Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

ExcelGen: generazione automatica della reportistica nei più comuni formati documentali

Tesi di Laurea Triennale in Informatica

Relatore

Prof. Ranzato Francesco

Laureando

Destro Stefano

Matricola 1229139

Anno Accademico 2023-2024

© Destro Stefano, Settembre 2024. Tutti i diritti riservati. Tesi di Laurea Triennale in Informatica: "*ExcelGen: generazione automatica della reportistica nei più comuni formati documentali*", Università degli Studi di Padova, Dipartimento di Matematica "Tullio Levi-Civita".

Ringraziamenti

Vorrei esprimere la mia più sincera riconoscenza ai docenti che hanno illuminato il mio percorso formativo: Cosimo Malorgio, Paola Bezze, Emilia Balielo, Annamaria Meneghesso, Valeria Bolzonaro e Cristian Cattarinussi. Il loro insegnamento ha lasciato un'impronta indelebile, guidandomi attraverso gli anni più cruciali della mia istruzione.

Un ringraziamento colmo di affetto va a mia sorella e ai miei genitori, con la loro costante presenza mi hanno sostenuto nel perseguire i miei obiettivi.

Infine, con il cuore colmo di gratitudine, rivolgo un pensiero speciale a Letizia Pastore e Thomas Franzolin. La loro instancabile presenza al mio fianco, è fonte di forza e ispirazione, rendendo ogni passo di questo cammino meno solitario e più significativo.

Padova, Settembre 2024 *Destro Stefano*

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di 315 ore, dal laureando Destro Stefano presso l'azienda RiskApp, con sede in via dell'Industria 2/F, Conselve (PD) 35026. L'obiettivo dello stage è lo sviluppo di una libreria Python per l'esportazione di dati da un *database* Django nei formati documentali `.xlsx`, `.docx` e `.pdf`. Questa libreria verrà integrata nell'applicazione Poseidon di RiskApp, azienda specializzata nello sviluppo di soluzioni *software insurtech* per la gestione del rischio in campo assicurativo.

Indice

Acronimi e abbreviazioni xi

Glossario xii

1 RiskApp 1

1.1 Profilo aziendale 1

1.2 Clientela e servizi 2

1.3 Architettura e *stack* tecnologico 2

1.4 Rapporto con gli stage 4

1.5 Conclusioni 5

2 Descrizione del progetto 7

2.1 Motivazioni e utilità 7

2.2 Scelta dell'attività di stage 7

2.3 Vincoli 9

2.3.1 Vincoli tecnologici 9

2.3.1.1 Python 3.7 9

2.3.1.2 Django 2.2.4 10

2.3.1.3 XlsxWriter 10

2.3.1.4 Python-docx 10

2.3.1.5 Django REST Framework 3.10.3 10

2.3.2 Vincoli di sviluppo e *test* 11

2.3.3 Vincoli temporali 11

2.4 Obiettivi 12

2.4.1 Obiettivi di progetto 12

2.4.2 Obiettivi di sviluppo personale 13

3 Stage in azienda **15**

3.1 Formazione sulle tecnologie 15

3.1.1 Django framework e ORM 15

3.1.2 Gestione dinamica dei QuerySet 16

3.1.3 Manipolazione avanzata di fogli di calcolo Excel 16

3.1.4 Obiettivi secondari 16

3.2 Analisi e pianificazione 17

3.2.1 Attori dei casi d'uso 17

3.2.2 Casi d'uso principali 17

3.2.3 Requisiti 17

3.2.3.1 Requisiti funzionali 18

3.2.3.2 Requisiti di vincolo 20

3.2.4 Piano di configurazione 20

3.3 Progettazione e codifica 21

3.3.1 Panoramica del progetto 21

3.3.2 Ambiente di sviluppo 22

3.3.3 Configurazione dell'ambiente di sviluppo 23

3.3.3.1 Visual Studio Code 23

3.3.4 Ambiente di *testing* 24

3.3.5 Architettura di sistema 26

3.3.6 Dizionari di configurazione 27

3.3.6.1 Elementi comuni di configurazione 27

3.3.6.2 Caratteristiche specifiche di `XlsxEngine` 28

3.3.6.3 Caratteristiche specifiche di `DocxEngine` 28

3.3.7 Struttura della libreria ExcelGen 28

3.3.7.1 Organizzazione dei moduli 28

3.3.7.2 Contenuto del modulo `common.py` 29

3.3.7.3 Contenuto del modulo `xlsx.py` 30

3.3.7.4 Contenuto del modulo `docx.py` 31

4 Collaudo e stesura della documentazione **32**

4.1 Documentazione 32

4.2 Collaudo finale 32

5 Conclusioni **34**

5.1 Raggiungimento Obiettivi 34

5.1.1 Obiettivi di progetto 34

5.1.2 Obiettivi personali 35

5.2 Consuntivo finale 36

5.3 Conoscenze acquisite 37

5.4 Valutazione personale di ExcelGen 39

5.5 Valutazione personale dell'esperienza di stage 40

A Dettagli casi d'uso **42**

B Dettagli dei dizionari di configurazione **44**

B.1 Configurazione per la classe `XlsxEngine` 44

B.2 Configurazione per la classe `DocxEngine` 48

C Dettagli sulla struttura dei moduli Python **55**

C.1 `common.py` 55

C.1.1 Classe DjangoStuff 55

C.1.2 Funzioni 56

C.1.3 Costanti 57

C.2 `xlsx.py` 57

C.2.1 `XlsxEngine` 57

C.2.2 `AuxWriter` 58

C.2.3 `SheetWriter` 59

C.3 `docx.py` 61

C.3.1 `Paragraph` 61

C.3.2 `DocxEngine` 62

Bibliografia **i**

Sitografia ii

Elenco delle figure

1.1 Logo RiskApp 1

2.1 Pianificazione delle attività di stage 11

3.1 Scenario principale 17

3.2 Contenuto della *repository* GitHub del progetto 22

3.3 Contenuto della cartella principale di ExcelGen 29

Elenco delle tabelle

2.1 Tabella degli obiettivi di progetto 13

2.2 Tabella degli obiettivi personali 14

3.1 Tabella dei requisiti funzionali 19

3.2 Tabella dei requisiti di vincolo 20

5.1 Tabella degli obiettivi di progetto raggiunti 35

5.2 Tabella degli obiettivi personali raggiunti 36

5.3 Tabella consuntivo orario 36

Elenco dei codici sorgenti

3.1 Esempio di *endpoint* per il *testing* della libreria 25

B.1 Esempio dimostrativo della configurazione 45

B.2 Esempio dimostrativo di una possibile lista accettata dalla *keyword* `fields` 47

B.3 Esempio di sezione di configurazione relativa lo stile globale del documento 52

B.4 Esempio sezione di configurazione relativa la definizione dei paragrafi 54

C.1 Esempio di come viene modificato il valore associato alla *keyword* `fields` 57

Acronimi e abbreviazioni

API Application Program Interface. 3, 16, 38

DRF Django REST Framework. 3, 10, 16, 24, 37, 38

HTTP HyperText Transfer Protocol. 42, 43

IDE Integrated Development Environment. 4, 24

ITS Issue Tracking System. 4

JSON JavaScript Object Notation. 21

MVC Model View Controller. 3

ORM Object-Relational Mapping. 15, 21

px *pixel*. 46, 50

XML eXtensible Markup Language. 16, 38, 63

Glossario

API Insieme di regole e protocolli che permettono a diverse applicazioni *software* di comunicare tra loro. Le API definiscono i metodi e i dati che possono essere richiesti e forniti, facilitando l'integrazione e l'interoperabilità tra sistemi diversi. 3

DataFrame Struttura dati bidimensionale etichettata, simile a una tabella, che può contenere dati di tipi diversi (numerici, stringhe, booleani, ecc.). Ogni colonna in un *DataFrame* è una *Serie*, e le righe e colonne sono indicizzate, permettendo operazioni efficienti di manipolazione, filtraggio e analisi dei dati. 20, 26, 30, 56

Debian Sistema operativo libero e *open-source* basato sul *kernel* Linux, noto per la sua stabilità, sicurezza e una vasta *repository* di pacchetti *software*. È utilizzato sia su *server* che su *desktop*, supportando diverse architetture *hardware*. Debian è la base di molte distribuzioni Linux. 33

endpoint URL che rappresenta un punto di accesso a un servizio *web*, permettendo la comunicazione tra *client* e *server*. Può essere utilizzato per inviare richieste e ricevere risposte tramite protocolli come HTTP. Gli *endpoint* sono fondamentali per le API e le applicazioni distribuite. 24

HTML Linguaggio *standard* per creare e strutturare contenuti su pagine *web*. Utilizza una serie di *tag* per definire elementi come testo, immagini, *link* e *form*, rendendo possibile la visualizzazione del contenuto nei *browser*. HTML è il fondamento su cui si costruiscono la maggior parte dei siti *web*. 16

io Libreria Python, fornisce strumenti per gestire flussi di *input* e *output*, permettendo di lavorare con file, stringhe e *buffer* di dati in modo astratto e flessibile, supportando operazioni di lettura e scrittura su vari tipi di *file*. 30, 31, 58, 64, 65

JSON Formato leggero di scambio dati, basato su testo, che utilizza una sintassi facilmente leggibile per rappresentare oggetti e *array*. È ampiamente utilizzato per trasmettere dati tra *server* e *client* in applicazioni *web*, grazie alla sua semplicità e compatibilità con diversi linguaggi di programmazione. 16

LaTeX Linguaggio di *markup* utilizzato per la composizione di documenti, particolarmente efficace nella gestione di testi scientifici e tecnici. Sfrutta comandi e *tag* per strutturare il contenuto, automatizzare la formattazione e gestire elementi complessi come formule matematiche, bibliografie e riferimenti incrociati. LaTeX è ampiamente utilizzato per la creazione di articoli, tesi e libri accademici. 17

LibreOffice *Suite* di *software* per ufficio *open-source* che include strumenti per l'elaborazione di testi, fogli di calcolo, presentazioni e formule matematiche. È compatibile con i formati di *file* Microsoft Office e altri *standard*, offrendo una valida alternativa gratuita per la produttività su più piattaforme. 31, 43, 65

NumPy Libreria Python per il calcolo scientifico, fornisce supporto per *array* e matrici multidimensionali, insieme a un'ampia raccolta di funzioni matematiche per operazioni su questi *array*. Grazie alla sua efficienza e velocità, è utilizzata come base per molte altre librerie scientifiche e di *machine learning*. 30, 56

OpenPyXl Libreria Python per leggere e scrivere file Excel. Offre funzionalità avanzate per manipolare fogli di calcolo, inclusa la formattazione delle celle, la gestione di formule e la creazione di grafici. OpenPyXL si distingue per la sua capacità di lavorare sia con file esistenti che di crearne di nuovi,

supportando un'ampia gamma di operazioni su dati tabulari in formato Excel. 10, 59

ORM Tecnica di programmazione che permette di interagire con un *database* relazionale utilizzando oggetti di programmazione anziché scrivere direttamente *query* SQL. L'ORM mappa le classi del linguaggio di programmazione ai modelli del *database*, consentendo di manipolare i dati del database attraverso metodi e proprietà degli oggetti, semplificando la gestione delle operazioni di *Create, Read, Update, Delete*. 3

Pandas Libreria Python per l'analisi e la manipolazione di dati, offre strutture dati come *DataFrame* e *Series*. Fornisce funzionalità avanzate per l'elaborazione e l'analisi di *dataset* strutturati. Nell'ecosistema Django, Pandas può essere utilizzata efficacemente per elaborare i dati estratti dai modelli Django, trasformando *queryset* in *DataFrame* per analisi avanzate o per preparare i dati prima di renderizzarli o esportarli. 10, 12, 20, 59, 60, 62

pip Gestore di pacchetti ufficiale di Python, usato per installare, aggiornare e rimuovere pacchetti e librerie. Gestisce le dipendenze e facilita l'integrazione di nuovi pacchetti nei progetti Python. 13, 34

playground Ambiente semplificato e isolato che emula le funzionalità di un sistema reale per consentire lo sviluppo, la sperimentazione tramite codice e l'apprendimento senza effetti collaterali su sistemi di produzione. Questi ambienti sono utilizzati per simulare scenari, verificare il comportamento di *software* e provare nuove idee in un contesto controllato e sicuro. 10–12, 37

query Richiesta di informazioni o di azioni specifiche eseguita su un *database*, utilizzando un linguaggio come SQL. Le *query* possono recuperare, inserire, aggiornare o cancellare dati, permettendo agli utenti di interagire con il *database* e manipolare i dati in esso contenuti. 15

QuerySet Rappresentazione astratta di una *query* al *database* che restituisce una collezione di oggetti del modello su cui è stato invocato. È valutato in

modo *lazy*, consentendo di costruire e filtrare dinamicamente le *query* senza eseguire nell'immediato le operazioni sul *database*. I *QuerySet* supportano operazioni come filtraggio, ordinamento e aggregazione. 12, 15, 16, 18, 20, 21, 26, 28–30, 33, 42, 43, 46, 55–57, 60, 62

SQL Linguaggio standardizzato per la gestione e l'interrogazione di *database* relazionali. Fornisce comandi per creare, modificare e eliminare strutture di *database*, nonché per inserire, aggiornare, recuperare e cancellare dati. SQL è ampiamente utilizzato in applicazioni che richiedono l'accesso e la manipolazione efficiente di grandi quantità di dati strutturati. 15

subprocess Libreria Python, consente di creare nuovi processi, collegarsi ai loro flussi di *input*/*output*/errori e ottenere i loro codici di ritorno. È utilizzata per eseguire comandi del sistema operativo e interagire con altri programmi direttamente dal codice Python, fornendo un controllo dettagliato sulla loro esecuzione e comunicazione. 64

tempfile Libreria Python, consente la creazione e gestione sicura di *file* e *directory* temporanei, utili per archiviare dati durante l'esecuzione di un programma, che vengono automaticamente eliminati alla chiusura del *file* o alla fine dell'esecuzione del programma. 64

Vim *Editor* di testo avanzato e altamente configurabile, ampiamente utilizzato per la programmazione e l'*editing* di *file* di testo. È noto per i suoi *keybindings*, che consentono un'elaborazione rapida ed efficiente attraverso combinazioni di tasti. Vim supporta diverse modalità di lavoro ed è estendibile tramite *plugin*. 23

Virtual DOM Copia leggera e *in-memory* del DOM reale, utilizzata principalmente da librerie come React per ottimizzare le operazioni di aggiornamento dell'interfaccia utente. Le modifiche vengono fatte sul Virtual DOM, e solo le differenze rispetto al DOM reale vengono applicate, migliorando così l'efficienza del *rendering*. 3

wrapper Codice che incapsula un'altra funzione, libreria o oggetto, fornendo un'interfaccia semplificata o aggiungendo funzionalità senza modificare l'elemento originale. Viene utilizzato per gestire compatibilità, estendere funzionalità o ridurre la complessità di utilizzo. 21

Capitolo 1

RiskApp

1.1 Profilo aziendale

RiskApp s.r.l (Fig. 1.1) è una *startup insurtech* con sede a Conselve, in provincia di Padova. L'azienda, fondata nel 2015 da un gruppo di studenti dell'Università Ca' Foscari di Venezia, si è rapidamente affermata nel campo della trasformazione digitale e dell'analisi avanzata del rischio per il settore assicurativo.

Figura 1.1: Logo RiskApp

In parallelo a RiskApp opera Insurance Advisor, una società di consulenza assicurativa. Sebbene le due aziende mantengano una propria identità e un proprio campo applicativo, lavorano in sinergia per offrire un servizio completo ai propri clienti. Insurance Advisor supporta privati e aziende nella scelta delle coperture assicurative più adatte alle loro specifiche necessità con un approccio personalizzato, basato su un'analisi approfondita delle diverse esigenze.

1.2 Clientela e servizi

RiskApp ha sviluppato una piattaforma digitale avanzata che si rivolge principalmente a due categorie di clienti:

- **Compagnie assicurative:** RiskApp supporta il processo di digitalizzazione, anticipando per quanto possibile le richieste dei clienti.
- **Intermediari assicurativi:** la piattaforma fornisce strumenti di analisi per offrire consulenze mirate e individuare le coperture assicurative più adeguate.

RiskApp mira a fornire strumenti che non solo migliorano l'efficienza operativa, ma consentono pure una gestione proattiva e a 360 gradi del rischio, rendendo le aziende più resilienti di fronte agli imprevisti. L'azienda integra tecnologie avanzate nelle sue soluzioni, offrendo:

- algoritmi avanzati di *machine learning* che analizzano grossi quantitativi di dati storici e attuali, identificando *pattern* che possono indicare potenziali rischi;
- un'infrastruttura *cloud* che garantisce accesso sicuro e continuo ai dati e agli strumenti di analisi, facilitando così la collaborazione tra le diverse parti coinvolte nel processo di *business*;
- un monitoraggio costantemente aggiornato delle variabili di rischio, particolarmente utile per le piccole e medie imprese che devono gestire rischi non statici e in continua evoluzione.

1.3 Architettura e *stack* tecnologico

L'architettura *software* di RiskApp, pur non essendo completamente nota nei suoi dettagli specifici, si basa su un insieme di tecnologie che supportano lo sviluppo e il *deployment* di applicazioni. L'azienda adotta un approccio flessibile, utilizzando diversi linguaggi di programmazione, *framework* e librerie in base alle necessità di ciascun componente del sistema.

Lo stack tecnologico principale include:

- ***Backend*:**
  - **Django:** *framework web* in Python che facilita lo sviluppo rapido di applicazioni. Esso fornisce un'architettura *Model View Controller (MVC)*$_G$, gestione dei database tramite *Object-Relational Mapping*$_G$, sistemi di autenticazione e un pannello di amministrazione integrato.
  - ***Django REST Framework (DRF)*$_G$:** estensione di Django per la creazione di *Application Program Interface*$_G$ *RESTful*, che include funzionalità per la gestione delle autenticazioni, la serializzazione dei dati e gli strumenti per il versionamento delle *Application Program Interface (API)*$_G$.
- ***Frontend*:**
  - **React:** libreria JavaScript per la costruzione di interfacce utente interattive e componenti riutilizzabili. Essa utilizza un modello basato su componenti e il concetto di *Virtual Document Object Model*$_G$ per ottimizzare le prestazioni di *rendering*.
- **Containerizzazione:**
  - **Docker:** una piattaforma per la creazione e gestione di *container*, che consente agli sviluppatori di automatizzare la distribuzione delle applicazioni e garantire consistenza tra ambienti di sviluppo, *testing* e produzione.

È importante sottolineare che questa descrizione è basata sulle informazioni disponibili riguardo le tecnologie utilizzate da RiskApp. L'architettura *software* specifica potrebbe includere ulteriori componenti o configurazioni non menzionate.

In riferimento alla mia esperienza in azienda, queste sono le tecnologie adottate che ho avuto modo di osservare:

- **Computer:** nello specifico portatili Apple con il loro sistema operativo proprietario.

- **Microsoft Office 365:** servizio Microsoft in abbonamento, gli strumenti prevalentemente usati sono Word, Excel e Outlook.
- **Slack:** piattaforma di telecomunicazione *online*, utile per le conversazioni interne all'azienda e lo scambio veloce di allegati.
- **GitHub:** piattaforma di *hosting* per progetti *software* basata su Git, che offre versionamento del codice, gestione collaborativa tramite *pull request*, integrazione continua e un *Issue Tracking System (ITS)*$_G$ per il monitoraggio di *bug* e richieste.
- **PyCharm Professional:** *Integrated Development Environment (IDE)*$_G$ avanzato per lo sviluppo Python, con funzionalità di *debug*, supporto per *framework web* e strumenti di *refactoring* e *testing*.
- **Jira:** una piattaforma di gestione dei progetti e *issue tracking*, ampiamente utilizzata per il monitoraggio delle attività, la gestione dei *bug* e la pianificazione agile, con strumenti di *reporting* e integrazione continua.
- **Visual Studio Code:** *editor* di codice *open-source* estensibile, che supporta numerosi linguaggi di programmazione, offre *debugging* integrato, supporto per i terminali e un ricco ecosistema di estensioni.
- **WebStorm Professional:** *IDE*$_G$ per lo sviluppo JavaScript e TypeScript, con strumenti integrati per *debugging*, *testing*, *refactoring* e supporto avanzato per *framework front-end* come React.

1.4 Rapporto con gli stage

RiskApp attribuisce alle attività di stage una funzione catalizzatrice per l'innovazione e lo sviluppo aziendale. I tirocinanti, prossimi alla laurea triennale, possono contribuire apportando nuove conoscenze nell'ambito di tecnologie emergenti. Pur rispettando i vincoli tecnologici aziendali, RiskApp incoraggia gli stagisti a introdurre soluzioni innovative nei progetti. Questa modalità permette di valutarne i potenziali benefici e considerarne l'eventuale integrazione nei prodotti esistenti.

L'approccio utilizzato dall'azienda mira a massimizzare il valore generato dagli stage. La procedura comincia con la selezione accurata di progetti che bilanciano complessità e rilevanza pratica, assicurandosi che questi rispondano a reali esigenze aziendali. Al termine del tirocinio, viene organizzata una sessione di confronto tra lo stagista e il *team* di sviluppo pertinente, facilitando uno scambio di idee sulle soluzioni implementate.

Oltre al valore tecnico, gli stage fungono da efficace strumento di *recruitment* per RiskApp. L'azienda identifica i tirocinanti più promettenti e intenzionati a entrare nel mondo del lavoro dopo la laurea triennale. Questi candidati vengono considerati per potenziali opportunità d'impiego, consentendo a RiskApp di arricchire il proprio organico con talenti già integrati nei processi e nella cultura aziendale.

Questo approccio *multifaceted* permette di investire sull'innovazione, creando un'opportunità di crescita e sviluppo aziendale.

1.5 Conclusioni

La sinergia tra RiskApp e Insurance Advisor offre una gamma completa di servizi che coprono tutte le fasi della gestione del rischio e della consulenza assicurativa:

- **Analisi del rischio su misura:** RiskApp fornisce gli strumenti tecnologici per l'analisi dei dati e la previsione dei rischi, mentre Insurance Advisor interpreta i dati e consiglia le coperture assicurative più adeguate.

- **Gestione del rischio continuativa:** particolarmente utile per le aziende che operano in settori ad alta volatilità o esposte a rischi significativi.

- **Consulenza assicurativa:** Insurance Advisor offre una consulenza mirata, supportando i clienti nella fase di scelta delle polizze, riducendo i costi ed evitando coperture ridondanti.

- **Supporto post-vendita:** assistenza nella gestione dei sinistri e nella revisione periodica delle coperture, utilizzando dati aggiornati per rinegoziare i termini delle polizze o adattarle alle nuove condizioni di rischio.

Capitolo 2

Descrizione del progetto

2.1 Motivazioni e utilità

La mia attività di stage ha origine da una necessità concreta di RiskApp, di perfezionare e uniformare il proprio sistema di generazione della reportistica. Nel suo ruolo di fornitore di soluzioni assicurative, RiskApp offre alla propria clientela la possibilità di accedere a *report* dettagliati in una varietà di formati, inclusi fogli di calcolo, documenti di testo e PDF. Questi documenti presentano una notevole diversità in termini di contenuto e struttura, riflettendo la specificità dei dati e dei servizi a cui si riferiscono.

L'approccio adottato da RiskApp per gestire questa complessità si è basato sull'utilizzo di *script* dedicati a casi specifici, talvolta anche isolati. Sebbene questa metodologia si sia dimostrata funzionale nel breve termine, ha evidenziato nel tempo significative limitazioni in termini di scalabilità ed efficienza operativa. In particolare, la necessità di apportare modifiche a cascata tra i vari *script* ha messo in luce l'esigenza di sviluppare una soluzione più flessibile e adattabile alle diverse esigenze.

2.2 Scelta dell'attività di stage

Il mio percorso verso lo stage in RiskApp è stato caratterizzato da una serie di circostanze favorevoli e scelte ponderate. A differenza di molti miei colleghi,

non ho partecipato all'evento StageIT, ma sono entrato in contatto con l'azienda attraverso un canale differente.

RiskApp, ha attirato il mio interesse per diversi motivi. In primo luogo, la dimensione contenuta dell'azienda e il suo ambiente dinamico promettevano un'esperienza di stage stimolante e ricca di opportunità di apprendimento. L'età media giovane del *team* e l'atmosfera energica che ne deriva hanno rappresentato per me un valore aggiunto significativo.

Ciò che ha reso il rapporto con RiskApp particolarmente interessante è stato il fatto che un membro interno all'azienda mi abbia contattato. Questo atteggiamento proattivo da parte sua ha influenzato positivamente la mia percezione dell'azienda e ha contribuito a farmi sentire valorizzato fin dall'inizio.

Durante i colloqui successivi, ho avuto modo di approfondire diversi aspetti cruciali:

- **Il progetto di stage:** mi è stata presentata un'opportunità che si è allineata perfettamente con i miei interessi e le mie competenze. Il progetto coinvolgeva anche l'impiego di tecnologie *backend*, un'area che mi appassiona particolarmente per la sua enfasi sulla logica e sulla risoluzione di problemi anche complessi.

- **Le tecnologie utilizzate:** la possibilità di lavorare con Django, un *framework* che già suscitava in me interesse, ha rappresentato un forte incentivo. Per questo motivo, tale opportunità mi è sembrata adatta per consolidare le mie competenze in un ambito di grande rilevanza nel mercato del lavoro attuale.

- **Il personale:** durante i colloqui, ho potuto apprezzare la cultura aziendale di RiskApp. Il *team* si è dimostrato da subito cordiale, disponibile e dotato di avanzate e affascinanti *skill* del settore.

- **L'implementazione del progetto:** le discussioni preliminari sull'approccio tecnico al progetto mi hanno permesso di intravedere delle sfide stimolanti, confermando la mia affinità con la metodologia di lavoro dell'azienda.

La decisione di accettare lo stage in RiskApp è stata ulteriormente rafforzata dalla buona reputazione di cui l'azienda gode. La mia impressione iniziale è stata confermata da una serie di *feedback* ricevuti da ex-studenti universitari.

2.3 Vincoli

2.3.1 Vincoli tecnologici

L'attività di stage si svolge all'interno di un ecosistema tecnologico ben definito, caratterizzato da una serie di specifiche e vincoli che influenzano significativamente lo sviluppo del progetto. Questi ultimi, anche se lontani dall'essere limitazioni arbitrarie, riflettono le esigenze concrete dell'ambiente aziendale di RiskApp. Di seguito viene analizzato ciascun vincolo tecnologico, esplorandone le implicazioni e le ragioni sottostanti.

2.3.1.1 Python 3.7

Questa versione specifica del linguaggio di programmazione è stata scelta per garantire la massima compatibilità con l'ecosistema *software* di RiskApp e per allinearsi con le versioni di Django e altre dipendenze utilizzate nel progetto.

L'adozione di Python 3.7 richiede un approccio strategico allo sviluppo:

1. **Gestione delle dipendenze:** bisogna assicurarsi che tutte le librerie di terze parti utilizzate nel progetto siano compatibili con Python 3.7, usando precedenti versioni di esse quando richiesto.

2. **Consapevolezza delle limitazioni:** è necessaria una certa consapevolezza delle funzionalità disponibili nella versione 3.7, evitando l'uso accidentale di sintassi o funzioni introdotte in versioni successive.

3. **Futura evoluzione:** pur lavorando all'interno dei vincoli di Python 3.7, il codice deve essere quanto più compatibile con l'ultima versione disponibile (ad oggi la 3.12.4).

2.3.1.2 Django 2.2.4

Il framework Django, nella sua versione 2.2.4, è stato adottato come spina dorsale del progetto RiskApp che implementerà la libreria da sviluppare.

Questa scelta richiede un approccio equilibrato allo sviluppo, sfruttando al massimo le potenzialità offerte da Django 2.2.4 mantenendo un occhio vigile sulle implicazioni di sicurezza dovute a possibili vulnerabilità note.

2.3.1.3 XlsxWriter

Per la generazione di file Excel (in `.xlsx`), si fa affidamento alla libreria XlsxWriter. Questa scelta è stata dettata dalla robustezza e dall'efficienza della libreria nel creare documenti Excel complessi e ricchi di funzionalità.

La scelta di XlsxWriter è stata motivata da: la sua facilità di utilizzo, l'eccellente *performance* rispetto a librerie come *OpenPyXL*$_G$ nella creazione di *file* da zero, la sua compatibilità con alcune specifiche funzioni di *Pandas*$_G$ e l'architettura complessiva del progetto.

2.3.1.4 Python-docx

Questa libreria offre una *pythonic interface* per la manipolazione di documenti Word, consentendo la creazione di *report* strutturati e formattati secondo le esigenze aziendali.

2.3.1.5 Django REST Framework 3.10.3

L'utilizzo di questo *framework* non rappresenta un vincolo in senso stretto. Si è preferito operare in un *playground*$_G$ che implementasse *DRF*$_G$ per garantire una maggior compatibilità della libreria con i sistemi di RiskApp.

In modo analogo a Django, questa versione specifica è stata selezionata per assicurare la massima compatibilità con l'ecosistema e le altre componenti *software* utilizzate dall'azienda.

2.3.2 Vincoli di sviluppo e *test*

Un vincolo cruciale del progetto è l'impossibilità di utilizzare dati propri dell'azienda durante lo sviluppo e i *test*. Questa restrizione, motivata da comprensibili esigenze di riservatezza e conformità normativa, porta inevitabilmente alla necessità di costruire un ambiente di sviluppo isolato detto *playground*$_G$.

Questo ambiente di *testing* dedicato richiede:

- la creazione di un progetto Django indipendente, separato dai sistemi di RiskApp;
- la generazione di dati di *test* realistici ma anonimi, che mimino la struttura e la complessità dei dati reali;
- l'implementazione di modelli di dati, viste e logiche di *business* simili o comunque riconducibili a quelle usate nell'ambiente di produzione.

2.3.3 Vincoli temporali

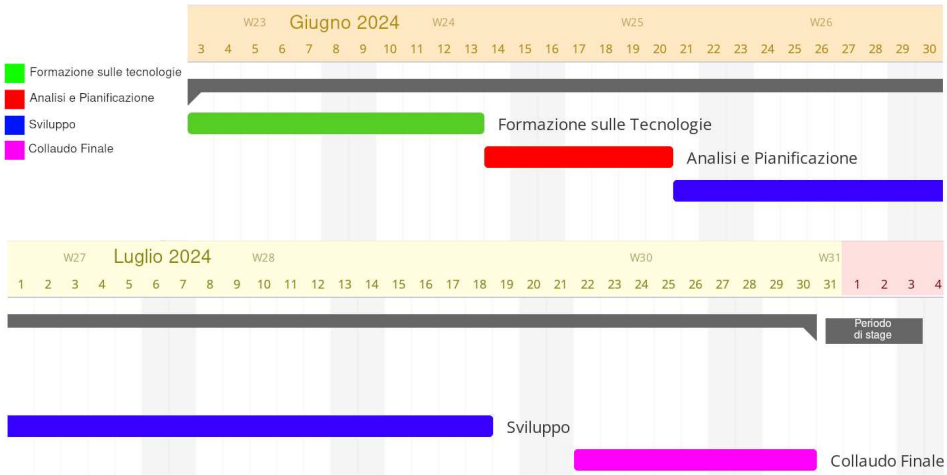

Figura 2.1: Pianificazione delle attività di stage

La pianificazione delle mie attività di stage così come riportato nel piano di lavoro (figura 2.1) è stata concordata tra me e il *tutor* aziendale e approvato

dal relatore. Il piano prevede la divisione delle attività in 4 macrosettori della durata complessiva di 315 ore.

1. **Formazione sulle tecnologie:** questa fase, della durata di 75 ore comprende lo studio approfondito del funzionamento dei modelli e delle viste Django, la creazione di *playground*$_G$ al fine di comprenderne il funzionamento, l'apprendimento della gestione dei *QuerySet*$_G$, lo studio della documentazione di Django, *Pandas*$_G$ e XlsxWriter, l'analisi della documentazione di python-docx, e la compilazione di una lista provvisoria delle funzionalità che la libreria dovrà soddisfare.

2. **Analisi e pianificazione:** questo periodo, della durata di 40 ore, prevede la definizione quanto più completa degli obiettivi del progetto, le specifiche tecniche e le strategie di implementazione. Questa attività include anche l'ideazione di un'architettura *software* adeguata alle tecnologie impiegate e agli obiettivi fissati.

3. **Sviluppo:** questa fase è stata stimata intorno alle 150 ore ed è dedicata alla creazione della libreria Python, implementando le funzionalità definite nelle fasi precedenti e integrando le varie tecnologie studiate.

4. **Collaudo finale:** l'ultima attività, della durata di 50 ore, includerà la stesura di alcuni *test* di unità, la creazione della documentazione, *test* di integrazione e l'eventuale *refactoring* del codice.

2.4 Obiettivi

Il progetto di stage si concentra sullo sviluppo di una libreria Python personalizzata per la generazione di reportistica, mirata a ottimizzare i processi interni di RiskApp.

2.4.1 Obiettivi di progetto

Gli obiettivi principali del progetto, concordati con l'azienda e riassunti in tabella 2.1, sono categorizzati come segue:

- M (*Mandatory*): Obiettivi obbligatori, essenziali per il completamento del progetto.
- D (*Desirable*): Obiettivi desiderabili, che aggiungerebbero valore significativo al progetto.
- O (*Optional*): Obiettivi opzionali, da perseguire se il tempo e le risorse lo consentono.

Id	Obiettivo
M01	Creazione di una libreria installabile tramite gestore di pacchetti *pip*$_G$ su diversi progetti.
M02	Implementazione dell'estrazione in formato `.xlsx` di dati non formattati.
M03	Implementazione dell'estrazione in formato `.xlsx` di dati formattati.
M04	Sviluppo di funzionalità per l'estrazione di dati con stile configurabile.
M05	Implementazione di regole condizionali per l'estrazione dei dati.
D01	Estrazione dei dati in formato `docx` .
O01	Estrazione dei dati in formato `pdf` .
O02	Stesura dei test di unità.
O03	Redazione della documentazione tecnica.

Tabella 2.1: Tabella degli obiettivi di progetto

2.4.2 Obiettivi di sviluppo personale

Parallelamente agli obiettivi ufficiali del progetto, ho definito una serie di obiettivi personali per la mia crescita professionale durante lo stage (illustrati in tabella 2.2).

Id	Obiettivo
M06	Approfondire la conoscenza di Python e nello specifico Django.
M07	Ridurre la complessità nella creazione dei *report* fornendo un approccio standardizzato alla generazione dei documenti.
M08	Accelerare il processo di generazione di *report*.
M09	Essere aperto all'implementazione di eventuali nuove *features* proposte da RiskApp.
M10	Migliorare le capacità personali di gestione di progetti *software* complessi.
M11	Scrivere codice efficiente e manutenibile al fine di fornire una piattaforma per future espansioni.
D02	Comprendere meglio il contesto architetturale e tecnologico dentro il quale la libreria sviluppata opererà.
D03	Collaborare con il *team* di sviluppo.

Tabella 2.2: Tabella degli obiettivi personali

Capitolo 3

Stage in azienda

3.1 Formazione sulle tecnologie

Questa fase di formazione intensiva ha fornito le basi tecniche necessarie per affrontare lo sviluppo della libreria con una più limpida comprensione delle sfide implementative. Ha permesso, inoltre, di acquisire non solo una conoscenza teorica delle singole tecnologie, ma anche una visione complessiva delle loro possibili interazioni.

3.1.1 Django framework e ORM

L'indagine iniziale si è maggiormente focalizzata sull'architettura dei *database* in Django. Il cuore pulsante di questo sistema è *Object-Relational Mapping (ORM)*$_G$, che fornisce un'astrazione ad alto livello per le operazioni sul *database*. Esso permette la manipolazione delle entità del *database* attraverso degli oggetti Python, eliminando la necessità di *query*$_G$ *Structured Query Language*$_G$ dirette.

Un'attenzione particolare è stata dedicata all'analisi dei *QuerySet*$_G$, strutture fondamentali che fungono da veri e propri costruttori di *query*$_G$, consentendo la definizione e il raffinamento delle operazioni di *data-retrieving*. La complessità intrinseca di questi concetti risulta amplificata dalla necessità di sviluppo di una libreria *data-agnostic*, quindi esige un approccio altamente flessibile e generalizzato.

Lo studio è proseguito con l'analisi delle viste Django. Esse di fatto hanno un ruolo cruciale nella costruzione di *API*$_G$. L'adozione di *DRF*$_G$ ha introdotto nuovi paradigmi. La differenza fondamentale tra le viste Django tradizionali e quelle *DRF*$_G$ risiede nella loro specializzazione: le prime sono destinate alla generazione di contenuti *HyperText Markup Language*$_G$, mentre le seconde sono progettate per la serializzazione efficiente dei dati in formati come *JavaScript Object Notation*$_G$. Le viste devono gestire le richieste degli utenti e restituire i file generati in modo adeguato e conforme agli obiettivi di sviluppo.

3.1.2 Gestione dinamica dei QuerySet

L'analisi della fase di estrazione dati ha rivelato la complessità della manipolazione di *QuerySet*$_G$ non completamente definiti a priori. I *dataset* spesso richiedono manipolazioni o arricchimenti pre-esportazione. Questa constatazione porta inevitabilmente alla necessità di architettare un sistema di configurazione altamente flessibile, in grado di orchestrare operazioni *ad hoc* sui *QuerySet*$_G$ preservando l'agnosticità della libreria rispetto alla struttura dei dati sottostanti.

3.1.3 Manipolazione avanzata di fogli di calcolo Excel

L'approfondimento della libreria XlsxWriter ha rappresentato un punto cruciale nel processo di formazione. Esso offre un controllo granulare sulla generazione e formattazione di file Excel. Lo studio ha richiesto non solo la padronanza delle funzionalità offerte da XlsxWriter, ma anche un'analisi approfondita delle capacità avanzate di Excel. La sfida principale di questa fase è stata la traduzione delle funzionalità di formattazione e calcolo di Excel in costrutti di programmazione.

3.1.4 Obiettivi secondari

Sebbene l'obiettivo primario fosse l'esportazione in formato Excel, lo studio si è esteso anche all'elaborazione di documenti Word e alla generazione di PDF. L'analisi della libreria python-docx ha evidenziato la complessità della struttura *eXtensible Markup Language (XML)*$_G$ sottostante ai documenti in formato `docx`.

Parallelamente, l'esplorazione delle tecniche di generazione PDF ha aperto nuove prospettive, sia attraverso la conversione da formati intermedi sia mediante la generazione diretta utilizzando sistemi come *LaTeX* $_G$.

3.2 Analisi e pianificazione

La fase di analisi prevede una serie di passi fondamentali per l'ideazione del sistema di reportistica, tra cui l'identificazione di casi d'uso e requisiti. In seguito vengono illustrati i principali aspetti considerati durante la pianificazione.

3.2.1 Attori dei casi d'uso

ExcelGen non prevede interazione utente, ma è il sistema a definire i casi specifici di esportazione servendosi della libreria stessa.

3.2.2 Casi d'uso principali

Per l'identificazione dei requisiti funzionali sono stati delineati i casi d'uso principali. In appendice A è possibile trovare una descrizione più approfondita dei casi d'uso individuati.

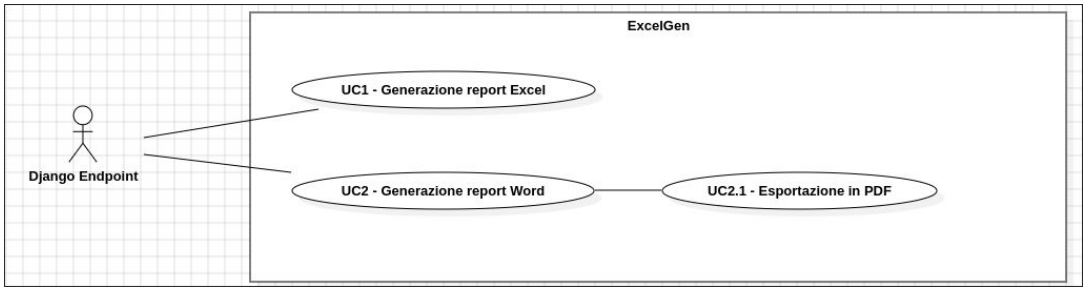

Figura 3.1: Scenario principale

3.2.3 Requisiti

Ogni requisito viene identificato da un codice che segue il seguente stile:

R[Tipo][Utilità][Id]

Tipo Indica il tipo del requisito. I valori sono:

F funzionali.

V di vincolo.

Utilità Indica l'utilità del requisito. I valori sono:

M (*mandatory*) obbligatorio.

D (*desirable*) desiderabile.

O (*optional*) opzionale.

3.2.3.1 Requisiti funzionali

Id	Descrizione
RFM1	La libreria deve disporre di un metodo per la creazione di documenti Excel dove esportare una tabella relativa i dati estratti dal *database* Django.
RFM1.1	Deve essere possibile configurare metodi di formattazione condizionale per specifiche colonne.
RFM1.2	La libreria deve poter applicare i filtri automatici (*autofilters*) alla tabella esportata.
RFD1.3	Possibilità di inserire la legenda nei *report*.
RFD1.4	Possibilità di inserire una tabellina relativa i filtri attivi sui *QuerySet*$_G$.
RFD1.5	Possibilità di includere tabelle *pivot* nei *report*.
RFD1.6	Possibilità di esportare più report per singolo documento Excel.
RFD1.7	Possibilità di stampare legenda e tabellina dei filtri attivi su un foglio dedicato.
RFD1.8	Possibilità di adattare la larghezza delle colonne al contenuto.
RFD1.9	Possibilità di mostrare più righe per cella.
RFD1.10	Possibilità di collassare gruppi di colonne.

Continua nella prossima pagina...

Tabella 3.1 – Continuo della tabella

Id	Descrizione
RFD1.11	Possibilità di nascondere singole colonne.
RFD1.12	Possibilità di definire la posizione della tabella nel foglio.
RFM2	La libreria può disporre di un metodo per la creazione di documenti Word dove esportare una tabella relativa i dati estratti dal *database* Django.
RFM2.1	La libreria deve poter offrire opzioni di formattazione.
RFD2.2	Possibilità di aggiungere paragrafi per dare contesto alla tabella dati esportata.
RFD2.3	Possibilità di limitare il numero di righe e colonne per pagina.
RFD2.4	Possibilità di definire il *layout* delle pagine.
RFO3	La libreria può disporre di un metodo per la creazione di documenti PDF dove esportare una tabella relativa i dati estratti dal *database* Django.
RFD4	Possibilità di inserire la locale in configurazione, questa risulta utile per i formati di date e ore.
RFD5	Possibilità di specificare l'ordinamento dei *record* secondo uno specifico campo.
RFD6	Possibilità di gestire ed eventualmente sostituire i valori `None` presenti fra i dati estratti.
RFM7	I casi speciali di stampa vanno gestiti, deve essere possibile definire dei metodi per la modifica dei dati prima della stampa.
RFM8	Devono essere presenti opzioni di formato che, se specificate, impongono un certo tipo di stile di formattazione ai dati (date, ore, valute monetarie, ecc...).

Tabella 3.1: Tabella dei requisiti funzionali

3.2.3.2 Requisiti di vincolo

Id	Descrizione
RVM1	Come minimo, due *input* devono essere presentati per la creazione delle tabelle dati: un *QuerySet*$_G$ per interrogare il *database* e una configurazione per guidare l'esportazione nei vari formati (vedi sezione 3.2.4).
RVM2	Per la gestione dei dati estratti ci si affida alla libreria *Pandas*$_G$. Durante questa fase, è necessario gestire relazioni di tipo *one-to-many* e *many-to-many*, garantendo che la struttura del *DataFrame*$_G$ popolato rifletta correttamente queste relazioni complesse.
RVM3	Deve essere implementata una modalità flessibile che permetta di selezionare o escludere i campi da esportare.
RVD4	Per gestire le discrepanze tra i nomi dei campi richiesti dal *front-end* e quelli presenti nel *backend*, è desiderabile implementare un dizionario *alias*-valore che permette una mappatura flessibile dei campi, facilitando così l'integrazione e la manutenzione del sistema.

Tabella 3.2: Tabella dei requisiti di vincolo

Per i vincoli tecnologici si fa riferimento alla sezione 2.3.1

3.2.4 Piano di configurazione

La progettazione del sistema di configurazione ha richiesto un'analisi approfondita dei potenziali casi d'uso e necessità implementative. Questo processo ha comportato non solo la definizione di una struttura di configurazione robusta, ma anche lo sviluppo di un meta-linguaggio di configurazione; questo deve essere sufficientemente espressivo per coprire un ampio spettro di scenari

di *reporting*, rimanendo al contempo accessibile a operatori con diversi livelli di competenza tecnica. Dopo un'attenta valutazione e confronto col *team* di sviluppo competente, ho optato per l'utilizzo di un dizionario Python come base della configurazione, preferendolo a formati più rigidi come *JavaScript Object Notation (JSON)*$_G$. Questa scelta è stata motivata dalla necessità di incorporare riferimenti a funzioni Python personalizzate per la manipolazione avanzata dei *QuerySet*$_G$.

Un requisito implicito nella progettazione di questa libreria è la generalizzazione. Data la complessità intrinseca del sistema *ORM*$_G$ di Django, la creazione di un *wrapper*$_G$ completo all'interno della libreria sarebbe un'impresa proibitiva e potenzialmente limitante. Per ovviare a questa sfida, si è deciso di implementare un meccanismo che consente la definizione esterna di funzioni specifiche per la manipolazione dei *QuerySet*$_G$. Queste funzioni possono essere referenziate direttamente nella configurazione, in modo analogo all'approccio che, a posteriori, si è scoperto essere già adottato da RiskApp per le sue soluzioni correnti.

3.3 Progettazione e codifica

3.3.1 Panoramica del progetto

La libreria si compone di diversi file (vedi fig 3.2), quali:

docs Cartella contenente due file *markdown* (`md`) relativi la guida alla stesura delle configurazioni accettate dalle classi `XlsxEngine` e `DocxEngine`.

examples Cartella contenente due progetti Django già predisposti per lavorare in armonia con la libreria.

excelgen Cartella principale del pacchetto Python.

Makefile Un *file* usato per automatizzare processi di compilazione e altre attività del progetto.

pyproject.toml Un file di configurazione per progetti Python moderni, usato per specificare dipendenze e impostazioni di *build*.

README.md *File* che contiene informazioni generali sul progetto.

requirements-dev.txt Un *file* che elenca le dipendenze di sviluppo del progetto.

requirements.text Un *file* che elenca le dipendenze principali del progetto.

setup.py Un file di *script* Python usato per la configurazione e distribuzione del pacchetto.

tests Contiene i *test* di unità del progetto.

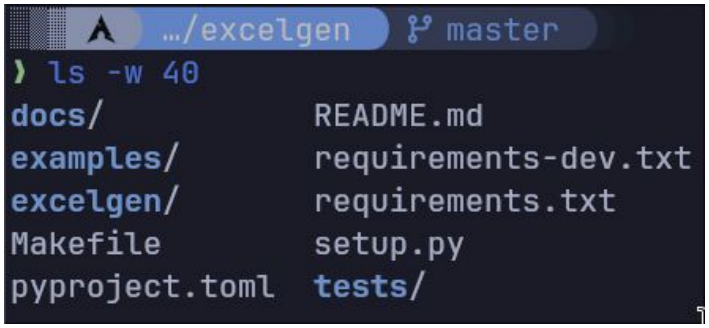

Figura 3.2: Contenuto della *repository* GitHub del progetto

3.3.2 Ambiente di sviluppo

Sistema operativo Lo sviluppo dell'applicativo è stato condotto su macOS 14 Sonoma, sistema operativo proprietario Apple.

***Software* per la codifica** Il principale strumento di sviluppo utilizzato è stato Visual Studio Code (versione 1.92), scelto per la sua flessibilità e l'ampia gamma di estensioni disponibili per il supporto allo sviluppo Python. Nelle fasi finali del progetto, è stato introdotto anche PyCharm Community Edition 2024.2, principalmente per beneficiare del suo rigoroso controllo sintattico Python, utile per una revisione finale del codice.

Containers Per replicare fedelmente l'ambiente di produzione di RiskApp e garantire la compatibilità delle versioni delle librerie, è stata impiegato l'applicativo Docker in combinazione con la funzionalità Dev Containers di Visual Studio Code.

Docker è stato utilizzato per creare *container* isolati che emulano le configurazioni e le dipendenze dei sistemi RiskApp. Questo approccio ha assicurato la coerenza tra l'ambiente di sviluppo e quello di produzione. Esso ha permesso di definire e personalizzare l'ambiente di sviluppo usando il file `devcontainer.json`, che permette di specificare l'immagine Docker, le estensioni VS Code necessarie e le impostazioni dell'*editor*.

3.3.3 Configurazione dell'ambiente di sviluppo

3.3.3.1 Visual Studio Code

Visual Studio Code (VS Code) è stato l'*editor* di codice designato per lo sviluppo dell'applicativo. In seguito viene riportata la configurazione di alcuni suoi aspetti:

Tema e interfaccia utente In base alle presenze in ufficio, la mia postazione di lavoro poteva subire variazioni, pertanto è stato necessario utilizzare due temi per adattarsi alle diverse condizioni di illuminazione:

- **Gruvbox Dark**: Utilizzato in condizioni di illuminazione ottimale. Offre un contrasto elevato e riduce l'affaticamento degli occhi.
- **Gruvbox Light**: Impiegato quando la luce solare diretta causava riflessi sullo schermo. Migliora la leggibilità del codice ed elimina fastidiosi riflessi.

Per gli indici delle righe viene adottata la modalità ibrida per consentire di sapere sempre a che riga si sta scrivendo e quanto distano le righe di codice al di sopra o al di sotto di essa.

Estensioni impiegate Le estensioni adottate vengono in seguito descritte:

- **Vim** Emulazione dei *keybindings* di *Vim*$_G$, permette una navigazione e modifica del codice a me familiare, rapida ed efficiente.
- **Pylance** Motore di analisi statica per Python. Offre suggerimenti di codice e rileva errori in tempo reale.

- **Python** Supporto completo per lo sviluppo Python, inclusi *syntax highlighting* e *intellisense*.
- **Python Debugger** Strumento essenziale per il *debugging* efficace del codice Python.
- **AutoDocstring** Generazione automatica di *docstring* per funzioni e classi Python.
- **Python Environment Manager** Gestione semplificata degli ambienti virtuali Python.
- **Python Indent** Miglioramento dell'indentazione automatica per Python.
- **Django** Supporto specifico per lo sviluppo con il *framework* Django.
- **Docker** Integrazione con Docker per la gestione dei *container* direttamente dall'*IDE*$_G$.

Questa configurazione di VS Code ha creato un ambiente di sviluppo adatto alla codifica, efficiente e personalizzato, contribuendo alla produttività.

3.3.4 Ambiente di *testing*

DRF$_G$ è stato utilizzato per creare un ambiente di *testing* efficace. È stato sviluppato un progetto Django configurato per rispondere a specifici *endpoint*$_G$, generando (tramite ExcelGen) e inviando i *file* in base ai parametri della richiesta. Questo *setup* rende possibile simulare e testare in tempo reale le funzionalità di generazione e *download* dei *report*, offrendo un contesto realistico al fine di supportare l'attività di sviluppo.

***Endpoint* configurabili** Sono state implementate viste Django che rispondono a richieste *GET*, avviando il *download* dei *file* nei formati richiesti (`.xlsx`, `.docx`, `.pdf`), codice di esempio in figura 3.1.

Modelli di dati sufficientemente complessi Per riflettere la complessità dei dati reali, sono stati creati modelli Django con:

- Relazioni *one-to-many* e *many-to-many*

- Una vasta gamma di tipi di campo, tra cui:
  - `BooleanField`
  - `CharField`
  - `DateField`
  - `DateTimeField`
  - `DecimalField`
  - `TextField`
  - `EmailField`

Segue esempio dimostrativo:

```
from django.http import StreamingHttpResponse
import .models as md
import datetime as dt
# Dizionario di configurazione
from .confs.xlsx_configs import config

class ToXLSX(APIView):
        def get(self, request):
                # Queryset
                qs = md.example.objects
                # Campi da includere o escludere nel report
                listqs = ['-description']
                aux_xlsx = XlsxEngine.single(qs, config, listqs)
                xlsx_bstream = aux_xlsx.export()
                response = StreamingHttpResponse(
                        xslx_bstream,
                        ...
                )
                return response
```

Codice 3.1: Esempio di *endpoint* per il *testing* della libreria

3.3.5 Architettura di sistema

Il progetto in questione non richiede un'architettura *software* complessa o rigidamente strutturata. La natura del sistema, incentrata sulla generazione di *file* si presta a un approccio più flessibile e orientato ai processi.

L'applicativo opera principalmente come una *pipeline* di elaborazione dati, dove le operazioni vengono eseguite sequenzialmente in base alle specifiche fornite nel dizionario di configurazione. Questa struttura lineare permette di:

1. leggere e interpretare il dizionario di configurazione all'avvio;
2. applicare eventuali operazioni aggiuntive sul $QuerySet_G$;
3. eseguire operazioni di *data-retrieving* tramite i $QuerySet_G$ elaborati;
4. effettuare manipolazioni e trasformazioni sul $DataFrame_G$ risultante;
5. generare l'*output* nei formati richiesti.

La scelta di non adottare un'architettura definita in modo rigido è giustificata dai seguenti fattori:

- **Flessibilità operativa:** il sistema deve adattarsi dinamicamente alle specifiche fornite nel dizionario di configurazione. Un'architettura più rigida rischia di rendere più complessa l'implementazione di nuove funzionalità o la modifica di quelle preesistenti, il tutto potrebbe non avere guadagni significativi dal punto di vista di efficienza e manutenibilità;
- **Semplicità del flusso di lavoro:** le operazioni di lettura, elaborazione e scrittura seguono un flusso lineare e prevedibile. Questo riduce la necessità di complesse interazioni tra componenti, che sarebbero tipicamente gestite in un'architettura più strutturata;
- **Natura *task-oriented*:** l'applicativo è essenzialmente orientato all'esecuzione di una serie di *task* che, anche potendo essere opzionali, sono ben definite. Questa caratteristica si allinea bene con un approccio procedurale piuttosto che con un'architettura a componenti fortemente strutturata;

È importante notare che, sebbene questo approccio sia adatto per le attuali esigenze del progetto, potrebbe richiedere una revisione se il sistema dovesse crescere drammaticamente in complessità, oppure se dovessero emergere nuovi requisiti che necessitano di una struttura più robusta.

3.3.6 Dizionari di configurazione

La stesura accurata dei dizionari di configurazione rappresenta un aspetto cruciale nell'utilizzo efficace delle classi `XlsxEngine` e `DocxEngine`. I dizionari costituiscono l'interfaccia primaria e, di fatto, il principale mezzo attraverso il quale è possibile interagire con il programma e personalizzarne il comportamento. Data la sua importanza centrale, è fondamentale comprendere a fondo ogni aspetto della struttura di questi dizionari per un utilizzo completo della libreria.

3.3.6.1 Elementi comuni di configurazione

Il sistema di configurazione per le classi `XlsxEngine` e `DocxEngine` offre un controllo dettagliato sul processo di esportazione dei dati in formati Excel e Word. Questo approccio flessibile si adatta a una vasta gamma di requisiti, consentendo una personalizzazione approfondita dell'*output* generato. Entrambe le configurazioni condividono elementi fondamentali come la definizione della localizzazione per il formato di date e orari, la gestione dei valori nulli e la struttura base del documento. Un aspetto cruciale comune riguarda la definizione dei campi da esportare, che va oltre la mera selezione dei dati. Il sistema permette una manipolazione sofisticata, consentendo di concatenare valori da diverse fonti, applicare funzioni di trasformazione complesse o accedere a dati correlati attraverso chiavi esterne. La flessibilità del sistema si estende alla gestione di scenari specifici, come il trattamento di valori nulli o la traduzione di nomi di campi per colmare eventuali discrepanze tra gli identificativi usati dal *front-end* e il *back-end* del sistema. Entrambe le configurazioni offrono un controllo granulare sulla formattazione visiva del documento, permettendo di definire stili personalizzati per vari elementi. Ciò garantisce che l'*output* non solo contenga i dati corretti, ma li presenti anche in modo visivamente efficace e accattivante.

3.3.6.2 Caratteristiche specifiche di `XlsxEngine`

La configurazione per Excel si distingue per la possibilità di specificare fogli aggiuntivi per contenuti supplementari come legende, tabelle dei filtri applicati al *QuerySet*$_G$ e *pivot*, ampliando le capacità di presentazione dei dati. Offre inoltre un controllo specifico su elementi funzionali del foglio di calcolo, come l'adattamento automatico della larghezza delle colonne, l'applicazione di *autofilters* e l'applicazione di molteplici regole di formattazione condizionale.

La struttura dettagliata del dizionario di configurazione per la classe `XlsxEngine` è ampiamente descritta e discussa in appendice B.1.

3.3.6.3 Caratteristiche specifiche di `DocxEngine`

La configurazione per Word introduce elementi che sfruttano appieno le potenzialità del formato, permettendo una personalizzazione più approfondita del *layout* e del contenuto del documento. Essa permette di definire lo stile globale controllando aspetti come *font*, stile del testo e colori. Per questioni di contesto e presentazione è stato reso possibile includere paragrafi di testo personalizzati all'interno del documento. Questi possono essere posizionati strategicamente prima o dopo la tabella dei dati, arricchendo il documento con contenuti contestuali o spiegazioni. Ogni paragrafo può essere formattato individualmente, offrendo un elevato grado di controllo sulla presentazione del contenuto. La configurazione della tabella contenente i dati in Word è particolarmente sofisticata, permettendo di controllare aspetti come i bordi, l'adattamento automatico della dimensione delle colonne, la ripetizione dell'intestazione e il numero di righe e colonne per pagina.

La struttura dettagliata del dizionario di configurazione per la classe `DocxEngine` è ampiamente descritta e discussa in appendice B.2.

3.3.7 Struttura della libreria ExcelGen

3.3.7.1 Organizzazione dei moduli

I moduli sono solo quattro e sono disposti come in figura 3.3.

`common.py` Qui sono presenti tutte le classi e funzioni che vengono impiegate nel *data-retrieving*. Tutta la logica interna a questo modulo è comune per entrambe le classi responsabili della generazione della reportistica.

`xlsx.py` Modulo principale, qui è definita la classe `XlsxEngine` che contiene tutta la logica utile alla generazione di file Excel.

`docx.py` Modulo dove viene definita la classe `DocxEngine`, questa contiene il codice per la generazione di file Word.

`xlsx_default_styles.py` Interno alla cartella `xlsx_modules`, questo modulo contiene un singolo dizionario dove sono definiti gli stili di *default* per la generazione di documenti Excel. Separato dagli altri moduli per rendere più intuitiva l'operazione di ridefinizione dei formati base.

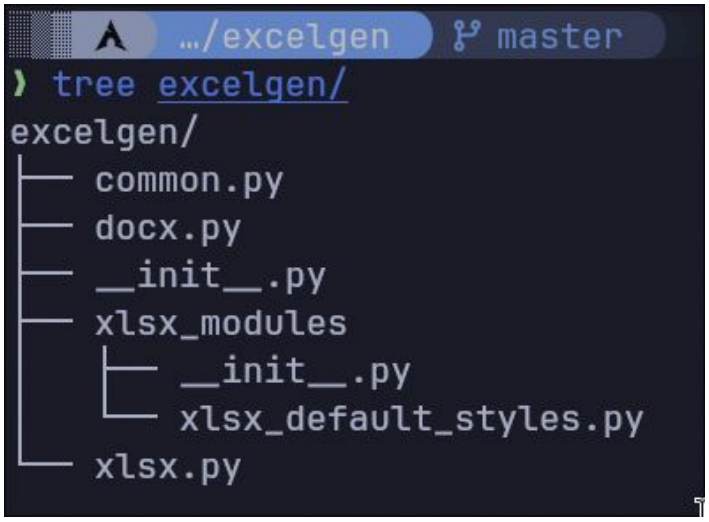

Figura 3.3: Contenuto della cartella principale di ExcelGen

3.3.7.2 Contenuto del modulo `common.py`

La componente principale è la classe `DjangoStuff`, interna al modulo, funge da contenitore per metodi statici che si appoggiano a componenti del *framework* Django. Tra le sue funzioni principali si distingue la capacità di concatenare campi, affinare *QuerySet*$_G$ basandosi su configurazioni specifiche e gestire un protocollo di inizializzazione. Il metodo `init_protocol` riveste particolare importanza, esso viene infatti chiamato all'interno dei costruttori delle classi

responsabili della generazione della reportistica, orchestrando una serie di operazioni fondamentali come il controllo della consistenza dei dati, l'affinamento del *QuerySet*$_G$ e la manipolazione del *DataFrame*$_G$ dei dati estratti. Questo processo include la sostituzione dei valori `NaN` (libreria *NumPy*$_G$) con valori `None` e la rimozione della consapevolezza del fuso orario dalle serie temporali, risolvendo i problemi legati all'esportazione in Excel di queste tipologie di dati.

Il modulo introduce inoltre una serie di funzioni ausiliarie che arricchiscono le capacità di manipolazione dei dati. Queste comprendono meccanismi per la traduzione ed estrazione dei campi rilevanti.

Per quanto riguarda la gestione delle localizzazioni, nel modulo sono presenti due costanti di tipo dizionario che associano stili di presentazione specifici per `date` e `datetime` a diverse impostazioni locali. Questo approccio assicura coerenza nella visualizzazione dei dati temporali attraverso diverse regioni e contesti d'uso, migliorando l'esperienza utente e la compatibilità internazionale delle applicazioni.

La struttura dettagliata del modulo `common.py` è ampiamente descritta e discussa in appendice C.1.

3.3.7.3 Contenuto del modulo `xlsx.py`

La classe `XlsxEngine` rappresenta il punto di ingresso principale per la generazione di *report* in formato Excel, sia singoli che multipli. Questa classe gestisce *QuerySet*$_G$, configurazioni, campi da esportare e filtri attivi, fornendo un'interfaccia versatile per l'esportazione dei dati. Il metodo `export` consente la generazione del documento Excel, restituendolo come oggetto `BytesIO` (libreria *io*$_G$) o come file salvato su disco, a seconda delle esigenze operative.

La classe `AuxWriter` è deputata alla scrittura di tabelle di legende e filtri attivi all'interno dei *worksheet*. Questa classe facilita l'inserimento di informazioni contestuali nei *report*, migliorandone la leggibilità e l'interpretazione.

La classe `SheetWriter` gestisce la creazione e l'organizzazione dei singoli fogli, operando sul *DataFrame*$_G$ dei dati e sul file di configurazione associato. Il metodo `write` orchestra operazioni complesse come:

- gestione degli *offset* per il posizionamento dei dati;
- applicazione di stili personalizzati in base al formato;
- impostazione di filtri automatici;
- adattamento automatico della larghezza delle colonne;
- applicazione di regole di formattazione condizionale;
- modifica dei dati tramite funzioni definite in configurazione;
- creazione di tabelle *pivot* per analisi avanzate;
- nascondimento o collasso delle colonne.

Una descrizione dettagliata della struttura del modulo `xlsx.py` è fornita in appendice C.2.

3.3.7.4 Contenuto del modulo `docx.py`

La classe `Paragraph` definisce una struttura per la gestione dei contenuti testuali. Questa classe incapsula il testo e le informazioni come la posizione nel documento, lo stile e il titolo.

La classe `DocxEngine` rappresenta il fulcro operativo. A differenza dell'approccio utilizzato per i *report* Excel, questa classe si concentra sulla costruzione di documenti con una struttura narrativa più complessa mantenendo però funzionalità come la manipolazione e formattazione dei dati in tabella.

La classe implementa diversi metodi per la gestione del *layout*, come la suddivisione automatica dei contenuti su più pagine e la ripetizione delle intestazioni.

Il sistema offre flessibilità nella strutturazione del contenuto permettendo la creazione di *report* con introduzioni, analisi e conclusioni strutturate.

Infine, `DocxEngine` offre versatilità nella generazione dell'*output*. Mette a dispozione metodi per la scrittura su *file* oppure per il ritorno del documento in formato `FileIO` (libreria *io*$_G$). Oltre alla generazione di file Word, la classe include funzionalità per la conversione in PDF tramite *LibreOffice*$_G$.

La struttura dettagliata del modulo `docx.py` è ampiamente descritta e discussa in appendice C.3.

Capitolo 4

Collaudo e stesura della documentazione

4.1 Documentazione

L'ultima settimana di stage è stata dedicata alla stesura della documentazione. Questa si compone di tre *file* in formato *Markdown* (`.md`):

`README.md` Questo file introduce la libreria, ne spiega gli scopi e fornisce indicazioni generali sul suo utilizzo.

`docs/xlsx.md` Approfondisce nel dettaglio la classe `XlsxEngine`, responsabile della generazione di file Excel, spiegando la struttura del dizionario di configurazione e le relative opzioni.

`docs/docx.md` In modo analogo al precedente, questo file documenta la classe `DocxEngine`, dedicata alla creazione di documenti Word e PDF, illustrando la configurazione accettata dalla suddetta classe.

4.2 Collaudo finale

Nell'ultima giornata di stage, insieme al *team* di sviluppo, si è proceduto all'installazione e al *deployment* di ExcelGen su un *branch* predisposto a scopo di *test* originato dal progetto Poseidon di RiskApp. Questo consiste in un ambiente

Debian$_G$ containerizzato tramite Docker. Questa fase ha permesso di testare l'integrazione e il funzionamento della libreria in un contesto simile se non uguale a quello di produzione.

Durante il collaudo finale, sono stati esaminati i casi d'uso più complessi di esportazione, usando *QuerySet*$_G$ che vanno a raccogliere migliaia di *record* spaziando fra tutti i possibili tipi di dato previsti.

Non sono stati riscontrati particolari problemi o malfunzionamenti, fatta eccezione per una limitazione relativa all'applicazione della formattazione condizionale, la quale ammetteva una sola condizione per colonna. Tuttavia tale criticità è stata prontamente risolta nell'arco di pochi minuti.

In particolare, la soluzione implementata ha sfruttato le funzionalità avanzate della libreria XlsxWriter, permettendo di applicare molteplici regole di formattazione condizionale per ciascuna colonna. Questo intervento ha consentito di superare efficacemente la limitazione inizialmente riscontrata, garantendo il pieno soddisfacimento dei requisiti del sistema *back-end* in materia di *download* di reportistica personalizzata.

In conclusione, il collaudo finale ha permesso di confermare il corretto funzionamento della libreria Python realizzata, la sua integrazione nell'ambiente di produzione e l'adeguatezza nel soddisfare i requisiti più complessi gestiti attualmente da RiskApp.

Capitolo 5

Conclusioni

5.1 Raggiungimento Obiettivi

5.1.1 Obiettivi di progetto

In seguito una tabella che indica gli esiti per ciascun obiettivo di progetto definito in sezione 2.4.1:

Id	Descrizione	Esito
M01	Creazione di una libreria installabile tramite gestore di pacchetti *pip*$_G$ su diversi progetti	Soddisfatto
M02	Implementazione dell'estrazione in formato `xlsx` di dati non formattati	Soddisfatto
M03	Implementazione dell'estrazione in formato `xlsx` di dati formattati	Soddisfatto
M04	Sviluppo di funzionalità per l'estrazione di dati con stile configurabile	Soddisfatto
M05	Implementazione di regole condizionali per l'estrazione dei dati	Soddisfatto
D01	Estrazione dei dati in formato `docx`	Soddisfatto

Continua nella prossima pagina...

Tabella 5.1 – Continuo della tabella

Id	Descrizione	Esito
O01	Estrazione dei dati in formato `pdf`	Soddisfatto
O02	Stesura dei test di unità	Parzialmente soddisfatto
O03	Redazione della documentazione tecnica	Soddisfatto

Tabella 5.1: Tabella degli obiettivi di progetto raggiunti

Come si nota, l'obiettivo **O02** non è stato svolto nella sua interezza. Questo perché è stata data precedenza allo sviluppo di nuove *features* in comune accordo con l'azienda.

5.1.2 Obiettivi personali

In seguito una tabella che indica gli esiti per ciascun obiettivo personale definito in sezione 2.4.2:

Id	Descrizione	Esito
M06	Approfondire la conoscenza di Python e nello specifico Django	Soddisfatto
M07	Ridurre la complessità nella creazione dei *report* fornendo un approccio standardizzato alla generazione dei documenti.	Soddisfatto
M08	Accelerare il processo di generazione di *report*	Soddisfatto

Continua nella prossima pagina...

Tabella 5.2 – Continuo della tabella

Id	Descrizione	Esito
M09	Essere aperto all'implementazione di eventuali nuove *features* proposte da RiskApp.	Soddisfatto
M10	Migliorare le capacità personali di gestione di progetti *software* complessi	Soddisfatto
M11	Scrivere codice efficiente e manutenibile al fine di fornire una piattaforma per future espansioni	Soddisfatto
D02	Comprendere meglio il contesto architetturale e tecnologico dentro il quale la libreria sviluppata opererà	Soddisfatto
D03	Collaborare con il *team* di sviluppo	Soddisfatto

Tabella 5.2: Tabella degli obiettivi personali raggiunti

5.2 Consuntivo finale

Lo stage è durato 315 ore come preventivato, tuttavia la ripartizione delle ore è stata differente rispetto quanto previsto.

Attività	Preventivo	Consuntivo
Formazione sulle tecnologie	75	64
Analisi e pianificazione	40	48
Sviluppo	150	163
Collaudo finale	50	40
Totale ore	**315**	**315**

Tabella 5.3: Tabella consuntivo orario

La ripartizione delle ore ha subito dei cambi per le seguenti ragioni:

Formazione sulle tecnologie Questa fase si è incentrata maggiormente sullo studio del *framework* Django e *DRF*$_G$ rispetto le librerie XlsxWriter e python-docx per la generazione dei documenti. Ho sovrastimato il tempo necessario all'apprendimento di Django per ciò che concerne la creazione di *playground*$_G$ dove effettuare i *test* della libreria.

Analisi e pianificazione Durante questa fase l'ideazione di una possibile configurazione ha richiesto molto più tempo del previsto in quanto andava architettata la struttura e tutte le possibili opzioni da implementare. Mentre per quel che riguarda l'ideazione dell'architettura dell'applicativo, questa ha richiesto estremamente meno tempo di quanto pianificato in quanto il *software* deve eseguire le stesse operazioni sequenzialmente.

Sviluppo Il tempo di sviluppo si è prolungato in quanto qualsiasi *feature* è stata proposta dal *team* di sviluppo è stata prontamente implementata, allungando il tempo realmente necessario al raggiungimento degli obiettivi di oltre una settimana.

Collaudo finale Il tempo rimanente è stato impiegato per:

- scrivere meticolosamente i documenti relativi la documentazione con eventuali esempi di utilizzo;
- implementare alcuni *test* di unità;
- eseguire il *refactoring* di determinate parti del codice;
- configurare *file* tipici delle librerie Python (vedi sezione 3.3.1);
- effettuare il collaudo sui sistemi RiskApp affiancato dal *team* di sviluppo.

5.3 Conoscenze acquisite

Lo sviluppo di ExcelGen ha rappresentato un'opportunità significativa per approfondire e ampliare le mie conoscenze in diversi ambiti dello sviluppo *soft-*

ware. In particolare, ho potuto consolidare ed espandere le mie competenze nelle seguenti aree:

- **Python**: ho affinato la mia padronanza del linguaggio Python, approfondendo aspetti avanzati come la gestione delle eccezioni e la modalità di programmazione orientata agli oggetti, che segue principi differenti da linguaggi come C++ e Java.

- **Django**: ho approfondito l'utilizzo di Django, in particolare per quanto riguarda la struttura generale dei progetti basati sul *framework* in questione e la gestione efficiente dei modelli di dati.

- **Django REST framework**: ho acquisito competenze nell'utilizzo di *DRF*$_G$ per la creazione di *API*$_G$ *RESTful*.

- **Struttura dei documenti Excel e Word**: ho approfondito la conoscenza della struttura interna dei formati `.xlsx` e `.docx`, comprendendo come manipolare efficacemente questi formati a livello programmatico. Inoltre nel caso specifico di `docx` ho avuto la possibilità di manipolare direttamente l'*XML*$_G$ tramite funzioni avanzate della libreria python-docx.

- **Excel**: ho avuto l'opportunità di riavvicinarmi a Microsoft Excel, approfondendo le sue funzionalità avanzate e le potenzialità di automazione.

- **Gestione della configurazione**: avendo già esperienza nell'*editing* di *file* di configurazione in ambito Linux ho sottovalutato le abilità necessarie per l'ideazione di una configurazione efficiente ed efficace. Questo mi ha dato l'opportunità di approfondire questa *skill* e di comprenderne meglio la complessità.

- ***Unit test***: non avendo mai scritto *test* di unità in Python questa è stata un'opportunità per apprendere concetti e costrutti fondamentali della libreria PyTest.

- **Documentazione tecnica**: ho affinato le mie capacità di produrre documentazione tecnica chiara e completa, essenziale per l'utilizzo efficace della libreria da parte di altri sviluppatori.

5.4 Valutazione personale di ExcelGen

L'esperienza di stage ha portato alla realizzazione di una libreria di esportazione dati che soddisfa efficacemente le esigenze di reportistica di RiskApp. Il prodotto finale si è dimostrato funzionale e versatile, adattandosi alle diverse necessità di esportazione dei *report* aziendali.

In retrospettiva, emergono alcune considerazioni significative:

- **Approccio alla programmazione**: l'esperienza ha evidenziato l'influenza del mio *background* in programmazione Java sull'approccio adottato nello sviluppo in Python. Con le conoscenze acquisite durante lo stage, riconosco che un approccio più idiomatico a Python avrebbe potuto portare a soluzioni più eleganti ed efficienti.

- **Strutturazione del codice**: riconosco la presenza di classi che non portano alcun vantaggio sia alla leggibilità del codice che all'efficienza operativa. La classe `DjangoStuff` in `common.py`, ad esempio, non ha ragione di esistere, in quanto funge semplicemente da contenitore di metodi, svolge un compito simile a un banale modulo.

- **Complessità dei moduli**: i moduli `xlsx.py` e `docx.py`, pur essendo comprensibili, ben organizzati e con una struttura che reputo manutenibile, si estendono per quasi 1000 righe ciascuno. La dimensione dei moduli è principalmente dovuta alla necessità di gestire molteplici casi limite e comportamenti anomali. In futuro, potrebbe essere vantaggioso esplorare strategie di *refactoring* per migliorare la modularità del codice.

- **Utilizzo delle caratteristiche di Python**: una maggiore familiarità con le caratteristiche native di Python avrebbe potuto portare a un *design*

più snello, facendo maggior uso delle componenti *built-in* del linguaggio e riducendo la dipendenza da strutture di classe.

Nonostante vi siano margini di miglioramento in termini di *design* e ottimizzazione, il prodotto sviluppato soddisfa pienamente i requisiti funzionali e di vincolo espressi dall'azienda.

5.5 Valutazione personale dell'esperienza di stage

L'esperienza di stage presso RiskApp si è rivelata formativa e gratificante sotto molteplici aspetti.

L'azienda ha offerto un ambiente di lavoro positivo, caratterizzato da:

- autonomia operativa bilanciata da una cultura di collaborazione;
- disponibilità costante del personale di riferimento, nonostante i loro impegni lavorativi;
- un'atmosfera che ha favorito l'apprendimento e la crescita professionale.

Il progetto assegnatomi ha presentato sfide stimolanti che hanno permesso di:

- espandere ExcelGen ben oltre la totalità dei requisiti, spingendomi a esplorare nuovi problemi che richiedevano soluzioni innovative;
- approfondire le mie conoscenze di Python, scoprendo l'impiego di molte nuove librerie;
- superare le iniziali difficoltà con Django, trasformandole in un'opportunità di apprendimento significativa.

La natura stimolante del progetto, unita all'ambiente professionale dell'azienda, ha reso questa esperienza la più coinvolgente dall'inizio del mio percorso di studi.

In conclusione, mi ritengo molto soddisfatto dell'esperienza di stage. Questa non solo è stata gratificante, ma ha superato le mie aspettative iniziali, fornendomi una visione pratica e approfondita dello sviluppo *software* in un contesto professionale e l'opportunità di vedere il mio lavoro integrato in un'applicazione reale.

Appendice A

Dettagli casi d'uso

UC 1: Generazione di *report* Excel

Attori Principali: Django *endpoint*

Descrizione: Su richiesta viene generato un *report* Excel

Precondizioni: Il sistema riceve una richiesta *HyperText Transfer Protocol (HTTP)*$_G$ *GET* all'*endpoint* designato, in base alla richiesta associa il *QuerySet*$_G$ ad un dizionario di configurazione e genera il *report*.

Postcondizioni: Il sistema ritorna al chiamante il *report* generato o viene sollevato un errore

UC 2: Generazione di *report* Word

Attori Principali: Django *endpoint*

Descrizione: Su richiesta viene generato un *report* Excel

Precondizioni: Il sistema riceve una richiesta *HTTP*$_G$ *GET* all'*endpoint* designato, in base alla richiesta associa il *QuerySet*$_G$ ad un dizionario di configurazione e genera il *report*.

Postcondizioni: Il sistema ritorna al chiamante il *report* generato o viene sollevato un errore

UC 2.1: Generazione di *report* in PDF

Attori Principali: Django *endpoint*

Descrizione: Su richiesta viene generato un *report* PDF

Precondizioni: Il sistema riceve una richiesta *HTTP*$_G$ *GET* all'*endpoint* designato, in base alla richiesta associa il *QuerySet*$_G$ ad un dizionario di configurazione e genera il *report* Word, poi viene convertito in PDF tramite l'applicativo *LibreOffice*$_G$.

Postcondizioni: Il sistema ritorna al chiamante il *report* generato o viene sollevato un errore

Appendice B

Dettagli dei dizionari di configurazione

B.1 Configurazione per la classe `XlsxEngine`

Parametri principali Essi includono:

- `locale` : Obbligatorio. Definisce la localizzazione per il formato di date e orari (*e.g.* `'en_US'`, `'it_IT'`).
- `sheet_name` : Obbligatorio. Specifica il nome del foglio Excel dove scrivere la tabella dei dati esportati.
- `legend_sheet_name` : Opzionale. Nome del foglio per la legenda e i filtri attivi, se utilizzati.
- `pivot_sheet_name` : Opzionale. Nome del foglio per le tabelle pivot, se utilizzate.
- `fields` : Obbligatorio. Lista di dizionari che rappresentano i campi da esportare e le loro proprietà.
- `none_value` : Opzionale. Valore personalizzato per i campi nulli (globale).
- `table_offsets` : Opzionale. Tupla o dizionario che definiscono la posizione della tabella nel foglio di calcolo (*e.g.* `(riga,colonna)`).

- `translate_fields` : Opzionale. Accetta un dizionario con coppie '*alias*':'valore' per la traduzione degli identificativi dei campi. Come già anticipato, questo viene predisposto per colmare le discrepanze fra i nomi adottati dal *front-end* e quelli usati dal *back-end*.
- `order_by` : Opzionale. Accetta l'identificativo di un *field*, preceduto o meno dal simbolo *dash* (`-`), definisce il criterio di ordinamento dei dati.
- `style_sets` : Opzionale. Accetta un dizionario per la definizione di stili personalizzati.
- `flags` : Opzionale. Opzioni booleane per il controllo del processo di scrittura.
- `collapsed` : Opzionale. Lista di liste contenenti due campi x e y. Nel foglio dei dati, dalla colonna relativa il *field* x a quella relativa y, i valori verranno collassati per una presentazione più ordinata.

Esempio dimostrativo:

```
CONFIG={
        'locale': 'it_IT',
        'sheet_name': 'Utenti',
        'none_value': 'N.D.',
        'table_offsets': (1,1),
        'translate_fields': {'xxx':'campo1'},
        'order_by': '-xxx',
        'fields'{['field': 'campo1', 'label':'Utenti']},
        'style_sets':{'header': {'valign': 'top'}},
        'flags':{'autofit': True, 'autofilter': True}
}
```

Codice B.1: Esempio dimostrativo della configurazione

È importante notare che i valori impostati su `None` verranno ignorati, applicando le impostazioni predefinite.

Valore della chiave `fields` Essa accetta una lista di dizionari (come in esempio B.2), ciascuno dei quali descrive un campo da esportare. La posizione all'interno della lista determina la posizione della colonna nel foglio di calcolo. Ogni dizionario relativo un *field* può includere:

- `field` : Identificativo del campo.
- `label` : Etichetta personalizzata da stampare sull'*header*.
- `legend` : Descrizione da includere nella legenda.
- `data_format` : Nome di un formato predefinito o personalizzato.
- `cond_format` : Formattazione condizionale. Segue le medesime regole predisposte della libreria XlsxWriter.
- `extend_qs`, `concat`, `F` : Metodi di manipolazione del *QuerySet*$_G$ (mutuamente esclusivi, in ordine di priorità).
  1. `extend_qs` : accetta un riferimento a una funzione che prenderà il *QuerySet*$_G$ come argomento, lo aggiorna secondo specifici parametri e lo ritorna;
  2. `concat` : concatena valori provenienti da più *field*, accetta un dizionario con due chiavi:
     (a) `what` : accetta una lista dei campi da concatenare nella forma `[modelx__field, modely_field]`;
     (b) `separator` : accetta una stringa che farà da separatore dei valori.
  3. `F`: utile per ottenere valori tramite chiave esterna. Accetta un valore nella forma `model__field`.
- `pivot` : Se `True`, permette di stampare una tabella pivot per questo campo (la *flag* `printPivot` deve avere valore `True`).
- `hidden` : Se `True`, la colonna avrà larghezza zero *pixel (px)*$_G$.
- `to_str` : Se `True`, converte i valori della colonna in stringhe prima della scrittura.

- `none_value` : Valore da sostituire ad eventuali valori `None` (specifico per il *field* in questione).

Esempio dimostrativo:

```
'fields': [
{
        'field': 'product_name', 'label': 'Prodotto',
        'legend': 'Prodotto a cui il commento fa
        ↪  riferimento', 'F': 'product__name',
},
{
        'field': 'content', 'label': 'Contenuto',
        'legend': 'Contenuto commento', 'to_str': True
},
{
        'field': 'created', 'label': 'Data creazione',
        'legend': 'Data creazione del commento',
        'data_format': 'datetime', 'hidden': True
},
{
        'field': 'userdata', 'label': 'Creatore',
        'legend': 'Creatore del commento',
        'extend_qs': add_userdata, 'pivot': True
}
]
```

Codice B.2: Esempio dimostrativo di una possibile lista accettata dalla *keyword* `fields`

Valore della chiave `style_sets` Essa accetta un dizionario che permette di definire stili personalizzati compatibili con la libreria XlsxWriter. Gli stili predefiniti includono `header`, `legend_label`, `filter_label`, `pivot_label`, `legend_description`, `filter_condition`, `pivot_count`, `euro`, `pound`,

`dollar`, `date`, e `datetime`. Tutti questi stili sono sovrascrivibili in questa sezione.

Valore della chiave `flags` Essa accetta un dizionario dove ad ogni chiave viene associato un valore booleano. Queste controllano vari aspetti del processo di esportazione. In seguito vengono descritte accoppiate ai loro valori di *default*:

- `formatHeader` : `True`. Formatta l'*header* della tabella.
- `textWrapAll` : `True`. Applica la proprietà `text_wrap` a tutte le celle (mostra più righe per cella).
- `autofit` : `True`. Adatta la larghezza delle colonne al contenuto stampato al loro interno (non tiene conto dei contenuti dell'eventuale legenda sotto la tabella).
- `autofilter` : `True`. Applica i filtri automatici alla tabella.
- `legendOnDedicatedSheet` : `True`. Stampa la legenda e la tabellina dei filtri attivi su un foglio dedicato.
- `printLegend` : `False`. Stampa la legenda.
- `printFilters` : `False`. Stampa i filtri attivi.
- `formatLegend` : `True`. Formatta la tabellina della legenda.
- `formatFilters` : `True`. Formatta la tabellina dei filtri attivi.
- `forceAutofit` : `False`. Applica la proprietà di *autofit* a tutte le colonne (legenda e filtri compresi se stampati sullo stesso foglio).
- `printPivot` : `False`. Stampa le tabelle *pivot*.

Esempio di utilizzo delle *flag* in B.1

B.2 Configurazione per la classe `DocxEngine`

Parametri principali La configurazione per la generazione di file `docx` risulta simile a quella accettata dalla classe `XlsxEngine`, i parametri in comune e che mantengono lo stesso comportamento vengono di seguito riportati:

- `locale`
- `none_value`
- `sheet_name`
- `translate_fields`
- `order_by`

I nuovi parametri principali introdotti nel dizionario di configurazione includono:

- `fields` : Obbligatorio. Elenco dei campi da esportare e le loro proprietà.
- `overall_style` : Opzionale. Stile globale del documento
- `paragraphs` : Opzionale. Elenco dei paragrafi da stampare e le loro proprietà.

Qualunque parametro impostato a `None` o non conferme viene ignorato e al suo posto un valore di *default* viene utilizzato.

Valore della chiave `fields` In modo analogo alla configurazione per `XlsxEngine`, essa accetta un elenco di dizionari, ciascuno dei quali descrive un campo da esportare. La posizione all'interno della lista determina la posizione della colonna nella tabella Word. Ogni dizionario di campo può includere:

- `field` : Identificativo del campo.
- `label` : Etichetta personalizzata del campo.
- `data_format` : Formati supportati:
  - `date`
  - `datetime`
  - `euro`
  - `pound`
  - `dollar`
  - `bool`

per esigenze specifiche esiste la proprietà `formatter`.

- `formatter` : Accetta un riferimento a una funzione che prenderà come argomento uno ad uno i valori di un *field*, l'*output* sarà l'effettivo valore stampato.
- `to_str` : Se `True`, converte i valori della colonna in stringhe prima della scrittura.
- `none_value` : Valore da sostituire ad eventuali `None` (specifico per il *field* in questione).

Valore della chiave `overall_style` Come già anticipato, questa chiave accetta un dizionario che determina la formattazione generale del testo.

Formattazioni comuni a tutte le componenti (tranne `margins`, `table` e `landscape`):

- `font` : Permette di specificare il *font* da utilizzare;
- `size` : Permette di specificare la grandezza del *font* in *px*$_G$;
- `bold` : Se impostato a `True` permette di applicare il grassetto;
- `italic` : Se impostato a `True` permette di applicare il corsivo;
- `color` : Permette di specificare il colore del testo. Accetta una stringa col valore esadecimale del colore.

Le chiavi illustrate in seguito accettano un dizionario come valore mediante il quale è possibile definire lo stile di formattazione:

- `text` : Formattazione generale del testo.
- `title1` : Formattazione del titolo.
- `title2` : Formattazione dei titoli dei paragrafi.
- `paragraphs` : Formattazione del testo dei paragrafi.
- `table` : Formattazioni relative la tabella. Le chiavi del dizionario accettato sono le seguenti:
  - `border` : Stile del bordo della tabella. Il dizionario accettato da questa chiave ha coppie chiave-valore come segue:

    * `size` : Grandezza del bordo.
    * `color` : Colore del bordo.
  - `autofit` : Se impostato a `True` adatta la larghezza della colonna al contenuto.
  - `repeat_header` : Se impostato a `True` ripete l'*header* in ogni pagina.
  - `cols_per_page` : Definisce il massimo numero di colonne per pagina.
  - `rows_per_page` : Definisce il massimo numero di righe per pagina.
- `header_cells` : Formattazione dell'*header* della tabella. Oltre alle comuni chiavi di stile ammette pure:
  - `fg_color` : Colore di sfondo delle celle dell'*header*. Accetta una stringa col valore esadecimale del colore.
  - `bottom_border` : Se impostato a `True` inserisce il bordo inferiore alle celle dell'*header*.
- `landscape` : Se impostato a `True` il *layout* sarà orizzontale.
- `margins` : Permette di specificare la grandezza dei margini. Accetta un dizionario con le seguenti chiavi:
  - `top`
  - `bottom`
  - `left`
  - `right`

  Ognuna di esse ammette un valore `float` che rappresenta la grandezza in centimetri.

Esempio dimostrativo:

```
'overall_style':{
        'text': {'font': 'Arial'},
        'title1': {'align': 'centre'},
        'title2': {'font': 'Times New Roman'},
        'paragraphs': {'size': 11},
        'table': {
                'border': {'size': 8, 'color': '000000'},
                'autofit': False, 'repeat_header': False,
                'cols_per_page': 3, 'rows_per_page': 3,
        },
        'header_cells': {
                'align': 'centre', 'fg_color': 'FF0000',
                'font': 'Times New Roman', 'size': 14,
                'bottom_border': False,
        },
        'landscape': True,
        'margins': {
                'left': 1.3, 'right': 1.3,
                'top': 1.3, 'bottom': 1.3
        },
}
```

Codice B.3: Esempio di sezione di configurazione relativa lo stile globale del documento

Valore della chiave `paragraphs` Essa accetta un dizionario che permette di definire dei paragrafi per accompagnare la tabella, così da spiegare oppure dare contesto ai dati estratti. Il metodo predisposto è di semplice comprensione, le *keyword* interne al dizionario determinano i titoli dei paragrafi e il dizionario a loro associato definisce: testo del paragrafo, ordine e stile di formattazione. La struttura del dizionario associato è qui descritta:

- `position` : Un numero qualunque che in relazione alla `position`

degli altri paragrafi definirà l'ordine di stampa di questi ultimi.

- `text` : Testo del paragrafo.
- `print_over_table` : Se impostato a `True` stamperà il paragrafo nella prima batteria di paragrafi sopra la tabella (*default*), altrimenti verrà stampato con la seconda sotto la tabella.
- `style` : Accetta un dizionario contenente due chiavi, questo viene usato dalla libreria per formattare il singolo paragrafo:
  - `title` : Formattazione specifica per il titolo del paragrafo, accetta lo stesso dizionario di configurazione delle *keyword* principali `text, title1` e `title2` .
  - `text` : Formattazione specifica per il testo del paragrafo, si comporta in modo analogo al precedente.
- `show_title`: Se impostato a `False` non stampa il titolo del paragrafo.

Esempio dimostrativo:

```
        'paragraphs':{
                'Titolo del paragrafo': {
                        'position': 0,
                        'text': 'Questo è un paragrafo sotto la
                        ↪  tabella stampata',
                        'print_over_table': False,
                        'style': {
                                'title': {
                                        'font': 'Comic Sans MS',
                                        'size': 20,
                                        'bold': True,
                                        'color': 'FF0000'
                                },
                                'text': {
                                        'align': 'left',
                                        'italic': True,
                                        'color': '000000'
                                }
                        },
                }
        }
```

Codice B.4: Esempio sezione di configurazione relativa la definizione dei paragrafi

Appendice C

Dettagli sulla struttura dei moduli Python

C.1 `common.py`

C.1.1 Classe DjangoStuff

Classe statica che raccoglie tutte le funzioni che si appoggiano al *framework* Django. Funge banalmente da contenitore di metodi.

Metodi statici:

- `__concat_func(qs: QuerySet, name: str, field: dict)`
  Concatena più campi sotto un unico *field*.

- `__annotate_engine(qs: QuerySet, fields: dict, order_by: str="")`
  Affina il *QuerySet*$_G$, nello specifico applica i valori inseriti in configurazione relativi `order_by`, `extend_qs` oppure `concat` oppure `F`.

- `init_protocol(queryset: QuerySet, config: dict, fields_to_include: list)`
  Si occupa dell'esecuzione ordinata delle seguenti *task*:

  1. controllo di consistenza dei dati;

2. traduzione degli identificativi dei campi interni all'argomento lista `fields_to_include`;
3. affinamento del *QuerySet*$_G$ in base ai valori presenti in configurazione;
4. generazione del *DataFrame*$_G$ dei dati;
5. sostituzione di tutti i valori `nan` (libreria *NumPy*$_G$) interni al *DataFrame*$_G$ con dei valori `None`;
6. rimozione della proprietà di *timezone awareness* a tutte le serie interne al *DataFrame*$_G$ (quando possibile);
7. modifica della struttura del dizionario di configurazione.

C.1.2 Funzioni

- `translate_fields(fields_list: list, translations: dict)`
  Data una lista traduce tutti valori interni che trovano corrispondenza nel dizionario delle traduzioni.
- `translate_dict(dictionary: dict, translations: dict)`
  Analogo al metodo precedente ma effettua le traduzioni sulle chiavi del dizionario passato.
- `get_relevant_fields_list(fields_to_include: list, config: dict)`
  Restituisce una lista dei campi da prendere in considerazione, basandosi su quelli definiti in configurazione e quelli presenti nella lista passata come argomento.
- `legacy_fields_format(fields: list)`
  Alla chiave `fields`, presente in configurazione, viene associata una lista nella forma descritta nel codice B.2. La funzione restituisce quella lista in forma di dizionario, seguendo questa struttura:

```
{
        'id_field1'{
                'label': 'label_name',
                'legend': 'legend...',
                ...
        },
        'id_field2':{...}
}
```

Codice C.1: Esempio di come viene modificato il valore associato alla *keyword* `fields`

- `order_by_translate(order_by_string: str, translated_fields: dict)`

  Il campo che definisce l'ordine dei *record* potrebbe in futuro arrivare dal *front-end*, è dunque necessario gestire l'evenienza secondo la quale il campo per l'ordinamento non sia nella sua forma corretta. Il metodo traduce il campo `order_by` tenendo conto che esso possa cominciare con il simbolo *dash* (`-`).

C.1.3 Costanti

Vengono infine definiti due dizionari che categorizzano le possibili locali associandole a uno stile di presentazione per `date` e `datetime`.

C.2 `xlsx.py`

C.2.1 `XlsxEngine`

Classe il cui compito è la generazione di *report* singoli o multipli.

Argomenti del costruttore:

- `querysets: list`

  Lista dei *QuerySet*$_G$.

- `configs: list`

  Lista dei dizionari di configurazione. La struttura è spiegata nella sezione B.1.

- `include_fields_collection: list =[]`

  Lista di liste relative i campi da esportare.

- `filters_dicts: list =[]`

  Lista di dizionari che descrivono i filtri attivi.

Attributi: analoghi agli argomenti del costruttore.

Metodi:

- **costruttore**

  Assegna banalmente i valori degli argomenti ai relativi attributi.

- `single(queryset : QuerySet, config: dict, field_list: list =[],`

  `active_filters: dict ={}):`

  `classmethod` che mette a disposizione un costrutture specifico per l'esportazione di un singolo *report*.

- `export(path: str =None)`

  Metodo per l'esportazione del documento Excel in formato `BytesIO` (libreria *io*$_G$ di Python), oppure salva il *file* nel percorso passato come argomento.

C.2.2 `AuxWriter`

Classe il cui compito è scrivere la legenda e i filtri attivi sul *worksheet* passato come argomento.

Argomenti del costruttore:

- **costruttore**

  Assegna banalmente i valori degli argomenti ai relativi attributi.

- `worksheet`

  Oggetto ricavato da `pandas.ExcelWriter`, permette di effettuare modifiche sul foglio di calcolo che rappresenta.

- `flags: dict`

  Dizionario delle *flags* interne alla configurazione.

Attributi: analoghi agli argomenti del costruttore.

Metodi:

- `write(data: list, row:int, column: int, title: str, label_format=None, description_format=None)`

  Scrive *label* e legenda, il tutto viene formattato.

- `write_just_text(data: list, row: int, column: int, title: str)`

  Analogo al precedente ma senza formattazione.

C.2.3 `SheetWriter`

Classe che contiene i metodi utili alla scrittura dei fogli Excel relativi un singolo *report*.

Argomenti del costruttore:

- `writer: ExcelWriter`

  Oggetto della libreria *Pandas*$_G$, appoggiandosi alla libreria XlsxWriter (oppure *OpenPyXL*$_G$), mette a disposizione tutti i metodi per la scrittura del *file* Excel.

- `queryset: QuerySet`

- `config: dict`

  Dizionario di configurazione. La struttura è spiegata nella sezione B.1.

- `fields_list: list=[]`

  Lista non ordinata dei *field* da esportare.

- `active_filters: dict={}`

  Dizionario rappresentante i filtri attivi.

Attributi:

- `df: DataFrame`

  Oggetto della libreria *Pandas*$_G$. Contiene tutti i dati ricavati dal *QuerySet*$_G$ per l'esportazione.

- `writer: ExcelWriter`

  Oggetto della libreria *Pandas*$_G$ che mette a disposizione i metodi per la scrittura del *file* Excel.

- `config: dict`

  Dizionario di configurazione.

- `relevant_fields: list`

  Lista ordinata dei *field* da esportare.

- `active_filters: dict`

  Dizionario rappresentante i filtri attivi.

Metodi:

- **costruttore**

  Assegna i valori agli attributi `df, config, relevant_fields` mediante il metodo `init_protocol` della classe `DjangoStuff` (vedi sezione C.1.1). Poi tramite la funzione `translate_dict` (vedi sezione C.1.2) traduce il dizionario `active_filters` basandosi sulla sezione per le traduzioni presente in configurazione.

- `locale_date_format(locale: str, is_datetime: bool = False)`

  Restituisce un dizionario relativo il formato di ora e data. Questo dizionario è compatibile con la libreria XlsxWriter e determina il formato di `date` e `datetime` interni al *workbook*.

- `write_pivot(worksheet, row: int, field_name: str, formats: dict)`

  Metodo predisposto per la stampa delle tabelle *pivot*.

- `__retrieve_offsets()`

  Restituisce una tupla con le coordinate per la stampa della tabella.

- `write()`

  Responsabile dell'attività di scrittura relativa a un singolo dizionario di configurazione. Esegue sequenzialmente le seguenti operazioni:

  1. recupero degli *offset* ;
  2. impostazione del valore *default* per i campi nulli;
  3. inizializzazione del foglio di lavoro;
  4. gestione degli stili personalizzati e creazione del dizionario dei formati;
  5. scrittura della tabella dati. Che prevede per ogni colonna:
     - Recupero di *label* e descrizione per la legenda;
     - applicazione del formato adatto alla colonna dati;
     - applicazione delle formattazioni condizionali;
     - scrittura dell'*header*.
  6. vengono applicate le funzionalità predefinite come *autofilter* e *autofit*, in base alle *flag* specificate nella configurazione;
  7. opzionale. Stampa la legenda;
  8. opzionale. Stampa i filtri attivi nello stesso foglio della legenda;
  9. opzionale. Stampa le tabelle *pivot* su un foglio dedicato;
  10. opzionale. Nasconde o collassa le colonne specificate nell'apposita sezione della configurazione.

C.3 `docx.py`

C.3.1 `Paragraph`

Utile a contenere i valori di un paragrafo generico.

Argomenti del costruttore

- `text: str`

  Testo del paragrafo.

- `position: int`

  Posizione che definisce l'ordine di stampa specifico del paragrafo.

- `style: dict={}`

  Determina la formattazione specifica del paragrafo.

- `title: str=""`

  Titolo del paragrafo.

Attributi: analoghi agli argomenti del costruttore.

Metodi:

- **costruttore**

  Assegna banalmente i valori degli argomenti ai relativi attributi.

C.3.2 `DocxEngine`

Classe che contiene i metodi utili all'esportazione di un *report* Word.

Argomenti del costruttore

- `queryset: QuerySet`

- `config: dict`

  Dizionario di configurazione. La struttura è spiegata nella sezione B.2.

- `field_to_include: list`

  Lista non ordinata dei *field* da esportare.

Attributi:

- `df: DataFrame`

  Oggetto della libreria *Pandas*$_G$ contenente tutti i dati ricavati dal *QuerySet*$_G$ raffinato per l'esportazione.

- `config: dict`

  Dizionario di configurazione.

- `relevant_fields: list`

  Lista ordinata dei *field* da esportare.

Metodi:

- `__set_table_cell_border(cell, border_size: str, edges_prop: dict)`

  Formatta i bordi della cella passata come argomento. Va a modificare l'*XML*$_G$ del file Word tramite funzioni avanzate della libreria python-docx.

- `__get_border_props()`

  Ritorna un dizionario che descrive grandezza e colore dei bordi.

- `__set_table_borders(table)`

  Formatta i bordi della tabella.

- `__aligner(align: str)`

  In base alla stringa passata ritorna la proprietà di allineamento compatibile con la libreria python-docx.

- `__formatter(elem_run, style: dict)`

  Formatta l'elemento passato in base al dizionario di stile.

- `__set_section(section)`

  Formatta l'oggetto `section` passato. Questa funzione è responsabile dell'orientamento del foglio, della formattazione dei margini e dell'impostazione delle dimensioni del foglio.

- `__set_col_header(cell)`

  Formatta la cella passata in base al dizionario di stile associato alla chiave `header_cells` specificata in configurazione.

- `__make_sublists(ordered_cols: list, cols_per_page: int)`

  Divide la lista passata in una lista di liste, questo viene fatto sulla base

del parametro `cols_per_page`. Questa funzione risulta utile nel caso fosse necessario stampare un numero limitato di colonne per ogni pagina.

- `__get_date_format()`
  Restituisce una tupla che comprende:
  1. lo stile di presentazione delle date;
  2. la locale utilizzata.

- `__write_rows(fields: dict, tab, startrow: int, col_list: list, repeat_header: bool=True)`
  Stampa tante righe della tabella, quante definite dal valore associato alla chiave `rows_per_page` in configurazione. Ritorna poi un intero che indica l'indice dell'ultima riga stampata.

- `__get_paragraph_list(paragraphs: dict)`
  Dato come argomento il dizionario dei paragrafi presente in configurazione ritorna una tupla che comprende due liste di oggetti `Paragraph` (vedi sezione C.3.1). La prima lista rappresenta i paragrafi da stampare prima della tabella, la seconda quelli da stampare dopo di essa.

- `write_file(file: str)`
  Svolge l'attività di scrittura sul *file* specificato dal percorso passato come argomento.

- `export()`
  Genera un file temporaneo (libreria *tempfile*$_G$), scrive su di esso il documento Word e lo ritorna in formato `FileIO` (libreria *io*$_G$)

- `export_pdf_libreoffice(libre_office_bin_path: str="/opt/homebrew/bin/soffice", dir_path: str=None)`
  Genera una cartella temporanea (libreria *tempfile*$_G$) dentro la quale scrive il documento Word, poi avvia un sottoprocesso trammite il metodo `Popen` (libreria *subprocess*$_G$). Esso converte il *file* Word in PDF tramite

l'applicativo *LibreOffice*$_G$. Se il parametro `dir_path` è specificato crea un file PDF nella cartella specificata, altrimenti restituisce il documento in formato `FileIO` (libreria *io*$_G$). È possibile specificare il percorso dell'eseguibile di *LibreOffice*$_G$ tramite l'opportuno parametro.

Bibliografia

- Buttu M., *Programmare con Python. Guida Completa*, Edizioni Lswr, s.l., 2014

Sitografia

- Python 3.7:
  https://docs.python.org/3.7/
- Django 2.2:
  https://devdocs.io/django 2.2/
- Django REST Framework:
  https://www.django-rest-framework.org/
- XlsxWriter:
  https://xlsxwriter.readthedocs.io/
- python-docx:
  https://python-docx.readthedocs.io/en/latest/
- Docker:
  https://docs.docker.com/