

UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Ingegneria "Enzo Ferrari"

Corso di Laurea Triennale in Ingegneria Informatica (L-08)

Progettazione, Implementazione e Configurazione di un Applicativo Web per la Raccolta di Dati Clinici Attraverso Sondaggi

Tutor:

Prof. Costantino Grana

Prof. Federico Bolelli

Candidato:

Michele Mosca

Matricola 118640

ANNO ACCADEMICO 2021/2022

Indice

| | |
|---|-----------|
| Introduzione | 1 |
| 1 Python | 2 |
| 1.1 Cenni storici | 2 |
| 1.2 Funzionalità e potenzialità | 2 |
| 1.3 Sviluppo Web | 3 |
| 2 Django | 4 |
| 2.1 Potenzialità | 4 |
| 2.2 Architettura, componenti e funzionamento | 5 |
| 2.2.1 Il paradigma MVC in Django | 5 |
| 2.2.2 Struttura di un sito Django | 6 |
| 2.2.3 Database | 7 |
| 2.3 Download ed installazione su un sistema Linux | 7 |
| 2.3.1 Creazione di un ambiente virtuale | 8 |
| 2.3.2 Installazione di Django e primo avvio | 8 |
| 2.4 Sviluppo Django con PyCharm | 9 |
| 2.4.1 Creazione di un progetto | 9 |
| 2.4.2 PyCharm Professional Edition | 11 |
| 3 Javascript | 12 |
| 3.1 Per cosa viene utilizzato Javascript? | 12 |
| 3.1.1 Aggiornamento dinamico dei contenuti | 12 |
| 3.2 Javascript lato client | 13 |
| 3.3 Ajax | 14 |
| 3.3.1 Implementazione di Ajax | 14 |

| | | |
|----------|--|-----------|
| 4 | Single Sign-On | 15 |
| 4.1 | Come funziona il Single Sign-On? | 15 |
| 4.1.1 | Vantaggi dell'autenticazione Single Sign-On | 17 |
| 4.2 | Implementazione SSO su un progetto Django nel server Unimore | 18 |
| 4.2.1 | Configurazione di Apache2 per l'uso del SSO | 18 |
| 4.2.2 | Creazione delle View per SSO | 19 |
| 4.2.3 | Inserimento degli Urls | 21 |
| 4.2.4 | Aggiunta del pannello di SSO in HTML | 22 |
| 5 | Installazione e configurazione del software | 23 |
| 5.1 | Requisiti software | 23 |
| 5.1.1 | Clone del progetto da GitHub | 23 |
| 5.1.2 | Creazione del Virtual Environment e successiva attivazione | 23 |
| 5.1.3 | Package Python | 23 |
| 5.2 | Configurazione in base al server Unimore | 24 |
| 5.2.1 | Modifica del file wsgi.py | 25 |
| 5.2.2 | Aggiunta della Web Application su Apache2 | 26 |
| 5.2.3 | Assegnazione dei permessi | 27 |
| 5.2.4 | Aggiunta del dominio | 27 |
| 5.3 | Configurazione dell'account amministratore e del database | 28 |
| 5.3.1 | Migrazione del database | 28 |
| 5.3.2 | Creazione account amministratore | 28 |
| 5.3.3 | Scrittura del file secrets.json | 28 |
| 5.3.4 | Assegnazione permessi ad Apache2 | 29 |
| 6 | Analisi del software | 30 |
| 6.1 | Struttura del progetto | 30 |
| 6.2 | Struttura dati del Model | 31 |
| 6.2.1 | Survey_collection | 32 |
| 6.2.2 | Survey | 33 |
| 6.2.3 | Image | 33 |

| | | |
|-------|---|-----------|
| 6.2.4 | Image_Collection | 33 |
| 6.2.5 | Image_Transformation | 34 |
| 6.2.6 | Choice | 34 |
| 6.2.7 | Answer | 35 |
| 6.3 | Struttura delle Views | 35 |
| 6.3.1 | Schermata Iniziale | 36 |
| 6.3.2 | Accesso alla piattaforma | 37 |
| 6.3.3 | Registrazione | 41 |
| 6.3.4 | Home utente | 43 |
| 6.3.5 | Visualizzazione della Collection | 45 |
| 6.3.6 | Pagina del quesito | 51 |
| 6.3.7 | Pannello di amministrazione | 55 |
| 6.3.8 | Visualizzazione dei risultati | 57 |
| 6.4 | Script di gestione delle collezioni | 60 |
| 6.4.1 | Scrittura del file di configurazione YAML | 60 |
| 6.4.2 | Analisi del codice | 62 |
| 6.5 | Documentazione | 70 |
| | Conclusioni | 71 |
| | Ringraziamenti | 73 |

Elenco delle figure

| | | |
|------|--|----|
| 1.1 | What do you use Python for the most? (Python Developers Survey 2018) | 3 |
| 2.1 | Pattern Model-View-Controller (MVC) | 5 |
| 2.2 | Filesystem di un sito Django | 6 |
| 2.3 | PyCharm, creazione di un nuovo progetto Django | 10 |
| 2.4 | PyCharm Professional Edition, interfaccia | 11 |
| 4.1 | Procedura di autenticazione mediante Single Sign-On | 16 |
| 4.2 | Procedura di verifica mediante Single Sign-On | 17 |
| 5.1 | Package Python | 24 |
| 6.1 | Struttura del progetto | 31 |
| 6.2 | Schema del database | 32 |
| 6.3 | Schermata iniziale dell'applicativo | 36 |
| 6.4 | Schermata di login mediante credenziali Unimore | 37 |
| 6.5 | Schermata di login completa con anche la sezione per amministratori | 39 |
| 6.6 | Schermata di registrazione | 41 |
| 6.7 | Schermata Home dell'utente | 43 |
| 6.8 | Visualizzazione della Collection | 45 |
| 6.9 | Risposta inviata con successo | 48 |
| 6.10 | Schermata del quesito | 51 |
| 6.11 | Pannello di amministrazione | 55 |
| 6.12 | Visualizzazione dei risultati | 57 |
| 6.13 | Visualizzazione dettagli dei risultati | 59 |

Elenco dei Codici

| | | |
|------|--|----|
| 3.1 | Codice di inclusione della tecnologia Ajax mediante CDN | 14 |
| 4.1 | Configurazione di Apache2 per l'uso del SSO | 18 |
| 4.2 | View per effettuare i test con il SSO | 20 |
| 4.3 | View per l'autenticazione mediante SSO | 20 |
| 4.4 | Aggiunta degli URLS dedicati al SSO | 21 |
| 5.1 | Clone del progetto da GitHub | 23 |
| 5.2 | Configurazione file wsgi.py | 25 |
| 5.3 | Configurazione di Apache2 | 26 |
| 5.4 | Assegnazione dei permessi | 27 |
| 6.1 | Definizione del modello Survey_Collection | 32 |
| 6.2 | Definizione del modello Survey | 33 |
| 6.3 | Definizione del modello Image | 33 |
| 6.4 | Definizione del modello Image_Collection | 33 |
| 6.5 | Definizione del modello Image_Transformations | 34 |
| 6.6 | Definizione del modello Choice | 34 |
| 6.7 | Definizione del modello Answer | 35 |
| 6.8 | Visualizzazione della schermata iniziale | 36 |
| 6.9 | Accesso mediante Single SignOn di Unimore | 37 |
| 6.10 | Accesso mediante credenziali dell'applicativo | 39 |
| 6.11 | Registrazione di un nuovo utente | 41 |
| 6.12 | Visualizzazione dei surveys assegnati all'utente | 43 |
| 6.13 | Visualizzazione della collection | 46 |
| 6.14 | Submit della risposta mediante Ajax | 48 |
| 6.15 | Visualizzazione del singolo quesito | 51 |
| 6.16 | Visualizzazione del pannello di amministrazione | 55 |
| 6.17 | Visualizzazione dei risultati | 57 |
| 6.18 | Esempio di configurazione per la creazione di una nuova collezione | 61 |

| | | |
|------|---|----|
| 6.19 | Esempio di configurazione per la modifica di una collezione | 62 |
| 6.20 | Funzione per la creazione o modifica delle collezioni | 62 |
| 6.21 | Dizionario dei messaggi di errore | 64 |
| 6.22 | Inserimento delle possibili trasformazioni | 65 |
| 6.23 | Aggiunta delle opzioni di scelta | 65 |
| 6.24 | Caricamento delle nuove immagini | 66 |
| 6.25 | Applicazione delle trasformazioni su un'immagine per un dato utente | 68 |
| 6.26 | Abilitazione degli utenti all'accesso alla collezione | 69 |

Introduzione

Il seguente elaborato descrive le fasi di installazione, configurazione e analisi della piattaforma di raccolta dati inerenti alle malattie della pelle attraverso l'uso di sondaggi clinici.

Il software è frutto dell'attività progettuale svolta presso i laboratori di ricerca dell'università di Modena e Reggio Emilia, il suo scopo è quello di realizzare una piattaforma in cui un gruppo di dermatologi può esprimere una diagnosi su dei campioni di possibili malattie della pelle, rappresentati mediante delle collezioni di immagini. Le diagnosi vengono effettuate attraverso la risposta a dei semplici quesiti, le risposte serviranno alla creazione di un *dataset* per il futuro sviluppo di una rete neurale in grado di riconoscere la malattie.

Si presenta ora un riassunto di ciò che sarà affrontato nei Capitoli successivi.

I Capitoli 1, 2, 3 e 4 affronteranno aspetti tecnici e teorici delle tecnologie utilizzate nel progetto, mentre i Capitoli 5 e 6 affronteranno aspetti pratici, legati all'installazione, utilizzo e manutenzione della Piattaforma.

- Nel **Capitolo 1** si tratterà di **Python**, delle sue funzionalità e delle sue potenzialità nello sviluppo Web.
- Nel **Capitolo 2** si tratterà di **Django**, dei suoi componenti, del suo funzionamento e delle sue potenzialità.
- Nel **Capitolo 3** si tratterà di **Javascript**, del perché del suo utilizzo, dell'integrazione con il Browser Web e della tecnologia **Ajax**.
- Nel **Capitolo 4** si tratterà del **Single Sign-On**, delle sue funzionalità, dei suoi vantaggi e di come implementarlo all'interno del proprio progetto *Django*.
- Nel **Capitolo 5** verranno presentati nel dettaglio quali sono i passi fondamentali da svolgere per l'**installazione e la configurazione** dell'applicativo sviluppato.
- Nel **Capitolo 6** si esplorerà la **struttura del progetto**, si andranno ad analizzare i **modelli** e le **view** create, commentando nel dettaglio il codice.

1. Python

Python è un linguaggio di programmazione dinamico di “alto livello”, orientato agli oggetti, adatto, tra gli altri usi, a sviluppare applicazioni distribuite, scripting, computazione numerica e system testing.

1.1 Cenni storici

Rilasciato pubblicamente per la prima volta nel 1991 dal suo creatore **Guido Van Rossum**, deriva il suo nome dalla commedia *Monty Python's Flying Circus*, in onda sulla BBC nel corso degli anni Settanta. Attualmente, lo sviluppo di Python, grazie all'enorme e dinamica comunità internazionale di sviluppatori, viene gestito dall'organizzazione no-profit *Python Software Foundation*. Con il rilascio della versione 3.0 di Python nel 2008 viene semplificato il linguaggio introducendo diversi miglioramenti, come ad esempio l'uso predefinito di stringhe Unicode.

1.2 Funzionalità e potenzialità

Python è un linguaggio **pseudocompilato**: un interprete si occupa di analizzare il codice sorgente e, se sintatticamente corretto, di compilarlo al momento in formato *bytecode* e di eseguirlo, garantendo quindi prestazioni elevate. Essendo un linguaggio **multi-paradigma**, supporta sia la programmazione procedurale, che la programmazione ad oggetti. Ogni installazione di Python include una collezione di oltre 200 moduli per svolgere i compiti più disparati, come ad esempio l'interazione con il sistema operativo e il filesystem, o la gestione di diversi protocolli.

Il meccanismo di Garbage Collection si occupa automaticamente dell'allocazione e del rilascio della memoria: questo consente di usare le variabili liberamente, senza doversi preoccupare di dichiararle e di allocare e rilasciare spazi di memoria manualmente.

1.3 Sviluppo Web

Esistono svariate possibilità per lo sviluppo Web in Python grazie ai diversi web framework disponibili, come:

- **Django**: uno dei framework più popolari, che fornisce diversi strumenti per la realizzazione di siti e Web Applications.
- **Flask**: un framework di dimensioni minori, che permette di creare rapidamente siti Web semplici.

Ne esistono chiaramente molti altri che permettono la realizzazione di ogni tipologia di Web Application: Web Frameworks for Python [17] è la pagina ufficiale che include un elenco aggiornato di Web Frameworks corredati da una breve descrizione.

Secondo le indagini di mercato indette da parte di *JetBrains*, nel 2018 l'ambito di utilizzo primario degli sviluppatori Python è infatti il **Web Development**.

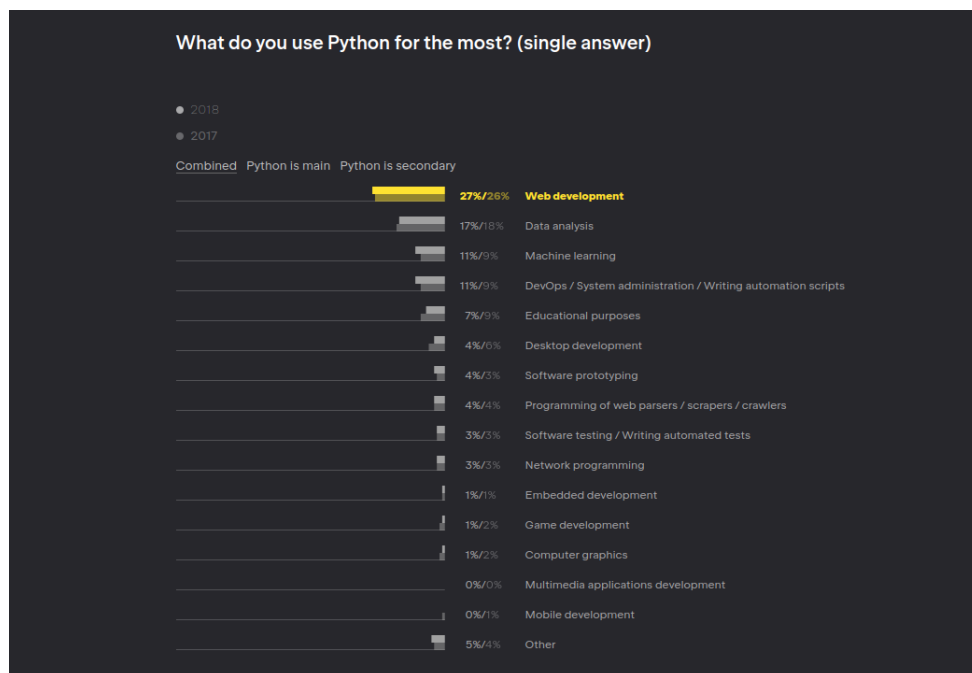


Figura 1.1: What do you use Python for the most? (Python Developers Survey 2018)

2. Django

Nato nel 2005, Django è un framework Web Python di alto livello che consente il rapido sviluppo di siti Web sicuri e gestibili. Gestito dall'organizzazione *Django Software Foundation*, è gratuito ed open source, ha una comunità fiorente e attiva, un'ottima documentazione e molte opzioni per il supporto sia gratuito che a pagamento.

2.1 Potenzialità

Django permette di scrivere un software che sia:

- **Completo:** ogni componente funziona coerentemente con gli altri, segue i principi di progettazione software e dispone di una documentazione ampia ed aggiornata.
- **Versatile:** Django può essere utilizzato per costruire quasi ogni tipo di sito Web, dai sistemi di gestione dei contenuti ai social network. Può funzionare con qualsiasi framework lato client e può fornire contenuti in quasi tutti i formati utili per lo sviluppo Web, come HTML, feed RSS, JSON, XML.
- **Sicuro:** Django abilita delle misure di protezione predefinite contro numerose vulnerabilità, come *SQL Injection*, *Cross-Site Scripting(XSS)*, *Cross-Site Request Forgery (CSRF)*, *Clickjacking*.
- **Scalabile:** ogni componente è indipendente dagli altri, quindi è possibile aggiungere hardware a qualsiasi livello in base alla necessità di traffico.
- **Mantenibile:** il codice di cui è composto Django segue il principio *Don't Repeat Yourself (DRY)*, per cui non vi sono duplicazioni futili di codice. Inoltre, essendo raggruppato in moduli riutilizzabili secondo il modello *Model-View-Controller (MCV)*, è facilmente mantenibile per lo sviluppatore.
- **Portatile:** essendo scritto in Python, le Web Applications possono essere eseguite su molte versioni di Linux, Windows, Mac OS X e provider di *Web Hosting*.

2.2 Architettura, componenti e funzionamento

2.2.1 Il paradigma MVC in Django

Il pattern *Model-View-Controller* (MVC) si propone di separare la rappresentazione del modello dei dati (**Model**), l'interfaccia utente (**View**) e la logica applicativa (**Controller**), anche detta logica di "business".

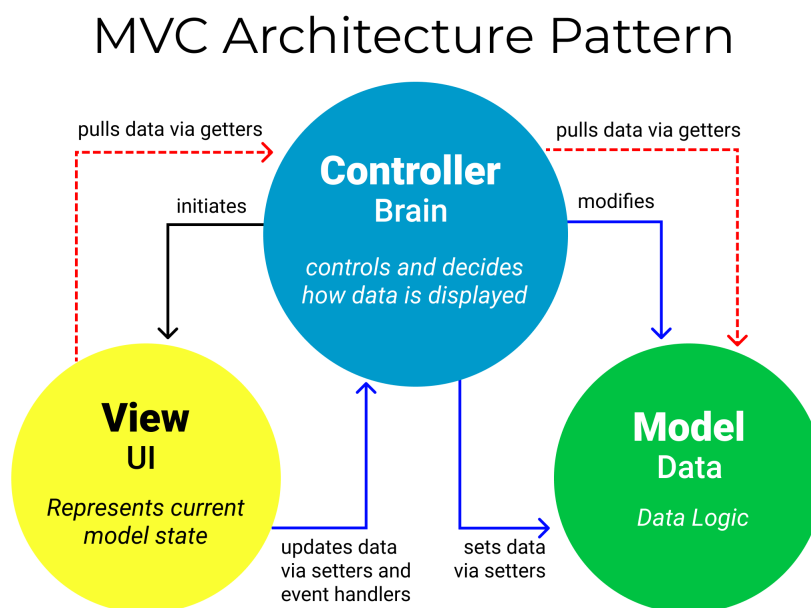


Figura 2.1: Pattern Model-View-Controller (MVC)

In modo analogo, ogni applicazione in Django è composta da *Model*, *View* e *Template*, come mostrato nella *Figura 2.1*

- **Model:** È costituito da un insieme di classi Python utili per la rappresentazione del modello dei dati. Queste classi forniscono una rappresentazione per le tabelle del database e ciò consente di utilizzare gli oggetti per effettuare diverse operazioni sui dati, al posto delle classiche query SQL: si parla quindi di **Object-Relational Mapping (ORM)**. Tutte le definizioni delle classi sono contenute nel file *models.py* nella directory dell'applicazione.

- **View:** È rappresentato da funzioni Python che gestiscono il controllo di flusso dell'applicazione ed implementano la componente *Controller* del pattern *MVC*. Grazie a tali funzioni possiamo definire le pagine all'interno dell'applicazione e i comportamenti che queste pagine avranno in funzione dell'interazione con l'utente. Ad esempio, in una *View* è possibile prendere una serie di dati dalla componente *Model*, elaborarli e decidere quale *Template* utilizzare per mostrare i dati richiesti oppure richiamare un'altra *View* per eseguire ulteriori funzioni.
- **Template:** È composto da file testuali i quali descrivono la presentazione dei dati, implementando di fatto la componente *View* del pattern *MVC*.

2.2.2 Struttura di un sito Django

Un sito Django può contenere più Web Applications e la sua struttura riflette nel relativo filesystem: il sito Web viene memorizzato in una directory contenente gli script Python di configurazione ed una sottodirectory per ciascuna applicazione, come mostrato nella *Figura 2.2*

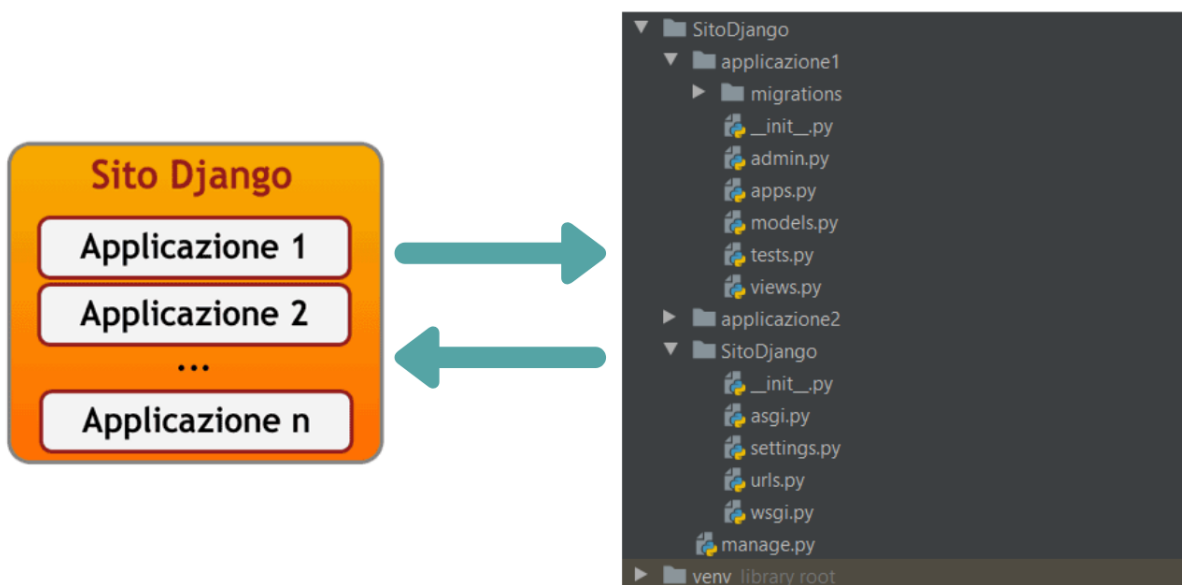


Figura 2.2: Filesystem di un sito Django

Di seguito vengono elencati i file principali che compongono il progetto:

- **__init__.py:** file vuoto che indica all'interprete di trattare la directory come un package Python.

- **settings.py**: file contenente tutte le impostazioni del sito Web. Qui è dove sono registrate tutte le applicazioni che vengono create, la posizione dei file statici, i dettagli di configurazione del database ed altre numerose opzioni.
- **urls.py**: definisce i mapping *url-to-view* del sito Web dichiarando gli URL gestiti dal progetto.
- **models.py**: contiene le definizioni delle classi *Model*, ossia delle strutture dati con cui interagisce l'applicazione.
- **views.py**: contiene le definizioni delle *View* dell'applicazione.
- **admin.py**: file utile sia per definire quali classi *Model* possono essere oggetto di modifica da parte dell'amministratore, sia per configurare il pannello di amministrazione Django.
- **manage.py**: script che viene eseguito da linea di comando al fine di creare applicazioni, lavorare con il database ed eseguire il server.

2.2.3 Database

Django supporta quattro database principali: *Postgre SQL*, *MySQL*, *Oracle* ed *SQLite*. Anche se Django è in grado di astrarre le differenze tra questi usando il proprio *Object-Relational Mapper (ORM)*, vi possono essere potenziali problemi di compatibilità per cui è bene adottare soluzioni che siano coerenti in fase di sviluppo, produzione e rilascio.

In ogni caso, Django è configurato in modo predefinito per usare **SQLite**: i dati vengono memorizzati in un file al fine di avere un database leggero e privo della necessità di impostare ulteriori configurazioni iniziali.

2.3 Download ed installazione su un sistema Linux

Per usare correttamente Django, è necessario che Python 3 e *Python Package Index (pip3)* siano presenti all'interno del sistema operativo. Nei sistemi Linux spesso Python è già presente nel setup di base. È comunque possibile verificare se c'è e che versioni sono presenti. Per farlo, digitiamo da shell quanto segue:

```
1 python3 --version
```

Nel caso in cui il comando fallisca, Python non è installato all'interno del proprio sistema ed in base alla propria macchina, vi sono diverse opzioni di download che sono reperibili all'interno del sito ufficiale di Python [13].

2.3.1 Creazione di un ambiente virtuale

Grazie a **Venv** è possibile creare facilmente un ambiente Python virtuale, all'interno del quale si possono creare progetti Django. Con il seguente comando si procede alla creazione dell'ambiente:

```
1 python -m venv project-name
```

Per attivare manualmente l'ambiente virtuale appena creato bisogna lanciare lo script **activate**, mediante il seguente comando:

```
1 source project-name-venv/bin/activate
```

Mentre per uscire dall'ambiente virtuale, lanciare il seguente comando:

```
1 deactivate
```

2.3.2 Installazione di Django e primo avvio

Django può essere adesso installato facilmente all'interno dell'ambiente virtuale, attraverso **Python Package Index** (pip):

```
1 python -m pip install Django
```

I comandi seguenti sono utili per testare l'installazione di Django creando una prima applicazione Web:

```
1 django-admin startproject myproject
2 cd myproject
3 manage.py startapp myapplication
4 manage.py runserver
```

2.4 Sviluppo Django con PyCharm

PyCharm è un ambiente di sviluppo integrato (IDE) sviluppato da *JetBrains* che fornisce supporto per framework Web come Django, un debugger integrato, ed è compatibile con *Git*.

Pycharm è disponibile all'interno del suo sito ufficiale [12] in due versioni compatibili per Windows, MAC e Linux:

- **PyCharm Community Edition:** versione gratuita dell'IDE, open-source ed indicata per il puro sviluppo di codice Python.
- **PyCharm Professional Edition:** versione a pagamento dell'IDE, indicata per lo sviluppo Web, ma anche scientifico con supporto di HTML, JS ed SQL.

Nei Capitoli seguenti sono mostrati procedimenti e caratteristiche proprie di quest'ultima versione.

2.4.1 Creazione di un progetto

La creazione di un progetto Django all'interno di tale IDE è molto semplice, dato che il framework Web è interamente supportato:

1. Dal menu principale, selezionare **File | New project**, oppure selezionare il pulsante **New Project** nella schermata di benvenuto. Si aprirà una finestra per la creazione di un nuovo progetto, come mostrato in Figura 2.3.

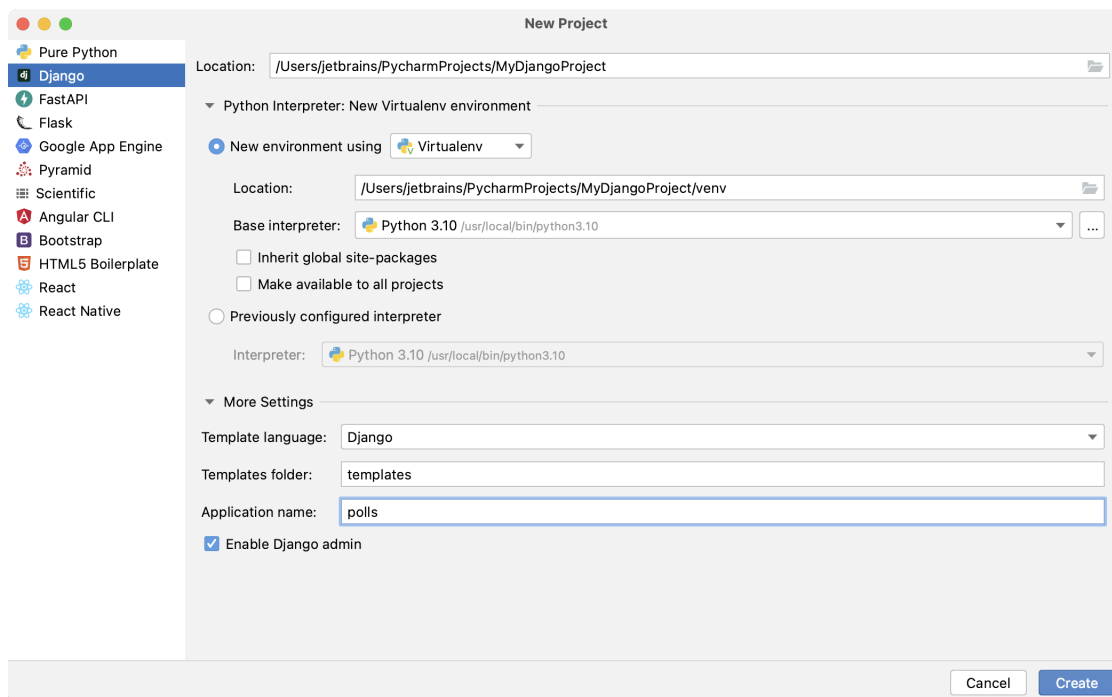


Figura 2.3: PyCharm, creazione di un nuovo progetto Django

2. All'interno di tale finestra di dialogo seguire le seguenti istruzioni:

- Specificare Django come tipo di progetto.
- Selezionando la prima voce: **Project Interpreter**, vengono mostrate le impostazioni di selezione di un nuovo ambiente oppure di un interprete esistente. È quindi possibile scegliere in base alle proprie preferenze quale interprete Python utilizzare.
- Se Django non è presente all'interno dell'interprete selezionato, *PyCharm* provvederà alla relativa installazione.
- Selezionando la seconda voce: **More settings**, vengono mostrate le impostazioni aggiuntive per specificare la cartella che conterrà i template Django, il nome della prima applicazione Web e l'opzione di creazione automatica dell'interfaccia di amministrazione.

3. Selezionare **Create**.

Una volta creato il progetto Django, *PyCharm* fornisce un'interfaccia completa con anche un terminale da cui è possibile lanciare gli stessi comandi mostrati nei Paragrafi precedenti.

2.4.2 PyCharm Professional Edition

Nei Capitoli successivi, verranno mostrate l'installazione e l'implementazione dell'applicativo per la somministrazioni di sondaggi clinici sviluppata mediante *PyCharm Professional Edition*, di cui è riportata l'interfaccia nella *Figura 2.4*.

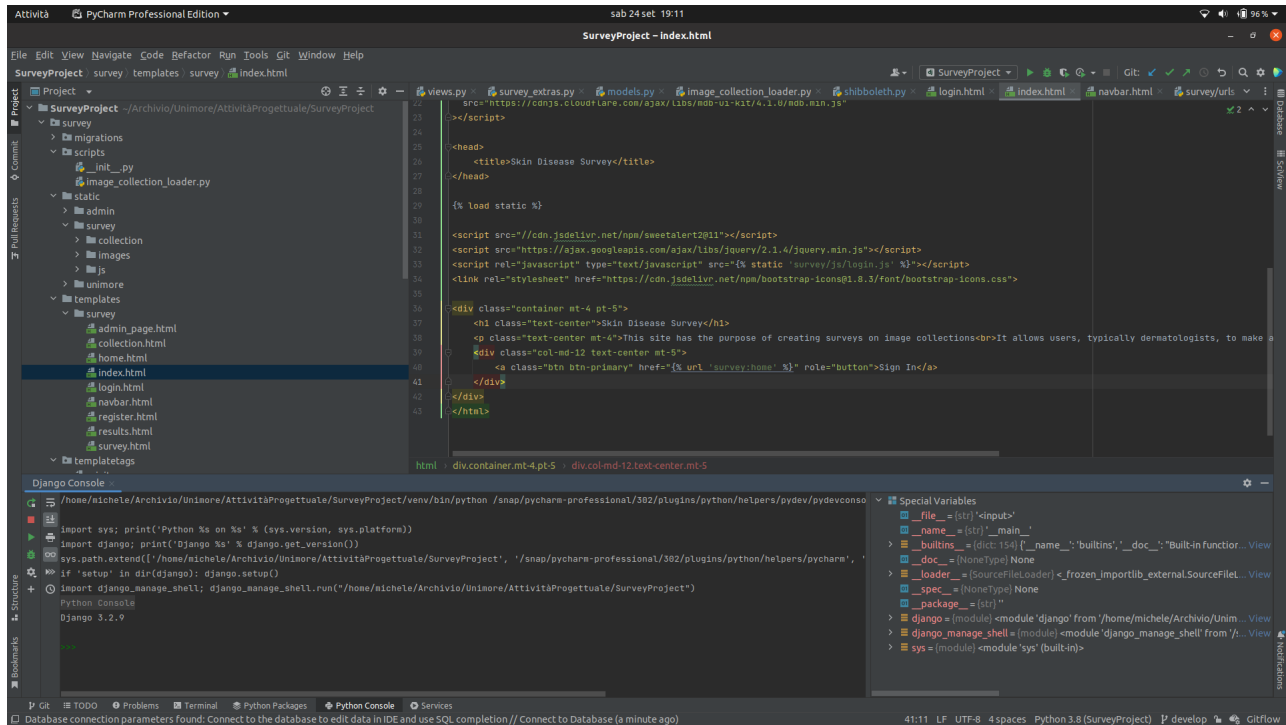


Figura 2.4: PyCharm Professional Edition, interfaccia

3. Javascript

Javascript è un linguaggio di programmazione utilizzato dagli sviluppatori per creare pagine Web interattive. Dall'aggiornamento dei feed dei social media fino alla visualizzazione di animazioni e mappe interattive, le funzioni di Javascript possono migliorare l'esperienza utente di un sito Web. Essendo un linguaggio di scripting lato client, è una delle tecnologie principali del World Wide Web. Ad esempio, durante la navigazione su Internet, quando appaiono una serie di immagini, un menù a discesa click-to-show, o quando i colori degli elementi che cambiano in modo dinamico su un sito Web, si tratta di effetti di Javascript.

3.1 Per cosa viene utilizzato Javascript?

In passato, le pagine Web erano statiche, simili alle pagine di un libro. Una pagina statica mostrava principalmente informazioni in un layout fisso e non funzionava come oggi ci aspettiamo da un sito Web moderno. Javascript è nato come tecnologia lato browser per rendere più dinamiche le applicazioni Web. Utilizzando Javascript, i browser potevano rispondere alle interazioni degli utenti e modificare il layout dei contenuti della pagina Web.

Con la maturazione del linguaggio, gli sviluppatori di Javascript hanno creato librerie, framework e pratiche di programmazione e hanno iniziato a usarlo al di fuori dei browser Web. Oggi è possibile utilizzare Javascript sia per lo sviluppo lato client che lato server. Nella sottosezione seguente vengono illustrati alcuni casi d'uso comuni:

3.1.1 Aggiornamento dinamico dei contenuti

È possibile utilizzare le funzioni Javascript per creare funzioni interattive che consentano agli utenti del sito Web di aggiornare dinamicamente i contenuti. Ecco alcuni esempi di queste caratteristiche interattive:

- Mostrare o nascondere informazioni facendo clic su un pulsante

- Convalidare la digitazione dei dati inseriti dagli utenti in un modulo, come i numeri di telefono e gli indirizzi e-mail.
- Modificare il colore di un pulsante della pagina Web al passaggio del mouse
- Generare annunci pubblicitari pop-up
- Ingrandire o ridurre un'immagine
- Riprodurre audio e video su una pagina Web
- Riproduzione di animazioni

3.2 Javascript lato client

Con "*Javascript lato client*" ci si riferisce al modo in cui Javascript funziona nel browser. In questo caso, il motore Javascript si trova all'interno del codice del browser. Tutti i principali browser Web sono dotati di motori Javascript integrati.

Gli sviluppatori di applicazioni Web scrivono codice Javascript con diverse funzioni associate a vari eventi, come il clic o il passaggio del mouse. Queste funzioni apportano modifiche all'HTML e al CSS.

Ecco una panoramica di come funziona Javascript lato client:

1. Il browser carica una pagina Web quando la si visita.
2. Durante il caricamento, il browser converte la pagina e tutti i suoi elementi, come pulsanti, etichette e caselle a discesa, in una struttura di dati chiamata *Document Object Model (DOM)*.
3. Il motore Javascript del browser converte il codice Javascript in bytecode. Questo codice è un intermediario tra la sintassi di Javascript e la macchina.
4. Eventi diversi, come il clic del mouse su un pulsante, attivano l'esecuzione del blocco di codice Javascript associato. Il motore interpreta quindi il bytecode e apporta modifiche al DOM.
5. Il browser mostra il nuovo DOM.

3.3 Ajax

AJAX, abbreviazione di *Asynchronous Javascript and XML*, indica una combinazione di tecnologie di sviluppo usate per creare pagine web dinamiche. La definizione AJAX indica le pagine web che utilizzano le tecnologie XHTML, CSS, DOM, XML e XSLT. Nello specifico, la maggior parte delle implementazioni AJAX usa l'API **XMLHttpRequest**, che include un elenco di richieste del server che possono essere richiamate all'interno del codice Javascript. I dati vengono solitamente inviati al browser in formato XML, formato facilmente leggibile e manipolabile. Ciò che caratterizza AJAX è la leggera sfasatura temporale tra la richiesta inviata dallo script al server web e la ricezione dei dati: avvenendo in momenti leggermente diversi, sono considerati asincroni. Inoltre gli script possono essere eseguiti lato client, consentendo di visualizzare immediatamente sulla pagina i dati ricevuti dal server, senza necessità di ricaricare la pagina stessa per visualizzarne i contenuti.

3.3.1 Implementazione di Ajax

Per poter implementare l'utilizzo di **Ajax** nel proprio codice *Javascript*, inseriamo nella pagina *.html*, in cui si vuole utilizzare tale tecnica, il seguente codice:

```
1 <script
2   type="text / Javascript"
3   src="https://cdnjs.cloudflare.com/ajax/libs/mdb-ui-kit/4.1.0/mdb.min.js"
4 >/script>
```

Codice 3.1: Codice di inclusione della tecnologia Ajax mediante CDN

In tal modo, vengono inclusi da un *Content Delivery Network (CDN)* gli script e le librerie per poter utilizzare **Ajax** nei propri script *Javascript*.

Un esempio di creazione di uno script che sfrutta la tecnica **Ajax** è possibile trovarlo al *Capitolo* 6.3.5

4. Single Sign-On

Il **Single Sign-On** facilita l'utilizzo delle risorse di rete in quanto consente ad un utente di accedere a più applicazioni e domini aziendali utilizzando un solo set di credenziali *username* e *password*. Con il *Single Sign-On* l'utente effettua il login **una sola volta** e ottiene l'accesso a diverse applicazioni senza la necessità di immettere nuovamente le credenziali di accesso in ogni applicazione.

4.1 Come funziona il Single Sign-On?

Fondamentalmente, l'autenticazione con **SSO** si basa su una relazione di fiducia tra domini (siti Web). Con l'accesso *Single Sign-On*, questo è ciò che accade quando si tenta di accedere ad un'applicazione oppure ad un sito Web:

1. il sito Web verifica innanzitutto se l'utente è già stato autenticato dalla soluzione *SSO*, caso in cui consente di accedere al sito.
2. in caso contrario, ci viene inviata la soluzione *SSO* per l'accesso.
3. immettiamo il singolo nome utente/password utilizzato per l'accesso aziendale.
4. la soluzione *SSO* richiede l'autenticazione dal provider di identità o dal sistema di autenticazione utilizzato dall'azienda. Verifichiamo la nostra identità e notificiamo la soluzione *SSO*.
5. la soluzione *SSO* passa i dati di autenticazione al sito Web e consente di tornare a tale sito.
6. dopo l'accesso, il sito passa i dati di verifica dell'autenticazione con l'utente mentre ci si sposta all'esterno del sito per verificare di essere autenticati ogni volta che si passa a una nuova pagina

Ciò che accade con il *Single Sign-On* è che i dati di verifica dell'autenticazione assumono la forma di **token**. Nelle due immagini seguenti viene esplicitata questa procedura.



Figura 4.1: Procedura di autenticazione mediante Single Sign-On

Il sito Web reindirizza l'utente al sito Web *SSO* per accedere. L'utente accede con un singolo nome utente e password. Il sito Web *SSO* verifica il provider di identità dell'utente, come ad esempio **Active Directory**.

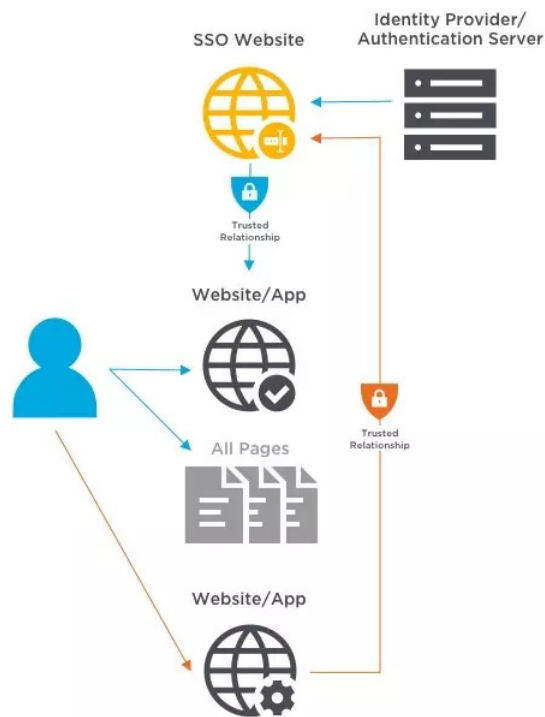


Figura 4.2: Procedura di verifica mediante Single Sign-On

Quando l'utente tenta di accedere a un sito Web diverso, il nuovo sito Web verifica con la soluzione SSO. Poiché l'utente è stato autenticato, verifica l'identità dell'utente nel nuovo sito Web senza richiedere un accesso aggiuntivo.

4.1.1 Vantaggi dell'autenticazione Single Sign-On

Alla luce di quanto detto finora, sono evidenti i vantaggi dell'autenticazione basata sul metodo SSO:

- **riduce "l'affaticamento delle password"**: ricordare una password invece di molte, semplifica la vita degli utenti. Come beneficio principale, dà un maggiore incentivo a creare password robuste.
- **semplifica la gestione di nomi e password**: quando si verificano cambiamenti del personale, SSO riduce sia lo sforzo IT che le possibilità di errore.

- **migliora la protezione dell'identità:** con *SSO* le aziende possono rafforzare la sicurezza dell'identità con tecniche quali l'autenticazione a due fattori (*2FA*) e l'autenticazione a più fattori (*MFA*).
- **aumenta la velocità ove necessario:** in ambienti come ospedali, industrie della difesa e servizi di emergenza, dove un gran numero di persone e reparti richiedono un accesso rapido e libero alle stesse applicazioni, *SSO* è particolarmente utile.
- **alleggerisce i carichi di lavoro dell'help desk:** un minor numero di utenti che richiedono aiuto con password perse consente di risparmiare denaro e migliora la sicurezza.
- **riduce i rischi per la sicurezza per i clienti, i fornitori e le entità partner:** le connessioni tra le aziende alleate presentano sempre vulnerabilità, che *SSO* può ridurre.
- **sono disponibili soluzioni SSO efficaci:** non c'è motivo per qualsiasi organizzazione di creare un proprio sistema o di sviluppare una profonda conoscenza riguardo *SSO*.

4.2 Implementazione SSO su un progetto Django nel server Unimore

Per implementare il *SSO* all'interno del proprio progetto *Django*, bisognerà innanzitutto assicurarsi che sia presente all'interno del proprio server un servizio *SSO*. Nel caso del server di Unimore è presente un Web server Apache2 e il servizio **Shibboleth**.

4.2.1 Configurazione di Apache2 per l'uso del SSO

Per prima cosa, bisognerà inserire all'interno del file di configurazione di Apache2 i comandi in cui specifichiamo di voler utilizzare l'autenticazione mediante *Shibboleth* per tutte le richieste dirette verso il nostro sito web. Per fare ciò modifichiamo il file `"/etc/apache2/sites-available/000-default-le-ssl.conf"` aggiungendo la seguente configurazione, sostituendo, dove è presente la voce, **service_x** con il nome del proprio progetto:

```
1 <Location /Shibboleth.sso>
2     AuthType None
```

```

3      Require all granted
4      SetHandler shib
5  </Location>
6  Alias /shibboleth-sp /usr/share/shibboleth/main.css
7  <Location /service_x/shiblogin>
8      AuthType shibboleth
9      ShibRequireSession on
10     require valid-user
11 </Location>
12 <Location /service_x/shibtest>
13     AuthType shibboleth
14     ShibRequireSession on
15     require valid-user
16 </Location>

```

Codice 4.1: Configurazione di Apache2 per l'uso del SSO

Effettuiamo un riavvio del servizio di Apache2 e da adesso ogni volta che avverrà una richiesta per le location `/service_x/shiblogin` e `/service_x/shibtest`, Apache2 reindirizzerà l'utente verso la pagina web del servizio *SSO Shibboleth* per l'autenticazione mediante le credenziali di ateneo. Per effettuare un riavvio del servizio Apache2, eseguire il seguente comando:

```

1  sudo service apache2 restart

```

4.2.2 Creazione delle View per SSO

Per l'acquisizione dei dati dell'utente che ha effettuato l'accesso mediante *SSO*, bisognerà creare una *View* la quale si occuperà di prelevare tali dati, controllare se l'utente è già registrato, in caso contrario inserire i suoi dati all'interno del database ed infine autenticarlo. Creiamo due *View*, una per effettuare i test sul *SSO* che chiameremo **shibboleth_test** ed una per l'effettiva autenticazione a cui daremo il nome di **shibboleth_login**. Prima di vedere nel dettaglio le due *View*, scriviamo le seguenti due funzioni **shibboleth_string** e **get_success_url**.

```

1  def shibboleth_string(field):
2      if type(field) is str:
3          return field.encode('latin1').decode()
4      else:

```

```

5         return str(field)
6
7     def get_success_url(request):
8         url = request.POST.get('next', request.GET.get('next', ''))
9         return url or resolve_url(settings.LOGIN_REDIRECT_URL)

```

Di cui la prima serve per assicurarci, nel caso in cui sono presenti dei caratteri speciali all'interno dei dati dell'utente, di codificarli correttamente prima della loro elaborazione. Mentre la seconda funzione, servirà per allegare alla richiesta il valore della variabile **next**, variabile che indica l'url a cui reindirizzare l'utente dopo la sua autenticazione (tipicamente assume l'url della pagina *home* dedicata all'utente).

Le informazioni dell'utente che ci vengono inviate dal server di autenticazione del *SSO*, sono contenute all'interno dei **metadati** della *request*.

```

1     def shibboleth_test(request):
2         meta = request.META
3
4         s = '<pre>\n'
5         for k, v in meta.items():
6             s += k + ': ' + shibboleth_string(v) + ', type: ' + escape(str(
7                 type(v))) + '\n'
8         s += '</pre>\n'
9         return HttpResponse(s)

```

Codice 4.2: View per effettuare i test con il SSO

In questa *View* di test, estraiamo le informazioni dai **metadati** per poi formattarli, mediante l'ausilio della funzione **shibboleth_string**, ed inviarli come risposta. Così facendo, al richiamo di tale *View*, otterremo una pagina che mostra come sono memorizzati i dati dell'utente all'interno dei **metadati**. La pagina, oltre a verificare che il sistema di *SSO* funzioni correttamente, è molto utile per scoprire quali sono i nomi delle chiavi per ottenere le informazioni dal dizionario dei *metadati*.

```

1     def shibboleth_login(request):
2         meta = request.META
3
4         user, created = User.objects.get_or_create(username=meta["eppn"])

```

```

5         if created:
6             user.set_unusable_password()
7
8             if user.email == '' and "mail" in meta:
9                 user.email = shibboleth_string(meta["mail"])
10            if user.first_name == '' and "givenName" in meta:
11                user.first_name = shibboleth_string(meta["givenName"]).title()
12            if user.last_name == '' and "sn" in meta:
13                user.last_name = shibboleth_string(meta["sn"]).title()
14
15            user.save()
16
17            login(request, user)
18
19            request.GET.urlencode()
20            return HttpResponseRedirect(get_success_url(request))

```

Codice 4.3: View per l'autenticazione mediante SSO

La *View* di login, si occuperà, invece, di estrarre le informazioni contenute all'interno dei *meta-dati*, controllare se l'utente è già presente all'interno del database, eventualmente aggiungerlo nel caso non sia presente, ed infine autenticarlo mediante il sistema di login di *Django*.

4.2.3 Inserimento degli Urls

Affinché le *View*, create nel Capitolo 4.2.2, vengano eseguite, bisogna creare due *URL* dedicati, che al raggiungimento da parte dell'utente richiamino le apposite *View*. Per fare ciò, aggiungiamo all'interno dello script *Python* dedicato agli *URLS* (*urls.py*), le seguenti due righe:

```

1     urlpatterns = [
2         path('shiblogin/', shibboleth.shibboleth_login, name='shiblogin'),
3         path('shibtest/', shibboleth.shibboleth_test, name='shibtest'),
4     ]

```

Codice 4.4: Aggiunta degli URLS dedicati al SSO

Così facendo, non appena arriverà una richiesta agli URL **shiblogin/** e **shibtest/**, verranno eseguite le *View* precedentemente create.

4.2.4 Aggiunta del pannello di SSO in HTML

Adesso non resta altro che inserire all'interno della propria pagina di *login*, il pannello per l'autenticazione mediante *Single SignOn*, con l'*URL* che porta all'autenticazione mediante *SSO*. Nel caso in esempio, il nome assegnato alla risoluzione del *URL* è quello di **shiblogin**.

```
1  <div class="row">
2    <div class="column col-12 text-left">
3      <h4><a href="{% url 'shiblogin' %}" id="modal-autorizza-dati">
4        Single
5        Sign-On Unimore</a></h4>
6      <div class="mb-2">
7        <b>Click</b> the <b><a href="{% url 'shiblogin' %}" id="modal-
8          autorizza-dati">Single SignOn</a></b> logo for <b>secure
9          login</b> with university e-mail.
10      </div>
11      <a href="{% url 'shiblogin' %}" id="modal-autorizza-dati"><img src
12       ="{% static 'unimore/images/unimore-sso.png' %}" alt="Login
13        with Shibboleth"></a>
14      <br>
15      <br>
16      <div class="alert alert-warning" role="alert">
17        Access to the portal is allowed only to regularly enrolled
18        students and staff of the Internship Office.
19      </div>
20    </div>
21  </div>
```

5. Installazione e configurazione del software

L'applicazione sviluppata arricchisce il sito Web relativo ai servizi offerti da parte dell'università di Modena e Reggio Emilia [15], ma è anche possibile installare l'applicazione in un qualunque altro server.

Il codice sorgente di tale applicativo, è presente nel repository dell'applicativo su GitHub [14].

L'installazione dell'applicazione Web relativa alla piattaforma dei Sondaggi necessita di alcune operazioni che verranno descritte nel dettaglio in questo capitolo.

5.1 Requisiti software

Affinché l'applicativo possa funzionare correttamente, occorre procedere come segue.

5.1.1 Clone del progetto da GitHub

Recarsi sulla pagina GitHub [14] dedicata al progetto ed effettuare il clone:

```
1 git clone https://github.com/MicheleMosca/SurveyProject.git
```

Codice 5.1: Clone del progetto da GitHub

5.1.2 Creazione del Virtual Environment e successiva attivazione

Creiamo, adesso, un Virtual Environment dedicato alla nostra applicazione, dove andremo ad installare tutti i pacchetti e i moduli che non sono presenti nelle librerie standard di Python.

```
1 cd SurveyProject/  
2 virtualenv SurveyProject-venv  
3 source SurveyProject-venv/bin/activate
```

5.1.3 Package Python

I pacchetti e i moduli necessari all'applicazione sono quelli elencati in figura 5.1.

| Package | Version | Latest version |
|--------------------|-------------|----------------|
| Django | 3.2.9 | ▲ 4.1.1 |
| Pillow | 9.1.1 | ▲ 9.2.0 |
| PyYAML | 6.0 | 6.0 |
| asgiref | 3.4.1 | ▲ 3.5.2 |
| beautifulsoup4 | 4.11.1 | 4.11.1 |
| certifi | 2022.6.15.1 | ▲ 2022.9.14 |
| charset-normalizer | 2.1.1 | 2.1.1 |
| django-extensions | 3.1.5 | ▲ 3.2.1 |
| docutils | 0.18.1 | ▲ 0.19 |
| graphviz | 0.20 | ▲ 0.20.1 |
| idna | 3.3 | ▲ 3.4 |
| pip | 21.1.2 | ▲ 22.2.2 |
| pyaml | 21.10.1 | 21.10.1 |
| pydot-ng | 2.0.0 | 2.0.0 |
| pygraphviz | 1.9 | ▲ 1.10 |
| pyparsing | 3.0.9 | 3.0.9 |
| pytz | 2021.3 | ▲ 2022.2.1 |
| requests | 2.28.1 | 2.28.1 |
| setuptools | 57.0.0 | ▲ 65.3.0 |
| soupsieve | 2.3.2.post1 | 2.3.2.post1 |
| sqlparse | 0.4.2 | 0.4.2 |
| urllib3 | 1.26.12 | 1.26.12 |
| wheel | 0.36.2 | ▲ 0.37.1 |

Figura 5.1: Package Python

Prima di installare i seguenti pacchetti è necessario installare le librerie: **graphviz**, **libgraphviz-dev**, **pkg-config** mediante il seguente comando bash:

```
1 sudo apt install graphviz libgraphviz-dev pkg-config
```

Adesso per installare velocemente tutti i pacchetti richiesti, utilizziamo il modulo Python **Pip** insieme all'argomento **-r** per specificare un file di requirements, file da dove selezionare i pacchetti da installare presente della directory del progetto.

```
1 pip install -r requirements.txt
```

5.2 Configurazione in base al server Unimore

Il *deployment* dell'applicazione è stato effettuato all'interno del server *services* [15] di Unimore, il quale ospita al suo interno un Web Server **Apache2** e il servizio di *Single SignOn* **Shibboleth**. Le operazioni effettuate in questo capitolo sono atte alla configurazione dell'applicativo in base a queste due tecnologie.

5.2.1 Modifica del file wsgi.py

Il file Python si occupa di stabilire un punto d'ingresso per web server compatibili con **WSGI** per servire il progetto. La configurazione è stata svolta grazie all'ausilio della *wiki* [16] offerta dal tutor.

Il file si trova nel seguente percorso (percorso assoluto del server utilizzato): /home/administrator/django_websites/SurveyProject/SurveyProject/wsgi.py

Modifichiamo lo script Python mediante questo comando:

```
1 vim /home/administrator/django_websites/SurveyProject/SurveyProject/wsgi.py
```

Sostituiamo il codice presente con il seguente:

```
1 import os
2 import sys
3 import site
4 from django.core.wsgi import get_wsgi_application
5
6 # Add the site-packages of the chosen virtualenv to work with
7 site.addsitedir('/home/administrator/django_websites/SurveyProject/
    SurveyProject-venv/lib/python3.8/site-packages')
8
9 # Add the app's directory to the PYTHONPATH
10 sys.path.append('/home/administrator/django_websites/SurveyProject/')
11 sys.path.append('/home/administrator/django_websites/SurveyProject/
    SurveyProject')
12
13 #to set enviroment settings for Django apps
14 os.environ['DJANGO_SETTINGS_MODULE'] = 'SurveyProject.settings'
15
16 # Activate your virtual env
17 activate_env=os.path.expanduser('/home/administrator/django_websites/
    SurveyProject/SurveyProject-venv/bin/activate_this.py')
18 exec(open(activate_env).read(), {'__file__': activate_env})
19
```



```
20 application = get_wsgi_application()
```

Codice 5.2: Configurazione file wsgi.py

5.2.2 Aggiunta della Web Application su Apache2

Inseriamo la nostra applicazione nel file di configurazione di Apache2 per poter essere indicizzata e dunque disponibile all'interno del Web server. Per fare ciò, modifichiamo il file: `"/etc/apache2/sites-available/000-default-le-ssl.conf"` Aggiungendo in coda la seguente configurazione:

```
1  #####
2  #      WEBSITE SurveyProject      #
3  #####
4  Alias /SurveyProject/survey/media/ /home/administrator/django_websites /
    SurveyProject/survey/media/
5  Alias /SurveyProject/survey/static/ /home/administrator/django_websites /
    SurveyProject/survey/static/
6  <Directory /home/administrator/django_websites/SurveyProject/survey/static
    />
7      Require all granted
8  </Directory>
9  <Directory /home/administrator/django_websites/SurveyProject/survey/media
    />
10     Require all granted
11  </Directory>
12  <Directory /home/administrator/django_websites/SurveyProject/SurveyProject
    />
13     <Files wsgi.py>
14         Require all granted
15     </Files>
16  </Directory>
17  <Location /Shibboleth.sso>
18     AuthType None
19     Require all granted
20     SetHandler shib
21  </Location>
22  Alias /shibboleth-sp /usr/share/shibboleth/main.css
```

```

23     <Location /SurveyProject/shiblogin>
24         AuthType shibboleth
25         ShibRequireSession on
26         require valid-user
27     </Location>
28     <Location /SurveyProject/shibtest>
29         AuthType shibboleth
30         ShibRequireSession on
31         require valid-user
32     </Location>
33     WSGIDaemonProcess SurveyProject python-path=/home/administrator/
        django_websites/SurveyProject:/home/administrator/django_websites/
        SurveyProject/SurveyProject-venv/lib/python3.8/site-packages
34     WSGIProcessGroup SurveyProject
35     WSGIScriptAlias /SurveyProject /home/administrator/django_websites/
        SurveyProject/SurveyProject/wsgi.py
36     #####
37     #         WEBSITE SurveyProject         #
38     #####

```

Codice 5.3: Configurazione di Apache2

5.2.3 Assegnazione dei permessi

Assegniamo i permessi di lettura, scrittura ed esecuzione alla root directory dell'applicazione:

```

1     sudo chmod -R 775 /home/administrator/django_websites/SurveyProject/

```

Codice 5.4: Assegnazione dei permessi

5.2.4 Aggiunta del dominio

Inseriamo il dominio del server che sta ospitando la *Web application*. Così facendo il framework *Django*, risponderà alle richieste provenienti da questo dominio, riportando l'utente alla pagina principale del sito.

Aggiungiamo il dominio "services.ing.unimore.it" alla variabile **ALLOWED_HOSTS** del file script *settings.py*, mediante il seguente comando:

```
1 vim /home/administrator/django_websites/SurveyProject/SurveyProject /  
    settings.py
```

Effettuiamo un riavvio del server *Apache2* per ricaricare le configurazioni rendendo disponibile il sito:

```
1 sudo service apache2 restart
```

5.3 Configurazione dell'account amministratore e del database

Il framework *Django* di default ha impostato l'utilizzo di un database **SQLite**, come approfondito nel Capitolo 2.2.3, è però possibile utilizzare un database differente in base alle proprie esigenze, modificando l'apposita variabile nel file *settings.py*.

5.3.1 Migrazione del database

Una volta scelto il database da utilizzare, eseguiamo il comando **migrate**, il quale effettuerà la traduzione dei modelli creati in tabelle, rendendo di fatto disponibile all'utilizzo il database.

```
1 python3 manage.py migrate
```

5.3.2 Creazione account amministratore

Creiamo un account amministratore, il quale si occuperà della gestione dell'intero sito web, avendo l'accesso al pannello amministrativo di Django e quello dedicato alla Web application.

```
1 python3 manage.py createsuperuser
```

5.3.3 Scrittura del file *secrets.json*

Creiamo il file **secrets.json**, il quale conterrà la *SECRET_KEY* e la password del database nel caso si decidesse di usare un database differente da **SQLite**:

```
1 {  
2     "SECRET_KEY": "django-insecure -+6qr1e+iz55$p-6#0p)6th*n%y=s-z(q8!kjc-k  
    &e&5tedk$bd",
```

```
3      "DB.PASSWORD": "PasswordDifficile"
4  }
```

5.3.4 Assegnazione permessi ad Apache2

Infine, assegniamo i permessi all'utente **www-data**, ciò permetterà ad *Apache2* di accedere ed applicare delle modifiche al database.

```
1  sudo chown www-data:www-data /home/administrator/django_websites /
    SurveyProject/db.sqlite3
```

6. Analisi del software

In questo capitolo verrà illustrata la struttura del software analizzando in dettaglio il codice per capirne meglio il suo funzionamento.

6.1 Struttura del progetto

Il progetto è formato da una directory radice, la quale, tra i numerosi file che ha al suo interno, contiene il database in formato *sqlite3*, il file *"requirements.txt"* per installare le librerie Python necessarie per l'applicato e il file *"secrets.json"* contenente le password e i *token* per il corretto funzionamento del sito.

All'interno della directory radice troviamo inoltre una sottodirectory nominata **"survey"**, la quale contiene al suo interno tutti gli script Python, Javascript, i template HTML e le immagini che compongono il sito Web. I più rilevanti sono:

- **survey/scripts**: contiene lo script *"image_collection_loader.py"*, il quale permette di caricare nuove configurazioni per la gestione delle collezioni. Tale script viene spiegato nel dettaglio nel capitolo 6.4.
- **survey/static/survey/collection**: contiene i file di configurazione *..yaml* di esempio.
- **survey/static/survey/images**: luogo dove sono memorizzate le immagini usate per i *survey*.
- **survey/static/survey/js**: contiene tutti gli script Javascript.
- **survey/templates/survey**: contiene i file *.html* dell'applicazione.
- **survey/templatetags**: contiene lo script *"survey_extras.py"*, il quale al suo interno ha la funzione che si occupa dell'encoding delle immagini.

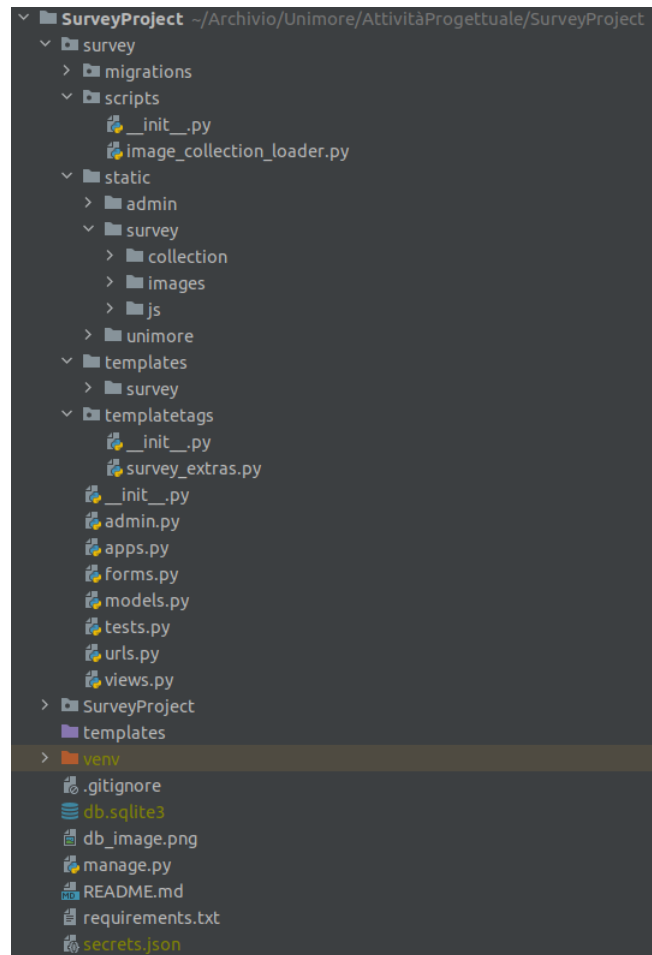


Figura 6.1: Struttura del progetto

6.2 Struttura dati del Model

Django utilizza il paradigma del *MVC*, descritto nel capitolo 2.2.1, dunque il database dell'applicativo è costruito in base alle classi che vengono descritte sul file `"survey/models.py"`. Una volta effettuata la migrazione dei modelli sul database, avremo il seguente schema:

6.2.2 Survey

Connette un **Survey_Collection** con un **auth.User**. Permette, dunque, ad un utente di interagire con le immagini che compongono il **Survey_Collection**.

```
1 class Survey(models.Model):
2     survey_collection = models.ForeignKey(Survey_Collection, on_delete=
3         models.CASCADE)
4     user = models.ForeignKey(User, on_delete=models.CASCADE)
```

Codice 6.2: Definizione del modello Survey

6.2.3 Image

Modello che identifica le immagini usate per la creazione delle collezioni. Un immagine è descritta da un **path** (che indica il percorso dell'immagine sul filesystem) e un **nome**.

```
1 class Image(models.Model):
2     path = models.CharField(max_length=200)
3     name = models.CharField(max_length=200, default='')
4
5     def __str__(self):
6         return self.path
7
8     class Meta:
9         unique_together = [['name', 'path']]
```

Codice 6.3: Definizione del modello Image

6.2.4 Image_Collection

Collega un **Image** con un **Survey_Collection**. Ciò permette di creare un set di immagini, la quale compone una **Survey_Collection**.

```
1 class Image_Collection(models.Model):
2     survey_collection = models.ForeignKey(Survey_Collection, on_delete=
3         models.CASCADE)
4     image = models.ForeignKey(Image, on_delete=models.CASCADE)
```



```

4
5     class Meta:
6         unique_together = [['image', 'survey_collection']]

```

Codice 6.4: Definizione del modello Image.Collection

6.2.5 Image_Transformation

Memorizza la lista delle trasformazioni che vengono effettivamente applicate ad una **Image.Collection** per un determinato **auth.User**.

```

1     class Image_Transformation(models.Model):
2         applied_transformation = models.CharField(max_length=200)
3         image_collection = models.ForeignKey(Image_Collection, on_delete=
4             models.CASCADE)
5         user = models.ForeignKey(User, on_delete=models.CASCADE)
6
7     class Meta:
8         unique_together = [['image_collection', 'user']]

```

Codice 6.5: Definizione del modello Image_Transformations

6.2.6 Choice

Immagazzina le possibili risposte che l'utente può selezionare.

```

1     class Choice(models.Model):
2         name = models.CharField(max_length=200)
3         survey_collection = models.ForeignKey(Survey_Collection, on_delete=
4             models.CASCADE)
5
6     def __str__(self):
7         return self.name

```

Codice 6.6: Definizione del modello Choice

6.2.7 Answer

Memorizza la risposta di un **auth.User** relativa all'**Image_Collection**, presa dall'elenco delle risposte del modello **Choice**. L'utente può inoltre scrivere un breve commento per giustificare la propria risposta.

```
1 class Answer(models.Model):
2     image_collection = models.ForeignKey(Image_Collection, on_delete=
3         models.CASCADE)
4     comment = models.CharField(max_length=200)
5     user = models.ForeignKey(User, on_delete=models.CASCADE)
6     choice = models.ForeignKey(Choice, on_delete=models.CASCADE)
7
8     class Meta:
9         unique_together = [['image_collection', 'user']]
```

Codice 6.7: Definizione del modello Answer

6.3 Struttura delle Views

Una view è un posto dove viene messa la "logica" della nostra applicazione. Essa richiederà informazioni dal **Model** creato e discusso precedentemente, analizzerà e formatterà tali dati per poi passarli ad un **Template** mediante un dizionario che chiameremo **context**.

Di seguito verranno elencate e spiegate le principali view, analizzando il regolare flusso dell'applicazione.

6.3.1 Schermata Iniziale

All'apertura del sito verrà mostrata la presente schermata la quale illustra lo scopo dell'applicativo ed attraverso il pulsante di '**Sign In**' sarà possibile accedere.

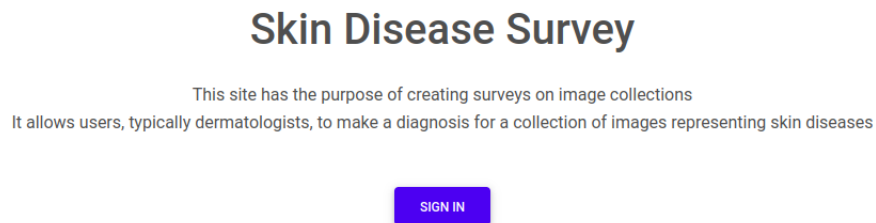


Figura 6.3: Schermata iniziale dell'applicativo

Per un utente che ha già eseguito un accesso in precedenza verrà direttamente indirizzato verso la sua *home*, mentre per tutti gli altri utenti verranno indirizzati verso la pagina di *login* dove potranno inserire le proprie credenziali o creare un nuovo account.

```
1  def indexView(request):  
2      return render(request, 'survey/index.html')
```

Codice 6.8: Visualizzazione della schermata iniziale

6.3.2 Accesso alla piattaforma

Mediante la seguente schermata l'utente può effettuare l'accesso alla piattaforma, effettuando un click sul logo del 'Single SignOn' di Unimore. Successivamente si verrà indirizzati verso la pagina di autenticazione di Unimore, dove l'utente potrà inserire le proprie credenziali di ateneo.

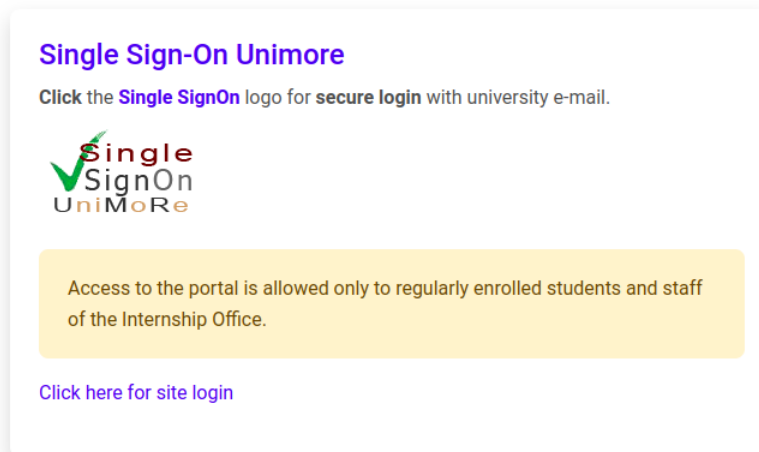


Figura 6.4: Schermata di login mediante credenziali Unimore

Utilizzando questa tipologia di accesso, se l'utente non era stato ancora registrato all'interno del database, la funzione **shibboleth_login** si occuperà di prelevare, dai *metadati* forniti da Unimore, le informazioni necessarie alla registrazione del nuovo utente. Nel caso in cui l'utente fosse già presente all'interno del database, verrà fatta l'autenticazione e si verrà indirizzati verso la propria *home*.

```
1  def shibboleth_login(request):
2      meta = request.META
3
4      user, created = User.objects.get_or_create(username=meta["eppn"])
5      if created:
6          user.set_unusable_password()
7
8      if user.email == '' and "mail" in meta:
```

```

9         user.email = shibboleth_string(meta["mail"])
10     if user.first_name == '' and "givenName" in meta:
11         user.first_name = shibboleth_string(meta["givenName"]).title()
12     if user.last_name == '' and "sn" in meta:
13         user.last_name = shibboleth_string(meta["sn"]).title()
14
15     user.save()
16
17     login(request, user)
18
19     request.GET.urlencode()
20     return HttpResponseRedirect(get_success_url(request))

```

Codice 6.9: Accesso mediante Single SignOn di Unimore

Effettuando un click nella voce *'Click here for site login'* comparirà la sezione di login dedicata agli amministratori del sito e agli utenti che non possiedono delle credenziali Unimore. Qui l'utente può inserire le proprie credenziali, le quali verranno inoltrate mediante una richiesta di tipo post alla funzione **loginView**.

Figura 6.5: Schermata di login completa con anche la sezione per amministratori

La funzione, una volta estratti i valori dei campi *username*, *password* e *remember_me* dalla richiesta, controlla se tali valori sono presenti all'interno del database ed in caso positivo reindirizza l'utente all'interno della **homeView**.

```

1  def loginView(request):
2      if request.method == 'POST':
3          username = request.POST.get('username')
4          password = request.POST.get('password')
5          remember_me = request.POST.get('remember_me')
6          next_page = request.GET.get('next')
7
8          user = authenticate(request, username=username, password=password)
9
10         if user is not None:
11             login(request, user)
12             if remember_me is None:

```

```

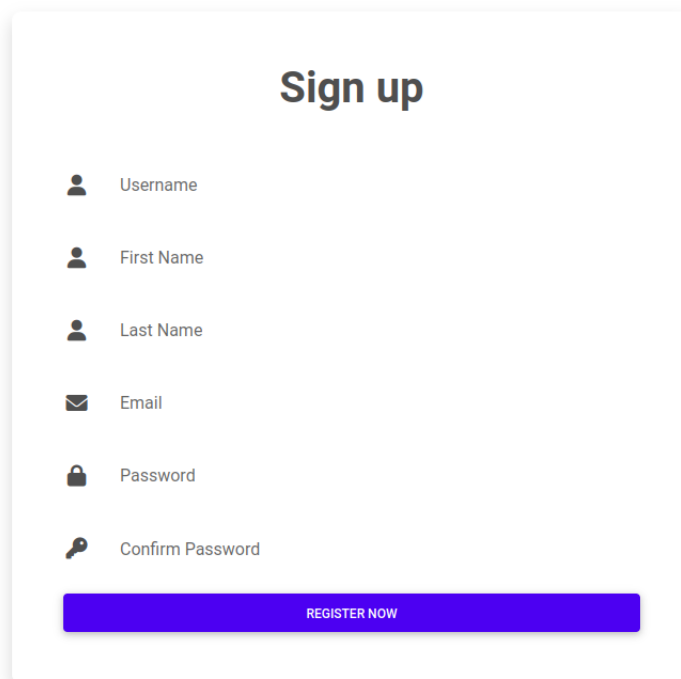
13         # if remember me is False it will close the session after
14         the browser is closed
15         request.session.set_expiry(0)
16
17         # else browser session will be as long as the session cookie
18         time "SESSION_COOKIE_AGE"
19
20     response = {
21         'msg': 'Login Success',
22         'next': next_page,
23     }
24     return JsonResponse(response)
25
26 else:
27     response = {
28         'error': 'Username or Password is incorrect'
29     }
30     return JsonResponse(response)
31
32 return render(request, 'survey/login.html')

```

Codice 6.10: Accesso mediante credenziali dell'applicativo

6.3.3 Registrazione

Un utente, attraverso questa pagina, potrà registrarsi all'interno del database degli utenti che hanno accesso all'applicativo compilando gli appositi campi richiesti.



The image shows a 'Sign up' form with a white background and a subtle shadow. At the top, the title 'Sign up' is centered in a bold, dark font. Below the title, there are six input fields, each with a small icon to its left: a person icon for 'Username', a person icon for 'First Name', a person icon for 'Last Name', an envelope icon for 'Email', a lock icon for 'Password', and a key icon for 'Confirm Password'. At the bottom of the form, there is a prominent blue button with the text 'REGISTER NOW' in white, uppercase letters.

Figura 6.6: Schermata di registrazione

Una volta che l'utente avrà completato il form, mediante un click sull'apposito pulsante "Register Now", inizierà una valutazione lato client di ciascuno dei campi per controllare che non vi siano eventuali errori. Se l'operazione è andata a buon fine, la richiesta verrà accolta dalla funzione **registerView**, la quale ha il compito di salvare all'interno del database i dati del nuovo utente.

```
1 def registerView(request):
2     if request.method == 'POST':
3         form = CreateUserForm(request.POST)
4         if form.is_valid():
5             form.save()
6
7         response = {
```



```

8         'msg': 'Account Created'
9     }
10    return JsonResponse(response)
11 else:
12    print(form.errors)
13    response = {
14        'error': f'{form.errors}',
15    }
16    return JsonResponse(response)
17
18 return render(request, 'survey/register.html')

```

Codice 6.11: Registrazione di un nuovo utente

6.3.4 Home utente

Una volta completata la fase di accesso al sito, il sistema rileverà che tipo di utente ha effettuato l'accesso. Nel caso di un utente amministratore la funzione reindirizza l'utente alla funzione *adminView*, diversamente nel caso di un utente non amministratore, la funzione *homeView* seleziona e mostra la lista delle **Collection** (gruppi di immagini su cui effettuare le diagnosi) che sono state assegnate al presente utente.

Active Surveys

| |
|--------------|
| Collection 1 |
| Collection 3 |
| Collection 4 |
| Collection 5 |
| Collection 6 |
| Collection 7 |

Figura 6.7: Schermata Home dell'utente

Nel caso di nuove **Collection** da valutare, una volta che l'utente amministratore avrà assegnato tale **Collection** al presente utente, essa comparirà in coda alla lista. Ogni **Collection** può essere selezionata anche una volta completata e le risposte possono essere visionate ed eventualmente modificate in qualunque momento. Selezionata una collezione, verrà inoltrata la richiesta alla *collectionView* la quale si occuperà di mostrare le immagini inerenti alla collezione.

```
1 @login_required(login_url='survey:login')
2 def homeView(request):
3     if request.user.is_superuser:
4         return redirect('survey:admin')
5
6     try:
7         survey_list = Survey.objects.filter(user_id=request.user.id)
8     except (KeyError, Survey.DoesNotExist):
```

```
9         survey_list = []
10
11     context = {
12         'survey_list': survey_list ,
13     }
14
15     return render(request , 'survey/home.html' , context)
```

Codice 6.12: Visualizzazione dei surveys assegnati all'utente

6.3.5 Visualizzazione della Collection

Raggiunta la *Collection View*, la funzione dopo aver controllato che l'utente ha effettivamente i permessi per valutare la presente **Collection**, preleva dal database i dati inerenti alle immagini che compongono la collezione, tra cui le possibili risposte e se l'utente ha già precedentemente risposto al quesito.

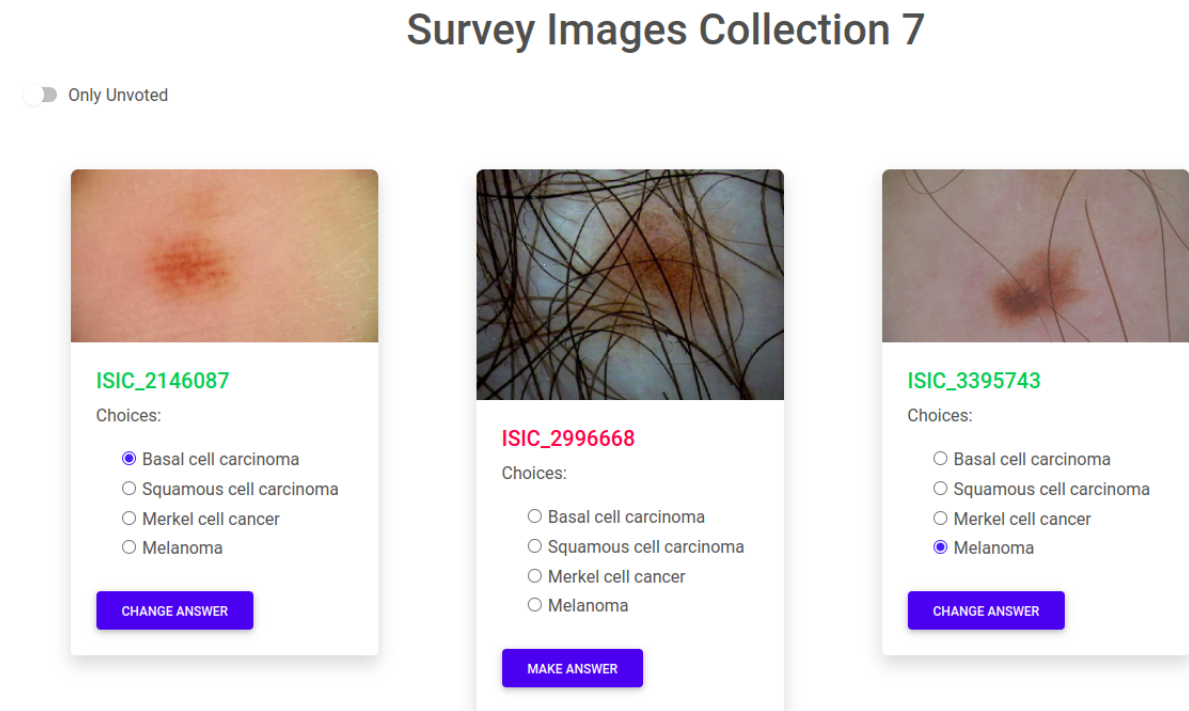


Figura 6.8: Visualizzazione della Collection

Le immagini aventi già una risposta da parte dell'utente vengono colorate in verde e hanno un pulsante *"Change Answer"* che permette di poter modificare la propria scelta, mentre le immagini che non hanno ancora ricevuto una diagnosi vengono colorate in rosso e hanno un pulsante *"Make Answer"* per inoltrare all'applicativo la risposta selezionata. È inoltre presente nella schermata uno switch per mostrare soltanto le immagini che non hanno ricevuto una diagnosi. Tale switch inoltrerà una get request alla stessa funzione *Collection View*, ponendo in *"on"* la variabile *"show_only_unvoted"*, la quale verrà valutata dalla funzione che eliminerà dalla lista delle immagini mostrate tutte quelle che hanno già ricevuto una diagnosi.

```

1  def get_images_dict(survey_images , user_answers , show_only_unvoted):
2      images_dict = dict()
3      for img in survey_images:
4          images_dict[img] = None
5          for ans in user_answers:
6              if ans.image_collection_id == img.id:
7                  images_dict[img] = ans
8
9      # Remove all voted images from the dict
10     img_list = list()
11     if show_only_unvoted:
12         for img, ans in images_dict.items():
13             if ans is not None:
14                 img_list.append(img)
15
16         for img in img_list:
17             images_dict.pop(img)
18
19     return images_dict
20
21     @login_required(login_url='survey:login')
22     def collectionView(request):
23         if not permissionOnSurvey(request):
24             return HttpResponseForbidden()
25
26         # Write changes on the db
27         if request.method == 'POST':
28             if request.POST.get('img') is not None:
29                 Answer.objects.update_or_create(
30                     image_collection_id=request.POST.get('img'),
31                     user_id=request.user.id,
32                     defaults={
33                         # 'comment': request.POST.get('comment'), Insert if
34                         # default comment is needed
35                         'choice_id': request.POST.get('answer')
36                     })

```

```

36         if request.is_ajax():
37             response = {
38                 'msg': 'Form submitted successfully!'
39             }
40             return JsonResponse(response)
41
42     user_id = request.user.id
43     survey_collection_id = request.GET.get('survey_collection_id')
44     user_answers = Answer.objects.filter(user_id=user_id)
45     survey_images = Image_Collection.objects.filter(survey_collection_id=
        survey_collection_id)
46
47     # Check if unvoted checkbox is selected
48     show_only_unvoted = False
49     if request.GET.get('show_only_unvoted') == 'on':
50         show_only_unvoted = True
51
52     # create a dictionary with image id as key and user answer as value ,
53     # if show_only_unvoted is flagged the dictionary
54     # contains only images that haven't an answer
55     images_dict = get_images_dict(survey_images , user_answers ,
        show_only_unvoted)
56
57     choices = Choice.objects.filter(survey_collection_id=
        survey_collection_id)
58
59     # create a dictionary with image_transformation_id as the key and the
60     # applied_transformations as value
61     img_transformation = {
62         Image_Transformation.objects.filter(user_id=user_id ,
63             image_collection=img).first()
64         .image_collection_id: Image_Transformation.objects.filter(user_id=
65             user_id , image_collection=img)
66         .first().applied_transformation for img in survey_images
67     }

```

```

65     context = {
66         'images_dict': images_dict ,
67         'show_only_unvoted': show_only_unvoted ,
68         'survey_collection_id': survey_collection_id ,
69         'choices': choices ,
70         'img_transformation': img_transformation ,
71     }
72     return render(request , 'survey/collection.html' , context)

```

Codice 6.13: Visualizzazione della collection

Una volta selezionata una risposta, successivamente al click del pulsante partirà l'esecuzione di uno script **Javascript**, il quale si occuperà di inoltrare la risposta alla medesima funzione mediante tecnologia **Ajax**, evitando l'aggiornamento della pagina e dunque velocizzando l'operazione di assegnazione di una risposta ai quesiti. Nel caso l'invio della risposta ha un esito positivo, verrà mostrata un'animazione per segnalare il successo dell'operazione.



Answer Submitted!

Figura 6.9: Risposta inviata con successo

```

1     $(document).ready(function () {
2         $('.post-form').on('submit', function (event) {
3
4             event.preventDefault();
5
6             const form = $(this);
7
8             const div = form.children('div').get(0);
9             div.className = "card border-success mb-3";

```

```

10
11     const h5 = form.children('div').children('div').children('h5').get
12       (0);
13
14     h5.className = "card-title text-success";
15
16
17     const input_submit = form.children('div').children('div').children
18       ('input').get(3);
19
20     input_submit.value = "Change Answer";
21
22     $.ajax({
23       type: "POST",
24       url: "",
25       data: form.serialize(),
26
27       success: function (response){
28         // handleAlerts('success', 'Answer submitted!')
29         Swal.fire({
30           position: 'top',
31           icon: 'success',
32           title: 'Answer Submitted!',
33           showConfirmButton: false,
34           timer: 1250
35         })
36       },
37
38       failure: function (error){
39         // handleAlerts('danger', 'Error!');
40         Swal.fire({
41           position: 'top',
42           icon: 'error',
43           title: 'Error!',
44           showConfirmButton: false,
45           timer: 1250
46         })
47       }
48     })

```



```
44         });  
45     })
```

Codice 6.14: Submit della risposta mediante Ajax

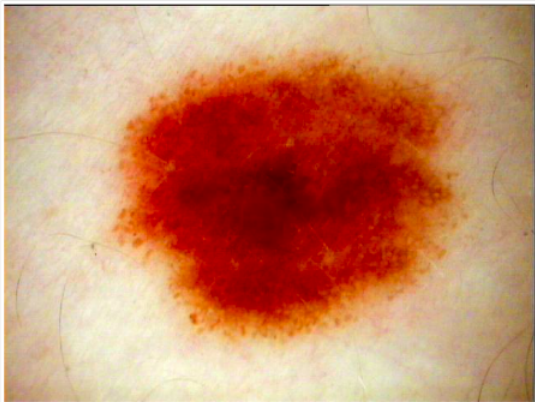
Effettuando un click su una delle immagini si verrà indirizzati nella *surveyView*, la quale ci mostrerà la pagina dedicata al quesito.

6.3.6 Pagina del quesito

In questa pagina è possibile vedere la singola immagine ad una maggiore risoluzione ed è anche possibile scrivere un piccolo commento opzionale che giustifichi la scelta effettuata.

ISIC_3898738

☐ Only Unvoted



Choose an answer:

- ☐ Basal cell carcinoma
- ☒ Squamous cell carcinoma
- ☐ Merkel cell cancer
- ☐ Melanoma

Comment (optional):

BACKSEND

Figura 6.10: Schermata del quesito

Anche qui è possibile scorrere tra i vari quesiti che compongono la collezione mediante le apposite freccette, inoltre è anche presente lo switch per indicare la preferenza di visualizzazione dei soli quesiti a cui non è stata selezionata alcuna risposta.

```
1 @login_required(login_url='survey:login')
2 def surveyView(request):
3     # check if the user is able to interact with this Survey Collection
4     if not permissionOnSurvey(request):
```

```

5         return HttpResponseRedirect()
6
7     # the user made a new answer, let's write changes on the db
8     if request.method == 'POST':
9         Answer.objects.update_or_create(
10             image_collection_id=request.POST.get('img'),
11             user_id=request.user.id,
12             defaults={
13                 'comment': request.POST.get('comment'),
14                 'choice_id': request.POST.get('answer')
15             })
16         if request.is_ajax():
17             response = {
18                 'msg': 'Form submitted successfully!'
19             }
20             return JsonResponse(response)
21
22     img_id = request.GET.get('img')
23     survey_collection_id = request.GET.get('survey_collection_id')
24     user_id = request.user.id
25     survey_images = Image_Collection.objects.filter(survey_collection_id=
26         survey_collection_id)
27     user_answers = Answer.objects.filter(user_id=user_id)
28     image_collection = Image_Collection.objects.filter(image_id=img_id,
29         survey_collection_id=survey_collection_id).first()
30
31     # Check if unvoted checkbox is selected
32     show_only_unvoted = False
33     if request.GET.get('show_only_unvoted') == 'on':
34         show_only_unvoted = True
35
36     # create a dictionary with image id as key and user answer as value ,
37     # if show_only_unvoted is flagged the dictionary
38     # contains only images that haven't an answer
39     images_dict = get_images_dict(survey_images, user_answers,
40         show_only_unvoted)

```

```

37     images_list = list(images_dict.keys())
38
39     prev_img = None
40     next_img = None
41
42     # Check if the current image is an unvoted image, otherwise use the
43     # first unvoted image
44     if image_collection not in images_list:
45         image_collection = images_list[0]
46
47     if images_list.index(image_collection) != 0:
48         prev_img = images_list[images_list.index(image_collection) - 1].
49         image_id
50
51     if images_list.index(image_collection) != len(images_list) - 1:
52         next_img = images_list[images_list.index(image_collection) + 1].
53         image_id
54
55     choices = Choice.objects.filter(survey_collection_id=
56         survey_collection_id)
57
58     selected_choice = Answer.objects.filter(image_collection_id=
59         image_collection.id, user_id=user_id).first()
60
61     comment = None
62     if selected_choice is not None:
63         comment = selected_choice.comment
64
65     # create a dictionary with image_transformation_id as the key and the
66     # applied_transformations as value
67
68     img_transformation = {
69         Image_Transformation.objects.filter(user_id=user_id,
70             image_collection=img).first()
71         .image_collection_id: Image_Transformation.objects.filter(user_id=
72             user_id, image_collection=img)
73         .first().applied_transformation for img in survey_images
74     }

```

```
65     context = {
66         'image_collection': image_collection ,
67         'choices': choices ,
68         'selected_choice': selected_choice ,
69         'comment': comment ,
70         'prev': prev_img ,
71         'next': next_img ,
72         'show_only_unvoted': show_only_unvoted ,
73         'img_transformation': img_transformation ,
74     }
75     return render(request , 'survey/survey.html' , context)
```

Codice 6.15: Visualizzazione del singolo quesito

6.3.7 Pannello di amministrazione

La funzione preleva dal database tutte le collezioni precedentemente create e dispone una lista di link nella quale è possibile consultare la pagina dedicata ai risultati di ogni singola collezione.

Administration Panel

Upload YAML configuration file to add/modify collections:

Collection Results:

| |
|--------------|
| Collection 1 |
| Collection 2 |
| Collection 3 |
| Collection 4 |
| Collection 5 |
| Collection 6 |
| Collection 7 |

Figura 6.11: Pannello di amministrazione

Nella schermata è inoltre presente un campo dove si può caricare un file di configurazione **YAML** mediante il quale si possono creare e modificare le collezioni ed inoltre è anche possibile decidere a quali utenti concedere il permesso di valutare le collezioni.

```
1  @permission_required('is_staff')
2  def adminView(request):
3      if request.method == 'POST':
4          file = request.FILES['file']
5          data = yaml.load(file, Loader=yaml.FullLoader)
6          print(data)
7          return_code = create_or_modify_collections(data)
8          if return_code != 0:
9              error = {
```

```

10         'error': errorMsg[return_code]
11     }
12     response = JsonResponse(error)
13     return response
14
15     if request.is_ajax():
16         response = {
17             'msg': 'Configuration Uploaded! '
18         }
19         return JsonResponse(response)
20
21     collection_list = Survey_Collection.objects.all()
22
23     context = {
24         'collection_list': collection_list
25     }
26
27     return render(request, 'survey/admin_page.html', context)

```

Codice 6.16: Visualizzazione del pannello di amministrazione

Il pulsante "invia" inoltrerà il *form* mediante una richiesta di tipo post alla medesima funzione usando anche questa volta la tecnologia **Ajax**, la quale chiamerà la funzione "*create_or_modify_collections*" dello script "*image_collection_loader.py*".

6.3.8 Visualizzazione dei risultati

Una volta selezionata la collection dal pannello di amministrazione, verrà mostrata una pagina dedicata ai risultati del quesito.

Nella pagina viene mostrata la lista degli utenti che hanno accesso a tale collection, la lista delle trasformazioni che possono essere applicate alle immagini delle collezione con tra parentesi la loro percentuale di probabilità di essere applicate.

Results Collection 7

User list: [prova1 prova2 prova3]

Possibly Transformations: flip(0.3),mirror(0.6),contrast(0.4)

| Image Name | Basal cell carcinoma | Squamous cell carcinoma | Merkel cell cancer | Melanoma |
|--------------|----------------------|-------------------------|--------------------|------------------|
| ISIC_2146087 | 33.3% (votes: 1) | 0.0% (votes: 0) | 66.7% (votes: 2) | 0.0% (votes: 0) |
| ISIC_2996668 | 0.0% (votes: 0) | 33.3% (votes: 1) | 33.3% (votes: 1) | 0.0% (votes: 0) |
| ISIC_3395743 | 33.3% (votes: 1) | 0.0% (votes: 0) | 0.0% (votes: 0) | 66.7% (votes: 2) |
| ISIC_1633379 | 0.0% (votes: 0) | 0.0% (votes: 0) | 66.7% (votes: 2) | 33.3% (votes: 1) |
| ISIC_3898738 | 33.3% (votes: 1) | 33.3% (votes: 1) | 0.0% (votes: 0) | 0.0% (votes: 0) |
| ISIC_0362853 | 0.0% (votes: 0) | 33.3% (votes: 1) | 0.0% (votes: 0) | 33.3% (votes: 1) |
| ISIC_8855901 | 0.0% (votes: 0) | 0.0% (votes: 0) | 66.7% (votes: 2) | 0.0% (votes: 0) |

Figura 6.12: Visualizzazione dei risultati

I risultati del quesito sono mostrati mediante una tabella avente come colonne le possibili risposte che possono essere assegnate alle immagini, mentre come righe troveremo le percentuali di selezione di ogni risposta, insieme al numero di voti che la risposta ha ottenuto.

```
1 @permission_required('is_staff')
2 def resultsView(request):
3     survey_collection_id = request.GET.get('survey_collection_id')
```



```

4      user_list = [User.objects.filter(id=query[0]).first() for query in
                    Survey.objects.filter(survey_collection_id=survey_collection_id).
                    values_list('user_id')]
5      img_collection = [img for img in Image_Collection.objects.filter(
                    survey_collection_id=survey_collection_id)]
6      choice = [choice for choice in Choice.objects.filter(
                    survey_collection_id=survey_collection_id)]
7      img_collection_to_choice_dict = {i: {c: 0 for c in choice} for i in
                    img_collection}
8
9      for user in user_list:
10         for img in img_collection:
11             ans_id = Answer.objects.filter(image_collection_id=img.id,
                    user_id=user.id)
12             if ans_id: (img_collection_to_choice_dict[img])[Choice.objects
                    .filter(id=ans_id.values_list('choice_id').first()[0]).
                    first()] += 1
13
14     users_answer = {
15         img_coll: [
16             (user, Image_Transformation.objects.filter(user_id=user.id,
                    image_collection_id=img_coll.id)
17             .first().applied_transformation, Answer.objects.filter(
                    user_id=user.id, image_collection_id=img_coll.id)
18             .first().choice if Answer.objects.filter(user_id=user.id,
                    image_collection_id=img_coll.id) else '')
19         for user in user_list
20     ] for img_coll in img_collection
21 }
22
23     context = {
24         'survey_collection_id': survey_collection_id,
25         'img_collection_to_choice_dict': {key: {k2.name: ('%.1f' % (v2 /
                    len(user_list) * 100), v2) for k2, v2 in value.items()} for key
                    , value in img_collection_to_choice_dict.items()}},
26         'user_list': user_list,

```

```

27         'choice_list': list(list(img_collection_to_choice_dict.values())
28                               [0].keys()),
29         'users_answer': users_answer,
30         'transformations': Survey_Collection.objects.filter(id=
31                       survey_collection_id).first().transformations,
32     }
33     return render(request, 'survey/results.html', context)

```

Codice 6.17: Visualizzazione dei risultati

Effettuando un click su ognuno dei risultati della tabella è possibile aprire un riquadro che mostrerà in dettaglio il voto di ogni utente, con le effettive trasformazioni che sono state applicate all'immagine per quel utente.

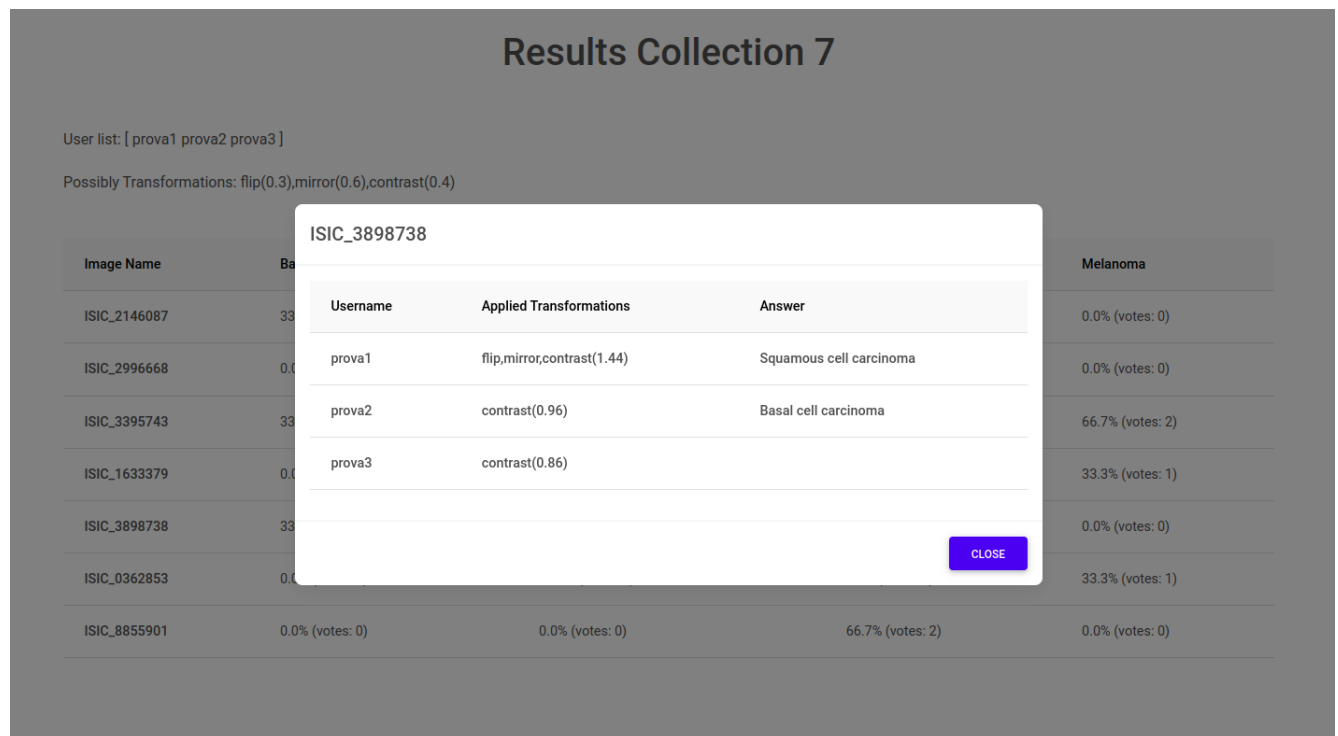


Figura 6.13: Visualizzazione dettagli dei risultati

6.4 Script di gestione delle collezioni

Per la gestione delle *Collection*, ho deciso di creare uno script in Python che dato un file in formato **YAML**, in cui è inserita la configurazione, lo analizza e dopo aver controllato la corretta scrittura della collezione esegue le operazioni sul database.

6.4.1 Scrittura del file di configurazione YAML

Per la creazione di un nuovo file di configurazione aprire un qualunque editor di testo e creare un nuovo documento prestando attenzione al fatto che, un file di configurazione accettabile dall'applicativo **dovrà** contenere all'interno del *filename* l'estensione **.yaml**. All'interno del documento per iniziare a scrivere una configurazione inserire il *tag*:

```
1 collection :
```

Dopo aver premuto il tasto "tab" della tastiera, per una corretta indentazione del codice, per descrivere una collezione avremo a disposizione sei differenti *tag*:

```
1 id : NUMBER
```

Con questo *tag* si specifica l'id della collezione, viene utilizzato **SOLO** se si vuole modificare una collezione esistente aggiungendo nuove immagini, risposte, utenti o per modificare la descrizione della collezione.

```
1 description : "DESCRIPTION"
```

Qui si può inserire una breve descrizione della *collection*.

```
1 transformations : [ 'TRANSFORMATION(PROBABILITY) ' , 'TRANSFORMATION(
    PROBABILITY) ' ]
```

Con questo *tag* si può specificare un elenco di trasformazioni che possono essere applicate alle immagini della *collection* con una percentuale di probabilità. Bisogna inserire una valida trasformazione, presente in questo elenco:

- **'flip'**: Applica una rotazione di 180° gradi all'immagine.
- **'mirror'**: Esegue un ribalto dell'immagine.

- **'contrast'**: Applica un aumento o una riduzione del contrasto in maniera casuale.

Scrivere poi tra parentesi tonde la probabilità che la trasformazione possa essere applicata per ogni utente che può interagire con la collezione.

Per la scrittura di nuove trasformazioni applicabili alle immagini bisogna inserire il codice della libreria Python: **Pillow** nello script *survey_extras.py* e se la trasformazione contiene dei particolari parametri, come nel caso del contrasto bisogna specificare se un aumento o una riduzione, andranno scritti all'interno dello script *image_collection_loader.py*.

```
1 images :
```

Questo è il *tag* per aggiungere nuove immagini alla raccolta, specificando il percorso del file ed è inoltre possibile inserire, opzionalmente, un nome per tale immagine. Se il campo nome non viene menzionato, il nome predefinito che verrà assegnato all'immagine sarà il nome del file, senza estensione. Come mostrato nel seguente esempio:

```
1 - path : "IMAGE_FILE_PATH"
2   name : "IMAGE_NAME"
```

A seguire il *tag* per la definizione delle possibili risposte che l'utente può selezionare. Bisogna specificare almeno una opzione.

```
1 choices :
2   - name : "OPTION 1"
3   - name : "OPTION 2"
```

Ed infine l'ultimo *tag*, il quale serve per aggiungere nuovi utenti alla lista degli utenti che hanno accesso alla *collection*.

```
1 users : [ 'USERNAME_USER1' , 'USERNAME_USER2' ]
```

Per completezza riporto un esempio di un file di configurazione per la creazione di una nuova collezione:

```
1 collection :
2   description : "Description of the collection"
3   transformations : [ 'flip(0.5)' , 'mirror(0.5)' , 'contrast(0.2)' ]
4
5   images :
6     - path : "survey/images/image1.jpg"
```

```

7         name: "image1"
8     -   path: "survey/images/image2.jpg"
9     -   path: "survey/images/image3.jpg"
10
11     choices:
12     -   name: "Collection1_Option1"
13     -   name: "Collection1_Option2"
14     -   name: "Collection1_Option3"
15     -   name: "Collection1_Option4"
16
17     users: [ 'user1', 'user2' ]

```

Codice 6.18: Esempio di configurazione per la creazione di una nuova collezione

A seguire un ulteriore esempio che mostra la modifica della collezione con id=1 cambiando la descrizione, aggiungendo una nuova immagine e un nuovo utente alla lista degli utenti che hanno accesso alla collezione.

```

1     collection:
2         id: 1
3         description: "Description of the collection modified"
4
5         images:
6         -   path: "survey/images/image4.jpg"
7
8         users: [ 'user3' ]

```

Codice 6.19: Esempio di configurazione per la modifica di una collezione

6.4.2 Analisi del codice

Analizziamo adesso nel dettaglio il comportamento dello script *"image_collection_loader.py"*.

```

1     def create_or_modify_collections(data):
2         collection = data.get('collection')
3         if collection is None:
4             print(f"Error: {errorMsg[3]}")
5             return 3

```

```

6
7     collection_id = collection.get('id')
8     description = collection.get('description', '')
9     choices = collection.get('choices')
10    transformations = collection.get('transformations')
11
12    if collection_id is not None:
13        print(f"Modifico la collection {collection_id}")
14        collection_object = Survey_Collection.objects.get(id=collection_id
15        )
16        print(f"Collecion id: {collection_object.id} Collection
17              description: {collection_object.description}")
18
19        if description != '':
20            collection_object.description = description
21            collection_object.save()
22            print(f"New description: {collection_object.description}")
23
24    else:
25        print("Creo una nuova collection")
26        if choices is None:
27            print(f"Error: {errorMsg[2]}")
28            return 2
29        collection_object = Survey_Collection(description=description)
30        collection_object.save()
31        print(f"Collection id: {collection_object.id}")
32
33    if transformations is not None:
34        add_transformations(transformations, collection_object)
35
36    if choices is not None:
37        add_choices(choices, collection_object)
38
39    images = collection.get('images')
40    if images is not None:
41        add_images(images, collection_object)

```

```

40
41     users = collection.get('users')
42     if users is not None:
43         add_users(users, collection_object)
44
45     return 0

```

Codice 6.20: Funzione per la creazione o modifica delle collezioni

Tale funzione si aspetta come parametro un *yaml data object* contenente la configurazione da applicare (come scrivere tale configurazione è descritto nel capitolo 6.4.1). Il programma a questo punto estrae le informazioni dall'oggetto e nel caso dovesse riscontrare degli errori nella struttura del file ritornerà un codice di errore rappresentante il motivo dell'interruzione del programma. È possibile ricavare una piccola descrizione dell'errore mediante l'accesso al seguente dizionario:

```

1     errorMsg = {
2         1: "Incorrect parameters! file .yaml is needed",
3         2: "The new collection doesn't have any choices for the answer!",
4         3: "The collection is empty!",
5         4: "Choices need a name!"
6     }

```

Codice 6.21: Dizionario dei messaggi di errore

L'ordine con la quale vengono estratte le informazioni è il seguente:

1. Si controlla se sono presenti le informazioni inerenti alla collezione, quali *id* della collezione, *descrizione*, possibili *scelte* e le *trasformazioni* che possono essere applicate alle immagini.
2. Se è presente l'id della collezione lo script passerà nella modalità di **modifica** di una collezione già esistente, dunque aggiornerà la descrizione se è presente una diversa all'interno del file di configurazione per poi procedere con i successivi step.
3. Una volta ottenuto il *collection_object*, che sia mediante ricerca in base all'id su un'operazione di modifica o dopo averne creato uno nuovo, si passerà all'inserimento delle **trasformazioni** da applicare, per poi passare all'aggiunta delle possibili **opzioni** da scegliere in fase di valutazione del quesito, al caricamento delle nuove **immagini** che compongono la collezione

ed infine ad abilitare l'accesso a tale collezione da parte degli **utenti** in base al *array* che viene scritto in configurazione.

Vediamo nel dettaglio ognuna di queste fasi:

Inserimento delle possibili trasformazioni

Per inserire la lista delle possibili trasformazioni che possono essere applicate alle immagini presenti all'interno della collezione, lo script utilizza la funzione *add_transformations*, la quale accetta come parametri: una lista di stringhe, contenenti le trasformazioni con la loro probabilità di essere applicate fra parentesi tonde, e l'oggetto *collection_object* che rappresenta la collezione alla quale applicare le modifiche.

```
1 def add_transformations(transformations , collection_object):
2     print(f"Transformations: {transformations}")
3     transformation_field = transformations[0]
4     for transformation in transformations[1:]:
5         transformation_field += ',' + transformation
6
7     collection_object.transformations = transformation_field
8     collection_object.save()
```

Codice 6.22: Inserimento delle possibili trasformazioni

La funzione, al suo interno, preleva ognuna delle stringhe presenti nella lista e formerà un'unica stringa contenente l'elenco delle trasformazioni. Tale stringa sarà formattata nel modo corretto per poi essere inserita all'interno del database.

Aggiunta delle opzioni di scelta

Per inserire delle nuove opzioni di scelta come risposte al quesito proposto, lo script fa uso della funzione *add_choices*, la quale accetta come parametri una lista di stringhe, dove in ogni stringa è scritta l'opzione di scelta, e l'oggetto *collection_object* che rappresenta la collezione alla quale applicare le modifiche.

```
1 def add_choices(choices , collection_object):
2     print(f"Choices: {choices}")
```



```

3         for choice in choices:
4             name = choice.get('name')
5             if name is None:
6                 print(f"Error: {errorMsg[4]}")
7                 return 4
8
9             choice_object = Choice.objects.get_or_create(name=choice['name'],
10                 survey_collection_id=collection_object.id)
11             print(f"Choice id: {choice_object[0].id} Choice name: {
                choice_object[0].name} "
                f"Collection id: {choice_object[0].survey_collection_id}
                added!")

```

Codice 6.23: Aggiunta delle opzioni di scelta

La funzione, al suo interno, per ogni stringa contenuta nella lista esegue il controllo che la stringa non sia vuota, dopodiché se la scelta non è già presente all'interno del database procederà con il suo inserimento.

Caricamento delle nuove immagini

L'inserimento di nuove immagini, avviene mediante l'utilizzo della funzione *add_images*, la quale accetta come parametri una lista di dizionari, i quali conterranno le informazioni inerenti alle immagini, come:

- Il **path** dov'è possibile reperire l'immagine all'interno del file system
- Il **nome** da assegnare all'immagine all'interno del database.

Oltre alla lista di dizionari, la funzione si aspetta che gli venga passato l'oggetto *collection_object* su cui applicare le modifiche.

```

1     def add_images(images, collection_object):
2         for img in images:
3             path = img['path']
4             name = img.get('name', (path.split('/')[-1]).split('.')[0])
5             transformations = collection_object.transformations
6

```

```

7         image_object = Image.objects.get_or_create(
8             path=path,
9             name=name,
10        )[0]
11        print(f"path: {image_object.path} name: {image_object.name}")
12        image_collection_object = Image_Collection.objects.get_or_create(
13            image_id=image_object.id, survey_collection_id=
14            collection_object.id)
15        image_collection_object[0].save()
16        print(f"Image_collection id: {image_collection_object[0].id} "
17              f"Image_collection image_id: {image_collection_object[0].
18              image_id} "
19              f"Image_collection survey_collection_id: {
20              image_collection_object[0].survey_collection_id}")
21
22        users_id = [user['user_id'] for user in Survey.objects.filter(
23            survey_collection_id=collection_object.id).values('user_id')]
24        for user_id in users_id:
25            image_transformation = Image_Transformation.objects.
26                update_or_create(
27                user_id=user_id, image_collection_id=
28                image_collection_object[0].id)[0]
29            apply_transformations(image_transformation, transformations,
30                                user_id)

```

Codice 6.24: Caricamento delle nuove immagini

La funzione, al suo interno, per ogni dizionario presente all'interno della lista, preleva i campi **path** e **name**, facendo attenzione in quest'ultimo campo nel controllare che il valore non sia nullo. In tal caso verrà utilizzato il nome del file senza la sua estensione, prelevandolo dal campo **path**.

Una volta prelevate le informazioni, si occuperà di controllare se l'immagine è già presente all'interno del database e in caso contrario inserirla.

Come ultima operazione, preleva tutti gli utenti che hanno accesso alla collezione e per ognuno di loro effettuerà al scelta e la memorizzazione delle trasformazioni che verranno applicate alla nuova immagine per il suddetto utente.

La scelta e la memorizzazione delle trasformazioni avviene mediante l'utilizzo della funzione *apply_transformations*, la quale accetta come parametri un oggetto *image_transformation* che rappresenta il record della trasformazione sul database, una stringa contenente la lista delle possibili trasformazioni che possono essere applicate all'immagine con le rispettive probabilità di applicazione e un oggetto **user_id** che rappresenta l'utente alla quale applicare le trasformazioni.

```
1  def decision(probability):
2      return random.random() < probability
3
4  def apply_transformations(image_transformation, transformations, user_id):
5      image_transformation.applied_transformation = ''
6      for transformation in transformations.split(','):
7          probability = float(transformation.split('(')[1].split(')')[0])
8          transformation = transformation.split('(')[0]
9          if decision(probability):
10             # Write here transformation parameter if required
11             if transformation == 'contrast':
12                 # Setting the factor parameter in a (0.5, 1.5) threshold
13                 factor = "%.2f" % random.uniform(0.5, 1.5)
14                 transformation += f'({factor})'
15
16             if image_transformation.applied_transformation == '':
17                 image_transformation.applied_transformation =
18                     transformation
19             else:
20                 image_transformation.applied_transformation += ',' +
21                     transformation
22
23     print(f"User_id: {user_id} Applied Transformations: {
24         image_transformation.applied_transformation}")
25     image_transformation.save()
```

Codice 6.25: Applicazione delle trasformazioni su un'immagine per un dato utente

Tale funzione per ogni trasformazione presente all'interno della stringa, preleva la sua corrispondente probabilità e mediante l'utilizzo della libreria **random**, prelevando un numero casuale in

un range che va da 0.00 a 1.00, se il numero ottenuto è minore del valore della probabilità allora la corrispondente trasformazione verrà applicata.

Nel caso della trasformazione **”contrast”**, verrà nuovamente utilizzata la funzione **random** per calcolare il valore di contrasto da applicare, in un range che va da 0.5 a 1.5.

Abilitazione degli utenti all’accesso alla collezione

L’aggiunta di nuovi utenti che hanno accesso alla collezione avviene mediante la funzione **add_users**, la quale si aspetta come parametri una lista di stringhe aventi gli **username** degli utenti alla quale abilitare l’accesso e l’oggetto *collection_object* alla quale applicare le modifiche.

```
1  def add_users(users , collection_object):
2      print(f"Adding new Users: {users}")
3      for user in users:
4          user_object = User.objects.filter(username=user).first()
5          obj, created = Survey.objects.get_or_create(survey_collection_id=
              collection_object.id , user_id=user_object.id)
6          if created:
7              print(f"User id: {user_object.id} Username: {user_object.
                  username} added!")
8              img_collection_ids = [img_collection['id'] for img_collection
                  in Image.Collection.objects.filter(survey_collection_id=
                      collection_object.id).values('id')]
9              for img_collection_id in img_collection_ids:
10                 image_transformation = Image_Transformation.objects.
                    update_or_create(user_id=user_object.id ,
                        image_collection_id=img_collection_id)[0]
11                 apply_transformations(image_transformation ,
                    collection_object.transformations , user_object.id)
```

Codice 6.26: Abilitazione degli utenti all’accesso alla collezione

La funzione, al suo interno, per ogni stringa presente nella lista preleva dal database le informazioni inerenti all’utente, in particolare gli interesserà lo **user id**, e successivamente crea la corrispondenza tra la collezione e l’utente mediante il model **Survey**.

Avendo appena inserito un nuovo utente alla collezione, necessiterà effettuare le scelte sulle trasformazioni da applicare per questo nuovo utente su tutte le immagini della collezione. Pertanto la funzione, per ogni immagine presente all'interno della collezione chiamerà la funzione **apply_transformation** (vista nella sezione precedente, *Capitolo 6.4.2*) per la scelta e l'applicazione delle trasformazioni.

6.5 Documentazione

La documentazione dell'intero applicativo è possibile reperirla attraverso il pannello di amministrazione di Django, in quanto è stata utilizzato il modulo di **admindocs**. Questo modulo permette di generare automaticamente la documentazione dalle *docstrings* inserite all'interno dei *Modelli*, delle *View*, dei *Template tags* e dei *Template filter*.

Per consultare la documentazione, una volta seguite le procedure elencate nel *Capitolo 5*, recarsi all'indirizzo: **admin/doc/**

Conclusioni

In conclusione, abbiamo analizzato tutte le componenti principali del software sviluppato, partendo dai richiami teorici delle tecnologie utilizzate per la sua realizzazione, per poi passare all'installazione dei componenti necessari al corretto funzionamento ed infine all'analisi dettagliata della struttura, dei componenti principali e degli script realizzati. L'applicativo web è al momento operativo e disponibile sul server services [15] dell'università, per la raccolta dei dati da parte dei dermatologi. Una volta completata la fase di raccolta dei dati, ottenendo un buon numero di risposte per la creazione del *dataset*, sarà dunque possibile sviluppare una rete neurale, o migliorare i modelli esistenti, in grado di riconoscere il tipo di malattia avendo in input un'immagine.

Sitografia

- [1] **Amazon Web Services.** *Cos'è JavaScript?*. URL:
<https://www.html.it/pag/16600/descrizione-tecnica-e-teorica/>.
- [2] **Cybersecurity360.it.** *Single Sign-On, accesso facilitato alle risorse di rete: ecco come funziona.* URL: <https://www.cybersecurity360.it/nuove-minacce/single-sign-on-accesso-facilitato-alle-risorse-di-rete-ecco-come-funziona/>.
- [3] **Django Software Foundation.** *Writing your first Django app.* URL:
<https://docs.djangoproject.com/en/3.0/intro/tutorial02/>.
- [4] **Freecodecamp.org.** *The Model View Controller Pattern.* URL:
<https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/>.
- [5] **HTML.it.** *Ajax, descrizione tecnica e teorica.* URL:
<https://www.html.it/pag/16600/descrizione-tecnica-e-teorica/>.
- [6] **HTML.it.** *Django, siti e applicazioni.* URL:
<https://www.html.it/pag/17905/siti-e-applicazioni/>.
- [7] **HTML.it.** *Il paradigma MVC in Django.* URL:
<https://www.html.it/pag/17904/il-paradigma-mvc-in-django/>.
- [8] **HTML.it.** *Perchè usare Python.* URL:
<https://www.html.it/pag/15608/perche-usare-python/>.
- [9] **HTML.it.** *Python, Introduzione e un po' di storia.* URL:
<https://www.html.it/pag/15607/introduzione-e-un-po-di-storia/>.
- [10] **JetBrains.** *Create a Django project.* URL:
<https://www.jetbrains.com/help/pycharm/creating-django-project.html>.
- [11] **JetBrains.** *Python Developers Survey 2018.* URL:
<https://www.jetbrains.com/research/python-developers-survey-2018>.
- [12] **PyCharm Official Site.** *Download PyCharm.* URL:
<https://www.jetbrains.com/pycharm/download/>.

- [13] ***Python Official Site***. *Download Python*. URL: <https://www.python.org/downloads/>.
- [14] ***Repository GitHub***. *Repository dell'applicativo*. URL:
<https://github.com/MicheleMosca/surveyProject>.
- [15] ***Server services***. *Server dei servizi offerti dall'università di Modena e Reggio Emilia*. URL:
<https://services.ing.unimore.it/SurveyProject>.
- [16] ***Services Machine***. *Wiki*. URL:
<https://github.com/AImageLab-zip/utilities/wiki/services-machine>.
- [17] ***Wiki Python***. *Web Frameworks for Python*. URL:
<https://wiki.python.org/moin/WebFrameworks>.

Ringraziamenti

Questo elaborato è il prodotto finale di un'attività progettuale svolta presso i laboratori di ricerca dell'Università. Vorrei, dunque, ringraziare il Prof. Federico Bolelli per la disponibilità e la cordialità che mi ha dedicato sia nello sviluppo dell'applicazione Web, che nella stesura dell'elaborato finale. Un ringraziamento non meno importante va anche al Prof. Costantino Grana, per i suoi preziosi insegnamenti trasmessi durante il suo corso e per aver avermi affidato il progetto.

I ringraziamenti più affettuosi vanno a tutta la mia famiglia, in particolare a mia madre, mio padre e mia sorella Arianna. Vi ringrazio per aver sempre creduto in me, dandomi la forza e il sostegno per affrontare questo duro percorso che senza di voi non sarei riuscito a concludere.

Grazie ai miei cari nonni e a tutta la mia grande famiglia per tutto il sostegno e gli incoraggiamenti che ogni giorno ci scambiamo.

Ringrazio il mio migliore amico e fratello Gianluca, per essere da sempre stato la mia guida e per tutta la forza che mi riesci a trasmettere, con te basta veramente uno sguardo per capirsi. Una menzione particolare va anche alla sua fidanzata Mirea, che con la sua dolcezza e buona cucina, riesce a migliorare anche i giorni più tristi.

Un grande ringraziamento va alla mia fidanzata Azzurra, mi sei stata accanto in tutte le fasi di questo percorso, mi hai sempre sostenuto e supportato spronandomi a dare sempre il meglio di me. Devo tanto a te e alla nostra relazione, in te ho sempre trovato tutto il conforto e l'amore di cui ho bisogno.

Un ringraziamento particolare va a tutto il team di Hacking del martedì sera, in voi ho trovato un gruppo con cui condividere non solo le mie passioni, ma anche preziosi momenti di svago. Ho imparato e imparo tanto da voi, non solo a livello informatico, per questo ve ne sono grato.

Ringrazio tutto il mio gruppo di studio, amici, colleghi e coinquilini. In particolare ringrazio: Giulio, Tano, Beppe, Erika, Berni, Angelo, Giuliana, Elme, Fabio, Fady, Alessandro, Dario e Federica. Siete stati fondamentali nel affrontare insieme questo percorso.

Infine ringrazio la Prof.ssa Donatella Giunta per avermi trasmesso, tramite le sue lezioni, la passione per l'informatica e per aver creduto in me, spingendomi a proseguire in questa appagante scienza.