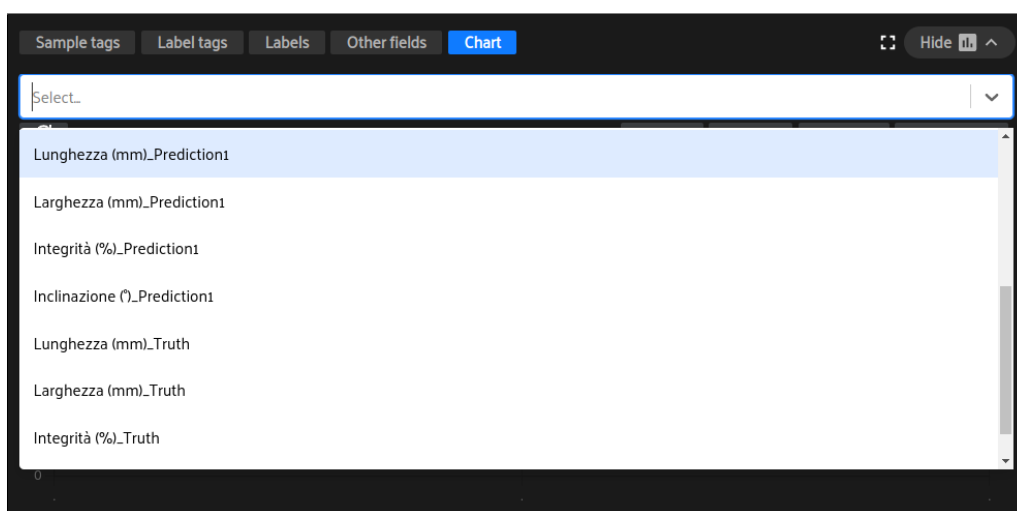


Figura 5.18: Selezione delle proprietà da visualizzare nel grafico

Ad esempio, scegliendo la proprietà “Lunghezza (mm)” inerente alla verità, il risultato sarà quello in Figura 5.19



Figura 5.19: Visualizzazione grafica di una proprietà verità

Successivamente, scegliendo la stessa proprietà inerente alla predizione, si avrà un grafico come quello nella Figura 5.20.



Figura 5.20: Confronto grafico tra i valori di una proprietà

Ogni punto del grafico è cliccabile e fa riferimento a un campione del dataset; pertanto, cliccandone uno verrà selezionata dalla griglia l'immagine corrispondente, come nella Figura 5.21.

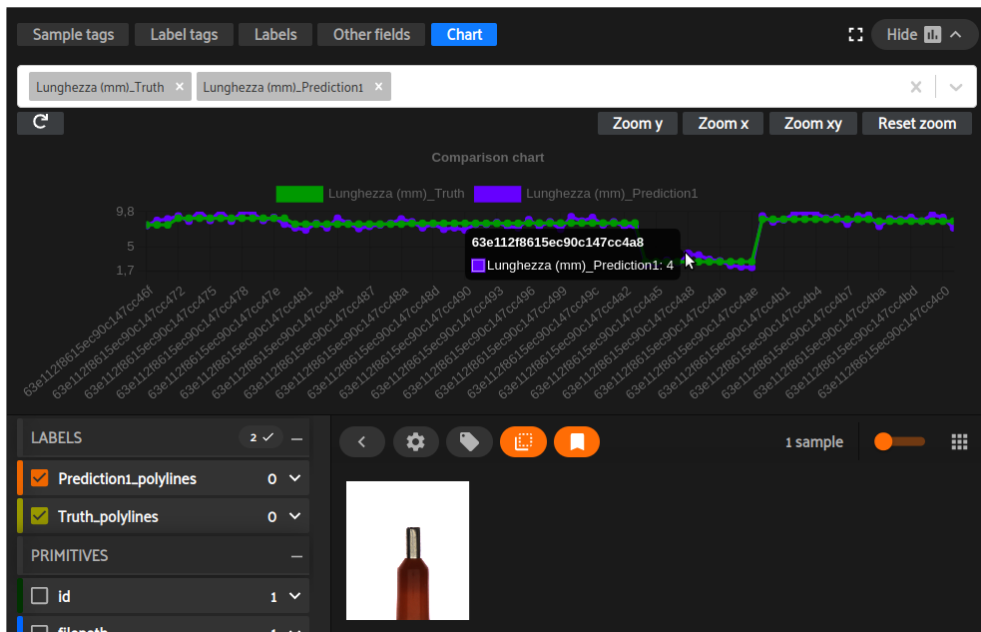


Figura 5.21: Click su un punto e selezione del relativo campione

Conclusioni

Nel corso della tesi è stato illustrato l'intero processo di sviluppo dell'applicativo MVE: si è partiti dalla descrizione del contesto di utilizzo, per poi passare alla definizione dell'architettura software e delle tecnologie utilizzate per attuare lo sviluppo; dopodiché, sono state definite le entità coinvolte e le funzionalità da soddisfare, seguite dalla specifica della configurazione di CVAT, di FiftyOne e del database MySQL e dall'implementazione dell'applicazione web e del plugin FiftyOne; infine, è stata fornita una guida operativa sul corretto utilizzo del software.

Durante l'esposizione del contesto di utilizzo, è stato precisato che l'azienda si occupa del controllo della qualità delle linee di produzione, utilizzando un sistema di visione che acquisisce ed elabora immagini. MVE introduce una procedura standard da eseguire sui dati ottenuti dal sistema di visione e crea un ambiente unificato per l'archiviazione delle immagini e della verità, permettendo il confronto di quest'ultima con il risultato degli algoritmi di elaborazione. Questo è possibile perché MVE è stato sviluppato per integrare CVAT, FiftyOne e MVS in un singolo ambiente operativo.

MVS è il sistema di visione attualmente in uso, su cui l'azienda ha pianificato delle attività per poterlo integrare con questo nuovo strumento. Le modifiche non sono ancora state sviluppate, pertanto l'applicazione MVE non è ancora stata utilizzata nel suo contesto finale.

Nel corso dello sviluppo di questo progetto vi è stato modo di interagire con tecnologie all'avanguardia, quali CVAT e FiftyOne, studiandole a fondo e prendendo familiarità con esse, per poterle utilizzare al meglio. Il loro punto

di forza è essere open source, in particolare CVAT fa parte di OpenCV, una delle più note librerie open source di Computer Vision, utilizzata sia nel mondo del lavoro che nel mondo della ricerca. L'uso di questi strumenti ha portato allo studio e all'utilizzo di altre tecnologie, come npm, Yarn e React, ampliando ulteriormente le conoscenze acquisite.

Bibliografia

- [1] *CNI Group*, <http://www.cnigroup.net/>.
- [2] *CVAT*, https://en.wikipedia.org/wiki/Computer_Vision_Annotation_Tool.
- [3] *CVAT Server API*, https://opencv.github.io/cvat/docs/api_sdk/api/.
- [4] *REST API*, <https://www.redhat.com/it/topics/api/what-is-a-rest-api>.
- [5] *REST API CVAT*, <https://app.cvat.ai/api/docs/>.
- [6] *FiftyOne GitHub*, <https://github.com/voxel51/fiftyone>.
- [7] *Documentazione FiftyOne*, <https://voxel51.com/docs/fiftyone/>.
- [8] *Documentazione plugin FiftyOne*, <https://docs.voxel51.com/plugins/index.html>.
- [9] *Documentazione React*, <https://it.reactjs.org/>.
- [10] *React*, [https://en.wikipedia.org/wiki/React_\(JavaScript_library\)](https://en.wikipedia.org/wiki/React_(JavaScript_library)).
- [11] *MySQL*, <https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html>.
- [12] *Flask*, [https://it.wikipedia.org/wiki/Flask_\(informatica\)](https://it.wikipedia.org/wiki/Flask_(informatica)).

-
- [13] *Python* <https://it.wikipedia.org/wiki/Python>.
 - [14] *Documentazione Python*, <https://docs.python.org/3/library>.
 - [15] *Documentazione Flask*, <https://flask.palletsprojects.com/>.
 - [16] *Python Package Index*, <https://pypi.org/>.
 - [17] *HTML*, <https://it.wikipedia.org/wiki/HTML>.
 - [18] *CSS*, <https://it.wikipedia.org/wiki/CSS>.
 - [19] *JavaScript*, <https://it.wikipedia.org/wiki/JavaScript>.
 - [20] *jQuery*, <https://it.wikipedia.org/wiki/jQuery>.
 - [21] *AJAX*, <https://it.wikipedia.org/wiki/AJAX>.
 - [22] *Bootstrap*, [https://it.wikipedia.org/wiki/Bootstrap_\(informatica\)](https://it.wikipedia.org/wiki/Bootstrap_(informatica)).
 - [23] *Jinja*, [https://it.wikipedia.org/wiki/Jinja_\(software\)](https://it.wikipedia.org/wiki/Jinja_(software)).
 - [24] *CVAT Guida installazione*, <https://opencv.github.io/cvat/docs/administration/basics/installation/>.
 - [25] *FiftyOne Guida installazione*, <https://github.com/voxel51/fiftyone#installing-from-source>.
 - [26] Jagreet Kaur, *Introduction to Python Flask Framework*, <https://www.xenonstack.com/blog/python-flask-framework>, (consultato il 12/01/2020).
 - [27] *CVAT API Login*, https://app.cvat.ai/api/docs/#tag/auth/operation/auth_create_login.
 - [28] *CVAT API Crea progetto*, https://app.cvat.ai/api/docs/#tag/projects/operation/projects_create.

-
- [29] *CVAT API Crea task*, https://app.cvat.ai/api/docs/#tag/tasks/operation/tasks_create.
- [30] *CVAT API Carica immagini task*, https://app.cvat.ai/api/docs/#tag/tasks/operation/tasks_create_data.
- [31] *CVAT API Lista progetti*, https://app.cvat.ai/api/docs/#tag/projects/operation/projects_list.
- [32] *CVAT API Lista task progetto*, https://app.cvat.ai/api/docs/#tag/projects/operation/projects_list_tasks.
- [33] *CVAT API Dettagli progetto*, https://app.cvat.ai/api/docs/#tag/projects/operation/projects_retrieve.
- [34] *CVAT API Elimina progetto*, https://app.cvat.ai/api/docs/#tag/projects/operation/projects_destroy.
- [35] *CVAT API Elimina task*, https://app.cvat.ai/api/docs/#tag/tasks/operation/tasks_destroy.
- [36] *CVAT API Download dataset task*, https://app.cvat.ai/api/docs/#tag/tasks/operation/tasks_retrieve_dataset.

Ringraziamenti

Vorrei dedicare questo spazio alle persone che mi hanno aiutata a realizzare questo elaborato e che mi hanno accompagnata in questo percorso di crescita personale e professionale.

Ringrazio il mio relatore, Gabbrielli Maurizio, per la sua disponibilità e per aver supervisionato la presente tesi.

Un ringraziamento speciale va al mio correlatore e tutor aziendale, Convertino Edoardo, che mi ha guidata in ogni fase della realizzazione della tesi e del progetto esposto in essa.

Ringrazio tutto lo staff dell'azienda CNI Informatica S.R.L., in cui ho svolto un tirocinio formativo della durata di 5 mesi e complementare alla redazione della tesi, per l'ospitalità e per le competenze acquisite.

Ringrazio i miei genitori e mio fratello Marco, che mi hanno sempre supportata e che mi hanno aiutata a superare i momenti più difficili. Senza il loro appoggio questo percorso sarebbe stato molto più difficile.

Un grandissimo grazie va ai miei amici, a quelli di sempre e a quelli che sono entrati a far parte della mia vita di recente. Grazie per i momenti di spensieratezza che mi avete regalato, per le risate e per le difficoltà superate insieme.

Ringrazio Veronica, che da quasi dieci anni mi sostiene in ogni mia impresa. Grazie per esserci e credere in me sempre. Grazie per dedicarmi sempre un po' del tuo tempo e per le serate passate sul tuo divano a guardare un film anche se puntualmente finiscono con te che ti addormenti dopo mezz'ora. Grazie perché so di poter contare su di te in ogni momento.

Ringrazio Giulia, che in poco tempo è diventata indispensabile. Grazie per avermi supportata e sopportata in questi ultimi mesi, per aver creduto in me quando tutto mi sembrava impossibile e per aver affrontato insieme a me tante fatiche. Grazie per non farmi sentire mai sbagliata, per darmi forza, per ascoltarmi sempre e per le parole giuste dette sempre al momento giusto, specialmente nei momenti di sconforto in cui tendo a sottovalutarmi. Grazie per esserci, perché non riesco ad immaginare questi mesi passati e quelli a venire senza di te al mio fianco.

Ringrazio Giada, con cui ho condiviso tanti momenti in questi anni, in particolare quelli passati a Bologna durante il mio primo anno della magistrale, senza i quali sarebbe sicuramente stato tutto più triste e noioso.

Grazie ai miei compagni di studio, Beatrice, Alberto e Federico, con cui ho condiviso gioie e fatiche in questi due anni pieni di progetti di gruppo svolti insieme.

Ringrazio la mia squadra di calcio a 5, che mi ha permesso di scaricare la tensione e di divertirmi.

Grazie a tutti.