AIESEC in Italia

Entity Financial Board

Case Study Tecnico e Manuale Architetturale

Digital Transformation in AIESEC

Architettura e Sviluppo di una Piattaforma Web per la Gestione Contrattuale e Finanziaria

Autore

Giona Mariani

LCVP F&L | AIESEC in Roma Tre 25.26

Implementation Manager | EFB 25.26

Anno 25.26

Ottobre 2025

Ringraziamenti

Questo progetto è il culmine di un viaggio lungo oltre nove mesi, un percorso fatto di intuizioni, sfide, notti insonni e continue evoluzioni. Un'opera di questa portata non nasce mai nel vuoto, ma è il risultato del sostegno, della fiducia e dell'ispirazione di molte persone a cui va la mia più profonda gratitudine.

Il mio primo pensiero va al mio comitato, AIESEC in Roma Tre, e a tutto l'Executive Board che mi ha accompagnato in questa avventura. Un ringraziamento speciale e sincero va alla mia Presidente, Beatrice Castelletti, per avermi non solo supportato in ogni fase, ma soprattutto "sopportato", con una pazienza e una fiducia che sono state il motore di questo progetto.

Questa webapp non esisterebbe senza l'ispirazione e il supporto ricevuto dal team MC Grinta (24.25). Un ringraziamento particolare va all'MCVP F&L Pietro Cottarelli che, con il suo esempio e la sua visione, mi ha fatto avvicinare e appassionare all'area Finance & Legal, ispirando la creazione del progetto. La mia profonda gratitudine va anche a Dafne Sagrati (MCVP Marketing) e a Roxana Ruiz Murcia (MCVP BD). Sono state loro a spingermi a credere nelle mie potenzialità e ad assumere ruoli a livello nazionale, offrendo una guida preziosa e sicura nei miei esordi in AIESEC.

La fase di sviluppo vera e propria è stata un'avventura condivisa. Ringrazio di cuore tutto l'EFB Team 25.1: in particolare, la mia gratitudine va a Giulia Romani (EFB Chair) e Alessandro Panetta (EFB Auditor). Sono stati i primi a credere in quest'idea, testando le versioni embrionali dell'applicazione con entusiasmo e fornendo feedback cruciali che l'hanno trasformata in ciò che è oggi.

Infine, un grazie enorme va a Matilde Zapparoli, MCVP Finance & Legal di MC Vetta (25.26), per avermi supportato e sopportato durante l'implementazione del sistema

e per aver creduto nel progetto fino alla sua adozione a livello nazionale.

Questo lavoro è dedicato a tutti gli AIESECer, e in particolare a ogni VP Finance & Legal. Spero possa rendere il vostro percorso un po' più semplice, così che possiate dedicare più tempo ed energie a ciò che conta davvero per la nostra organizzazione e la vostra crescita.

Introduzione

Le organizzazioni complesse e distribuite a livello globale, come AIESEC, fondano il proprio successo non solo sulla forza della loro visione, ma anche sull'efficienza dei loro processi operativi interni. In un contesto in cui le risorse sono limitate e prevalentemente volontarie, l'ottimizzazione dei flussi di lavoro amministrativi cessa di essere un semplice dettaglio tecnico per diventare un fattore strategico determinante per la sostenibilità e la crescita dell'organizzazione stessa.

Il presente elaborato documenta e analizza la progettazione, lo sviluppo e l'implementazione di una web application gestionale su misura, concepita per rispondere a una precisa esigenza emersa all'interno di AIESEC Italia: la necessità di superare le inefficienze e i rischi associati a una gestione documentale e amministrativa basata su procedure manuali e strumenti frammentati come i fogli di calcolo. L'analisi preliminare, infatti, ha evidenziato criticità sistemiche in termini di dispendio di tempo, mancanza di standardizzazione, rischio di errore umano e difficoltà nel tracciamento dei dati a livello nazionale.

Questo documento si propone quindi come un **case study** completo del progetto. Il suo scopo è duplice: da un lato, agire come **manuale tecnico** e guida pratica per i futuri manutentori e per gli utenti finali della piattaforma; dall'altro, offrire una riflessione analitica sull'intero ciclo di vita dello sviluppo software in un contesto reale, esaminando le scelte architetturali, le sfide tecniche affrontate e l'impatto finale della soluzione implementata.

L'opera è strutturata in otto capitoli. Il **primo capitolo** introduce il contesto organizzativo di AIESEC e analizza nel dettaglio le problematiche operative che hanno motivato il progetto. Il **secondo capitolo** è dedicato all'architettura software e all'in-

frastruttura di deployment, descrivendo lo stack tecnologico e le fondamenta del sistema. Il **terzo capitolo** costituisce il cuore dell'analisi funzionale, illustrando in dettaglio le capacità operative della piattaforma, dalla gestione dei contratti ai moduli di finanza e audit. Il **quarto capitolo** sposta l'attenzione sull'esperienza utente e sul design dell'interfaccia. Il **quinto capitolo** assume una prospettiva narrativa, raccontando le principali sfide tecniche incontrate e le soluzioni ingegneristiche adottate per superarle. Il **sesto capitolo** funge da guida pratica, con sezioni dedicate sia all'utente finale che allo sviluppatore. Infine, il **settimo capitolo** delinea una possibile roadmap per gli sviluppi futuri, mentre l' **ottavo capitolo** trae le conclusioni, valutando i risultati raggiunti e l'impatto del progetto sull'organizzazione.

Indice

Elenco delle figure

Capitolo 1

Il Contesto - Il "Perché" del Progetto

1.1 AIESEC: Una Panoramica

AIESEC si configura come la più grande organizzazione internazionale interamente gestita da giovani, con una presenza capillare in oltre 110 paesi e territori [3]. La sua missione fondamentale è lo sviluppo del potenziale umano e della **leadership** attraverso esperienze di scambio interculturale. Tali esperienze si articolano principalmente in programmi di tirocinio professionale (**Global Talent** e **Global Teacher**) e progetti di volontariato internazionale (**Global Volunteer**), che coinvolgono ogni anno migliaia di studenti universitari e neolaureati in tutto il mondo.

La struttura organizzativa è intrinsecamente complessa e decentralizzata. A livello globale, l'ente è coordinato da AIESEC International, mentre a livello nazionale le operazioni sono gestite da un comitato direttivo (es. **Member Committee - MC** per l'Italia). La vera forza propulsiva, tuttavia, risiede nella vasta rete di comitati locali (**Local Committees - LC**), tipicamente ospitati all'interno delle università [2]. Se da un lato questa struttura garantisce agilità e una profonda connessione con il tessuto giovanile locale, dall'altro introduce significative sfide di **coordinamento**, **standardizzazione** e **gestione amministrativa**. È proprio in questo scenario di elevata complessità operativa che emerge la necessità critica di adottare strumenti digitali avanzati, capaci

di supportare, ottimizzare e rendere scalabili i processi interni dell'organizzazione.

1.2 Lo Scenario Iniziale: Processi e Criticità

Prima dell'introduzione della webapp gestionale, l'ecosistema operativo di AIESEC Italia si fondava su un insieme di strumenti eterogenei e procedure prevalentemente manuali. Questo approccio, sebbene funzionale in contesti di scala ridotta, manifestava crescenti limiti in termini di **efficienza**, **scalabilità** e **controllo** all'aumentare della complessità e del volume delle attività. L'analisi dello scenario preesistente rivela tre aree di criticità interdipendenti.

1.2.1 La Gestione Manuale dei Documenti

Il ciclo di vita di un documento amministrativo, dal contratto per un partecipante (Exchange Participant - EP) a un verbale di comitato, rappresentava un processo a elevata intensità di lavoro e a forte rischio di errore. L'analisi procedurale di tale flusso evidenzia una sequenza di passaggi critici:

1. **Reperimento del template:** Il membro doveva individuare il modello corretto del documento (solitamente in formato '.docx') all'interno di repository condivisi, con il rischio concreto di utilizzare versioni obsolete o non conformi.
2. **Compilazione manuale:** I dati venivano inseriti manualmente, un'operazione che introduceva un'alta probabilità di errori di battitura, omissioni o compilazioni incomplete.
3. **Ciclo di firma e digitalizzazione:** Il documento doveva essere stampato, firmato a mano, scansionato per la digitalizzazione e infine salvato in formato PDF, un processo dispendioso in termini di tempo e risorse.
4. **Archiviazione:** Il file finale doveva essere nominato secondo convenzioni specifiche e archiviato manualmente, con frequenti inconsistenze che ne complicavano il successivo recupero e la consultazione.

Ciascun passaggio di questo flusso costituiva un potenziale collo di bottiglia, moltiplicando le ore di lavoro dedicate ad attività a basso valore aggiunto.

1.2.2 I Limiti dei Fogli di Calcolo Condivisi

Per il tracciamento dei dati anagrafici, contrattuali e finanziari, lo strumento de facto era il foglio di calcolo (es. Google Sheets). L'utilizzo di tale strumento come surrogato di un database relazionale introduceva, tuttavia, significative vulnerabilità sistemiche:

- **Integrità dei Dati:** L'assenza di meccanismi di validazione strutturata degli input portava a un "decadimento" della qualità del dato. Formati incoerenti, duplicazioni e errori di inserimento rendevano le analisi quantitative e i report intrinsecamente inaffidabili.
- **Scalabilità e Performance:** All'aumentare del volume di dati, i file diventavano estremamente lenti e difficilmente gestibili, compromettendo la produttività.
- **Rischio Operativo:** La possibilità di modifica o cancellazione accidentale di dati o formule da parte di utenti inesperti rappresentava un rischio costante per l'integrità dell'intero dataset, senza adeguati sistemi di controllo o log delle modifiche.

1.2.3 Mancanza di Tracciamento e Standardizzazione

La conseguenza strategica delle criticità operative descritte era l'assenza di una visione d'insieme, centralizzata e in tempo reale. Per il Comitato Nazionale (MC) risultava estremamente complesso monitorare lo stato delle operazioni dei Comitati Locali (LC) e garantire l'aderenza agli standard nazionali. Questa frammentazione informativa impediva di rispondere a domande gestionali fondamentali in modo rapido e accurato, trasformando i processi di **audit** e di **controllo di conformità** in attività onerose e reattive, anziché proattive. Mancava, in sintesi, una visione sistemica, fondamentale per il governo di un'organizzazione complessa.

1.3 La Visione: Obiettivi e Impatto Atteso della WebApp

A fronte delle criticità sistemiche emerse dall'analisi dello scenario, la formulazione di una soluzione ha richiesto la definizione di una visione strategica chiara. L'obiettivo non

era la semplice informatizzazione di un processo esistente, bensì la sua **reingegnerizzazione** attraverso la creazione di una piattaforma web centralizzata. Tale piattaforma doveva agire come **unica fonte di verità** (Single Source of Truth), un concetto cardine nell'ingegneria del software che mira a eliminare le ridondanze e le inconsistenze dei dati, garantendo che ogni elemento informativo sia memorizzato in un'unica, autorevole posizione [9].

La visione si è articolata in una serie di obiettivi strategici interconnessi. In primo luogo, l'introduzione di un elevato livello di **automazione** nella generazione documentale, per trasmutare un'attività manuale e dispendiosa in un processo guidato e istantaneo. A ciò si affiancava la **centralizzazione** dei dati in un database strutturato, superando la frammentazione e la vulnerabilità intrinseca dei fogli di calcolo. Parallelamente, si perseguiva la **standardizzazione** dei flussi di lavoro, imponendo procedure uniformi a livello nazionale per assicurare conformità e coerenza. Infine, la piattaforma doveva offrire strumenti di **tracciabilità** in tempo reale, fornendo dashboard capaci di trasformare i dati operativi grezzi in informazioni strategiche a supporto del processo decisionale, sia per i comitati locali che per l'ente nazionale.

L'impatto atteso di questa visione era la trasformazione del lavoro quotidiano dei membri, liberandoli dalle incombenze amministrative ripetitive per permettere loro di concentrarsi sulle attività a maggior valore aggiunto, in linea con la missione fondamentale di AIESEC.

1.4 Metodologia di Sviluppo Adottata

La scelta della metodologia di sviluppo rappresenta un fattore critico che influenza la capacità di un progetto di adattarsi al cambiamento e di fornire valore in modo efficace. I modelli di sviluppo software tradizionali, come il **modello a cascata** (Waterfall), si basano su una pianificazione rigida e su fasi sequenziali (analisi, progettazione, sviluppo, test), un approccio che si rivela poco efficiente in contesti ad alta incertezza o con requisiti in evoluzione come quello di AIESEC [**?**].

Per questo progetto è stato quindi adottato un approccio metodologico di tipo **Iterativo e Incrementale**, che si ispira ai principi della filosofia **Agile** [4]. Anziché mirare a un rilascio monolitico finale, lo sviluppo è progredito attraverso cicli brevi e ripetuti

(iterazioni). Ciascun ciclo ha prodotto un "incremento" funzionante del software, ovvero una versione dell'applicazione con un set di funzionalità testabili e potenzialmente rilasciabili. Questo paradigma si fonda su due pilastri concettuali:

- **Feedback continuo:** Il dialogo costante con gli utenti finali al termine di ogni iterazione ha permesso di validare le scelte implementative, correggere la rotta in corso d'opera e garantire che le funzionalità sviluppate rispondessero a esigenze reali e non solo a requisiti ipotizzati in fase di analisi.
- **Rilascio di valore incrementale:** Gli utenti hanno potuto beneficiare delle funzionalità chiave sin dalle prime fasi del progetto, favorendo l'adozione dello strumento e fornendo un valore tangibile e immediato all'organizzazione, senza dover attendere il completamento dell'intero ciclo di sviluppo.

Tale metodologia si è rivelata fondamentale per gestire la complessità e l'evoluzione dei requisiti, garantendo che il prodotto finale fosse non solo tecnicamente solido, ma anche e soprattutto allineato con le necessità operative degli utenti.

Capitolo 2

Architettura e Infrastruttura - Le "Fondamenta"

2.1 Panoramica dell'Architettura: Frontend, Backend, Database

L'architettura della web application è stata progettata seguendo un modello a tre livelli (three-tier architecture), un paradigma consolidato nell'ingegneria del software che separa la logica di presentazione, la logica di elaborazione e la gestione dei dati. Questa separazione garantisce modularità, manutenibilità e scalabilità al sistema. I tre livelli principali sono: il Frontend (Client-Side), il Backend (Server-Side) e il Data Layer (Persistence Layer).

- **Frontend (Livello di Presentazione):** Rappresenta tutto ciò con cui l'utente interagisce direttamente attraverso il browser. È responsabile della visualizzazione dei dati e della cattura degli input dell'utente. Questo livello è stato costruito utilizzando tecnologie web standard come **HTML5** per la struttura, **CSS3** per lo stile e **JavaScript** per l'interattività. Nello specifico, si è fatto uso del framework **Tailwind CSS** [20] per un rapido sviluppo di un'interfaccia moderna e responsiva, e di **Alpine.js** [15] per gestire le interazioni dinamiche in modo leggero e dichiarativo.

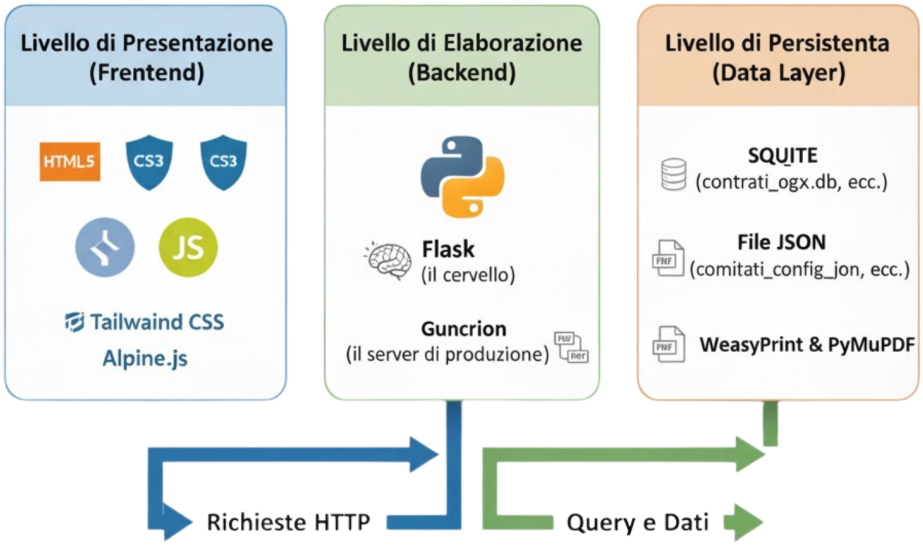

Figura 2.1: Diagramma dell'architettura a tre livelli della WebApp AIESEC.

- **Backend (Livello di Elaborazione):** Costituisce il "cervello" dell'applicazione. Risiede sul server e ha il compito di ricevere le richieste HTTP dal frontend, eseguire la logica di business (business logic), processare i dati e generare le risposte. Il cuore del backend è sviluppato in **Python**, utilizzando il micro-framework **Flask** [16]. Flask è stato scelto per la sua leggerezza, flessibilità e per il suo vasto ecosistema di estensioni, che lo rendono ideale per progetti che richiedono un controllo granulare senza le rigidità di framework più monolitici.

- **Data Layer (Livello di Persistenza):** È il livello responsabile della memorizzazione e del recupero dei dati in modo persistente. Per questo progetto, la scelta è ricaduta su **SQLite** [8], un motore di database SQL serverless, self-contained e transazionale. Essendo basato su un singolo file, SQLite elimina la necessità di un processo server separato, semplificando notevolmente la configurazione e il deployment. Questa soluzione si è rivelata ottimale per la scala del progetto, offrendo un eccellente compromesso tra performance, semplicità di gestione e affidabilità.

2.2 Lo Stack Tecnologico: Giustificazione delle Scelte

La selezione dello stack tecnologico è stata guidata dal principio di efficienza e pragmatismo, privilegiando strumenti consolidati, leggeri e adatti alla scala del progetto. L'analisi del file `requirements.txt` rivela un ecosistema di librerie coerente, scelto per massimizzare la velocità di sviluppo senza sacrificare la robustezza.

2.2.1 Backend e Logica di Business

Il cuore pulsante dell'applicazione è stato sviluppato in **Python**, un linguaggio apprezzato per la sua sintassi pulita e il suo vasto ecosistema. La scelta del web framework è ricaduta su **Flask** [16], un micro-framework che offre una solida base per la gestione delle richieste HTTP e il routing, lasciando allo sviluppatore piena libertà sulle componenti da integrare. Questa filosofia "minimale" ha permesso di costruire un'applicazione su misura, senza il peso di funzionalità superflue.

Per la complessa sfida della generazione di documenti PDF, è stato adottato un approccio duale di grande efficacia:

- **WeasyPrint** [11] è stato utilizzato per la conversione iniziale di template HTML e CSS in documenti PDF. Questo permette di definire il layout dei documenti utilizzando tecnologie web standard, separando nettamente la struttura (HTML) dalla presentazione (CSS).
- **PyMuPDF** [13] è stato impiegato per le manipolazioni avanzate post-generazione, come l'aggiunta dinamica di loghi, timbri e firme in coordinate precise, o la fusione di più documenti PDF in un unico file.

2.2.2 Frontend e Rendering

Sebbene gran parte della logica sia gestita lato server, l'interfaccia utente è stata curata per essere moderna e reattiva. Il rendering delle pagine HTML avviene attraverso il motore di templating **Jinja2** [18], perfettamente integrato con Flask. Per lo stile, la scelta di **Tailwind CSS** [20] ha permesso di applicare uno stile consistente e professionale attraverso un sistema di classi di utilità, accelerando notevolmente lo sviluppo della UI.

L'interattività lato client è stata affidata ad **Alpine.js** [15], una libreria minimalista che consente di aggiungere comportamento dinamico direttamente nel markup HTML, evitando la complessità di framework JavaScript più pesanti come React o Vue.js, non necessari per questo progetto.

2.2.3 Data Layer e Gestione Dati

Per la persistenza dei dati, la scelta di **SQLite** [8] si è rivelata strategica. La sua natura *serverless*, basata su file, ha semplificato il deployment e la manutenzione, offrendo al contempo tutte le funzionalità di un database SQL transazionale necessarie per l'applicazione. Per operazioni di analisi e esportazione dei dati, come la generazione di report in formato Excel, è stata utilizzata la libreria **Pandas**, uno standard de facto nell'ecosistema Python per la manipolazione di dati tabulari.

2.2.4 Servizi e Operazioni in Background

Per gestire compiti schedulati e non bloccanti, è stato integrato **APScheduler**. Come visibile nel file `app.py`, questo strumento gestisce operazioni cruciali come la pulizia periodica dei file temporanei e la sincronizzazione notturna dei dati con Google Sheets. In un ambiente di produzione, l'applicazione viene servita tramite **Gunicorn** [5], un server WSGI robusto e performante, che si interfaccia con un web server come Nginx per gestire il traffico in modo efficiente.

2.3 Il Cuore del Backend: Struttura a Blueprint in Flask

Per evitare la creazione di un'applicazione monolitica – un singolo e massiccio file `app.py` difficile da navigare e manutenere – è stato adottato il pattern architetturale dei **Blueprint**, una delle funzionalità più potenti di Flask [19]. Un Blueprint è un componente che permette di organizzare un'applicazione Flask in moduli più piccoli, indipendenti e riutilizzabili. Ogni Blueprint può definire le proprie rotte (routes), template, file statici e logica di business, incapsulando una specifica area funzionale dell'applicazione.

L'analisi della struttura del progetto e del file `app.py` mostra una chiara implementazione di questo pattern. Invece di definire tutte le rotte nel file principale, l'applicazione è stata scomposta in diversi Blueprint, ciascuno dedicato a un macro-processo:

- `ep_contracts`: Gestisce l'intero flusso dei contratti per gli Exchange Participant (EP) dei programmi oGX. Contiene la logica per i form multi-step, la generazione dei PDF contrattuali e la gestione del loro stato nel database.
- `ep_membership`: Un modulo specializzato per il processo di adesione a socio.
- `igv_contracts` e `igv_ep_contracts`: Blueprint dedicati alla gestione delle diverse tipologie di contratti per il programma iGV (Incoming Global Volunteer), separando la logica per i partner (Enabler) da quella per i partecipanti.
- `protocol_manager`: Un Blueprint di servizio, cruciale per la gestione centralizzata e la generazione dei numeri di protocollo univoci per tutti i documenti, garantendo consistenza e prevenendo conflitti.

Questi moduli vengono poi "registrati" nell'applicazione principale, come evidenziato dalle chiamate `app.register_blueprint(...)` in `app.py`. Questa architettura modulare non solo migliora l'organizzazione logica del codice sorgente, ma facilita anche lo sviluppo parallelo, il debugging e la futura estensione dell'applicazione con nuove funzionalità, che possono essere sviluppate come nuovi Blueprint isolati senza interferire con il resto del sistema.

2.4 Gestione dei Dati: Database SQLite e File di Configurazione JSON

Una corretta strategia di persistenza dei dati è fondamentale per la stabilità e l'integrità di qualsiasi applicazione. L'architettura di questo progetto adotta un approccio duale, distinguendo nettamente tra i dati operativi dinamici e i dati di configurazione semi-statici. Per queste due diverse esigenze, sono state scelte due tecnologie complementari: un database relazionale SQLite e file di configurazione in formato JSON.

2.4.1 Database Relazionale con SQLite

Per la gestione dei dati operativi e transazionali — come i dettagli dei contratti, le anagrafiche dei partecipanti e lo storico delle adesioni — è stato impiegato il motore di database **SQLite** [8]. Come evidenziato dalla presenza di file come `contratti_ogx.db` e `igv_contracts.db` nella directory principale del progetto, ogni macro-area funzionale tende a utilizzare un proprio database dedicato, favorendo l'isolamento dei dati.

La scelta di SQLite è strategica per diverse ragioni. In primo luogo, la sua natura **serverless** (non richiede un processo server separato) semplifica drasticamente l'ambiente di deployment e riduce i costi di manutenzione, un fattore non trascurabile in un contesto gestito da volontari. In secondo luogo, pur essendo leggero, SQLite è un motore di database **ACID-compliant**, garantendo l'affidabilità delle transazioni. Le funzioni di inizializzazione, come `init_db()` richiamate all'avvio nel file `app.py`, assicurano che lo schema del database sia sempre presente e consistente, rendendo l'applicazione robusta e auto-configurante al primo avvio.

2.4.2 File di Configurazione JSON

Per i dati di configurazione dell'applicazione, ovvero quelle informazioni che cambiano di rado e che spesso necessitano di essere leggibili o modificabili dall'amministratore, è stato scelto il formato **JSON** (JavaScript Object Notation). Il file più rappresentativo di questo approccio è `comitati_config.json`, che agisce come un registro centrale per tutti i comitati. Contiene informazioni semi-statiche come il nome del comitato, il PIN di accesso, i dati del presidente e del VP Finance, le impostazioni SMTP e altri parametri specifici.

L'utilizzo di JSON per questi scopi offre notevoli vantaggi:

- **Leggibilità Umana:** Il formato è intuitivo e facilmente ispezionabile, semplificando il debug e le modifiche manuali in caso di necessità.
- **Interoperabilità:** Essendo uno standard de facto per lo scambio di dati sul web, è gestito nativamente da Python (attraverso la libreria `json`) e da JavaScript, rendendo semplice la comunicazione tra backend e frontend.

- **Flessibilità dello Schema:** Permette di aggiungere nuove chiavi di configurazione senza la necessità di migrazioni strutturate del database, offrendo grande agilità durante lo sviluppo.

Le funzioni di utilità come `load_json()` e `save_json()`, presenti nel codice, astrae l'accesso a questi file, centralizzando la logica di lettura e scrittura.

2.5 Integrazioni con Servizi Esterni

Per arricchire le funzionalità e automatizzare i flussi di lavoro, l'applicazione si integra con due servizi esterni fondamentali: le API di Google Sheets per la reportistica e il protocollo SMTP per le notifiche email. Queste integrazioni estendono le capacità del sistema, connettendolo a piattaforme standard e consolidate.

2.5.1 Google Sheets API per Audit e Reportistica

Una delle integrazioni più strategiche è quella con **Google Sheets**. Questa connessione trasforma un semplice foglio di calcolo in un'estensione dinamica della webapp, utilizzata principalmente per i processi di **audit** e per la sincronizzazione dei dati finanziari. L'interazione con le API di Google è gestita attraverso la libreria Python **gspread** [6], come specificato nel file `requirements.txt`.

L'autenticazione avviene tramite un **Service Account**, una modalità sicura che permette all'applicazione di accedere ai dati di Google Sheets per conto proprio, senza l'intervento manuale di un utente. Le credenziali sono archiviate in modo sicuro nel file `google_credentials.json`. Come si evince dalle funzioni `refresh_audit_cache()` e `scheduled_sheets_sync()` presenti in `app.py`, questa integrazione è cruciale per le operazioni schedulate in background, permettendo all'applicazione di leggere e scrivere dati in modo asincrono per mantenere aggiornati i report e le dashboard di controllo.

2.5.2 Protocollo SMTP per le Notifiche Email

Per automatizzare le comunicazioni, la webapp implementa un client di posta elettronica basato sul **Simple Mail Transfer Protocol (SMTP)** [10], lo standard universale per la trasmissione di email. Questa funzionalità, sebbene non ancora pienamente sfruttata

in tutte le sue potenzialità, pone le basi per l'invio di notifiche automatiche, come ad esempio i promemoria per i contratti in scadenza o le conferme di avvenuta generazione di un documento.

L'architettura è stata progettata per essere flessibile: le credenziali e i parametri del server SMTP non sono codificati nell'applicazione, ma sono memorizzati all'interno del file `comitati_config.json`, in un dizionario dedicato `smtp_settings` per ciascun comitato. Questo design permette a ogni comitato locale di configurare e utilizzare il proprio server di posta, garantendo una maggiore decentralizzazione e personalizzazione del servizio di notifica.

2.6 Infrastruttura di Deployment

L'architettura software di un'applicazione è incompleta senza un'adeguata infrastruttura che ne garantisca l'esecuzione in un ambiente di produzione stabile, sicuro e performante. Per questa webapp, è stato configurato uno stack di deployment basato su componenti open-source consolidati, ospitato su un Virtual Private Server (VPS).

2.6.1 Virtual Private Server e Sistema Operativo Ubuntu

L'applicazione è ospitata su un **VPS**, una soluzione che offre un ambiente server isolato con accesso root completo, garantendo un controllo granulare sulla configurazione del sistema a un costo contenuto. Come sistema operativo è stato scelto **Ubuntu Server** [7], una delle distribuzioni Linux più diffuse e supportate per il web hosting. La sua stabilità, l'ampia documentazione, la vasta community e il robusto sistema di gestione dei pacchetti (APT) ne fanno una base solida e affidabile per l'esecuzione di servizi web.

2.6.2 Web Server Nginx e Application Server Gunicorn

Per servire l'applicazione Flask al pubblico, è stata implementata una configurazione standard basata su un'accoppiata di server, ciascuno con un ruolo specifico:

- **Gunicorn** [5] agisce come **Application Server WSGI** (Web Server Gateway Interface). Il suo compito è eseguire l'applicazione Python, gestire un pool di

processi "worker" per poter rispondere a più richieste contemporaneamente e fare da interfaccia tra il web server e l'applicazione Flask.

- **Nginx** [14] funge da **Web Server** e **Reverse Proxy**. Posizionato "davanti" a Gunicorn, Nginx riceve tutte le richieste in ingresso dal mondo esterno. Gestisce in modo estremamente efficiente la distribuzione dei file statici (CSS, JavaScript, immagini), alleggerendo il carico sull'applicazione Python. Inoltre, inoltra (proxy pass) le richieste dinamiche a Gunicorn e può essere configurato per gestire la terminazione SSL (HTTPS), la compressione e il caching, migliorando performance e sicurezza.

Questa architettura a due server è uno standard de facto per il deployment di applicazioni web Python, poiché combina l'efficienza di Nginx nella gestione del traffico con la capacità di Gunicorn di eseguire il codice applicativo in modo robusto.

2.6.3 Gestione del Servizio con Systemd

Per garantire che l'applicazione sia in esecuzione continua e si riavvii automaticamente in caso di crash o al riavvio del server, il processo Gunicorn è gestito come un servizio di sistema tramite **Systemd**. Systemd è l'init system e service manager standard per la maggior parte delle distribuzioni Linux moderne, inclusa Ubuntu.

È stato creato un file di servizio (*unit file*) personalizzato, solitamente collocato in `/etc/systemd/system/`, che definisce come avviare, fermare e gestire il processo Gunicorn. Questo approccio trasforma l'applicazione da un semplice script da lanciare manualmente a un vero e proprio **servizio (daemon)** integrato nel sistema operativo, garantendo alta affidabilità (*high availability*) e una gestione centralizzata dei log di esecuzione.

Capitolo 3

Il Cuore dell'Applicazione - Le Funzionalità Principali

Dopo aver analizzato l'architettura e l'infrastruttura del sistema, questo capitolo si addentra nel dominio funzionale dell'applicazione, descrivendo in dettaglio le sue capacità operative. L'analisi procederà per moduli logici, partendo dal sistema di accesso fino ai flussi di lavoro più complessi.

3.1 Autenticazione e Gestione dei Comitati

Il primo punto di contatto con l'applicazione è il suo sistema di autenticazione, progettato per essere al contempo sicuro e flessibile, adattandosi alla struttura organizzativa di AIESEC. Questo modulo non solo protegge l'accesso alle risorse, ma definisce anche i permessi e le configurazioni specifiche per ogni comitato.

3.1.1 Flusso di Login, Sessioni e Modalità Demo

Il meccanismo di autenticazione si discosta dai tradizionali sistemi basati su database di utenti. L'accesso è invece governato da una combinazione di una **chiave comitato** (selezionata da un menu a tendina) e un **PIN** numerico associato. Le credenziali valide sono definite all'interno del file di configurazione `comitati_config.json`, una scelta

che semplifica la gestione degli accessi per un numero limitato e stabile di entità come i comitati AIESEC.

Al momento di un login valido, la funzione `_create_session()` in `app.py` si occupa di istanziare la sessione utente. La sessione di Flask è un meccanismo che permette di conservare informazioni tra le richieste di un utente, tipicamente utilizzando cookie firmati crittograficamente. Nel contesto dell'applicazione, la sessione viene popolata con un dizionario contenente tutte le informazioni utili del comitato (nome, codice, stato di expansion, etc.), escludendo per sicurezza i dati sensibili come il PIN. È stata inoltre implementata una rotta `/login/demo`, che consente l'accesso tramite un utente fittizio "test", ideale per scopi di dimostrazione e debugging senza interagire con i dati di produzione.

3.1.2 Amministrazione e Controllo degli Accessi Basato sui Ruoli

L'applicazione implementa un semplice ma efficace sistema di controllo degli accessi basato sui ruoli (Role-Based Access Control - RBAC). I ruoli sono implicitamente definiti dalle proprietà del comitato nella sessione:

- **Utente Standard (LC):** Un comitato locale o un'expansion, con accesso alle funzionalità operative standard.
- **Amministratore (MC):** Il comitato nazionale (`mc_italy`), identificato dal flag `is_admin` nella sessione, che ha accesso a pannelli di controllo aggiuntivi, statistiche aggregate e funzionalità di gestione degli altri comitati.

La protezione delle rotte è garantita dall'uso di **decoratori** Python, come `@login_required` e `@admin_required`. Questi decoratori, definiti nel modulo `auth.py` e basati sulle funzionalità offerte da librerie come **Flask-Login** [12], agiscono come "guardiani" per le diverse sezioni dell'applicazione. Prima di eseguire la logica di una rotta, il decoratore verifica se l'utente in sessione soddisfa i requisiti di autenticazione e di permesso, reindirizzandolo alla pagina di login in caso contrario. Questo approccio è uno standard del settore per la messa in sicurezza di endpoint web.

3.1.3 Gestione delle Impostazioni di Comitato

Per conferire autonomia ai singoli comitati, è stata creata una sezione dedicata `/impostazioni`. Da questa interfaccia, gli utenti autorizzati possono modificare dinamicamente una serie di parametri relativi al proprio comitato, che vengono poi persistiti nel file `comitati_config.json`. Le funzionalità di gestione includono:

- **Modifica Dati Anagrafici:** Aggiornamento dei nomi del Presidente e del VP Finance, IBAN, indirizzi e altri dati identificativi.
- **Gestione Credenziali:** Possibilità di modificare il PIN di accesso del proprio comitato.
- **Upload di File:** Caricamento delle immagini delle firme del Presidente e del VP Finance, e del timbro del comitato. Questi file vengono salvati in una cartella dedicata e associata al comitato, pronti per essere impressi dinamicamente sui PDF.
- **Regolazione dei Progressivi:** Possibilità per gli amministratori di visualizzare e, se necessario, modificare manualmente i contatori dei numeri di protocollo, gestiti dal Blueprint `protocol_manager`.

Questo sistema di auto-gestione riduce il carico di lavoro per l'amministratore di sistema e responsabilizza i comitati locali sulla correttezza delle proprie informazioni.

3.2 Il Flusso Unificato di Gestione Contratti

Il modulo di gestione dei contratti rappresenta la funzionalità principale (core feature) della piattaforma, progettato per sostituire interamente il processo manuale, frammentato e prono a errori. La soluzione implementata è un **flusso di lavoro unificato e guidato**, che accompagna l'utente dalla selezione del tipo di contratto fino alla sua archiviazione finale, garantendo coerenza, integrità dei dati e un'esperienza utente ottimale.

3.2.1 L'Interfaccia Guidata Multi-step

Dal punto di vista della User Experience (UX), la sfida principale era gestire la raccolta di una grande quantità di informazioni eterogenee senza sopraffare l'utente. La soluzione adottata è stata l'implementazione di un'interfaccia **multi-step**. Anziché presentare un unico, interminabile modulo, il processo di inserimento dati è stato scomposto in più passaggi (step) logici e sequenziali. Ogni step raggruppa campi omogenei (es. "Anagrafica Partecipante", "Dettagli dell'Esperienza", "Contatti di Emergenza").

Questo pattern di progettazione offre due vantaggi principali: primo, riduce il **carico cognitivo** dell'utente, che può concentrarsi su un numero limitato di informazioni alla volta; secondo, crea un senso di progressione e guida l'utente verso il completamento del task. Come emerge dall'analisi del codice, questo approccio è stato applicato in modo **unificato** a tutte le principali tipologie di contratto, inclusi i contratti per i programmi oGX, iGV Enabler e iGV EP, creando un'esperienza utente consistente in tutta l'applicazione.

3.2.2 Logica di Salvataggio Progressivo e Validazione Dati

A supporto dell'interfaccia multi-step, il backend implementa una logica di **salvataggio progressivo**. Al completamento di ogni passaggio, i dati inseriti vengono inviati al server tramite una richiesta HTTP e salvati temporaneamente nel database SQLite, associati a un identificatore univoco del contratto. Questo meccanismo garantisce che nessun dato vada perso nel caso in cui l'utente interrompa la compilazione per poi riprenderla in un secondo momento.

Contestualmente al salvataggio, il backend esegue una **validazione server-side** dei dati ricevuti. Vengono controllati i formati (es. validità di un'email o di un codice fiscale), la completezza dei campi obbligatori e la coerenza delle informazioni. Se la validazione fallisce, l'utente non può procedere allo step successivo e riceve un feedback puntuale sui campi da correggere. Questo approccio garantisce l'integrità e la qualità dei dati fin dalla fase di inserimento, prevenendo la registrazione di informazioni errate o incomplete nel sistema.

3.2.3 Generazione, Archiviazione e Tracciamento dello Stato

Una volta completato l'ultimo step del form, il sistema avvia il processo di finalizzazione. Il backend recupera tutti i dati salvati progressivamente e invoca il **"Compilatore"** Python specifico per quella tipologia di contratto. Come analizzato nel Capitolo 2, questo modulo specializzato utilizza i dati per popolare un template HTML con Jinja2, che viene poi convertito in un documento PDF finale tramite WeasyPrint e PyMuPDF.

Il contratto generato viene archiviato in due modi:

1. Il **file PDF** viene salvato sul file system del server, in una directory protetta e associata all'identificativo univoco del contratto.

2. I **dati strutturati** del contratto vengono salvati in modo permanente nel database SQLite, e lo stato del contratto viene aggiornato (es. da "Bozza" a "Generato").

L'applicazione gestisce il ciclo di vita del contratto attraverso una semplice ma efficace **macchina a stati** (State Machine). Ogni contratto può trovarsi in diversi stati (es. `Bozza`, `Inviato all'EP`, `In Revisione`, `Approvato`), permettendo un tracciamento puntuale e trasparente dell'intero processo.

3.2.4 La Dashboard di Gestione Contratti

Per i membri AIESEC, il punto di accesso principale per il monitoraggio è la **dashboard di gestione contratti**. Questa interfaccia centralizzata offre una visione aggregata di tutti i contratti e fornisce strumenti per la loro gestione. Le funzionalità chiave, implementate nelle rotte amministrative, includono:

- **Visualizzazione e Ricerca:** Una vista tabellare che mostra le informazioni salienti di ogni contratto, dotata di potenti filtri per comitato, stato, tipo di contratto e range di date, oltre a una funzionalità di ricerca testuale libera.

- **Azioni rapide:** Per ogni contratto, sono disponibili azioni contestuali come il download del PDF generato o la visualizzazione di una pagina di dettaglio con tutti i dati inseriti.

- **Esportazione Dati:** È presente una funzionalità di esportazione dei dati filtrati nei formati **CSV** e **Excel**. Questa operazione, come evidenziato dalla presenza della libreria nel `requirements.txt`, è gestita in modo robusto dalla libreria **Pandas**, che costruisce un DataFrame in memoria e lo serializza nel formato richiesto, pronto per analisi esterne.

Questa dashboard trasforma i dati operativi in uno strumento di controllo e management per i responsabili dei comitati.

3.3 La Generazione Documentale "Statica"

Oltre al flusso multi-step per i contratti complessi, la piattaforma offre un meccanismo di generazione più snello per documenti che richiedono un numero limitato di input. Questa modalità, definita "statica" in contrapposizione a quella "progressiva", si basa su form a pagina singola ed è utilizzata per una vasta gamma di documenti amministrativi, in particolare i verbali di comitato.

3.3.1 Verbali e Altri Documenti Amministrativi

La generazione di un verbale (es. approvazione nuovi soci, elezioni, budget) segue un flusso operativo semplificato. L'utente seleziona il tipo di documento desiderato da un'interfaccia dedicata (es. `/verbali/seleziona`), viene reindirizzato a un form web a pagina singola e, una volta inseriti i dati necessari, avvia la generazione.

A livello di backend, il sistema gestisce questa logica in modo centralizzato. La rotta di generazione, come `/documenti/genera/singolo`, riceve i dati del form e una chiave identificativa del documento (`doc_key`). Utilizzando una mappatura definita nella variabile `COMPILER_MAP` in `app.py`, il sistema invoca dinamicamente il modulo Python corretto, situato nella directory `/compilatori`. Questo modulo esegue la consueta logica di rendering del template e di generazione del PDF. Un caso di particolare interesse è quello dei verbali elettivi (es. `v_election_eb`), che richiedono l'upload di allegati come le schede dei voti. Per questi, viene utilizzata una rotta dedicata, `/documenti/genera/con_upload`, che gestisce la ricezione dei file e li pas-

sa al compilatore specifico per l'inclusione nel PDF finale, dimostrando la flessibilità dell'architettura.

3.3.2 Generazione di Bundle Documentali

Per ottimizzare ulteriormente i processi che richiedono la produzione di più documenti correlati (es. l'onboarding di un nuovo membro, che necessita di Contratto, Modulo di Adesione e Verbale di Approvazione), è stata implementata la funzionalità di **generazione di bundle**. Questa feature permette all'utente di compilare un unico

3.4 Modulo Avanzato: Accountancy e Finanza

Oltre alla gestione contrattuale, l'applicazione integra un sofisticato modulo dedicato alla contabilità (Accountancy) e all'analisi finanziaria. Questo sistema estende le capacità della piattaforma da semplice generatore di documenti a strumento di **governance finanziaria**, fornendo ai comitati gli strumenti per standardizzare la rendicontazione e monitorare la propria salute economica.

3.4.1 Generazione di Documenti Contabili

Il modulo automatizza la creazione di documenti contabili complessi, come gli **Estratti Conto** e le **Fatture di Costo**. Il flusso di generazione per questi documenti si distingue per la sua interattività e per le capacità avanzate di manipolazione dei file. L'interfaccia utente, sebbene basata su un form a pagina singola, include elementi dinamici come tabelle in cui l'utente può aggiungere righe per inserire più movimenti o fatture.

La caratteristica tecnicamente più rilevante di questo processo è la capacità di **fondere documenti PDF**. L'utente, oltre a inserire i dati nei form, carica anche i documenti originali (es. l'estratto conto bancario ufficiale o le fatture dei fornitori in formato PDF). Il backend esegue quindi un'operazione di "stitching" (cucitura) documentale:

1. Utilizza **WeasyPrint** per generare un frontespizio standardizzato e delle pagine di riepilogo tabellare, basate sui dati inseriti dall'utente.

2. Utilizza **PyMuPDF** per unire in un unico file il PDF appena generato con i PDF originali caricati dall'utente.

Il risultato è un documento unico, completo e professionale, che combina dati strutturati e fonti originali, garantendo massima trasparenza e conformità per i processi di audit. Durante la generazione, il frontend mostra un overlay di caricamento, fornendo un'esperienza utente fluida e asincrona.

3.4.2 La Dashboard Finanza: KPI e Metriche

La piattaforma integra una **dashboard finanziaria** che agisce come un cruscotto di Business Intelligence (BI) per il monitoraggio delle performance economiche. A differenza di altre sezioni, i dati visualizzati qui non risiedono nel database primario dell'applicazione, ma vengono attinti da un foglio di calcolo Google Sheets che funge da fonte dati principale.

Per garantire performance ottimali e non dipendere da chiamate API sincrone a ogni caricamento di pagina, è stato implementato un meccanismo di **caching**. Un processo in background, gestito da APScheduler, interroga periodicamente Google Sheets tramite la libreria `gspread` [6] e salva una copia dei dati in un file locale, `financial_cache.json`. Le dashboard leggono i dati da questo file di cache, garantendo tempi di caricamento istantanei.

Vengono monitorati e visualizzati Key Performance Indicators (KPI) finanziari essenziali per la gestione di un'entità AIESEC, tra cui: **Liquidità** (Liquidity), **Profit & Loss** (PnL), **Months of Financial Reserves** (MoFR) e il rapporto tra **Spese Eseguite e Budget Previsto** (Executed/Budgeted ratio). L'applicazione fornisce viste differenziate: una per i Comitati Locali (LC) con i propri dati specifici, e una aggregata per il Comitato Nazionale (MC).

3.4.3 Sincronizzazione Dati con Google Sheets

L'integrazione con Google Sheets è bidirezionale e orchestrata con attenzione. Come descritto, la piattaforma *consuma* dati dal foglio di calcolo per alimentare le dashboard.

Al contempo, altre parti dell'applicazione (come il modulo di audit) sono in grado di *scrivere* dati sul foglio, che agisce quindi come un hub dati centrale.

La sincronizzazione dei dati è gestita tramite due meccanismi:

- **Automatica e Schedulata:** Un task gestito da **APScheduler**, visibile in `app.py`, esegue un "pull" dei dati finanziari da Google Sheets a intervalli regolari (es. ogni notte), aggiornando il file `financial_cache.json`.
- **Manuale On-Demand:** Gli amministratori dispongono di una funzionalità nel pannello di controllo che permette di forzare un aggiornamento immediato della cache. Questa opzione è vitale quando sono state apportate modifiche significative al foglio di calcolo sorgente e si desidera che i cambiamenti siano immediatamente visibili sulla piattaforma.

Questo sistema ibrido garantisce che i dati siano al contempo aggiornati e performanti, combinando l'efficienza di una cache locale con la flessibilità di un data source collaborativo basato su cloud.

3.5 Modulo Avanzato: Audit e Compliance

Per rispondere alla necessità del Comitato Nazionale (MC) di monitorare la conformità e l'aderenza agli standard operativi da parte dei Comitati Locali (LC), è stato sviluppato un modulo di **Audit**. Questo sistema automatizza la visualizzazione dei risultati del processo di audit mensile, trasformando i dati presenti in un foglio di calcolo in un'interfaccia di controllo chiara e accessibile direttamente dalla piattaforma.

L'architettura di questo modulo ricalca quella del sistema finanziario, basandosi su un'integrazione con un foglio di lavoro Google Sheets specifico, denominato **'AppCheckAudit'**, che funge da unica fonte di verità. Anche in questo caso, per ottimizzare le performance, è stato implementato un sistema di **caching**. Una funzione dedicata, `refresh_audit_cache()`, si occupa di interrogare periodicamente il foglio di calcolo tramite `gspread` [6] e di salvare i dati elaborati in un file cache locale, `audit_data_cache.json`.

I dati di audit vengono poi presentati all'utente attraverso due interfacce distinte, a seconda del ruolo:

- **Interfaccia Utente (LC):** All'interno della dashboard principale, ogni comitato locale visualizza un widget dedicato che mostra esclusivamente il proprio stato di audit per il mese corrente. Vengono presentati i punteggi parziali (es. *Legalities*, *Accountancy*), il punteggio totale e uno stato di conformità chiaramente identificabile tramite testo e colori (es. "Passato al primo check" in verde, "Non Passato" in rosso). Questo fornisce un feedback immediato e inequivocabile al comitato sulla propria performance.

- **Interfaccia Amministratore (MC):** Il pannello di controllo dell'amministratore offre una visione aggregata dell'intero processo di audit nazionale. Vengono mostrate statistiche riassuntive, come il numero totale di comitati, le percentuali di conformità al primo e al secondo check del mese, e altri indicatori chiave. Questo permette al Comitato Nazionale di avere un controllo strategico sull'andamento dell'intera rete e di identificare tempestivamente le aree di criticità.

Anche per questo modulo, è prevista una funzionalità che permette agli amministratori di forzare un aggiornamento manuale della cache (`/admin/force_refresh_audit_cache`), garantendo l'accesso ai dati più recenti in qualsiasi momento.

Capitolo 4

User Experience e Design dell'Interfaccia

La validità di una soluzione software non si misura unicamente dalla sua correttezza funzionale, ma anche dalla qualità dell'interazione con l'utente finale. Un'interfaccia complessa o poco intuitiva può vanificare i benefici di un backend potente, portando a frustrazione e a un basso tasso di adozione. Per questa ragione, grande attenzione è stata posta alla progettazione dell'esperienza utente (User Experience - UX) e dell'interfaccia utente (User Interface - UI). Questo capitolo analizza i principi di design e i componenti chiave che definiscono l'interazione con la piattaforma.

4.1 Filosofia di Design: Semplicità e Reattività

La filosofia di design che ha guidato lo sviluppo dell'interfaccia si fonda su due principi cardine: **semplicità** e **reattività**. L'obiettivo primario era creare uno strumento che apparisse immediatamente familiare e accessibile anche a utenti con scarse competenze tecniche, riducendo al minimo la curva di apprendimento.

Il principio di **semplicità** si è tradotto in un'interfaccia pulita, minimale e priva di elementi superflui. La navigazione è stata strutturata in modo gerarchico e prevedibile, e ogni schermata è stata progettata per presentare solo le informazioni e le azioni necessarie per il contesto specifico, evitando di sovraccaricare l'utente.

Il principio di **reattività** (o *responsiveness*) garantisce che l'applicazione sia pienamente utilizzabile su un'ampia gamma di dispositivi, dai monitor desktop ai tablet e agli smartphone. Il layout di ogni pagina è stato costruito secondo un approccio *mobile-first*, adattandosi fluidamente alle dimensioni dello schermo. Questa caratteristica è fondamentale per un'organizzazione dinamica come AIESEC, i cui membri accedono agli strumenti da dispositivi diversi e in contesti differenti. La rapida implementazione di un design responsive è stata resa possibile dall'adozione del framework **Tailwind CSS** [20], il cui approccio basato su classi di utilità si è rivelato ideale per costruire interfacce complesse e reattive in modo efficiente.

4.2 Layout Responsive per Desktop e Mobile

La realizzazione pratica di un'interfaccia reattiva si basa su un insieme di tecniche di CSS e su una progettazione strutturale dell'HTML pensata per la flessibilità. L'approccio adottato in questo progetto sfrutta appieno le potenzialità del framework **Tailwind CSS** [20] per creare un'esperienza utente fluida e consistente su qualsiasi dispositivo.

La strategia di responsiveness si articola su tre pilastri principali:

1. **Sistema a Griglia Flessibile (Flexbox & Grid):** Anziché utilizzare un sistema a griglia predefinito, il layout di ogni pagina è costruito utilizzando le utility di **Flexbox** e **CSS Grid**. Questo permette un controllo granulare sul posizionamento e l'allineamento degli elementi. Ad esempio, una dashboard che su desktop presenta una sidebar laterale e un'area di contenuto principale, su schermi più piccoli riorganizza automaticamente questi blocchi, impilandoli verticalmente per una migliore leggibilità.

2. **Breakpoint Schedulati:** Il cambiamento del layout avviene in corrispondenza di specifici punti di interruzione (*breakpoints*), ovvero larghezze dello schermo predefinite. Tailwind CSS fornisce una serie di breakpoint standard (`sm`, `md`, `lg`, `xl`) che vengono utilizzati per applicare stili condizionali. Per esempio, una serie di card disposte in tre colonne su schermi grandi (`lg:grid-cols-3`) possono diventare due colonne su tablet (`md:grid-cols-2`) e una singola colonna su mobile (comportamento di default).

3. **Componenti Adattivi:** Ogni componente dell'interfaccia, dalle barre di navigazione alle tabelle di dati, è stato progettato per essere intrinsecamente responsivo. La barra di navigazione principale, ad esempio, si trasforma da un menu orizzontale esteso su desktop a un menu compatto, accessibile tramite un'icona "hamburger", su mobile. Le tabelle di dati, che su schermi larghi mostrano tutte le loro colonne, su schermi stretti diventano scorrevoli orizzontalmente per evitare di comprimere il contenuto o di rompere il layout della pagina.

Questo approccio sistematico al design responsivo garantisce che l'usabilità dell'applicazione rimanga elevata, indipendentemente dal dispositivo utilizzato dall'utente finale per accedervi.

4.3 Componenti Interattivi Chiave

Oltre al layout generale, l'esperienza utente è definita dalla qualità dei singoli componenti con cui l'utente interagisce per inserire dati e compiere azioni. Sono stati sviluppati componenti specifici per guidare l'utente, fornire feedback chiari e semplificare le operazioni complesse.

4.3.1 Il Form Multi-step: Progettazione e Logica

Come anticipato nel Capitolo 3, il form multi-step è il componente centrale per la gestione dei contratti. Dal punto di vista del design dell'interfaccia, la sua implementazione include elementi specifici volti a migliorare l'usabilità:

- **Indicatore di Progressione:** In cima alla pagina, una barra di stato o un elenco di passaggi evidenzia lo step corrente e il numero totale di step. Questo funge da "mappa" per l'utente, dandogli un senso di posizione e di progressione all'interno del flusso di lavoro.
- **Navigazione Chiara:** Pulsanti "Avanti" e "Indietro" ben visibili permettono una navigazione intuitiva tra i passaggi, consentendo all'utente di rivedere e correggere i dati inseriti prima della sottomissione finale.

- **Feedback di Validazione Contestuale:** In caso di errore di inserimento, il sistema non mostra un messaggio generico, ma evidenzia visivamente il campo specifico che richiede attenzione (es. con un bordo rosso) e mostra un messaggio di errore pertinente nelle immediate vicinanze.

L'interattività di questi elementi, come l'abilitazione/disabilitazione dei pulsanti di navigazione, è gestita con un uso mirato di JavaScript, sfruttando la leggerezza di librerie come **Alpine.js** [15] per aggiungere comportamento dinamico senza appesantire il frontend.

4.3.2 Gestione degli Upload: Feedback Visivo e Validazione

L'upload di file è un'operazione che può generare incertezza nell'utente. Per mitigare questo aspetto, i componenti di upload standard del browser sono stati sostituiti da elementi personalizzati che forniscono un feedback visivo immediato e chiaro. Il design di questi componenti segue un ciclo di stati:

1. **Stato Iniziale:** Il componente mostra un invito all'azione chiaro, come "Seleziona un file o trascinalo qui".

2. **Stato di Successo:** Una volta che l'utente seleziona un file valido, il componente cambia stato visivamente. Ad esempio, l'etichetta viene sostituita dal nome del file caricato e lo sfondo del componente può diventare verde, comunicando in modo inequivocabile che l'operazione è andata a buon fine.

3. **Stato di Errore:** Se l'utente tenta di caricare un file non consentito (es. per tipo o dimensione), il componente fornisce un feedback di errore immediato, ad esempio con un bordo rosso e un messaggio esplicativo, prevenendo l'invio di dati non validi al server.

Questa attenzione al feedback visivo trasforma un'operazione potenzialmente ambigua in un processo trasparente e rassicurante per l'utente.

4.3.3 Navigazione e Pulsanti Funzionali

La coerenza visiva e funzionale è un pilastro dell'usabilità. All'interno dell'applicazione, tutti gli elementi di navigazione e i pulsanti di azione seguono una gerarchia cromatica e stilistica definita. Le azioni primarie (es. "Salva", "Genera PDF") sono associate a pulsanti con un colore predominante, mentre le azioni secondarie (es. "Annulla") utilizzano stili meno evidenti. Questa convenzione guida l'attenzione dell'utente verso le azioni più importanti.

Inoltre, l'applicazione include funzionalità pensate per migliorare il comfort dell'utente, come il **toggle per il tema Chiaro/Scuro**. Questa opzione di personalizzazione, implementata tramite variabili CSS e un semplice script JavaScript, permette di adattare l'interfaccia alle preferenze personali o alle condizioni di illuminazione ambientale, un dettaglio che denota una particolare attenzione alla qualità complessiva dell'esperienza utente.

4.4 Meccanismi di Feedback per l'Utente

Un'interfaccia efficace non solo accetta input, ma "risponde" all'utente, comunicando lo stato delle operazioni in corso e l'esito delle azioni completate. L'applicazione implementa due principali meccanismi di feedback per garantire una comunicazione sistema-utente chiara e costante.

Il primo meccanismo gestisce il **feedback asincrono immediato**. Per le operazioni che richiedono un tempo di elaborazione lato server (come la generazione di un PDF complesso), l'interfaccia fornisce un riscontro visivo istantaneo per evitare che l'utente percepisca il sistema come bloccato. Al click su un pulsante di azione, uno script JavaScript si occupa di:

- Disabilitare il pulsante per prevenire invii multipli.
- Mostrare un indicatore di caricamento, come uno *spinner* o un intero *overlay* semi-trasparente con un messaggio (es. "Elaborazione in corso...").

Questo approccio migliora drasticamente la *latenza percepita*, rassicurando l'utente che la sua richiesta è stata presa in carico e che il sistema sta lavorando.

Il secondo meccanismo è il sistema di **feedback di stato**, utilizzato per comunicare l'esito di un'operazione dopo un ricaricamento di pagina (es. dopo il salvataggio di un form). Per questo, l'applicazione sfrutta il sistema di **message flashing** di Flask [17]. Quando un'azione viene completata con successo o con un errore nel backend, una funzione `flash()` salva un messaggio categorizzato (es. `success`, `error`, `warning`) nella sessione dell'utente. Il template di base dell'applicazione contiene una logica che, a ogni caricamento di pagina, controlla la presenza di questi messaggi e, se ne trova, li visualizza in un'area prominente, solitamente in cima alla pagina. I messaggi sono stilisticamente differenziati per colore (es. verde per il successo, rosso per l'errore), fornendo un riscontro cromatico e testuale immediatamente comprensibile.

Capitolo 5

Sfide Tecniche e Soluzioni Implementate

Lo sviluppo di un'applicazione software non è un processo puramente lineare. Oltre all'implementazione delle funzionalità pianificate, emergono inevitabilmente sfide tecniche impreviste che richiedono analisi, sperimentazione e l'adozione di soluzioni specifiche. Questo capitolo analizza alcune delle sfide più significative incontrate durante lo sviluppo della piattaforma e le strategie ingegneristiche adottate per superarle.

5.1 La Complessità della Generazione PDF Dinamica

Una delle sfide tecniche più ardue del progetto è stata la generazione di documenti PDF dinamici, professionali e conformi a layout complessi. La semplice conversione di un file HTML in PDF si scontra con le peculiarità del formato "a pagina fissa", che introduce problematiche non presenti nel rendering web standard.

Il problema principale risiede nel controllo del layout paginato. Era necessario gestire in modo affidabile il posizionamento assoluto di elementi come loghi, firme e timbri, garantire che le interruzioni di pagina (*page breaks*) non avvenissero in punti critici (es. lasciando una firma isolata all'inizio di una nuova pagina) e replicare intestazioni su più pagine. Librerie di conversione diretta HTML-to-PDF, pur essendo potenti, mostravano dei limiti nel garantire questo livello di precisione granulare.

La soluzione adottata è stata un'architettura di generazione a **pipeline in due fasi**, che sfrutta i punti di forza di due librerie specializzate e complementari:

1. **Fase 1 - Rendering del Layout con WeasyPrint:** Inizialmente, i dati raccolti dal form vengono utilizzati per popolare un template HTML. Questo template viene quindi processato da **WeasyPrint** [11], un motore di rendering scelto per il suo eccellente supporto alle specifiche CSS per i media impaginati (*CSS Paged Media Module*). WeasyPrint si occupa di interpretare il layout, gestire le interruzioni di pagina e produrre un PDF "di base" ben strutturato e stilisticamente corretto.

2. **Fase 2 - Manipolazione e Finitura con PyMuPDF:** Il PDF generato da WeasyPrint viene passato come input a **PyMuPDF** [13]. Questa libreria agisce a un livello più basso, permettendo di manipolare direttamente gli oggetti di una pagina PDF. In questa fase vengono "sovraimpressi" (overlay) sul documento base gli elementi che richiedono un posizionamento assoluto e preciso, come le immagini delle firme e dei timbri, utilizzando coordinate esatte. PyMuPDF è inoltre lo strumento che permette di fondere più documenti PDF, come visto nel modulo di Accountancy.

Questo approccio a pipeline si è rivelato vincente, poiché combina la flessibilità del design basato su standard web (HTML/CSS) con la precisione di una libreria di manipolazione PDF a basso livello, risolvendo efficacemente la sfida della generazione di documenti complessi e dinamici.

5.2 Centralizzare la Gestione dei Progressivi: la Creazione di un Blueprint Dedicato

Una delle funzionalità apparentemente più semplici, ma tecnicamente più insidiose, è la generazione di numeri di protocollo univoci e sequenziali per ogni documento. Nelle prime versioni dell'applicazione, questo era gestito tramite la lettura e la scrittura di un file JSON condiviso, `progressivi.json`. Sebbene funzionale in un ambiente a

utente singolo, questo approccio presentava una vulnerabilità critica in un contesto web multi-utente: il rischio di **race condition**.

La sfida si manifesta quando due o più richieste di generazione di documenti avvengono quasi simultaneamente. Entrambi i processi potrebbero leggere lo stesso numero progressivo dal file JSON, incrementarlo dello stesso valore e poi riscriverlo. Il risultato sarebbe la generazione di due documenti distinti con lo stesso identico numero di protocollo, una grave violazione dell'integrità dei dati che vanificherebbe lo scopo stesso del sistema di protocollazione. Inoltre, la logica per la gestione dei progressivi era sparsa in diversi punti dell'applicazione, rendendo complesse eventuali modifiche.

La soluzione è stata un'importante opera di **refactoring architetturale**: la creazione di un **Blueprint dedicato**, il `protocol_manager`. Questa scelta ha risolto la sfida su due fronti:

1. **Centralizzazione della Logica:** Tutto il codice relativo alla generazione e alla gestione dei numeri di protocollo è stato spostato all'interno di questo nuovo modulo. Il Blueprint espone un'interfaccia chiara e unica al resto dell'applicazione, diventando l'unica autorità responsabile dei progressivi. Questo aderisce al **Principio di Singola Responsabilità** (Single Responsibility Principle), migliorando notevolmente la manutenibilità e la leggibilità del codice.

2. **Migrazione su Database Transazionale:** La persistenza dei contatori è stata spostata dal file JSON a una tabella dedicata nel database SQLite. Questa è la soluzione tecnica alla race condition. Sfruttando le proprietà **ACID** (Atomicità, Consistenza, Isolamento, Durabilità) del database, l'operazione di "leggi, incrementa e salva" il contatore può essere eseguita come una **transazione atomica**. Il database stesso si fa carico di gestire i lock a basso livello, garantendo che anche in caso di richieste concorrenti, ogni transazione venga eseguita in modo isolato e sequenziale, eliminando alla radice la possibilità di generare numeri di protocollo duplicati.

Questa evoluzione da un semplice file di configurazione a un servizio transazionale centralizzato rappresenta un significativo passo avanti nella maturità e nella robustezza dell'applicazione.

5.3 Scalare l'Interfaccia Multi-step da un solo flusso a tutti i contratti

L'introduzione del form multi-step per il primo tipo di contratto (oGX) si era rivelata un grande successo in termini di User Experience. La sfida successiva è stata estendere questo modello di interazione a tutte le altre tipologie di contratto (iGV Enabler, iGV EP, etc.), ciascuna con le proprie specificità in termini di numero di passaggi, campi richiesti e logiche di validazione.

L'approccio più ingenuo, ovvero duplicare e adattare il codice per ogni nuovo tipo di contratto, avrebbe portato a una massiccia **duplicazione del codice**, violando il principio "Don't Repeat Yourself" (DRY) [9]. Questo avrebbe reso la manutenzione un incubo: una modifica alla logica di navigazione o alla grafica avrebbe richiesto un intervento su decine di file diversi, con un alto rischio di introdurre inconsistenze.

La soluzione adottata è stata un profondo **refactoring** del componente multi-step per trasformarlo in un sistema **generico e configurabile**. Anziché avere una logica "hard-coded" per un singolo flusso, è stato creato un motore di rendering e di gestione dello stato più astratto. L'idea centrale è stata quella di separare la *struttura* del processo multi-step (la logica di navigazione, la visualizzazione della barra di progresso, il salvataggio progressivo) dalla *configurazione* specifica di ogni contratto.

In questa nuova architettura:

- Il backend gestisce il flusso tramite una rotta dinamica che accetta come parametri il tipo di contratto e il numero dello step.
- Per ogni tipo di contratto, viene definito un "manifesto" o un oggetto di configurazione che descrive la sequenza degli step, il template HTML da renderizzare per ciascuno e le regole di validazione da applicare.
- I template Jinja2 sono stati scomposti in componenti riutilizzabili. Esiste un template "scheletro" (*shell*) per il multi-step che contiene la barra di progresso e i pulsanti, e che al suo interno carica dinamicamente il contenuto specifico dello step corrente.

Questo disaccoppiamento tra la logica del "contenitore" multi-step e il "contenuto" specifico di ogni contratto ha portato a enormi benefici. L'aggiunta di un nuovo flusso contrattuale è diventata un'operazione molto più rapida, che richiede solo la creazione dei nuovi template e di un nuovo file di configurazione, senza più toccare la logica di base. Questo ha garantito una totale consistenza dell'esperienza utente in tutta l'applicazione e ha reso il sistema infinitamente più manutenibile e scalabile.

5.4 Ottimizzazione delle Prestazioni nel Caricamento Dati delle Dashboard

Le dashboard finanziarie e di audit sono tra le funzionalità a più alto valore aggiunto, ma presentavano una sfida prestazionale intrinseca: i dati sorgente risiedono su un servizio esterno (Google Sheets). Un'implementazione ingenua, che interrogasse le API di Google a ogni visita della pagina, avrebbe comportato tempi di caricamento inaccettabilmente lunghi (nell'ordine di svariati secondi), una pessima esperienza utente e un eccessivo consumo delle quote API.

La sfida era quindi quella di **disaccoppiare** il caricamento della pagina da parte dell'utente dalla lenta operazione di recupero dei dati dalla fonte esterna. La soluzione è stata l'implementazione di un'architettura basata su uno strato di **caching asincrono**.

Il meccanismo funziona nel seguente modo:

1. **Recupero Dati in Background:** Un processo schedulato, gestito dalla libreria **APScheduler** [1], viene eseguito a intervalli regolari (es. una volta ogni notte, durante le ore di minor traffico). Questo processo è l'unico autorizzato a contattare le API di Google Sheets tramite `gspread` [6] per recuperare l'intero dataset finanziario e di audit.

2. **Creazione della Cache Locale:** I dati recuperati vengono pre-elaborati, aggregati e infine salvati localmente sul server in file JSON ottimizzati per la lettura (`financial_cache.json` e `audit_data_cache.json`). La lettura da un file locale è un'operazione quasi istantanea rispetto a una chiamata di rete.

3. **Servizio Dati dalla Cache:** Quando un utente visita una pagina della dashboard, la rotta Flask non contatta più le API di Google, ma si limita a leggere i dati dal file di cache locale. Questo riduce il tempo di risposta da diversi secondi a pochi millisecondi.

Per garantire la possibilità di visualizzare dati aggiornati su richiesta, è stata inoltre implementata una funzionalità nel pannello di amministrazione per forzare manualmente l'aggiornamento della cache. Questa architettura di caching non solo ha migliorato drasticamente le performance e l'esperienza utente, ma ha anche reso l'applicazione più **resiliente**: le dashboard rimangono consultabili (mostrando l'ultimo dato disponibile) anche nel caso in cui le API di Google siano temporaneamente irraggiungibili.

Capitolo 6

Guide Pratiche (Manutenzione e Uso)

Dopo aver analizzato l'architettura e le funzionalità della piattaforma, questo capitolo fornisce una serie di guide pratiche destinate ai due principali profili di utenza: il membro AIESEC, che utilizza l'applicazione quotidianamente, e il futuro sviluppatore, che potrebbe doverla mantenere o estendere. L'obiettivo è tradurre i concetti discussi in precedenza in istruzioni operative chiare e dirette.

6.1 Guida per l'Utente AIESEC

Questa sezione è concepita come un manuale d'uso essenziale per i membri dei Comitati Locali e Nazionali. Illustra i flussi di lavoro più comuni, dalla configurazione iniziale alla generazione dei documenti e alla consultazione dei dati.

6.1.1 Primo Accesso e Configurazione del Comitato

Al primo utilizzo della piattaforma, è fondamentale eseguire una configurazione iniziale per garantire che tutti i documenti vengano generati con i dati corretti.

1. **Effettuare l'Accesso:** Navigare all'URL dell'applicazione. Selezionare il proprio comitato dal menu a tendina e inserire il PIN numerico fornito dal proprio responsabile (es. LCP) o dall'amministratore di sistema.

2. **Controllare la Dashboard Iniziale:** Dopo il login, si viene reindirizzati alla dashboard principale. È probabile che vengano mostrati dei messaggi di avviso (*warnings*) che segnalano la mancanza di informazioni o file essenziali.

3. **Accedere alle Impostazioni:** Dal menu di navigazione, cliccare sulla voce "Impostazioni". Questa pagina è il pannello di controllo per tutti i dati specifici del comitato.

4. **Compilare i Dati e Caricare i File:** Compilare in modo accurato tutti i campi richiesti nella sezione "Dati Comitato" (es. nome Presidente, nome VP Finance, IBAN, etc.). Successivamente, utilizzare i moduli di upload per caricare i file richiesti: l'immagine della firma del Presidente, quella del VP Finance e, se applicabile, il timbro del comitato. Questi dati e file verranno utilizzati automaticamente dall'applicazione per popolare tutti i documenti generati.

6.1.2 Come Creare e Gestire un Contratto con il Flusso Guidato

La generazione di un contratto è il processo principale dell'applicazione e segue un flusso guidato.

1. **Avviare il Processo:** Dalla dashboard o dal menu principale, selezionare l'opzione per creare un nuovo contratto (es. "Nuovo Contratto EP"). Scegliere la tipologia specifica di contratto da generare (es. Global Volunteer, Global Talent, etc.).

2. **Seguire la Compilazione Multi-step:** L'applicazione presenterà un form suddiviso in più passaggi. Compilare i campi richiesti in ogni step. Il progresso viene salvato automaticamente a ogni passaggio, quindi è possibile interrompere e riprendere la compilazione in un secondo momento.

3. **Finalizzare la Generazione:** Una volta completato l'ultimo step e verificati i dati, confermare la generazione. Il sistema creerà il documento PDF e il contratto apparirà nella dashboard di gestione con uno stato iniziale (es. "Generato").

4. **Consultare e Scaricare:** Dalla dashboard di gestione contratti, è possibile ricercare il contratto appena creato, visualizzarne i dettagli e scaricare il file PDF finale per l'invio al partecipante.

6.1.3 Come Utilizzare le Dashboard di Finanza e Audit

Le dashboard forniscono una visione sintetica e immediata dello stato di salute del comitato dal punto di vista della conformità e della finanza.

- **Widget di Audit:** Nella dashboard principale, il riquadro dedicato all'audit mostra lo stato di conformità del comitato per il mese in corso. I colori (verde, arancione, rosso) e un messaggio di stato chiaro indicano se l'audit è stato superato e in quale fase. Cliccando sui dettagli è possibile visualizzare i punteggi parziali per ogni area di valutazione.

- **Widget Finanziario:** Analogamente, il widget finanziario presenta i principali indicatori di performance economica (KPI) del comitato, come la liquidità attuale e il P&L (Profit & Loss). Fornisce una stima immediata della situazione finanziaria.

- **Approfondimento:** Per un'analisi più dettagliata dei dati finanziari, è possibile navigare alla pagina "Statistiche Finanziarie". Questa sezione offre una vista storica completa, con grafici e tabelle che mostrano l'andamento dei ricavi, dei costi e di altri indicatori su base mensile, trimestrale e semestrale.

6.2 Guida per lo Sviluppatore

Questa guida è destinata a uno sviluppatore che debba prendere in carico la manutenzione o l'estensione della piattaforma. Fornisce le istruzioni per l'installazione dell'ambiente di sviluppo, una mappa per orientarsi nella codebase e un tutorial per l'aggiunta di nuove funzionalità.

6.2.1 Setup dell'Ambiente di Sviluppo Locale

Per poter eseguire e modificare l'applicazione in locale, è necessario seguire i seguenti passaggi:

1. **Prerequisiti:** Assicurarsi di avere installato Python (versione 3.8 o superiore) e Git.

2. **Clonazione del Repository:** Clonare il codice sorgente del progetto da GitHub o dal repository in uso.

3. **Creazione dell'Ambiente Virtuale:** All'interno della cartella del progetto, creare un ambiente virtuale per isolare le dipendenze. Eseguire i comandi:

   ```
   python -m venv venv
   source venv/bin/activate  # Su Windows: venv\Scripts\activate
   ```

4. **Installazione delle Dipendenze:** Installare tutte le librerie Python necessarie, elencate nel file `requirements.txt`, con un singolo comando:

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

5. **Configurazione Iniziale:** Creare una copia del file `comitati_config.json` e assicurarsi che contenga almeno la configurazione per il comitato 'test'. Creare un file `google_credentials.json` (anche vuoto, se non si necessita dell'integrazione con Google Sheets in fase di test). Verificare che le cartelle `/uploads` e `/temp_generated` esistano nella root del progetto.

6. **Avvio del Server di Sviluppo:** Eseguire il file principale per avviare l'applicazione.

   ```
   python app.py
   ```

Il server di sviluppo di Flask si avvierà in modalità *debug*, rendendo l'applicazione accessibile all'indirizzo `http://127.0.0.1:5000` e riavviandola automaticamente a ogni modifica del codice.

6.2.2 Struttura del Codice: Navigare tra File e Cartelle

La codebase è organizzata in modo modulare per facilitare la navigazione e la manutenzione. I punti di interesse principali sono:

- `app.py`: È il file di avvio e il cuore orchestratore dell'applicazione. Qui vengono inizializzati Flask, le configurazioni globali, i task schedulati e, soprattutto, vengono registrati i Blueprint.
- **Le cartelle dei Blueprint** (es. `/ep_contracts`, `/protocol_manager`): Ciascuna di queste directory rappresenta una macro-funzionalità dell'applicazione. Al loro interno si trovano tipicamente i file `routes.py` (con la logica delle viste) e `models.py` (con la logica di interazione con il database).
- `/compilatori`: Questa cartella contiene tutti gli script Python ("compilatori") responsabili della generazione dei singoli documenti PDF. Ogni file corrisponde a un tipo di documento.
- `/interfaces` (o `/templates`): Contiene tutti i template HTML **Jinja2** che costituiscono l'interfaccia utente dell'applicazione.
- `/static`: Contiene i file statici come CSS, immagini e JavaScript lato client.
- **File `.json` e `.db`**: I file di configurazione e i database SQLite si trovano nella directory principale del progetto.

6.2.3 Tutorial: Aggiungere un Nuovo Tipo di Documento "Statico"

Grazie all'architettura modulare, l'aggiunta di un nuovo documento (es. un nuovo tipo di verbale) è un processo strutturato:

1. **Creare il Template PDF:** Creare un nuovo file HTML (es. `nuovo_verbale.html`) nella cartella dei template PDF (es. `/templates`) che definirà la struttura del documento finale. Utilizzerà le variabili Jinja2 per i dati dinamici.

2. **Creare il Form di Input:** Creare un nuovo file HTML (es. `form_nuovo_verbale.html`) nella cartella `/interfaces`. Questo file conterrà il form web per l'inserimento dei dati da parte dell'utente.

3. **Creare il Compilatore Python:** Creare un nuovo file Python (es. `nuovo_verbale.py`) nella cartella `/compilatori`. Questo script deve contenere una funzione `genera(...)` che riceve i dati del form, renderizza il template PDF e restituisce i byte del file finale.

4. **Registrare il Nuovo Documento:** Aprire il file `app.py`, importare il nuovo modulo compilatore e aggiungerlo al dizionario `COMPILER_MAP` con una chiave univoca (es. `'nuovo_verbale': nuovo_verbale`). Infine, aggiungere un link al nuovo form nella pagina di selezione appropriata (es. in `verbali_seleziona.html`).

6.2.4 Debugging e Log

Per l'individuazione e la risoluzione dei problemi, sono disponibili diversi strumenti:

- **Debugger di Flask:** Eseguendo l'applicazione in modalità debug, qualsiasi eccezione non gestita nel backend attiverà un debugger interattivo direttamente nel browser, che permette di ispezionare lo stack di chiamate e i valori delle variabili.

- **Logging su File:** L'applicazione implementa una funzione `log_activity()` che scrive le operazioni significative in un file di log strutturato, `activity_log.json`. Questo file può essere consultato per tracciare le azioni degli utenti e diagnosticare problemi.

- **Strumenti di Sviluppo del Browser:** Per problemi relativi al frontend (JavaScript, CSS, richieste AJAX), gli strumenti di sviluppo integrati nei moderni browser (Console, Network tab) sono essenziali.

Capitolo 7

Sviluppi Futuri e Roadmap

Un'applicazione di successo è un'entità in continua evoluzione. Questo capitolo delinea una roadmap di possibili sviluppi futuri, pensati per estendere le capacità della piattaforma e aumentarne ulteriormente il valore strategico per AIESEC. Le proposte sono raggruppate in aree tematiche: evoluzioni funzionali, miglioramenti dell'esperienza utente e potenziamento tecnico.

7.1 Evoluzioni Funzionali Core

7.1.1 Modulo di Analytics e "Conversioni"

L'evoluzione più significativa consisterebbe nella creazione di un modulo di **Business Intelligence** avanzato. Tale modulo aggregherebbe i dati operativi di tutte le aree funzionali (es. Marketing, oGV, iGV) per calcolare e visualizzare i tassi di conversione lungo l'intero customer journey di AIESEC. Questo permetterebbe ai comitati di passare da una gestione basata sull'intuizione a una strategia **data-driven**, monitorando l'efficacia delle proprie strategie in tempo reale.

7.1.2 Integrazione con Servizi di Firma Elettronica

Per completare la digitalizzazione del processo contrattuale, un passo successivo naturale sarebbe l'integrazione con un servizio di **Firma Elettronica Avanzata (FEA)** o **Firma Digitale**. Questo eliminerebbe l'ultimo passaggio manuale (stampa, firma, scan-

sione), rendendo il flusso 100% digitale e aumentando il valore legale e la professionalità dei documenti scambiati con partner e partecipanti.

7.1.3 API Pubblica e Webhooks

L'esposizione di un'**API REST** sicura e di un sistema di **Webhooks** aprirebbe la piattaforma a innumerevoli possibilità di integrazione con altri sistemi, sia interni che esterni ad AIESEC. Ad esempio, si potrebbero inviare notifiche automatiche su Slack o Discord alla firma di un contratto, o sincronizzare i dati con altri strumenti gestionali in uso a livello nazionale o internazionale.

7.2 Miglioramenti dell'Esperienza Utente (UI/UX)

7.2.1 Sistema Multilingua (Italiano/Inglese)

L'internazionalizzazione dell'interfaccia, rendendola pienamente disponibile in lingua inglese, aumenterebbe drasticamente l'accessibilità per i membri internazionali e per gli stakeholder esterni. L'implementazione di un sistema di **i18n** (internationalization), ad esempio tramite la libreria `Flask-Babel`, renderebbe la piattaforma un modello potenzialmente replicabile in altre nazioni della rete AIESEC.

7.2.2 Potenziamento dell'Interfaccia e Accessibilità

Un'ulteriore evoluzione del frontend potrebbe includere un affinamento del tema scuro, un miglioramento dei contrasti e l'adozione di standard di **accessibilità** web (es. WAI-ARIA) per garantire che l'applicazione sia utilizzabile dal maggior numero di persone possibile, indipendentemente da eventuali disabilità.

7.3 Potenziamento Tecnico e di Sicurezza

7.3.1 Migrazione del Database a PostgreSQL/MySQL

Con l'aumentare del volume dei dati, potrebbe diventare strategico pianificare una migrazione da SQLite a un sistema di database server-based più robusto, come **Postgre-**

SQL. Questo garantirebbe maggiore scalabilità, performance superiori in condizioni di alta concorrenza e strumenti di gestione e backup più avanzati.

7.3.2 Sistema di Backup Automatizzato

Per garantire la business continuity e la protezione dei dati, sarebbe fondamentale implementare un sistema di **backup automatico e schedulato**. Tale sistema dovrebbe eseguire copie di sicurezza regolari sia del database che dei file caricati (firme, timbri, documenti), archiviandole in una location esterna e sicura (es. un object storage su cloud come Amazon S3).

Appendice A

Screenshot Significativi dell'Interfaccia

Questa appendice presenta una selezione di schermate rappresentative dell'interfaccia utente della web application, al fine di fornire un riferimento visivo alle funzionalità e al design discussi nel corpo del documento.

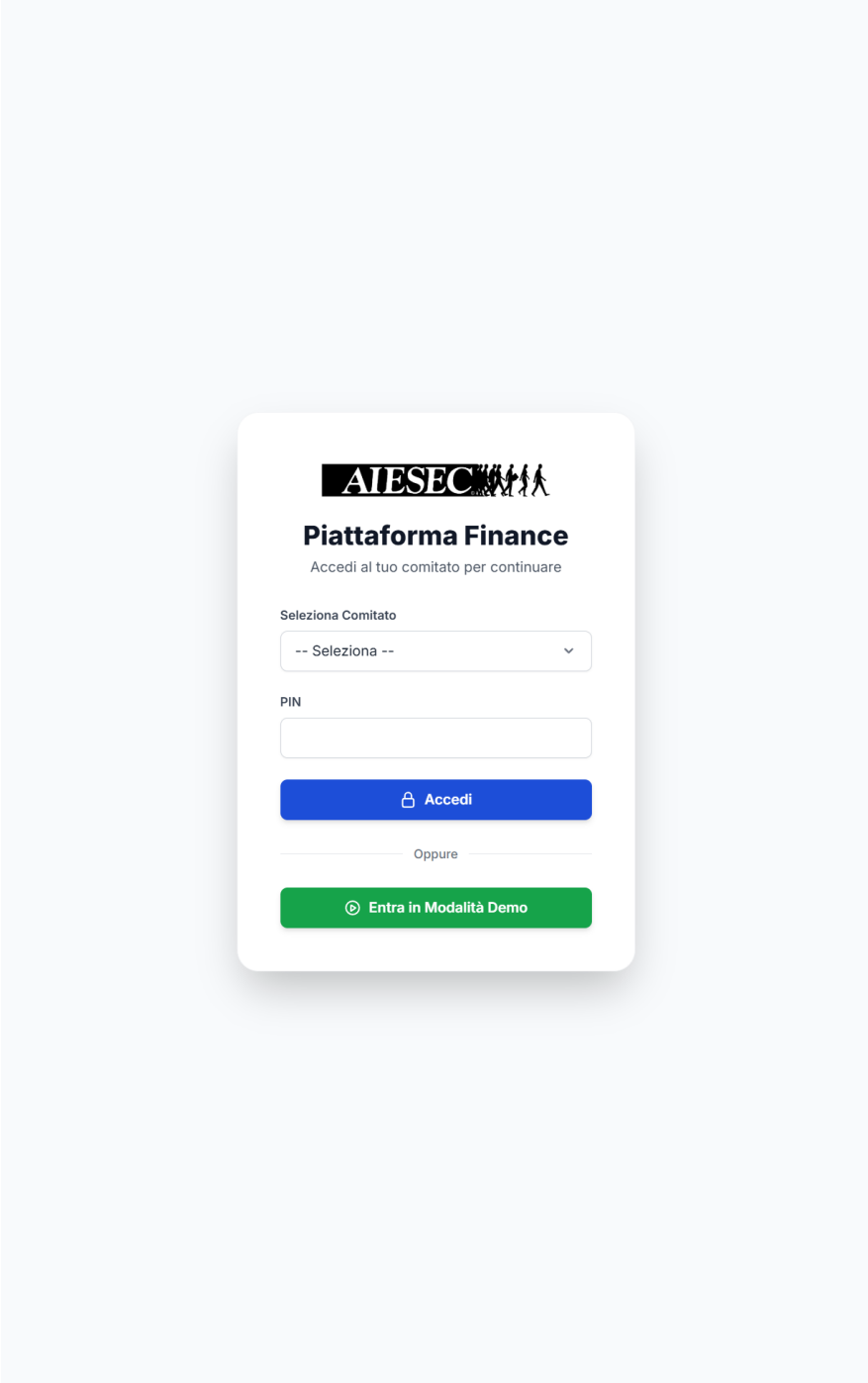

Figura A.1: La pagina di accesso all'applicazione, con la selezione del comitato e l'inserimento del PIN.

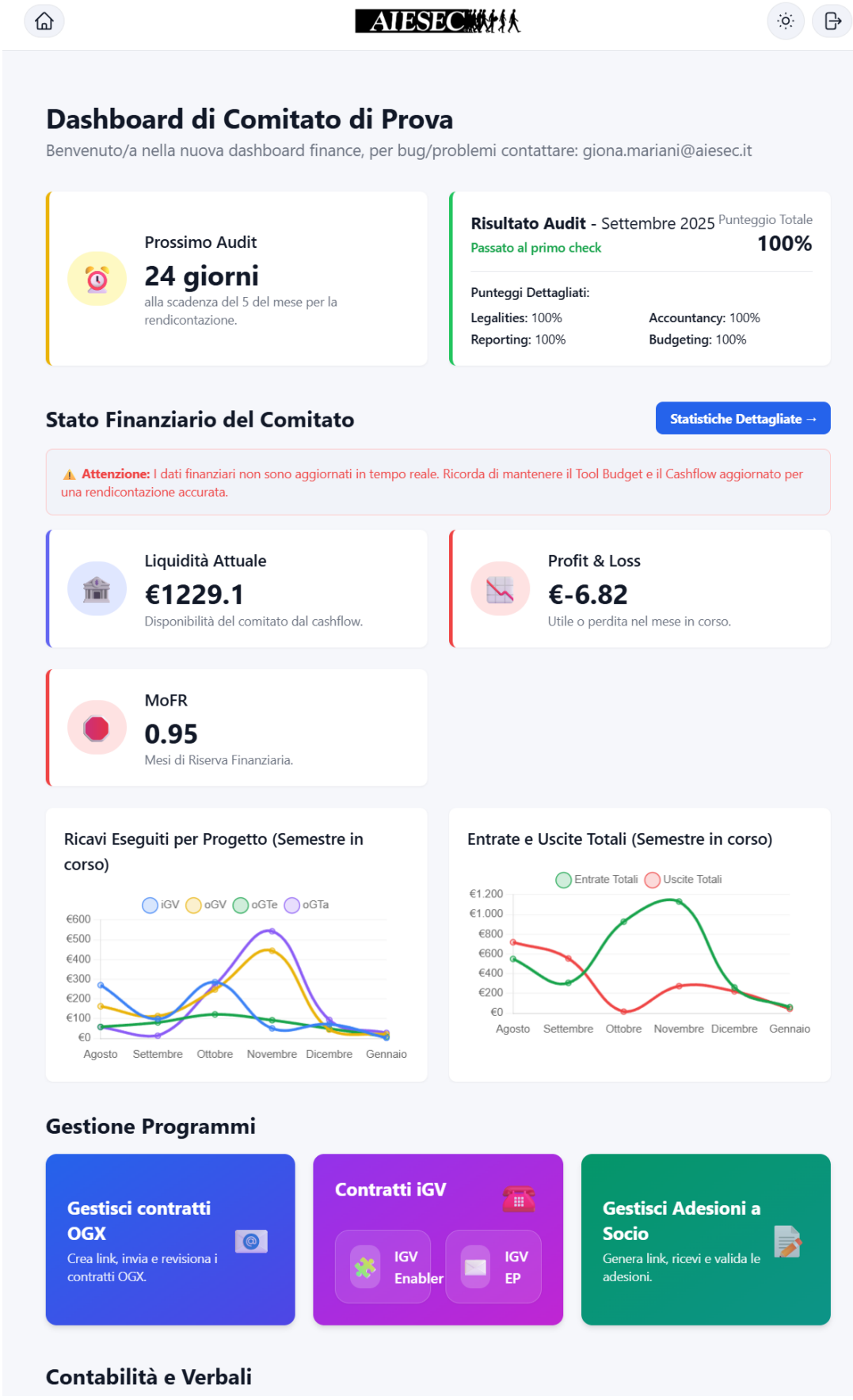

Figura A.2: La dashboard principale per un utente di un Comitato Locale (LC), con i widget riassuntivi per l'Audit, la Finanza e le notifiche.

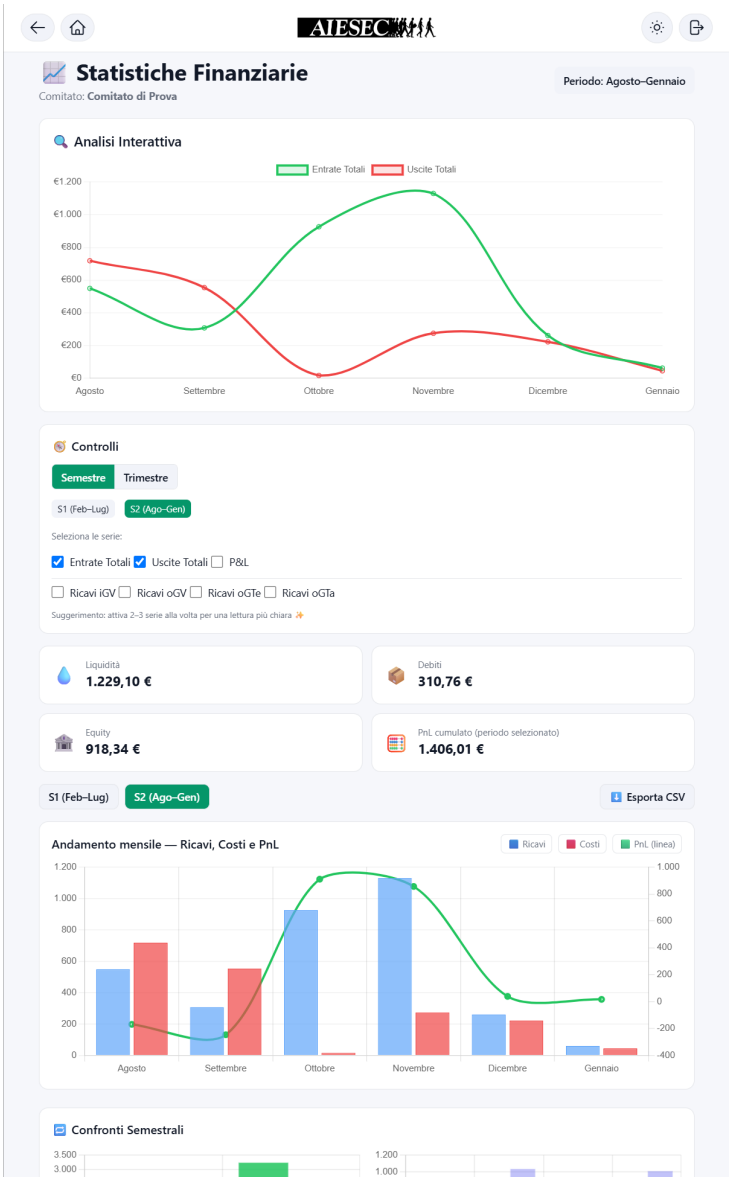

Figura A.3: La pagina dedicata alle Statistiche Finanziarie, che offre un'analisi dettagliata con grafici sull'andamento di KPI come P&L e Liquidità.

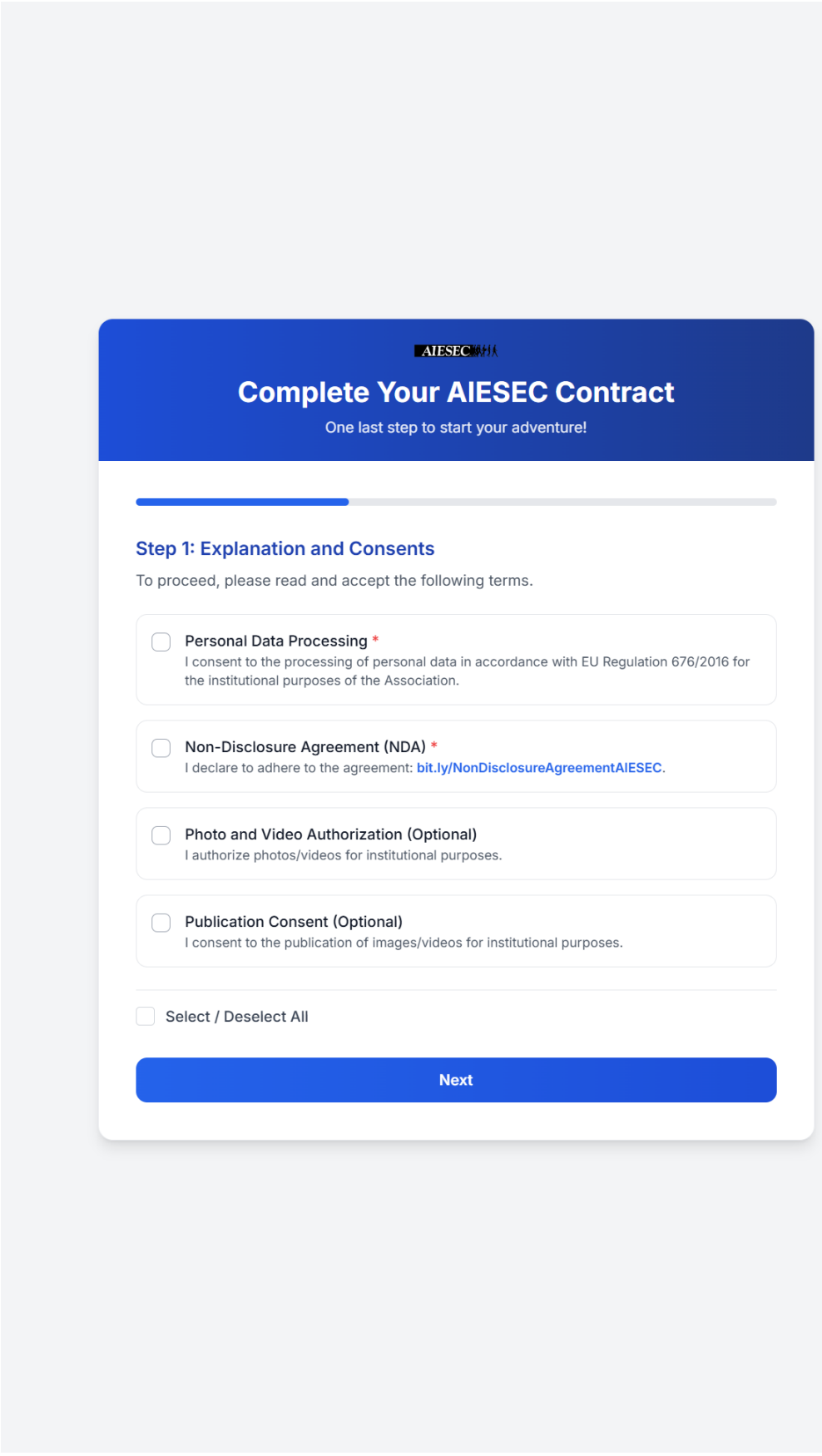

Figura A.4: Esempio dell'interfaccia guidata multi-step per la compilazione di un contratto. Si noti l'indicatore di progressione in alto.

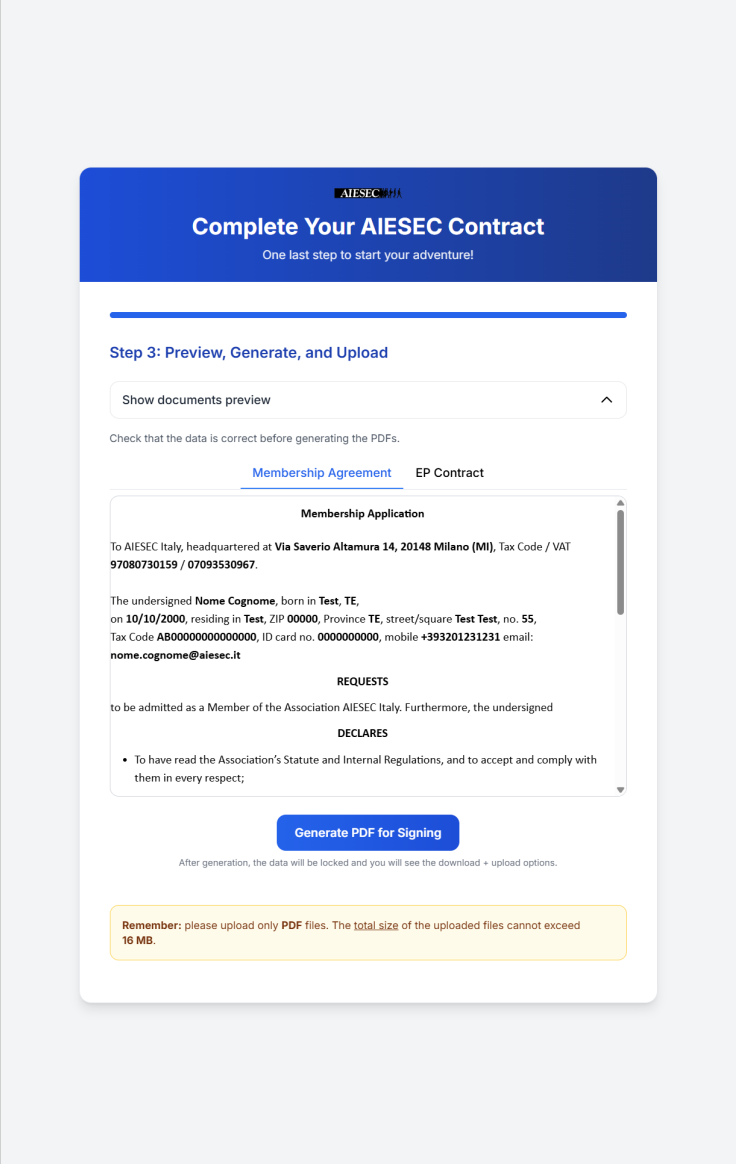

Figura A.5: La funzionalità di anteprima del contratto all'interno del form multi-step, che permette all'utente di visualizzare il documento finale prima della generazione definitiva.

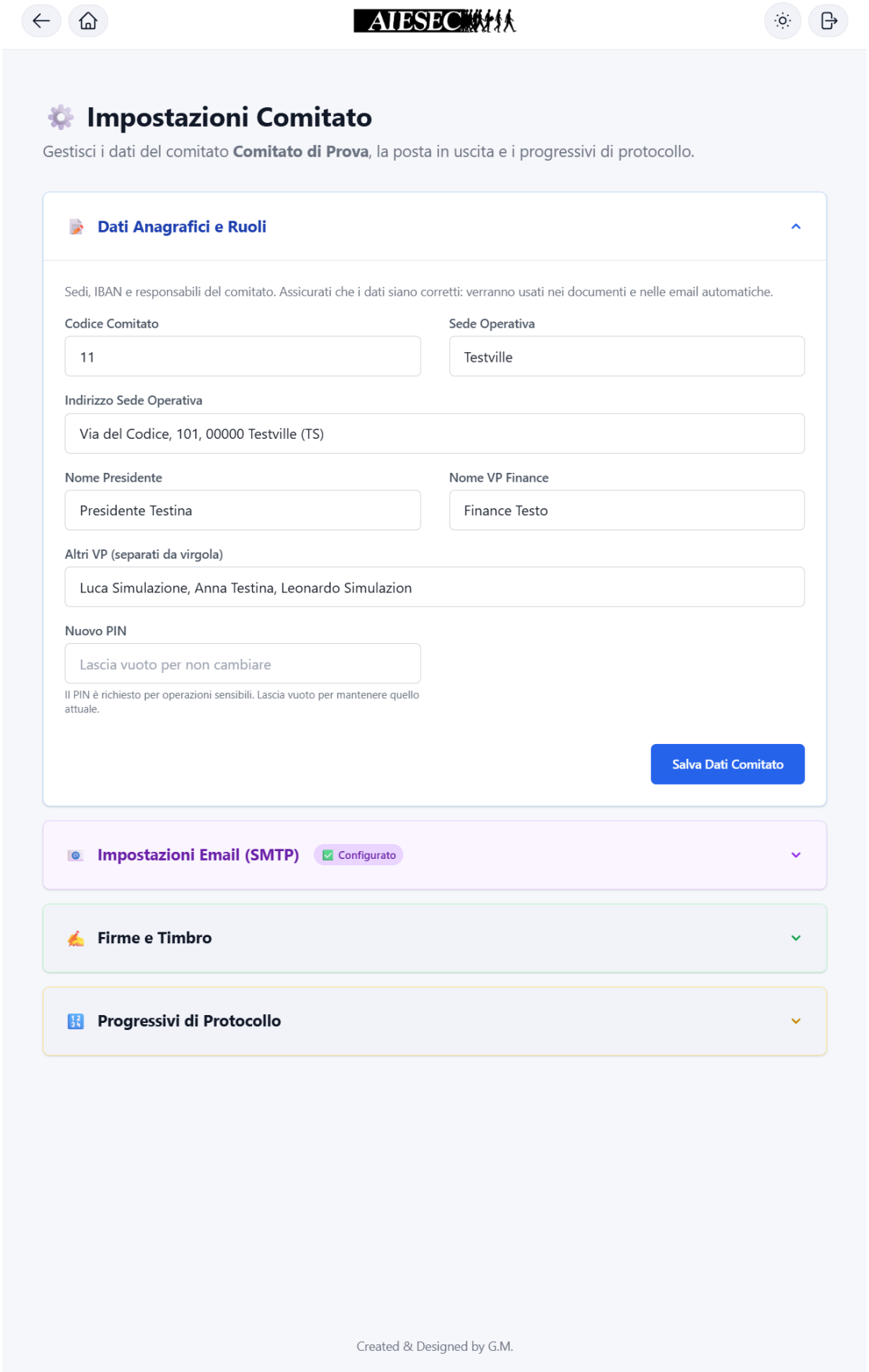

Figura A.6: La pagina delle Impostazioni, da cui l'utente può aggiornare i dati del proprio comitato e caricare i file per le firme e il timbro.

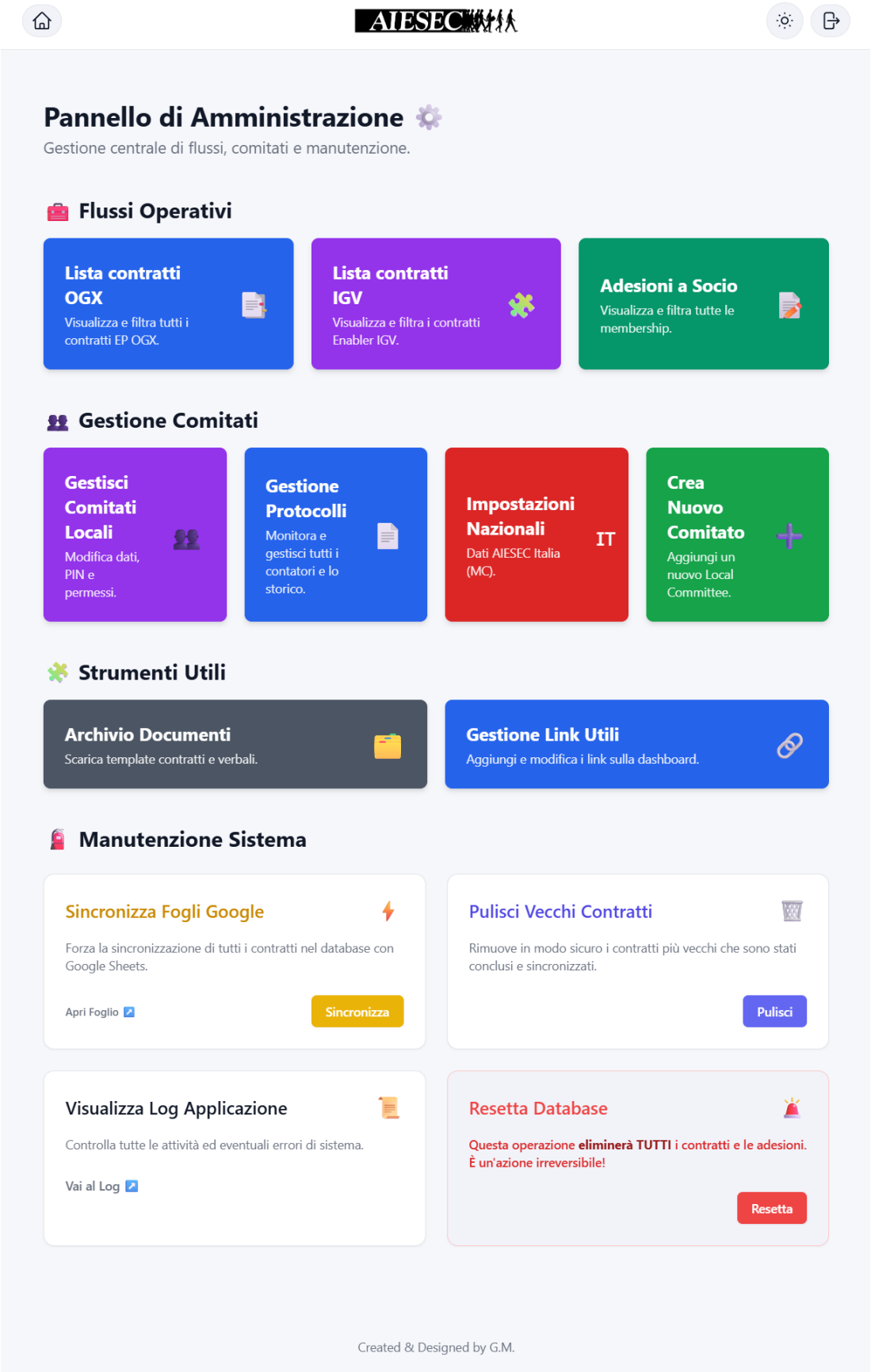

Figura A.7: La dashboard riservata all'amministratore (MC), con una vista aggregata dei dati finanziari e dello stato di compliance nazionale.

Appendice B

Estratti di Codice Commentati

Questa appendice contiene una selezione di estratti di codice significativi, commentati per illustrare l'implementazione pratica di alcuni dei concetti architetturali discussi nel corpo del documento.

Registrazione dei Blueprint in `app.py`

Come analizzato nella Sezione 2.3, l'applicazione è strutturata in moduli funzionali indipendenti utilizzando i Blueprint di Flask. L'estratto di codice seguente, proveniente dalla fine del file `app.py`, mostra come questi moduli vengono importati e poi "registrati" nell'applicazione Flask principale. Ogni chiamata a `app.register_blueprint()` collega un intero set di rotte e logiche al sistema centrale, specificando un prefisso URL (es. `/ep-contracts`) per mantenere l'ordine. Questa pratica è il cuore dell'architettura modulare del progetto.

```
from ep_contracts.routes import bp as ep_flow_bp
from ep_membership import bp as ep_membership_bp
from igv_contracts.routes import igv_contracts_bp
from igv_ep_contracts.routes import bp as igv_ep_bp
from protocol_manager.routes import protocol_manager_bp

app.register_blueprint(ep_flow_bp, url_prefix='/ep-contracts')
app.register_blueprint(ep_membership_bp, url_prefix='/ep-membership')
app.register_blueprint(igv_contracts_bp, url_prefix="/igv-contracts")
```

```
app.register_blueprint(igv_ep_bp, url_prefix="/igv-ep")
app.register_blueprint(protocol_manager_bp, url_prefix='/
    protocol_manager')

if __name__ == '__main__':
    app.run(debug=True)
```

Algoritmo B.1: Registrazione dei Blueprint in `app.py`.

Implementazione di un Decoratore di Autenticazione

La messa in sicurezza delle rotte, discussa nella Sezione 3.1.2, è gestita tramite decoratori Python. L'estratto seguente mostra un esempio rappresentativo di come il decoratore `@login_required` (dal file `auth.py`) è implementato. Questo "wrapper" controlla se un utente è presente nella sessione prima di eseguire la funzione della rotta richiesta. Se l'utente non è autenticato, viene reindirizzato alla pagina di login. Questo pattern permette di separare la logica di controllo accessi dalla logica di business in modo pulito ed elegante.

```
# Esempio dal file auth.py
from functools import wraps
from flask import session, flash, redirect, url_for

def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if 'user' not in session:
            flash("Per accedere a questa pagina e necessario
    effettuare il login.", "warning")
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function
```

Algoritmo B.2: Esempio di implementazione del decoratore `@login_required`.

La Pipeline di Generazione PDF in un "Compilatore"

La soluzione alla sfida della generazione PDF, analizzata nella Sezione 5.1, si basa su una pipeline a due fasi. Il codice seguente è un esempio semplificato di una funzione `genera()` all'interno di un modulo "compilatore". Mostra chiaramente la sequenza: (1) rendering di un template HTML con Jinja2, (2) conversione dell'HTML in un PDF base con WeasyPrint, e (3) manipolazione del PDF base con PyMuPDF per aggiungere elementi con posizionamento assoluto come le firme.

```python
# Esempio semplificato di una funzione in /compilatori/ogv.py
from weasyprint import HTML
import fitz  # PyMuPDF
from flask import render_template

def genera(dati_form, dati_comitato, numero_protocollo, app_root_path):
    # FASE 1: Rendering HTML con i dati dinamici
    html_string = render_template(
        "pdf_templates/ogv.html",
        dati_form=dati_form,
        dati_comitato=dati_comitato,
        protocollo=numero_protocollo
    )

    # FASE 2: Conversione HTML -> PDF base con WeasyPrint
    pdf_base_bytes = HTML(string=html_string, base_url=app_root_path).write_pdf()

    # FASE 3: Manipolazione e finitura con PyMuPDF
    doc_pdf = fitz.open("pdf", pdf_base_bytes)
    page = doc_pdf[0] # Seleziona la prima pagina

    # Aggiunge il logo in coordinate precise
    logo_rect = fitz.Rect(50, 50, 150, 100)
    page.insert_image(logo_rect, filename="path/to/logo.png")

    # Aggiunge la firma in fondo alla pagina
    firma_rect = fitz.Rect(400, 700, 550, 750)
```

```
    page.insert_image(firma_rect, filename="path/to/firma_presidente.
    png")

    final_pdf_bytes = doc_pdf.tobytes()
    doc_pdf.close()

    return final_pdf_bytes
```

Algoritmo B.3: Esempio semplificato della pipeline di generazione PDF.

Appendice C

Glossario dei Termini AIESEC

Questa appendice fornisce una spiegazione degli acronimi e dei termini specifici dell'organizzazione AIESEC utilizzati nel presente documento, al fine di facilitarne la comprensione a un pubblico non specializzato. Le definizioni sono state integrate con quelle presenti nel "Welcoming Handbook" ufficiale di AIESEC in Roma Tre.

AI (AIESEC International) L'entità globale che coordina l'intera rete di AIESEC nel mondo[cite: 170, 189].

AIESEC La più grande organizzazione internazionale al mondo gestita da giovani, focalizzata sullo sviluppo della leadership attraverso scambi interculturali [cite: 45, 52-56].

EB (Executive Board) Il comitato direttivo di un Comitato Locale (LC), composto dai Vice Presidenti (VP) e dal Presidente (LCP)[cite: 160, 179].

EFB (Entity Financial Board) Il team nazionale di supporto all'area Finance & Legal, responsabile della compliance e della sostenibilità finanziaria dell'intera nazione.

EP (Exchange Participant) Un partecipante a un programma di scambio di AIESEC[cite: 156, 175].

F&L (Finance & Legalities) L'area funzionale che si occupa della gestione finanziaria, legale e amministrativa di un comitato[cite: 207, 208].

ICX (Incoming Exchange) Termine generico che si riferisce a tutti i programmi di scambio in entrata[cite: 203, 204].

IGV (Incoming Global Volunteer) Un programma di volontariato in entrata, in cui partecipanti internazionali vengono in Italia per lavorare su progetti legati agli Obiettivi di Sviluppo Sostenibile (SDG) dell'ONU[cite: 201, 202].

JD (Job Description) La descrizione delle responsabilità e dei compiti associati a un determinato ruolo all'interno dell'organizzazione[cite: 171, 190].

LC (Local Committee) Un Comitato Locale di AIESEC, solitamente basato in una città o università[cite: 157, 176].

LCP (Local Committee President) Il Presidente di un Comitato Locale[cite: 158, 177].

LCVP (Local Committee Vice President) Un Vice Presidente di un Comitato Locale[cite: 159, 178].

MC (Member Committee) Il Comitato Nazionale che gestisce le operazioni di AIESEC in un determinato paese o territorio (in questo caso, AIESEC in Italia)[cite: 164, 183].

MCP (Member Committee President) Il Presidente del Comitato Nazionale.

MCVP (Member Committee Vice President) Un Vice Presidente del Comitato Nazionale[cite: 166, 185].

MKTG (Marketing) L'area funzionale che si occupa della promozione e della comunicazione[cite: 194, 195].

MM (Middle Management) Ruoli di gestione intermedi, come i Team Leader, che coordinano i membri del team[cite: 161, 180].

MoFR (Months of Financial Reserves) Un indicatore finanziario che misura per quanti mesi un'entità potrebbe sopravvivere utilizzando le sue riserve liquide, senza ulteriori entrate.

MXP (Membership Experience) L'area funzionale che si occupa della gestione dell'esperienza dei membri all'interno dell'organizzazione[cite: 205, 206].

OC (Organisational Committee) Un comitato organizzatore dedicato alla gestione di un evento o conferenza specifica[cite: 172, 191].

OGT (Outgoing Global Talent) Un programma di tirocinio professionale all'estero[cite: 193, 196].

OGV (Outgoing Global Volunteer) Un programma di volontariato all'estero[cite: 197, 198].

oGX (Outgoing Global Exchange) Termine generico che si riferisce a tutti i programmi di scambio in uscita (es. oGV, oGTa, oGTe), in cui i giovani italiani vanno all'estero[cite: 199, 200].

PnL (Profit & Loss) Conto economico che riassume i ricavi, i costi e le spese sostenute durante un periodo specifico.

TL (Team Leader) Il leader di un team all'interno di un'area funzionale[cite: 162, 181].

VP (Vice President) Un Vice Presidente, leader di una specifica area funzionale all'interno di un comitato (es. VP Finance & Legal).

XP (Experience) Esperienza, termine generico per indicare un'esperienza di scambio o all'interno dell'organizzazione[cite: 155, 174].

Appendice D

Schema del Database

Questa appendice descrive la struttura delle tabelle principali utilizzate dall'applicazione per la persistenza dei dati operativi nel database SQLite. Lo schema, sebbene distribuito su più file di database, segue una logica coerente, distinguendo tra i dati dei contratti oGX, quelli più articolati del mondo iGV e la gestione centralizzata dei protocolli.

Tabella: `contratti_ep` (Modulo oGX)

Questa è la tabella centrale per la gestione dei contratti dei programmi in uscita (Outgoing Exchange - oGX). Ogni riga rappresenta un singolo contratto generato attraverso il flusso multi-step e costituisce la "Single Source of Truth" per tutte le informazioni relative a un partecipante.

Tabella D.1: Schema della tabella `contratti_ep`.

Nome Colonna	Tipo Dati	Descrizione
`id`	INTEGER	Chiave primaria autoincrementante.
`unique_link`	TEXT	Stringa alfanumerica unica per l'accesso pubblico al form di compilazione.
`committee_key`	TEXT	Chiave identificativa del comitato (es. "roma_tre").
`contract_type`	TEXT	Tipo di contratto (es. "ogv", "ogta_short").
`status`	TEXT	Stato attuale del contratto (es. "Bozza", "Approvato", "In Revisione").
`filled_data_json`	TEXT	Campo JSON contenente la serializzazione di tutti i dati inseriti dall'utente nel form. Questo approccio garantisce flessibilità allo schema.
`contract_path`	TEXT	Percorso relativo del file PDF finale salvato sul server.
`created_at`	DATETIME	Timestamp della creazione del record nel database.

Tabella: `igv_contracts` (Modulo iGV)

Questa tabella, definita in `igv_contracts/models.py`, rappresenta un'evoluzione del modello dati. È progettata come una tabella **unificata** per gestire diversi flussi del mondo iGV (es. 'enabler', 'ep'). L'uso estensivo di campi JSON (`subject_json`, `payload_json`) le conferisce una grande flessibilità, permettendo di archiviare strutture di dati complesse e variabili (come i dettagli dei progetti o le anagrafiche dei partner) senza la necessità di alterare continuamente lo schema della tabella.

Tabella D.2: Schema della tabella `igv_contracts`.

Nome Colonna	Tipo Dati	Descrizione
`id`	INTEGER	Chiave primaria autoincrementante.
`igv_type`	TEXT	Tipologia di record ('enabler' o 'ep'), che permette di distinguere i diversi flussi gestiti da questa tabella.
`unique_link`	TEXT	Stringa alfanumerica unica per l'accesso pubblico.
`subject_json`	TEXT	Campo JSON per i dati del "soggetto" principale del contratto (es. anagrafica dell'EP o del Partner).
`payload_json`	TEXT	Campo JSON per i dati "oggetto" del contratto (es. dettagli del progetto, periodi di collaborazione, condizioni economiche).
`status`	TEXT	Stato attuale del flusso (es. 'Inviato', 'In attesa di revisione', 'Approvato').
`provisional_protocol`	TEXT	Numero di protocollo provvisorio, assegnato alla prima generazione del documento.
`final_protocol`	TEXT	Numero di protocollo definitivo, assegnato solo al momento dell'approvazione finale.
`files_generated_json`	TEXT	Campo JSON che archivia i percorsi dei file PDF generati dal sistema.
`files_uploaded_json`	TEXT	Campo JSON che archivia i percorsi dei file firmati e caricati dall'utente finale.
`created_at`	TEXT	Timestamp (ISO 8601) della creazione iniziale del record.
`submitted_at`	TEXT	Timestamp (ISO 8601) che segna il momento in cui l'utente ha caricato i documenti firmati.
`status_changed_at`	TEXT	Timestamp (ISO 8601) dell'ultima modifica dello stato da parte di un membro dello staff.

Tabella: `protocol_counters` (Gestore Centralizzato)

Questa tabella è il cuore del Blueprint `protocol_manager` e rappresenta la soluzione architetturale al problema delle *race condition* nella generazione dei numeri di protocollo. Ogni operazione di incremento viene gestita come una transazione atomica dal database, garantendo l'univocità e la sequenzialità dei numeri anche in caso di richieste concorrenti.

Tabella D.3: Schema della tabella `protocol_counters` del gestore centralizzato.

Nome Colonna	Tipo Dati	Descrizione
`committee_key`	TEXT	Chiave primaria. Identifica univocamente il comitato (es. "milano") oppure l'entità aggregata "expansion".
`ogx_counter`	INTEGER	Contatore progressivo per tutti i documenti della categoria oGX.
`igv_counter`	INTEGER	Contatore progressivo per tutti i documenti della categoria iGV (usato dal flusso Enabler).
`verbali_counter`	INTEGER	Contatore progressivo per tutti i verbali.
`last_updated`	DATETIME	Timestamp dell'ultimo aggiornamento di uno qualsiasi dei contatori per quel comitato.

Conclusioni e sviluppi futuri

Giunti al termine di questa analisi, è possibile tracciare un bilancio del lavoro svolto, valutandone i risultati, l'impatto sull'organizzazione e le lezioni apprese durante l'intero ciclo di vita del progetto.

Riepilogo dei Risultati Raggiunti

Il progetto ha raggiunto con successo gli obiettivi strategici prefissati, traducendosi nello sviluppo e nel deployment di una web application pienamente operativa che ha introdotto un cambiamento paradigmatico nei processi di AIESEC Italia. Sono stati conseguiti i seguenti risultati chiave: l'**automazione** dell'intero ciclo di vita della documentazione; la **centralizzazione** dei dati operativi in un sistema strutturato; la **standardizzazione** dei flussi di lavoro a livello nazionale; e l'introduzione di strumenti di **tracciabilità** e **analisi** per il monitoraggio della compliance e della performance finanziaria.

Valutazione dell'Impatto e dei Benefici per AIESEC

L'introduzione della piattaforma ha generato un impatto trasformativo sui processi operativi, portando benefici concreti e misurabili. In primo luogo, un drastico aumento dell' **efficienza operativa**, con una significativa riduzione delle ore-lavoro dedicate a compiti amministrativi manuali e ripetitivi. In secondo luogo, un netto miglioramento della **compliance** e della qualità dei dati, grazie all'adozione di flussi guidati e validati. Infine, si è registrato un aumento della **professionalità percepita** dell'organizzazione. L'applicazione ha di fatto liberato risorse umane, permettendo ai membri di

AIESEC di concentrarsi sulle attività a maggior valore aggiunto, in linea con la missione dell'organizzazione.

Lezioni Apprese e Considerazioni Personali

Questo progetto è stato, prima di tutto, un percorso di crescita personale intrecciato indissolubilmente con la mia esperienza in AIESEC. L'idea è nata nel novembre 2024, dopo la mia elezione a Vice Presidente Finance & Legal in AIESEC in Roma Tre. Fu in quel ruolo che toccai con mano la complessità e la frammentazione dei processi gestionali, una sfida che mi spinse, nel febbraio 2025, ad applicare per l'Entity Financial Board (EFB) nazionale con un'ambizione: introdurre un sistema che potesse semplificare la vita di tutti i miei successori.

L'idea iniziale era modesta: un semplice bot Telegram. Tuttavia, analizzando la portata del problema, divenne chiaro che serviva una soluzione più robusta e scalabile. Fu così che il progetto evolse nella creazione di una web application completa, con un'interfaccia dinamica e accessibile. Oggi, nel mio secondo mandato nel team EFB, continuo a sviluppare e ingrandire questa piattaforma, che è stata adottata da tutti i comitati di AIESEC Italia.

Da un punto di vista tecnico, la scelta di uno stack "leggero" si è rivelata vincente per uno sviluppo iterativo, e la progettazione di un'architettura modulare ha consolidato la mia comprensione di principi di ingegneria del software fondamentali. Ma la lezione più grande è stata vedere come un'esigenza sentita in prima persona possa trasformarsi in uno strumento di valore per un'intera comunità. Questo lavoro è la sintesi di quel percorso: la dimostrazione che la tecnologia, guidata da un bisogno reale e dalla passione, può diventare un potente abilitatore di cambiamento.

Bibliografia

[1] Alex Agronholm. Advanced python scheduler, 2025. Documentazione Ufficiale.

[2] AIESEC in Italy. Aiesec italia website, 2025. Sito Ufficiale Nazionale.

[3] AIESEC International. Aiesec global website, 2025. Sito Ufficiale Internazionale.

[4] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development. `http://agilemanifesto.org/`, 2001.

[5] C. Benoit. Gunicorn - green unicorn - a python wsgi http server for unix, 2025. Documentazione Ufficiale.

[6] Anton Burns. gspread - google spreadsheets python api, 2025. Documentazione Ufficiale.

[7] Canonical Ltd. Ubuntu - the world's most popular open source os, 2025. Sito Ufficiale.

[8] D. Richard Hipp. Sqlite home page, 2025. Sito Ufficiale.

[9] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, 1st edition, 2000.

[10] J. Klensin. Rfc 5321: Simple mail transfer protocol, 2008. Internet Engineering Task Force (IETF). Disponibile online.

[11] Kozea. Weasyprint - the awesome document factory, 2025. Documentazione Ufficiale.

[12] Matthew Le. Flask-login - flask extension that provides user session management, 2025. Documentazione Ufficiale.

[13] Jorj X. McKie and Ruikai Liu. Pymupdf documentation, 2025. Documentazione Ufficiale.

[14] Nginx. Nginx - web server, load balancer, reverse proxy, mail proxy and http cache, 2025. Sito Ufficiale.

[15] Caleb Porzio. Alpine.js - a rugged, minimal framework for composing javascript behavior in your markup, 2025. Sito Ufficiale.

[16] The Pallets Projects. Flask - web development, one drop at a time, 2025. Documentazione Ufficiale.

[17] The Pallets Projects. Flask documentation - message flashing, 2025. Documentazione Ufficiale.

[18] The Pallets Projects. Jinja - the pallets projects, 2025. Documentazione Ufficiale.

[19] The Pallets Projects. Modular applications with blueprints, 2025. Documentazione Ufficiale di Flask.

[20] Adam Wathan and Jonathan Reinink. Tailwind css - a utility-first css framework, 2025. Sito Ufficiale.