UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica e dell'Automazione

TESI DI LAUREA

Progettazione e realizzazione in Django di un portale per il monitoraggio delle finanze e degli investimenti

Design and implementation in Django of a portal for monitoring finances and investments

Relatore

Prof. Domenico Ursino

Candidato

Bianco Giacomo

ANNO ACCADEMICO 2023-2024

Sommario

La gestione delle proprie finanze è diventata una pratica sempre più importante e diffusa nell'attuale contesto economico, caratterizzato da incertezze e alta inflazione. In questa tesi sono state descritte la progettazione e l'implementazione di una piattaforma web concepita per supportare gli utenti nella pianificazione e gestione delle proprie finanze. Sviluppata utilizzando il framework Django, la piattaforma ha l'obiettivo di offrire agli utenti tutte le funzionalità necessarie per la gestione delle finanze personali, familiari e per il monitoraggio degli investimenti in strumenti finanziari. Il principale scopo è creare una piattaforma semplice e accessibile, pensata per avvicinare anche i neofiti al mondo della finanza, rendendo la gestione delle proprie risorse più intuitiva e alla portata di tutti. La realizzazione di questo progetto ha seguito un percorso ben definito, articolato in diverse fasi. Inizialmente, è stata condotta una fase di studio sul framework Django per acquisire le competenze necessarie allo sviluppo della piattaforma. Successivamente, è stata effettuata un'analisi delle esigenze degli utenti mediante la raccolta dei requisiti e la formulazione dei casi d'uso. Questi passaggi hanno fornito la base per la progettazione del database, elemento fondamentale per una gestione efficiente dei dati all'interno della piattaforma. Infine, l'ultima fase è stata l'implementazione effettiva.

Keyword: Gestione Finanziaria, Budgeting Personale, Budgeting Famigliare, Investimenti Finanziari, Piattaforma Web, Framework Django, Usabilità

Indice

Elenco delle figure

Introduzione

L'idea alla base di questo progetto nasce dalla crescente passione e dallo studio del mondo della finanza, nonché dall'osservazione delle dinamiche economiche che caratterizzano e influenzano la nostra quotidianità. Con il tempo, tali dinamiche sono diventate sempre più complesse e difficili da comprendere senza uno studio approfondito, mentre le loro variazioni si sono fatte più frequenti e imprevedibili. Tra le principali dinamiche riscontrabili nella quotidianità vi è l'aumento del costo della vita, che rappresenta uno dei principali fattori alla base del cambiamento delle esigenze economiche delle persone. Questo trend ha evidenziato la necessità di una gestione economica più attenta: è sempre più importante, al giorno d'oggi, tenere traccia delle proprie spese e risparmiare in vista di scenari futuri incerti. La capacità di gestire le proprie finanze in modo consapevole non solo aiuta a evitare problematiche economiche, ma consente di vedere il presente e il futuro con maggiore serenità. Il progresso tecnologico, che procede di pari passo con il peggioramento delle dinamiche economiche quotidiane, permette di creare e utilizzare strumenti più efficienti rispetto ai tradizionali metodi cartacei per il tracciamento delle spese. Questo scenario ha spinto a sviluppare una piattaforma dedicata al controllo delle proprie finanze, che consenta agli utenti di monitorare entrate, uscite, risparmi e investimenti, e di pianificare il proprio futuro economico. Il progetto si inserisce in un contesto di crescente digitalizzazione dei servizi finanziari, in un periodo in cui le soluzioni tecnologiche offrono nuove opportunità per semplificare e ottimizzare la gestione del denaro. L'obiettivo principale di questa tesi è progettare e sviluppare un sistema che renda la gestione delle finanze il più semplice possibile, rendendo più accessibile e comprensibile un tema che spesso viene percepito come complesso e persino elitario. In linea con l'obiettivo di semplicità e facilità d'uso, il sistema è stato implementato come piattaforma web, accessibile da qualunque dispositivo tramite Internet. La realizzazione della piattaforma si sviluppa attraverso diverse fasi, che includono lo studio, la progettazione e l'implementazione della piattaforma web. Inizialmente, si conduce uno studio approfondito del framework Django, per acquisire le competenze necessarie per un suo corretto utlizzo. Questa fase è fondamentale per comprendere come funziona e quali sono le funzionalità offerte da esso. A seguito di questa fase, si effettua un'analisi dei requisiti e dei casi d'uso. Durante questo processo, vengono identificati i bisogni degli utenti e le funzionalità principali che la piattaforma dovrà supportare. Sulla base dei requisiti raccolti, si definiscono i casi d'uso. Una volta raccolti i requisiti e definiti i casi d'uso, si avvia la progettazione del database; questo passaggio consiste nella definizione delle entità e delle loro relazioni, per poi tradurle nelle definitive tabelle. Infine, si avvia la fase di implementazione della piattaforma. Tutte queste fasi sono analizzate nei capitoli successivi; in particolare, questa tesi è strutturata come di seguito specificato:

- Nel Capitolo 1 verranno presentati i sistemi di gestione finanziaria, analizzandone le caratteristiche principali, proponendone una classificazione dettagliata e descrivendo le funzionalità distintive delle principali piattaforme attualmente disponibili sul mercato.
- Nel Capitolo 2 verrà introdotto Django, approfondendo il suo funzionamento e le sue logiche interne. Inoltre, si analizzerà il concetto di web framework, esplorando l'evoluzione storica e le motivazioni che hanno portato alla sua nascita e diffusione.
- Nel Capitolo 3 verranno approfondite le specifiche del progetto, iniziando con un'analisi dettagliata dei requisiti per comprende appieno le necessità degli utenti. Successivamente, si procederà con la descrizione dei casi d'uso. Questo ci aiuterà a comprendere le modalità di interazione degli utenti con il sistema e le funzionalità principali che il progetto deve offrire.
- Nel Capitolo 4 verrà approfondita la progettazione del database, partendo dall'individuazione e dall'analisi delle principali entità coinvolte, per poi esaminare le relazioni che le collegano. Infine, verrà illustrato come questa progettazione venga implementata concretamente all'interno di framework.
- Nel Capitolo 5 verrà illustrata l'implementazione della piattaforma, ponendo particolare attenzione su specifiche funzionalità rese disponibili dal framework utilizzato. Successivamente, verranno presentate le caratteristiche della piattaforma completata, includendo un manuale d'uso, arricchito da diverse immagini per guidare l'utente nell'utilizzo della piattaforma e facilitarne la comprensione.
- Nel Capitolo 6 verrà illustrata un'analisi di mercato sulla piattaforma, attraverso un confronto con le principali soluzioni esistenti sul mercato. Successivamente, sarà presentata un analisi SWOT e, infine, verranno esplorati i possibili sviluppi futuri del progetto.

CAPITOLO 1

Sistemi di monitoraggio delle finanze e degli investimenti

Negli ultimi anni, la gestione e il monitoraggio delle proprie finanze ha subito un importante cambiamento spinto dalle nuove tecnologie e dall'aumento della necessità di un maggiore consapevolezza finanziaria da parte degli individui e delle famiglie, passando da strumenti tradizionali, come registri cartacei e fogli di calcolo, a delle piattaforme digitali accessibili tramite internet. Questo capitolo si propone di fornire una panoramica sui sistemi di monitoraggio delle proprie finanze e degli investimenti, analizzando la varietà di soluzioni attualmente disponibili sul mercato. Partendo da uno spettro generale, verranno esaminati i principali approcci al budgeting personale e famigliare, nonché gli strumenti dedicati al monitoraggio degli investimenti. Successivamente, esploreremo le più popolari piattaforme esistenti, come PocketGuard, You Need A Budget (YNAB) e PocketSmith, analizzando le loro principali funzionalità e cercando di individuare le opportunità di miglioramento per sviluppare un innovativo portale basato sul framework Django.

1.1 Introduzione ai sistemi di gestione finanziaria

Le piattaforme di gestione finanziaria permettono molto più della semplice registrazione delle spese; esse infatti permettono di monitorare investimenti, spese ricorrenti, debiti e offrono anche funzionalità di pianificazione e controllo di specifici obbiettivi finanziari da raggiungere. Attraverso queste piattaforme è più facile e immediato prendere decisioni riguardo le proprie spese, i propri risparmi e propri investimenti.

Sul mercato sono presenti diversi tipi di piattaforme che possono essere raggruppante in 3 macro-sezioni:

- i sistemi di budgeting personale;
- i sistemi di budgeting famigliare;
- i sistemi di monitoraggio degli investimenti.

Da un lato i sistemi di budgeting personale e familiare permettono di tenere traccia delle entrate e delle uscite, di pianificare il risparmio e di stabilire obiettivi economici a breve e lungo termine per gli utenti e le loro famiglie. Dall'altro, i sistemi di monitoraggio degli investimenti aiutano a gestire i propri asset finanziari, visualizzando l'andamento dei portafogli azionari e permettendo agli utenti di avere un quadro chiaro dei propri investimenti.

1.1.1 I sistemi di budgeting personale

I sistemi di budgeting personale sono strumenti digitali progettati per permettere alle persone di organizzare e monitorare le loro finanze quotidiane. Attraverso questi strumenti, un utente può tracciare le proprie entrate, come lo stipendio, e le spese di tutti i tipi, incluse quelle pianificate in futuro o quelle ricorrenti, come bollette e affitti.

L'obiettivo principale di queste piattaforme è aiutare gli utenti a sviluppare abitudini di spesa consapevoli, sane e sostenibili, agevolando la comprensione di dove finiscono i propri soldi e su quali aree è possibile risparmiare.

Utilizzare questi sistemi rispetto ai metodi tradizionali ha diversi vantaggi, come la possibilità di definire obiettivi di risparmio mensili, destinare parte del budget a specifiche categorie o sottocategorie di spesa e di ricevere avvisi in caso si stiano per superare i limiti impostati. Questo approccio rende il controllo delle proprie finanze meno gravoso e spesso più gratificante.

Questi sistemi sono spesso dotati di funzionalità analitiche che offrono, sotto forma di report e grafici, una panoramica della propria situazione finanziaria. Questo rende facile individuare abitudini di spesa che potrebbero essere modificate o eliminate, migliorando così la gestione delle risorse economiche.

1.1.2 I sistemi di budgeting famigliare

I sistemi di budgeting famigliare sono strumenti digitali progettati per permettere alle famiglie di gestire le proprie finanze collettive, rendendo più semplice la pianificazione e il monitoraggio di entrate e uscite condivise. Attraverso questi strumenti diversi membri possono tenere traccia delle fonti di reddito della famiglia, come stipendi e altre entrate, e di organizzare le spese comuni, come bollette, affitti, spesa alimentare, l'istruzione dei figli e le attività ricreative condivise.

L'obbiettivo principale di queste piattaforme è promuovere una gestione condivisa e trasparente, superando il tabù legato al parlare e discutere di denaro in famiglia, che spesso rende difficile affrontare apertamente le decisioni economiche che coinvolgono tutti i membri.

Attraverso di esse si possono pianificare obbiettivi comuni di risparmio e fare previsioni a lungo termine per spese importanti, come vacanze e spese per gli studi dei figli, nonché essere pronti per spese future impreviste. Utilizzare questi sistemi rispetto ai metodi tradizionali ha gli stessi vantaggi dell'utilizzare i sistemi di budgeting personale; inoltre la possibilità di avere una visione chiara non solo sulle proprie spese, ma anche su quelle degli altri membri della famiglia rende più semplice e meno stressante la gestione economica di quest'ultima.

Come gli strumenti di budgeting personale, i sistemi di budgeting familiare permettono di visualizzare i dati finanziari sotto forma di grafici e report periodici, facilitando l'identificazione di eventuali aree di miglioramento.

Oltre al concetto tradizionale di famiglia, questi sistemi possono essere usati da un qualunque gruppo di individui che è in una situazione in cui è importante tenere traccia delle spese fatte da persone diverse, come, ad esempio, un gruppo di amici in vacanza, che al termine del viaggio deve suddividere equamente le spese sostenute.

1.1.3 I sistemi di monitoraggio degli investimenti

I sistemi di monitoraggio degli investimenti sono strumenti digitali progettati per permettere agli utenti di gestire e monitorare i propri asset finanziari. Attraverso questi strumenti è possibile visionare l'andamento del proprio portafogli di investimento, che può contenere diversi strumenti finanziari, come azioni, obbligazioni, fondi comuni di investimento, materie prime e criptovalute.

Attraverso le funzionalità di analisi e reporting, gli utenti possono osservare le performance dei loro investimenti, effettuare un confronto con l'andamento generale del mercato, valutare se stanno raggiungendo i propri obbiettivi, visionare l'andamento storico e identificare trend di mercato da sfruttare.

Rispetto ai metodi tradizionali, questi strumenti offrono vantaggi significativi, come la possibilità di ricevere notifiche sull'andamento in tempo reale dei propri investimenti e di rendere più comprensibile l'analisi dei parametri più importanti attraverso UI e UX più chiare e semplici.

Poiché queste piattaforme non consentono di eseguire direttamente operazioni di acquisto o vendita sul mercato, ma soltanto il monitoraggio, utilizzarle invece di accedere direttamente al broker permette di ridurre il rischio di decisioni impulsive, favorendo una gestione più riflessiva e consapevole degli investimenti.

1.2 Piattaforme esistenti

Disponibili online esistono diverse piattaforme che offrono soluzioni per gestire le proprie finanze personali, famigliari e gli investimenti. Di seguito, esploreremo alcune delle caratteristiche principali che ogni piattaforma presenta.

1.2.1 Caratteristiche principali delle piattaforme esistenti

La principali caratteristiche che ogni piattaforma presenta sono:

- *interfaccia utente facile e intuitiva*: le piattaforme devono presentare un'interfaccia utente chiara e facilmente utilizzabile; ciò permette agli utenti di accedere rapidamente a tutte le funzionalità della piattaforma;
- *personalizzazione*: in queste piattaforme è prevista la possibilità di personalizzare il tracciamento del proprio budget, con opzioni per modificare e creare le proprie personali categorie e sottocategorie di spesa;
- *gestione simulata dei conti*: queste piattaforme consentono di creare conti virtuali per simulare quelli reali, sui quali è possibile registrare transazioni, per monitorare entrate e uscite in modo preciso e dettagliato;
- *tracciamento delle spese*: gli utenti possono registrare e monitorare le spese quotidiane, categorizzandole e specificando dettagli, come la categoria di spesa, la data, il tipo di transazione e su quale conto è stata effettuata;
- *reportistica*: queste piattaforme generano report periodici che forniscono analisi delle spese, dei risparmi e dei rendimenti degli investimenti, includendo grafici e tabelle;
- *monitoraggio degli investimenti*: questi ultimi offrono funzionalità per visualizzare l'andamento del proprio portafogli d'investimento, potendo monitorare azioni, obbligazioni, fondi comuni e criptovalute, con aggiornamenti in tempo reale.
- *obiettivi di risparmio*: queste piattaforme consentono di impostare e monitorare degli obiettivi di risparmio specifici, decidendo la quantità di denaro da raggiungere e altri parametri opzionali, come la particolare categoria di spesa sulla quale si vuole risparmiare;
- *creazione di una famiglia virtuale*: queste piattaforme permettono di creare una o più famiglie virtuali con conti virtuali condivisi su cui diversi utenti registrati possono effettuare transazioni, facilitando la gestione delle finanze comuni;

- *notifiche e avvisi*: queste piattaforme inviano notifiche per ricordare scadenze di pagamento, significative variazioni in una particolare categoria di spesa, oppure il raggiungimento di un limite massimo di spesa impostato;
- *accessibilità multi-dispositivo*: queste piattaforme offrono una versione web, consentendo agli utenti di gestire le proprie finanze in qualsiasi luogo, in qualsiasi momento e qualunque dispositivo dispositivo;
- *reattività*: queste piattaforme sono responsive, ovvero si adattano automaticamente a diverse dimensioni di schermo e dispositivo;
- *aggiornamento dinamico dei contenuti*: a seguito di qualsiasi azione svolta dall'utente, queste piattaforme aggiornano la loro interfaccia senza la necessità di ricaricare la pagina.

1.3 PocketGuard

PocketGuard è una piattaforma che semplifica la gestione delle finanze e aiuta gli utenti a capire quanto denaro "disponibile" hanno in base alle loro entrate e alle spese fisse. Nella figura 1.1 viene riportata la pagina principale di pocketguard.

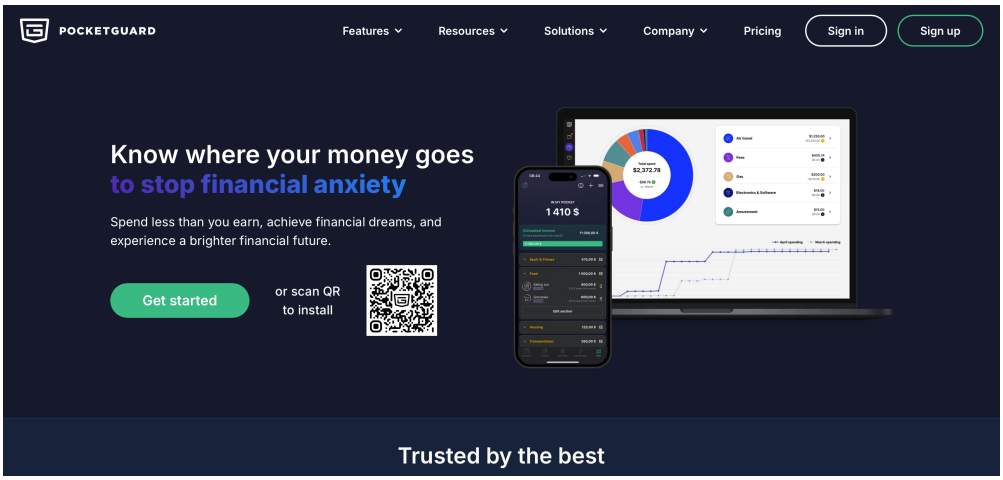

Figura 1.1: Pagina principale di PocketGuard

1.3.1 Funzionalità principali e interfaccia utente

Le principali funzionalità fornite da PocketGuard sono le seguenti:

- *metodo di budgeting semplice*: PocketGuard segue un approccio semplice e visivo per il budgeting, mostrando quanto denaro rimane dopo aver preso in considerazione le spese fisse, i risparmi e i pagamenti ricorrenti; nelle Figure 1.2 e 1.3 sono mostrati l'inserimento e la gestione delle transazioni in PocketGuard;
- *in my pocket*: una delle funzionalità distintive di PocketGuard è il concetto di "In My Pocket"; tramite questa funzionalità PocketGuard calcola automaticamente il denaro disponibile che l'utente può spendere senza intaccare i propri risparmi o superare il budget mensile;

- *abbonamenti ricorrenti*: PocketGuard rileva automaticamente gli abbonamenti e le spese ricorrenti, come le bollette o gli abbonamenti a servizi in streaming, aiutandoci a evitare spese non necessarie;
- *report finanziari*: la piattaforma fornisce report finanziari che aiutano a comprendere le tendenze delle spese e a identificare aree in cui è possibile risparmiare denaro, questi report sono mostrati nella Figura 1.4.

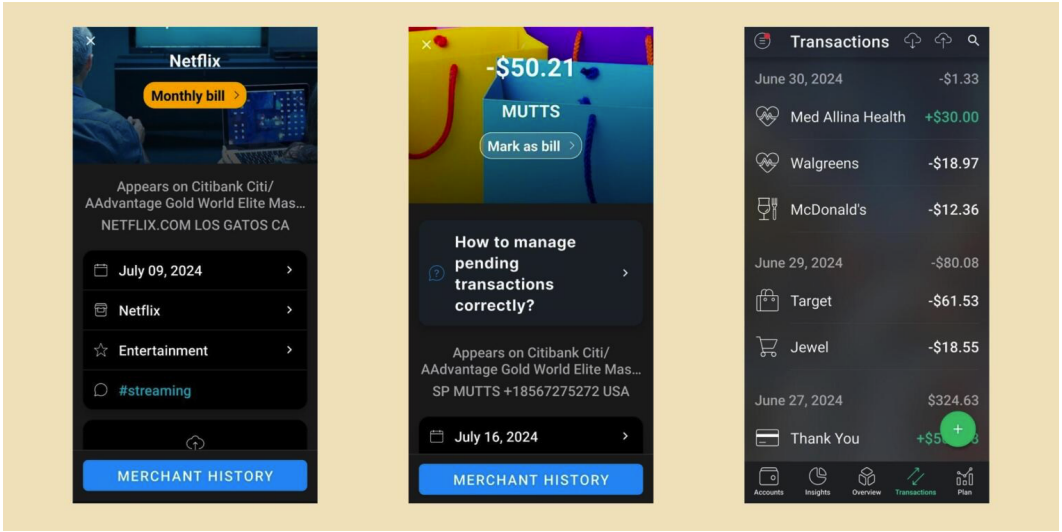

Figura 1.2: Inserimento di una transazione in PocketGuard

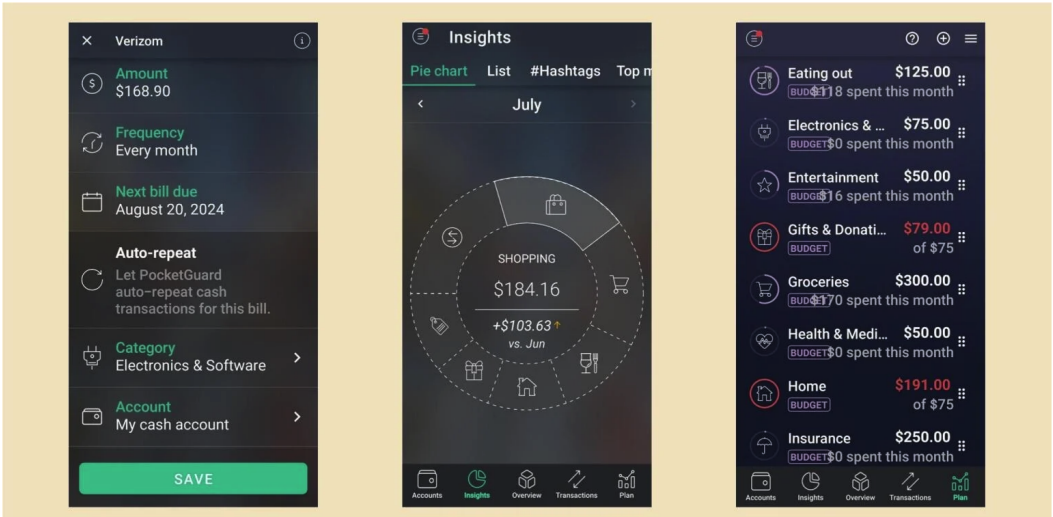

Figura 1.3: Gestione delle transazioni in PocketGuard

1.4 YNBA (You need a budget)

YNAB (You Need A Budget) è una popolare piattaforma di budgeting che aiuta gli utenti a prendere il controllo delle proprie finanze e a gestire il proprio denaro in modo più consapevole. Nella Figura 1.5 viene riportata la pagina principale di YNAB.

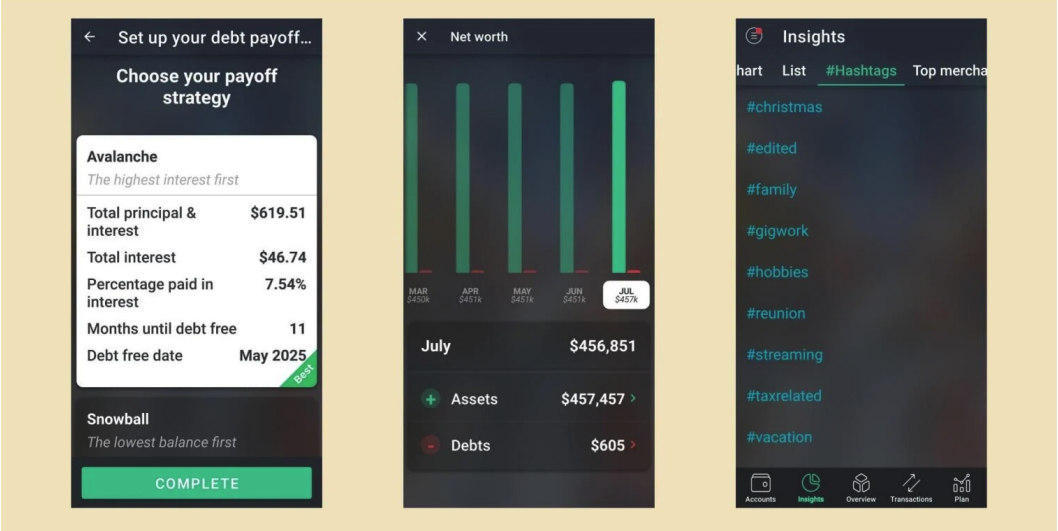

Figura 1.4: Report e grafici in PocketGuard

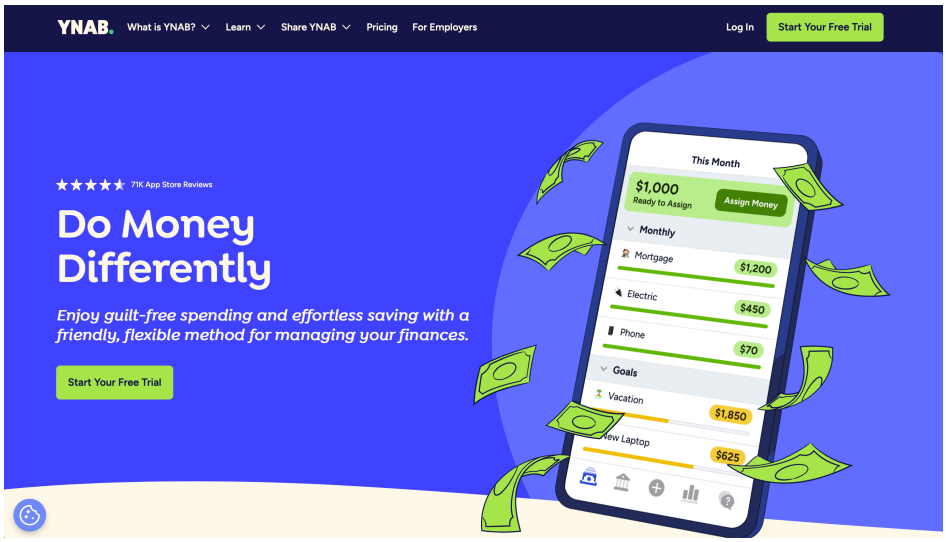

Figura 1.5: Pagina principale di YNAB

1.4.1 Funzionalità principali e interfaccia utente

Una volta effettuato l'accesso, viene presentata la dashboard principale (mostrata in Figura 1.6). Le principali funzionalità fornite da YNAB sono le seguenti:

- *metodo di budgeting*: YNAB si basa sul principio del "dollaro che lavora", secondo cui ogni dollaro guadagnato deve essere allocato per un obiettivo specifico (sia che si tratti di una spesa quotidiana, di un debito o di un risparmio);
- *creazione di un budget personalizzato*: YNAB consente di creare un budget per ogni categoria di spesa, e di allocare specifiche somme di denaro per ciascuna categoria;
- *obiettivi di risparmio*: YNAB consente di impostare obiettivi di risparmio per determinate categorie e ci aiuterà a determinare quanto risparmiare ogni mese per raggiungere l'obiettivo;
- *educazione finanziaria*: YNAB offre numerosi corsi e risorse educative per aiutare gli utenti a migliorare le proprie conoscenze in ambito finanziario;

- *supporto per i conti condivisi*: YNAB consente di gestire conti condivisi, rendendosi utile per le coppie o le famiglie che vogliono gestire insieme le proprie finanze, monitorando entrate e spese comuni.

Le funzionalità di obiettivi di risparmio, creazione di un budget personalizzato e supporto per i conti condivisi sono illustrate nella Figura 1.7.

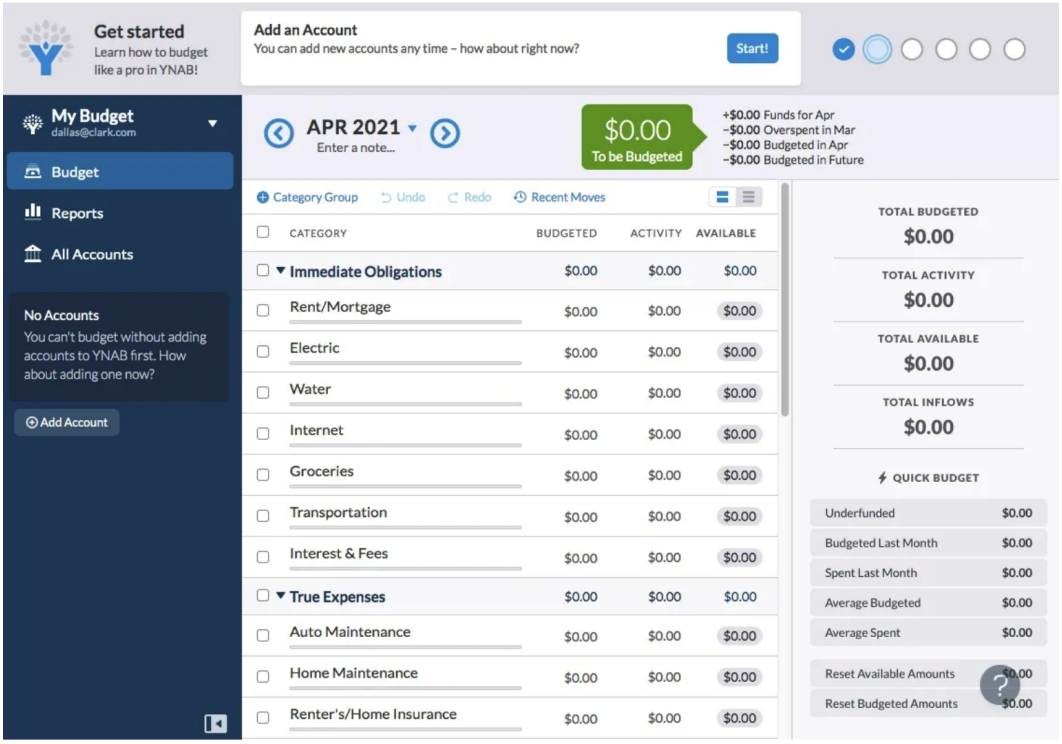

Figura 1.6: Dashboard di YNAB una volta effettuato l'accesso

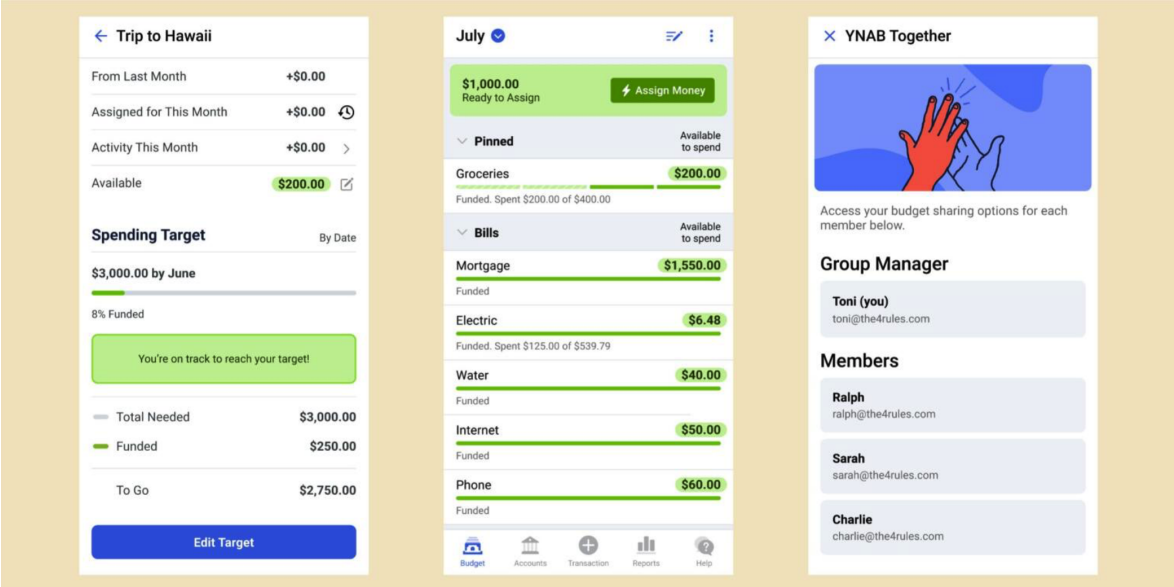

Figura 1.7: Illustrazione delle principali funzioni di YNAB

1.5 PocketSmith

PocketSmith è una piattaforma di budgeting e pianificazione finanziaria che si concentra sulla gestione delle spese quotidiane e sulla pianificazione a lungo termine, permettendo agli utenti di pianificare il proprio futuro finanziario attraverso previsioni basate su dati. Nella figura 1.8 viene riportata la pagina principale di PocketSmith.

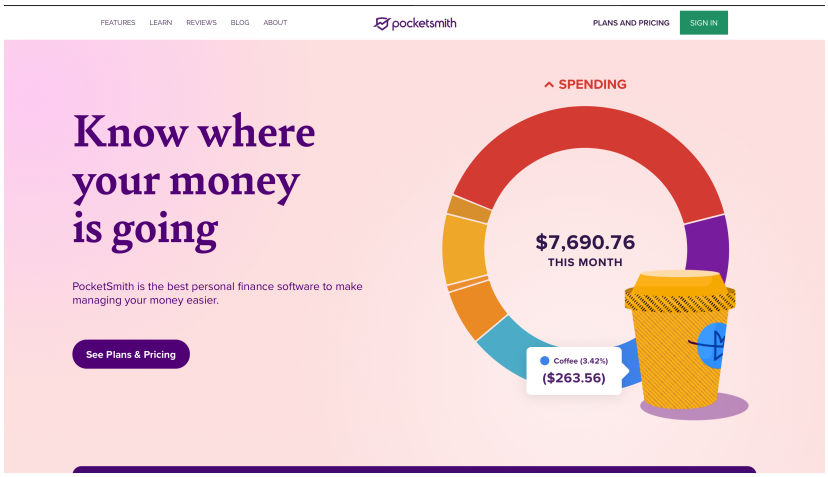

Figura 1.8: Pagina principale di PocketSmith

1.5.1 Funzionalità principali e interfaccia utente

Una volta effettuato l'accesso, viene presentata la dashboard principale (mostrata in Figura 1.9). Le principali funzionalità fornite da PocketSmith sono le seguenti:

- *pianificazione e previsioni a lungo termine*: PocketSmith offre strumenti di pianificazione a lungo termine che utilizzano previsioni basate sulle proprie abitudini di spesa, consentendo di avere una visione chiara delle proprie finanze future; gli utenti possono visualizzare scenari finanziari fino a 30 anni nel futuro;
- *calendario finanziario*: una delle funzionalità distintive di PocketSmith è il calendario delle transazioni, che permette di vedere entrate e uscite in un formato calendario, la funzione è mostrata nella Figura 1.10;
- *gestione del budget basata su scenari*: con PocketSmith è possibile creare budget per diversi scenari di vita, come "aumento delle spese" o "incremento dello stipendio", per vedere come variabili diverse potrebbero influenzare le proprie finanze;
- *conti multivaluta*: con PocketSmith è possibile simulare conti in diverse valute; ciò è particolarmente indicato per chi gestisce conti in valute diverse o effettua transazioni internazionali, come viaggiatori frequenti o professionisti che lavorano con mercati esteri;
- *category sankey diagram*: è un tipo di diagramma che permette di visualizzare il flusso di valori tra diverse categorie e sottocategorie in modo intuitivo. Questo strumento mostra graficamente come le risorse (ad esempio, il denaro) si spostano tra categorie principali e sottocategorie, la funzione è mostrata nella Figura 1.11.

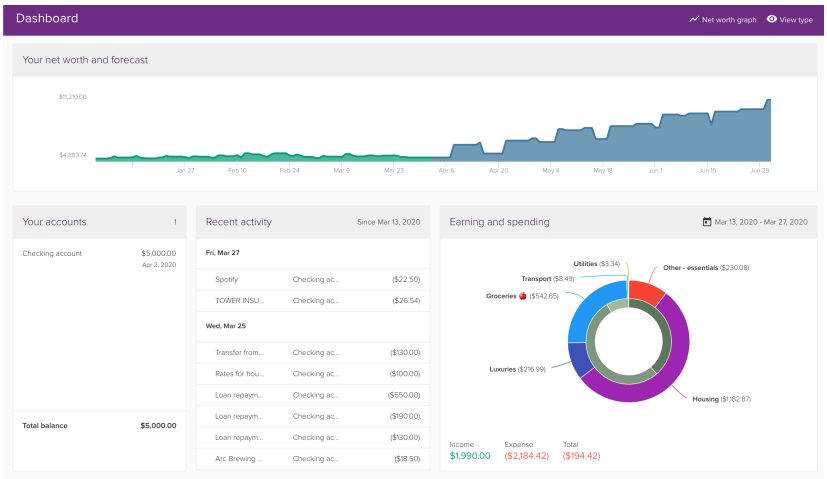

Figura 1.9: Dashboard di PocketSmith

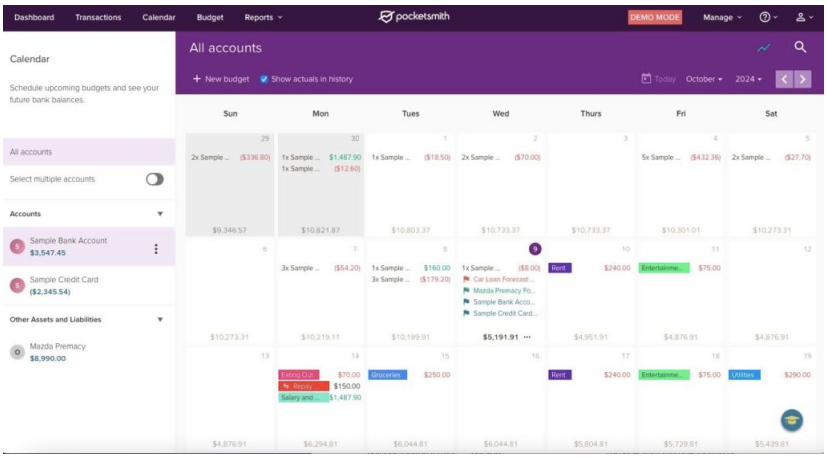

Figura 1.10: Funzione "Calendario finanziario" di PocketSmith

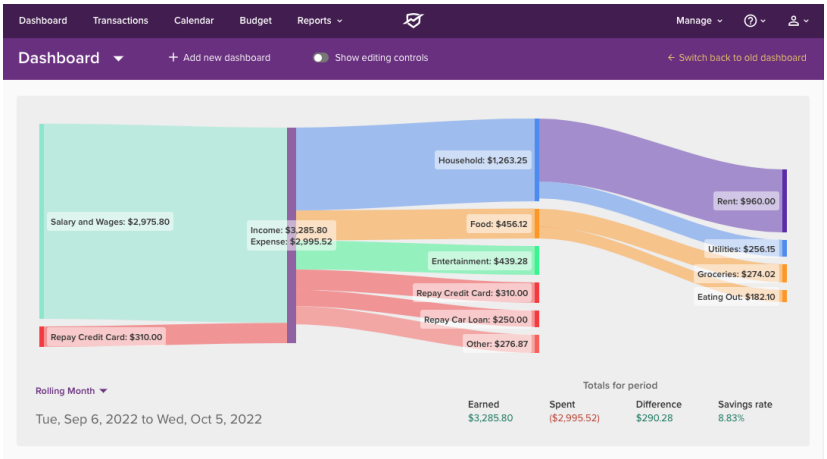

Figura 1.11: Funzione "Sankey diagram" di PocketSmith

CAPITOLO 2

Django

In questo capitolo verrà analizzato il framework Django, utilizzato per sviluppare la piattaforma descritta in questa tesi. Inizieremo con un introduzione storica riguardo la nascita di questa tecnologia, esplorando i motivi che hanno portato alla sua creazione, fino ad arrivare al concetto di web framework. Metteremo a confronto le principali soluzioni disponibili e descriveremo i pattern più comuni adottati da queste. Successivamente spiegheremo come funziona Django nello specifico e chi oggi utilizza questa tecnologia.

2.1 Introduzione storica

Inizialmente, quando gli utenti volevano accedere a dati o applicazioni dovevano essere fisicamente vicini a dei computer chiamati Mainframe, ovvero computer di grandi dimensioni progettati per elaborare un'enorme quantità di dati. L'accesso avveniva tramite terminale (dispositivo semplice che serviva solo per inserire comandi tramite una tastiera e visualizzare i risultati tramite uno schermo, un esempio è illustrato nella Figura 2.1) che era fisicamente collegato al Mainframe (Figura 2.2). Questi terminali dipendevano completamente dal computer centrale, il che rendeva necessario che gli utenti fossero presenti nel luogo dove il Mainframe era ospitato. Di fronte alla necessità di rendere i sistemi accessibili da postazioni remote nacque il modello client-server.

Figura 2.1: Immagine del terminale "VT100"

Figura 2.2: Immagine di una sala computer degli anni '70, con terminali collegati a un mainframe centrale

2.1.1 Il modello client-server

Il modello client-server (la cui struttura è riportata in Figura 2.3) è un modello di comunicazione che suddivide in due attori principali i partecipanti ai vari processi: da un lato i client che sono coloro che richiedono i dati, e dall'altra i server, che offrono determinati servizi. Un server in ambito web è un host (un computer collegato a una rete capace di fornire servizi o risorse ad altri dispositivi) che fa funzionare uno o più programmi. Lo schema di funzionamento di un modello client-server è il seguente:

1. il client invia una richiesta al server;
2. il server riceve la richiesta;
3. il server esegue il servizio richiesto;
4. il server invia una risposta e, se necessario, i dati rilevanti al client;
5. il client riceve la risposta e i dati dal server.

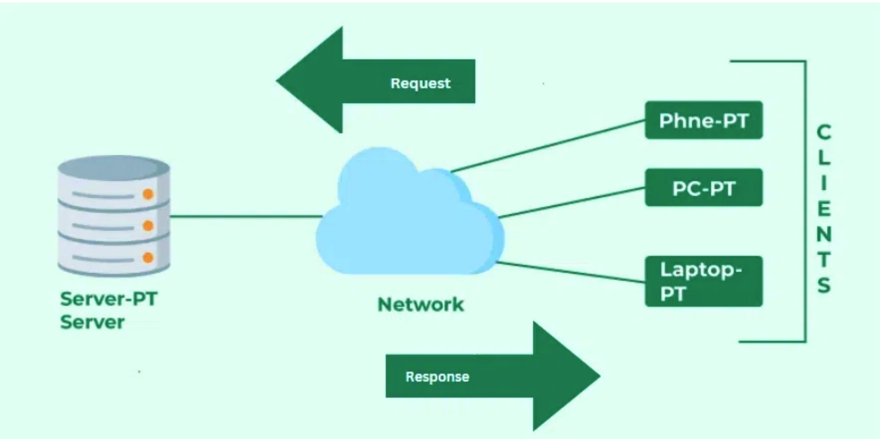

Figura 2.3: Modello client-server

Negli anni '80 utilizzare il modello client-server significava dover installare sui dispositivi un programma precompilato che fungeva da client. Ogni aggiornamento sul lato server necessitava di un nuovo aggiornamento lato client; questo significava che il client precompilato andava reinstallato o aggiornato manualmente ogni volta. Tale approccio creava problemi

significativi soprattutto per aziende con molti dipendenti distribuiti in più sedi, poichè era essenziale che tutti i dispositivi avessero la versione aggiornata del programma client.

Con l'avvento del World Wide Web negli anni '90, i browser hanno rivoluzionato l'utilizzo del modello client-server; infatti essi consentono agli utenti di accedere e utilizzare i programmi tramite una semplice interfaccia, ovvero le applicazioni web, senza la necessità di installare un programma client specifico su ogni dispositivo.

Le prime applicazioni web erano statiche, ovvero il client poteva solo visualizzare documenti HTML (Figura 2.4) senza interagire realmente con essi. HTML ("Hyper Text Markup Language") è la principale tecnologia utilizzata nelle applicazioni web; con questo linguaggio possiamo creare e impaginare documenti destinati alla visualizzazione su un browser. Inizialmente ogni azione da parte dell'utente portava al caricamento di un diverso documento HTML, e quindi un ricaricamento completo della pagina; oltre a questo non era possibile nessun altro tipo di interazione. L'introduzione di JavaScript (Figura 2.5) cambiò radicalmente questo scenario. JavaScript è un linguaggio di programmazione che gli sviluppatori utilizzano per realizzare pagine Web interattive senza coinvolgere necessariamente il server per ogni azione. Un classico esempio di interazione resa possibile da JavaScipt è la validazione dei dati in un modulo; infatti se l'utente inserisce un valore non valido in un campo del modulo JavaScript può immediatamente far visualizzare un messaggio di errore senza dover attendere una risposta dal server. Altri esempi di interazione sono:

- *eventi di click*: gli sviluppatori possono utilizzare JavaScript per eseguire funzioni quando un utente fa click su un elemento (come un pulsante o un link);
- *animazioni*: con JavaScript è possibile creare animazioni per migliorare l'esperienza visiva dell'utente;
- *modifica del contenuto della pagina*: con JavaScript è possibile, in risposta ad un evento, modificare dinamicamente il contenuto della pagina, ovvero senza il doverla ricaricare.

Figura 2.4: HTML5

Figura 2.5: JavaScript

Le applicazioni web inizialmente seguivano un modello sincrono per l'invio di richieste al server. Questo significava che ogni volta che un utente eseguiva un'azione che richiedeva nuovi dati (come inviare un modulo) l'intera pagina si ricaricava; questo non è ottimale perchè il ricaricamento causava un'interruzione visibile per l'utente dato che la pagina si bloccava temporaneamente in attesa dei nuovi dati; inoltre ogni richiesta di questo tipo comportava un certo tempo di latenza che variava a seconda della velocità della connessione e della velocità di risposta del server. Questo modello sincrono aumentava i tempi di attesa e riduceva la fluidità dell'esperienza utente.

Nel 2005, con l'introduzione di Ajax (Figura 2.6), ci fu un altro salto significativo nello sviluppo delle applicazioni web. Ajax (Asynchronous JavaScript and XML) indica una combinazione di tecnologie di sviluppo usate per creare pagine web dinamiche; infatti, essa ha permesso di creare applicazioni web asincrone migliorando l'interattività del client senza la necessità di ricaricare l'intera pagina. Con le applicazioni web asincrone le richieste di dati al server possono essere effettuate senza bloccare l'interfaccia utente; questo permetteva agli utenti di continuare ad utilizzare la pagina mentre i dati venivano caricati in background.

Figura 2.6: Ajax

Lo sviluppo di queste tecnologie ha portato ad avere applicazioni web sempre più complesse; in questo scenario gli sviluppatori cominciarono a trovare difficoltà nel gestire il codice in modo efficente; da qui nacquero i Web Framework, ovvero strumenti progettati per supportare e velocizzare lo sviluppo di applicazioni web.

2.2 Cos'è un web framework

Per definizione, un framework è una struttura di base per creare applicazioni e sistemi. È costituito da un insieme di componenti software che permette di ottimizzare, strutturare e automatizzare il codice. I framework sono versatili e possono essere usati per tante tipologie di progetti come, ad esempio, un'applicazione web (Web Framework). Per lo sviluppatore il framework rappresenta un risparmio di tempo e gli permette di concentrarsi sugli aspetti specifici dell'applicazione senza doversi preoccupare di implementare ogni singolo dettaglio tecnico da zero. Le principali funzionalità che un web framework mette a disposizione sono:

- *Separazione delle preoccupazioni*: la separazione delle preoccupazioni è un concetto fondamentale nello sviluppo software; questo concetto implica che diverse parti di un'applicazione devono essere gestite in modo indipendente, ognuna con le proprie responsabilità e funzioni. I web framework, per rispettare questo, principio adottano solitamente un pattern architetturale; il più comune tra questi è il pattern "MVC" che verrà illustrato nella prossima sezione.

- *Gestione delle rotte*: i web framework permettono in modo semplice di gestire le rotte; una rotta è una regola che collega un URL specifico ad una funzione da noi implementata: quando un utente inserisce un URL nel browser o fa click su un link, il framework controlla le rotte definite per determinare quale codice eseguire in risposta a quella richiesta.

- *Templating*: i web framework mettono a disposizione un sistema di templating che consente di definire e utilizzare dei template, ovvero dei documenti che hanno delle parti statiche (come layout e design) e parti dinamiche, in cui verrano inseriti dei dati variabili solo nel momento della generazione effettiva della pagina.

- *Integrazione con il database*: i web framework mettono a disposizione uno strumento per interagire con i database, ovvero un ORM (Object-Relational Mapping), che semplifica l'interazione dello sviluppatore con il database utilizzato.

- *Autenticazione*: i web framework offrono strumenti per gestire l'autenticazione degli utenti consentendo agli sviluppatori di implementare in modo semplice funzioni come registrazione, login e logout.

- *Middleware*: nei web framework è possibile utilizzare i middleware, strumenti software che svolgono un ruolo crucile nella gestione delle richieste e risposte (tra client e server) e nella gestione della sicurezza. Un utilizzo tipico dei middlewere è quello di garantire che solo gli utenti autenticati possano accedere a determinati URL.

2.3 Modello MVC e MTV

Molti web framework adottano un pattern architetturale chiamato Model-View-Controller (MVC). Un pattern è una soluzione riutilizzabile e documentata per un problema ricorrente in un determinato contesto di progettazione del software. Un pattern architetturale, nello specifico, è un modello con cui organizzare e strutturare le componenti principali del software. Questo modello è pensato per separare l'interfaccia utente dalla gestione dei dati e dalla logica applicativa dividendo la nostra applicazione web in 3 componenti:

- *Model*: è la componente dedicata all'accesso e alla gestione dei dati della nostra applicazione web. Il Model agisce da intermediario tra l'applicazione e il database non solo permettendo l'accesso fisico ai dati, ma anche creando un livello di astrazione che rende i dati compatibili con le altre componenti.
- *View*: questa componente si occupa della presentazione dei dati e della gestione dell'interazione con l'utente. Le View visualizzano le informazioni fornite dal Model permettendo all'utente di interagire con il sistema.
- *Controller*: è la componente che riceve i comandi dell'utente tramite le View e risponde eseguendo operazioni sul Model. Queste operazioni possono modificare lo stato dell'applicazione, aggiornando di conseguenza anche la View.

Nella Figura 2.7 è illustrato il funzionamento di questa architettura.

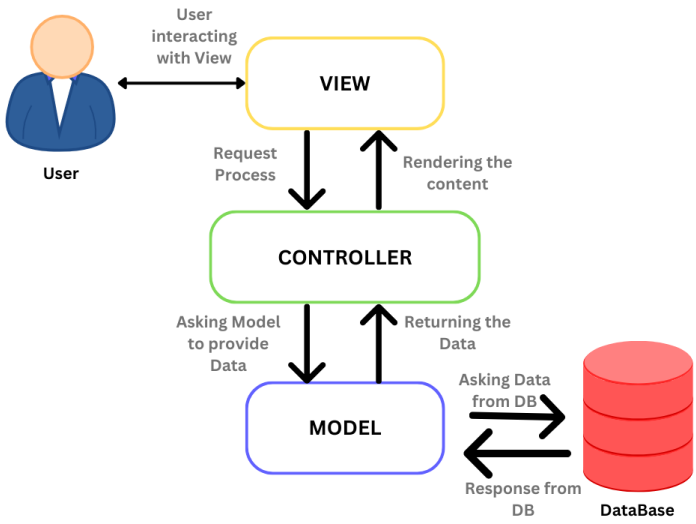

Figura 2.7: Modello MVC

Tutto parte dall'utente che interagendo con la View genera un input che viene inviato al Controller in ascolto sotto forma di richiesta; il controller elabora questa richiesta coinvolgendo (se necessario) il Model per accedere a modificare i dati nel database. Una volta completata l'elaborazione, il Controller invia una risposta alla View; quest'ultima può consistere in una nuova interfaccia utente oppure in dati organizzati in un formato ben definito (ad esempio in formato JSON); la View utilizzerà questi nuovi dati per aggiornare o modificare dinamicamente la propria interfaccia.

Django, invece, utilizza un altro pattern architetturale, ovvero il pattern Model-Template-View (MTV). Il pattern MTV è una diversa interpretazione del pattern MVC progettato

specificatamente per lo sviluppo di applicazioni web. La principale differenza consiste nell'interpretazione del Controller; infatti riprendendo la definizione del Controller nel pattern MVC, nel pattern MTV è il framework stesso che funge da Controller gestendo molte delle sue tipiche responsabilità, come il routing delle richieste, l'invocazione delle View e la gestione delle risposte. In questo modello, le componenti principali sono:

- *Model*: come nel modello MVC il Model è responsabile della gestione dei dati; esso si occupa del loro accesso, della loro manipolazione e dell'interazione con il database.
- *Template*: il Template corrisponde alla View nel modello MVC. Esso è responsabile della presentazione dei dati, ovvero definisce come i dati dovrebbero essere visualizzati all'utente; in pratica descrive come i dati vengono rappresentati nella pagina web.
- *View*: nel modello MTV la View è il componente che gestisce la logica dell'applicazione e l'interazione tra l'utente e il sistema. La View riceve le richieste dell'utente, interagisce con il Model per ottenere i dati necessari, e quindi seleziona il Template da visualizzare. La View è il componente che fa da ponte eseguendo operazioni sul Model e passando i dati al Template per la visualizzazione.

Da questa descrizione emerge un'altra differenza tra i due modelli, ovvero il ruolo della "View". Nel modello MVC la View è responsabile della presentazione e dell'interfaccia utente, mentre nel modello MTV la View è più orientata alla gestione della logica dell'applicazione mentre la presentazione vera e propria è affidata al Template.

Nella Figura 2.8 è illustrato come Django interpreta il pattern MTV.

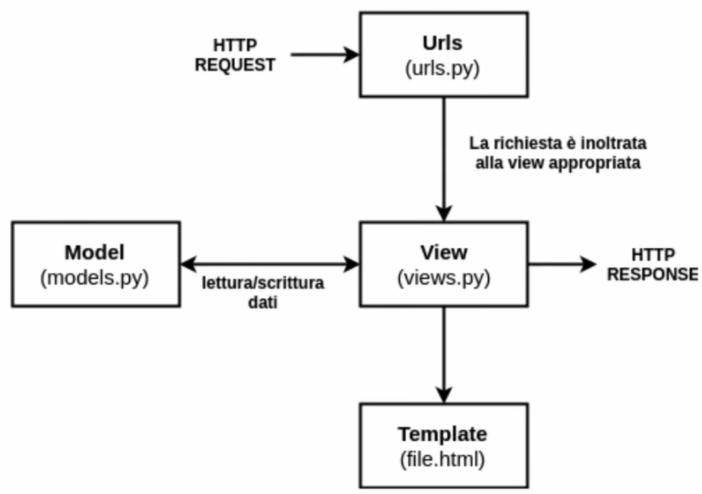

Figura 2.8: Modello MTV

Tutto parte con l'utente che, attraverso l'interazione con un'interfaccia, manda una richiesta HTTP che viene confrontata con gli URL definiti nell'URL dispatcher, ovvero una mappa che associa gli Url alle View. La richiesta viene, quindi, inoltrata alla View appropriata che dispone della logica per sapere a quali dati accedere tramite Model e che elabora una risposta. Questa risposta può essere un insieme di dati organizzati in un formato ben definito (risposta HTTP) o una nuova pagina HTML. In quest'ultimo caso la View delega la formattazione della risposta al Template.

2.4 I principali Web Framework

Ci sono diversi Web Framework disponibili sul mercato, la loro principale distinzione è tra framework front end e framework back end. In particolare:

- Il Framework front end è un insieme di strumenti, librerie e convenzioni che forniscono una struttura predefinita per lo sviluppo dell'interfaccia utente di un'applicazione web. Tuttavia dal nome capiamo che questi framework si focalizzano esclusivamente sulla parte visibile dell'applicazione (front end) delegando tutte le operazioni relative alla logica di business, alla gestione dei dati e al server al back-end, che deve essere sviluppato separatamente o integrato con altre tecnologie.
- il Framework back end è un framework back end è un insieme di strumenti, librerie e convenzioni che forniscono una struttura predefinita per lo sviluppo della logica di business, della gestione dei dati e delle funzionalità del server di un'applicazione web. Tuttavia dal nome capiamo che questi framework si focalizzano esclusivamente sulla parte non visibile dell'applicazione (back end) delegando la costruzione e la gestione dell'interfaccia utente al front end, che deve essere sviluppato separatamente o integrato con altre tecnologie.

Django è un web framework back end; infatti anche se include funzionalità per la gestione delle interfacce utente (come il sistema di templating) la sua principale area di applicazione riguarda la parte logica e server-side dell'applicazione (back end). Per questo motivo, di seguito, illustreremo i principali web framework back end utilizzati:

- *Django*: Django (Figura 2.9) è un framework open source basato su Python che adotta il pattern MTV; questo framework si distingue per la sua completezza offrendo strumenti per coprire vari aspetti dello sviluppo web, come un ORM (Django ORM) per l'interazione con il database e un sistema di templating per la gestione delle viste. Django adotta il principio "Don't Repeat Yourself" (DRY), ovvero evitare la ripetizione del codice attraverso strumenti come le funzioni di utilità e i template tag.
- *Laravel*: Laravel (Figura 2.10) è un framework open source che adotta il pattern MVC, questo framework è definito come "The PHP Framework for Web Artisans" ed è uno dei framework PHP più diffusi. Come Django anch'esso include componenti che coprono qualsiasi aspetto della progettazione di un applicazione web, come un ORM (Eloquent) e un sistema di templating (Blade).
- *Ruby On Rails*: Ruby On Rails (Figura 2.11) è un framework open source che funziona secondo il principio "Convention over Configuration" (CoC); quest'utlimo è un paradigma di progettazione software che cerca di ridurre il numero di decisioni che uno sviluppatore deve prendere durante l'utilizzo del framework. In pratica Ruby on Rails fornisce regole e convenzioni "standard" che coprono diversi aspetti della progettazione eliminando la necessità di configurare ogni dettaglio manualmente, ad esempio i file che contengono i controller, le viste e i modelli devono stare in cartelle specifiche con nomi standard.
- *Flask*: Flask (Figura 2.12) è un web framework open source basato su Python caratterizzato da flessibilità, leggerezza e semplicità d'uso. Fa parte della categoria dei microframework, ovvero framework minimali progettati per fornire solo le funzionalità di base necessarie per costruire un'applicazione.

Figura 2.9: Django

Figura 2.10: Laravel

Figura 2.11: Ruby On Rails

Figura 2.12: Flask

2.5 Come funziona Django

Un progetto in Django è un insieme di applicazioni e file di configurazione collegati tra loro in modo da comporre un applicazione web. Un'applicazione in Django è un modulo autonomo che viene creato per eseguire un compito specifico; questo approccio favorisce la modularità del progetto: è, infatti, possibile utilizzare la stessa applicazione in progetti diversi oppure importare applicazioni progetatte da altri. La struttura di un progetto Django (Figura 2.13) è organizzata in modo da avere al primo livello:

- la cartella principale, che contiene dei file che gestiscono gli aspetti fondamentali del workflow di Django;
- una cartella per ogni applicazione.

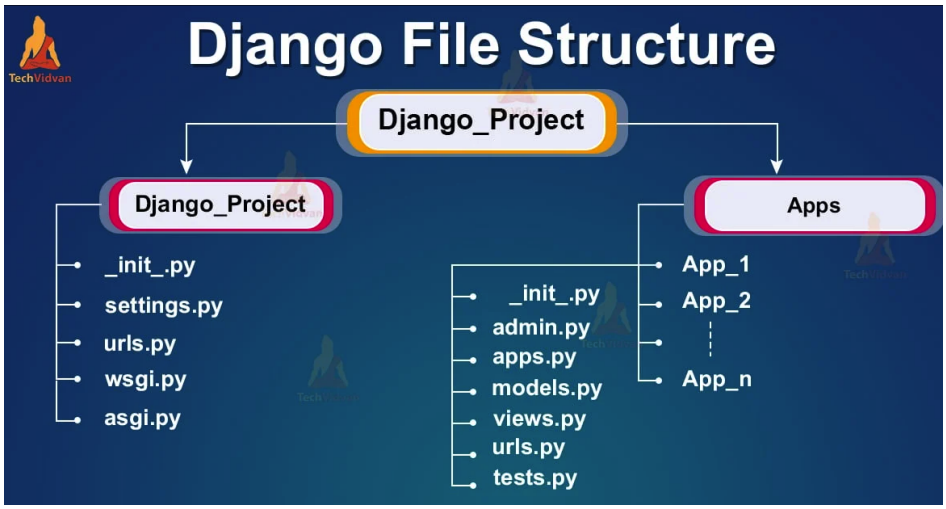

Figura 2.13: Struttura del primo livello di un progetto Django

Di seguito illustreremo nel dettagli quali sono i file della cartella principale:

- `manage.py`: questo è un file che viene creato automaticamente in ogni progetto Django, esso include una serie di istruzioni da eseguire attraverso Python che permettono di gestire gli aspetti più importanti del progetto, come creare un'applicazione, avviare il server di sviluppo o eseguire migrazioni.

- `__init__.py`: questo è un file che viene creato automaticamente in ogni directory Python che deve essere trattata come un modulo; un modulo è un file che contiene codice Python e che può essere importato e riutilizzato in altri programmi.
- `settings.py`: è un file che contiene i dettagli del nostro progetto Django come:
  - *Configurazione del database*: in questo file verifichiamo il tipo di database da utilizzare e le relative credenziali, il database predefinito è "sqlite3".
  - *Applicazioni*: in questo file verifichiamo la lista delle applicazioni che il nostro progetto utilizza. Una volta creata un'applicazione attraverso `manage.py` essa va registrata nella sezione `INSTALLED APPS` di questo file.
  - *Middlewere*: in questo file definiamo i middleware che gestiscono le richieste e le risposte. Un middlewere è un componente software che permette di gestire richieste e risposte aggiungendo delle funzionalità come l'autenticazione.
  - *File statici*: in questo file possiamo definire il percorso di base in cui il browser dovrà recuperare i file statici (file CSS, file JavaScript e file HTML).
- `urls.py`: questo file contiene la lista dei pattern URL della nostra applicazione; un pattern URL collega un percorso specifico (definito dall'utente) a una funzione vista che viene eseguita quando l'utente accede a quell'URL. Ogni pattern URL ha 3 parametri:
  - *url*: il percorso specifico da associare;
  - *funzione vista*: la funzione da richiamare quando si accede a quel percorso;
  - *nome del pattern*: un identificatore che consente di riutilizzare il pattern in altre parti del codice.

  Di solito ogni applicazione possiede il proprio file `urls.py`; per questo motivo il file `urls.py` della cartella principale spesso è utilizzato per includere i pattern URL presenti nei diversi file `urls.py` definiti nelle diverse app.
- `wsgi.py`: questo è un file che serve per connettere la nostra applicazione Django con un server web, ovvero quando distribuiamo il progetto in produzione. WSGI sta per Web Server Gateway Interface, ed è uno standard Python per la comunicazione tra i server web e le applicazioni web. Esso permette al server (come Apache o Nginx) di interagire con applicazioni Python come Django.
- `asgi.py`: questo file è simile a `wsgi.py` ma è usato per connettere la nostra applicazione Django a un server compatibile con ASGI (Asynchronous Server Gateway Interface). ASGI è un'evoluzione di WSGI che supporta la gestione di richieste asincrone.

Ogni volta che creiamo una nuova app, come già detto in precedenza, viene creata una cartella specifica che, oltre ai file `urls.py` e `__init__.py`, contiene altri file; questi utlimi sono:

- `models.py`: questo è il file dove definiamo sotto forma di classi i modelli della nostra applicazione, ovvero le tabelle del nostro database. In queste classi definiamo il design effettivo di ogni tabella, come il nome e il tipo di ogni singola colonna, eventuali vincoli e le relazioni tra tabelle.
- `admin.py`: una volta definito il nostro modello, possiamo registrarlo in questo file in modo da poter gestire i record della specifica tabella attraverso il pannello amministrazione. Quest'ultimo (Figura 2.14) è un'interfaccia generata da Django accessibile tramite un URL speficico che consente di gestire e amministrare i dati del database.

- `app.py` : questo è un file nel quale possiamo modificare i dettagli di configurazione della nostra app.
- `views.py`: questo è il file definiamo le view della nostra applicazione sotto forma di funzioni; ogni funzione rappresenta una vista che gestisce una specifica richiesta e restituisce una risposta. Queste funzioni rappresentano il secondo parametro da specificare nella definizione di un pattern URL nel file `urls.py`.
- `test.py`: questo è un file utilizzato per scrivere test automatizzati per verificare il corretto funzionamento dell'applicazione.

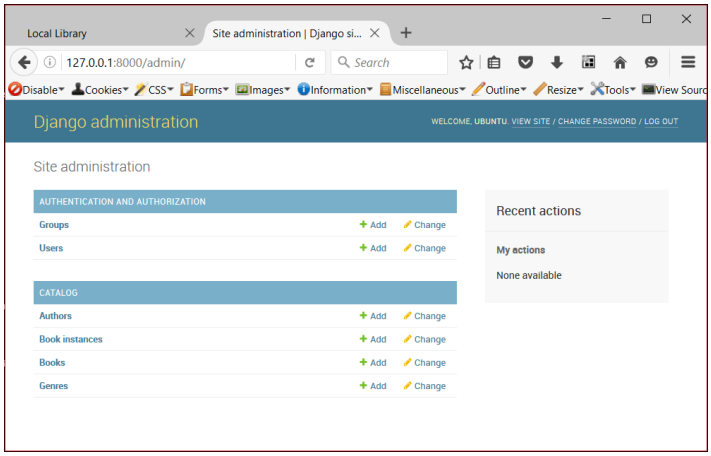

Figura 2.14: Pannello di amministrazione

In aggiunta ad essi ogni applicazione può contenere per convenzione altri elementi, ovvero:

- *cartella templates*: questa è una cartella nella quale sono contenuti i template dell'applicazione;
- *cartella static*: questa è una cartella nella quale sono contenuti i file statici dell'applicazione;
- `services.py`: questo è un file che viene comunemente utilizzato come convenzione per separare la logica di utilizzo di servizi esterni dal resto dell'applicazione (ad esempio API di terze parti), oppure per definire funzioni che verranno richiamate in diverse viste per evitare ridondanze nel codice.

2.6 Chi utilizza Django

Grazie alle sue carattestiche Django è uno dei web framework più utilizzati sul mercato, infatti, grazie alla sua flessibiltà e facilità d'uso, il suo utilizzo è cresciuto esponenzialmente negli ultimi anni. In dimostrazione di ciò la Figura 2.15 mostra quali sono i framework più utilizzati con dati aggiornati ad inizio 2024; invece, in Figura 2.16 viene mostrata l'evoluzione negli anni della quantità di "GitHub Stars" dei diversi web framework. Le "GitHub Stars" sono un metodo utilizzato dagli utenti della piattaforma GitHub per mostrare apprezzamento verso una determinata tecnologia.

Django è utilizzato da organizzazioni di tutte le dimensioni per le proprie applicazioni web, dalle startup alle aziende su larga scala. In Figura 2.17 sono riportate alcune aziende

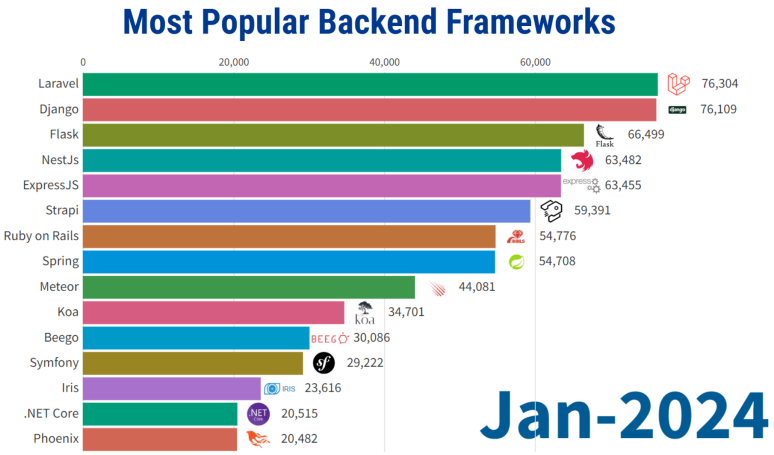

Figura 2.15: Grafico della popolarità dei back end framework

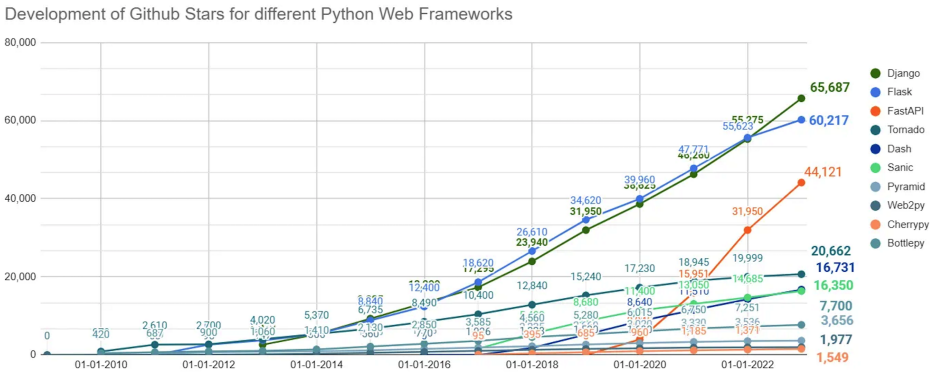

Figura 2.16: Grafico che mostra l'andamento negli anni delle "GitHub Stars"

che utilizzano Django nelle loro operazioni quotidiane sfruttando la sua capacità di gestire applicazioni robuste, sicure e scalabili.

Figura 2.17: Principali aziende che utilizzano Django

Secondo il sito "TheirStack.com" il numero di aziende che utilizza Django supera le 30.000.

CAPITOLO 3

Analisi dei requisiti

In questo capitolo verrà descritto Budget Nest, il sistema oggetto della presente tesi. Nella prima sezione illustreremo le funzionalità principali della piattaforma, definendo i diversi tipi di utenti (attori) che la utilizzeranno e analizzando le loro specifiche esigenze. Tali necessità saranno successivamente tradotte in requisiti di sistema, ovvero mediante una descrizione dettagliata delle caratteristiche e dei comportamenti che la piattaforma deve garantire. Verranno, poi, presentati i casi d'uso, che definiscono come gli attori interagiscono con il sistema, insieme alla matrice di mapping, che associa i requisiti alle funzionalità effettive del sistema.

3.1 Descrizione del progetto

Budget Nest è una piattaforma progettata per semplificare la gestione finanziaria personale e familiare, offrendo un servizio organizzato e intuitivo per ottimizzare e tenere sotto controllo le proprie finanze.

In merito al budgeting personale, *Budget Nest* offre diverse funzionalità:

- l'utente può registrare entrate e uscite;
- l'utente può creare categorie di spesa personalizzate;
- l'utente può pianificare spese future e impostare spese ricorrenti;
- l'utente può definire obiettivi di spesa per categorie specifiche.

Inoltre, la piattaforma permette di impostare piani di risparmio personalizzati, fornendo suggerimenti pratici per raggiungere obiettivi economici prefissati. Grazie a dei grafici dettagliati gli utenti possono monitorare l'andamento delle loro finanze in modo chiaro e intuitivo.

Sul fronte del budgeting famigliare, *Budget Nest* offre diverse funzionalità:

- la gestione condivisa di entrate e uscite famigliari;
- la possibilità di creare e monitorare conti comuni;
- la possibilità di gestire piani di risparmio personalizzati per tutta la famiglia.

Un'ulteriore caratteristica distintiva è il sistema di *gamification*, che permette ai membri della famiglia di partecipare a sfide di risparmio. Il vincitore di ciascuna sfida guadagna monete virtuali, utilizzabili per ottenere premi all'interno della piattaforma.

Budget Nest include anche una sezione dedicata al monitoraggio degli investimenti finanziari.

La piattaforma web è strutturata su due livelli:

1. *primo livello*: esso presenta una panoramica delle funzionalità disponibili e permette agli utenti di registrarsi o effettuare il login in pochi passaggi;
2. *secondo livello*: una volta che l'utente ha effettuato l'accesso, il secondo livello mette a disposizione tutte le funzionalità della piattaforma.

3.1.1 Analisi dei requisiti

L'analisi dei requisiti è una fase cruciale nel processo di sviluppo di un progetto software, in cui si cerca di comprendere le necessità e le aspettative degli stakeholder, ossia di tutte le persone, i gruppi o le organizzazioni che hanno un interesse nella realizzazione del progetto. In questa fase, l'obiettivo principale è raccogliere tutte le informazioni necessarie per definire chiaramente cosa il sistema dovrà fare e come dovrà comportarsi.

3.1.2 StakeHolder

Uno dei primi passi nella realizzazione di un progetto software è quello di identificare tutti gli stakeholder. Questi ultimi possono essere classificati in due categorie:

- *stakeholder esterni*: gli stakeholder esterni sono figure esterne alla realizzazione del progetto che possono, però, influenzarne lo sviluppo, essere interessate ai suoi risultati o beneficiarne direttamente;
- *stakeholder interni*: gli stakeholder interni sono figure che fanno parte dell'organizzazione o del team di progetto; essi sono direttamente coinvolti nella realizzazione.

In questo caso, il progetto è stato sviluppato interamente da un singolo individuo che assume, quindi, il ruolo di stakeholder interno. I principali stakeholder esterni sono gli utilizzatori finali della piattaforma, ovvero coloro che interagiranno direttamente con l'applicazione per gestire le proprie finanze. Questi tipi di stakeholder sono denominati "attori" e sono riportati di seguito:

- *utente singolo*: l'utente singolo utilizza la piattaforma per gestire le proprie finanze personali; le funzionalità messe a disposizione dal sistema sono quelle riguardanti la sezione di budgeting personale e monitoraggio degli investimenti finanziari;
- *membro della famiglia*: il membro della famiglia utilizza la piattaforma per gestire le finanze della propria famiglia in modo collaborativo; le funzionalità messe a disposizione della piattaforma sono quelle riguardanti la sezione di budgeting famigliare.

La Figura 3.1 mostra gli attori.

Una volta individuati gli stakeholder, il passo successivo consiste nella raccolta dei requisiti funzionali e non funzionali.

Figura 3.1: Attori

3.1.3 Requisiti funzionali

I requisiti funzionali descrivono in modo dettagliato cosa un sistema deve fare, ossia le funzionalità e i comportamenti che l'applicazione deve implementare per soddisfare le esigenze degli utenti e degli stakeholder. Nel nostro progetto, possiamo individuare i seguenti requisiti funzionali, raggruppati in base alle aree di utilizzo della piattaforma:

- *Registrazione e autenticazione*: questa sezione illustra i requisiti necessari per consentire agli utenti di accedere alla piattaforma in modo sicuro. In particolare, essa fornisce le seguenti funzionalità:
  - *creazione account*: gli utenti devono poter creare un account personale sulla piattaforma;
  - *autenticazione utente*: gli utenti devono poter autenticarsi con username e password;
  - *gestione password*: gli utenti devono poter modificare la propria password.
- *Gestione del Budget Personale*: questa sezione descrive i requisiti per la gestione delle finanze personali degli utenti. In particolare, , essa fornisce le seguenti funzionalità:
  - *crud entrate uscite*: l'utente singolo o il membro della famiglia deve poter registrare o eliminare entrate e uscite;
  - *crud sottocategorie di spesa*: l'utente singolo o il membro della famiglia deve poter creare, modificare ed eliminare sottocategorie di spesa personalizzate;
  - *crud spese future*: l'utente singolo o il membro della famiglia deve poter pianificare, modificare o eliminare spese future;
  - *visualizzazione grafici*: l'utente singolo deve poter visualizzare grafici sui propri andamenti finanziari;
  - *crud piani risparmio*: l'utente singolo o il membro della famiglia deve poter impostare, modificare o eliminare piani di risparmio personalizzati;
  - *crud conti*: l'utente singolo o il membro della famiglia deve poter creare, modificare o eliminare i conti su cui registrare le transazioni;
  - *monitoraggio investimenti*: l'utente singolo deve poter monitorare i propri investimenti finanziari;
  - *crud investimenti*: l'utente singolo deve poter inserire, modificare o eliminare una operazione di investimento;

  - *crud spese ricorrenti*: l'utente singolo deve poter impostare, modificare o eliminare le spese ricorrenti;
  - *visualizzazione storico transazioni*: l'utente singolo o il membro della famiglia deve poter accedere allo storico delle transazioni;
  - *crud obbiettivi di spesa*: l'utente singolo deve poter impostare, modificare o eliminare obiettivi di spesa mensili per categorie specifiche.

- *Gestione del budget familiare*: questa sezione definisce i requisiti per la gestione delle finanze familiari e la condivisione delle spese. In particolare, essa fornisce le seguenti funzionalità:

  - *accesso famiglia*: l'utente singolo deve poter entrare in una famiglia o crearne una nuova;
  - *monitoraggio spese famigliari*: il membro della famiglia deve poter monitorare le entrate e le uscite di tutta la famiglia;
  - *condivisione conti*: il membro della famiglia deve poter condividere conti comuni;
  - *crud sfide di risparmio*: il membro della famiglia deve poter creare, modificare ed eliminare delle sfide di risparmio su particolari categorie di spesa;
  - *partecipazione sfide di risparmio*: il membro della famiglia deve poter partecipare a delle sfide di risparmio su particolari categorie di spesa create da altri membri.
  - *acquisto premi*: l'utente deve poter ottenere premi virtuali grazie alle monete guadagnate dalle sfide di risparmio.

3.1.4 Requisiti non funzionali

I requisiti non funzionali specificano gli standard e le qualità che un sistema deve soddisfare per funzionare in modo efficace, concentrandosi su come il sistema opera, piuttosto che su cosa il sistema fa. Nel nostro progetto possiamo individuare i seguenti requisiti non funzionali:

- *semplicità e intuitività dell'interfaccia*: il sistema deve essere intuitivo e facile da usare, con un'interfaccia chiara e ben organizzata;
- *supporto multidispositivo*: l'interfaccia deve adattarsi correttamente a schermi di diverse dimensioni;
- *utilizzo del framework Django*: il sistema deve essere sviluppato utilizzando il framework Django.

3.2 Casi d'uso

Per modellare i requisiti funzionali individuati nel nostro progetto è stata utilizzata la tecnica di modellazione dei casi d'uso. Un caso d'uso può essere definito come una sequenza di azioni che un sistema esegue per soddisfare un obiettivo specifico di un attore. In altre parole, un caso d'uso rappresenta l'interazione tra un attore (nel nostro caso, l'utente singolo o un membro della famiglia) e il sistema per raggiungere uno scopo definito.

I casi d'uso del nostro progetto sono stati organizzati in package, suddivisi in base alle macro-aree funzionali della piattaforma. In particolare, i package che compongono il nostro sistema sono i seguenti:

- *Autenticazione e gestione utente*: questo package include i casi d'uso relativi alla gestione dell'accesso e dei dati personali degli utenti, applicabili sia per l'utente singolo che per il membro della famiglia. Esso copre le operazioni di registrazione, login, modifica delle credenziali e gestione del profilo. In particolare, esso prevede le seguenti funzionalità:
  - *Gestione accesso piattaforma*: questo caso d'uso descrive l'interazione del membro della famiglia oppure dell'utente singolo con il sistema nella fase di accesso alla piattaforma, che può avvenire tramite la registrazione oppure tramite login. In caso di esito positivo, l'utente autenticato viene reindirizzato alla dashboard principale della piattaforma. Il controllo delle credenziali avviene utilizzando il framework Django, sfruttando il sistema di autenticazione integrato.
  - *Gestione credenziali utente*: questo caso d'uso descrive l'interazione del membro della famiglia oppure dell'utente singolo con il sistema quando quest'ultimo vuole modificare dei dati personali. L'utente dalla dashboard accede alle impostazioni ed entra nella sottosezione relativa alla modifica del profilo. In caso di effettiva modifica, una conferma viene mostrata all'utente al termine dell'operazione e i dati aggiornati vengono salvati nel database utilizzando le funzionalità ORM di Django.

  La Figura 3.2 mostra il diagramma dei casi d'uso relativi a questo package.

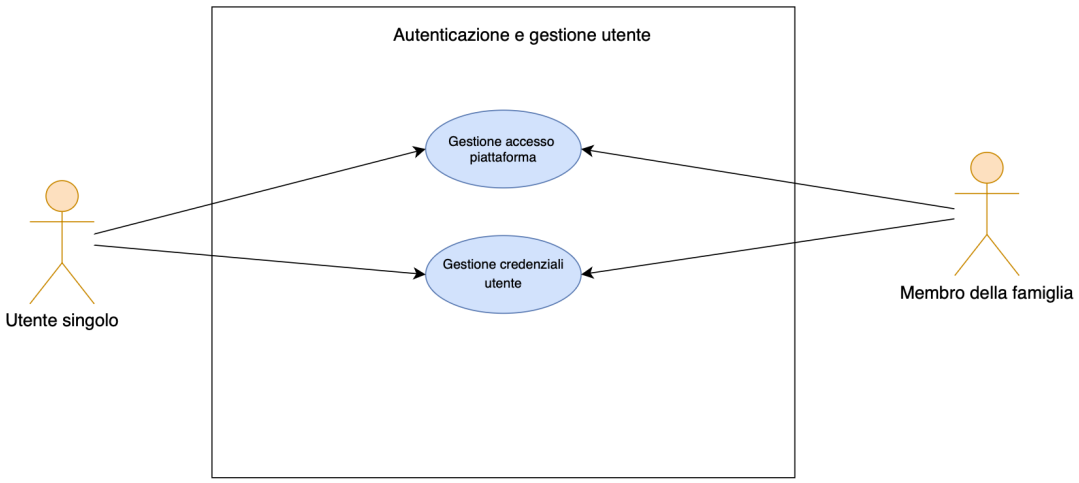

Figura 3.2: Casi d'uso relativi al package "Autenticazione e gestione utente"

- *Finanze personali*: questo package include i casi d'uso relativi alla gestione delle finanze personali, coprendo tutte le possibili interazioni dell'utente singolo nella sezione del budgeting personale. I casi d'uso che hanno come attore anche il membro della famiglia sono casi d'uso la cui logica non cambia rispetto all'utente singolo. In particolare, i casi d'uso inclusi in questo package sono i seguenti:
  - *Gestione obbiettivi di spesa*: questo caso d'uso descrive l'interazione dell'utente singolo con il sistema per la gestione degli obiettivi di spesa. Nella sezione relativa al budgeting personale, l'utente può modificare o eliminare un obiettivo esistente, oppure creare un nuovo obiettivo specificando nome, categoria di riferimento e scadenza. Ogni volta che un utente singolo o un membro della famiglia inserisce

una transazione, il sistema verifica se esiste un obiettivo di spesa associato alla categoria della transazione appena inserita. In caso affermativo, aggiorna il progresso dell'obiettivo e modifica dinamicamente la sua rappresentazione grafica.

- *Visualizzazione e interazione grafici*: questo caso d'uso descrive l'interazione dell'utente singolo con il sistema per la visualizzazione e l'interazione con i grafici relativi alle spese. Nella sezione dedicata al budgeting personale, l'utente può visualizzare grafici a torta che mostrano la distribuzione delle spese per categoria in diversi periodi di tempo. L'interazione prevede la possibilità di riformulare lo stesso grafico a torta escludendo una o più categorie di spesa selezionate. Ogni volta che un utente singolo o un membro della famiglia inserisce una transazione, il sistema aggiorna e rigenera dinamicamente questi grafici, riflettendo in tempo reale l'andamento delle spese.
- *Gestione entrate e uscite*: questo caso d'uso descrive l'interazione dell'utente singolo oppure del membro della famiglia con il sistema per la registrazione di una nuova transazione. Accedendo alla sezione dedicata al budgeting personale, oppure alla sezione relativa alla gestione famigliare, l'utente può inserire una nuova transazione specificando diversi parametri, tra cui il tipo di transazione. In base al tipo di transazione selezionata, il sistema richiede parametri aggiuntivi; in caso di trasferimento tra conti l'utente deve selezionare il conto di destinazione, mentre per una transazione periodica l'utente deve specificare la frequenza.
- *Gestione sottocategorie di spesa*: questo caso d'uso descrive l'interazione del membro della famiglia, oppure dell'utente singolo, con il sistema per la gestione delle sottocategorie di spesa. Accedendo alla dashboard, l'utente può navigare nelle impostazioni e selezionare la sottosezione relativa alle sottocategorie di spesa. Da questa sezione l'utente può modificare o eliminare una sottocategoria creata in precedenza oppure crearne una nuova.
- *Gestione piani risparmio*: questo caso d'uso descrive l'interazione dell'utente singolo con il sistema per la gestione dei piani di risparmo. Accedendo alla sezione dedicata al budgeting personale l'utente può modificare o eliminare un piano di risparmio già esistente, oppure crearne uno nuovo specificando obbiettivo, conto di riferimento e scadenza. Ogni volta che un utente singolo o un membro della famiglia registra una nuova transazione il sistema controlla se esiste un piano di risparmio relativo al conto della transazione appena inserita. In caso affermativo il sistema aggiorna il progresso attuale del piano cambiando dinamicamente la sua rappresentazine grafica.
- *Gestione storico transazioni*: questo caso d'uso descrive l'interazione dell'utente singolo, oppure del membro della famiglia, con il sistema per la consultazione dello storico delle transazioni. Accedendo alla sezione dedicata al budgeting personale, oppure alla sezione relativa al budgeting famigliare, l'utente può visionare lo storico delle transazioni selezionando un conto specifico. Nel caso in cui una transazione sia stata registrata dall'utente, quest'ultimo ha la facoltà di eliminarle.
- *Gestione conti*: questo caso d'uso descrive l'interazione dell'utente singolo con il sistema per la gestione dei conti. Accedendo alla sezione dedicata al budgeting personale, l'utente singolo può creare un nuovo conto specificando nome, tipo di conto e saldo iniziale.

La Figura 3.3 mostra il diagramma dei casi d'uso relativi a questo package.

- *Finanze famigliari*: questo package include i casi d'uso relativi alla gestione delle finanze famigliari, coprendo tutte le possibili interazioni del membro della famiglia nella

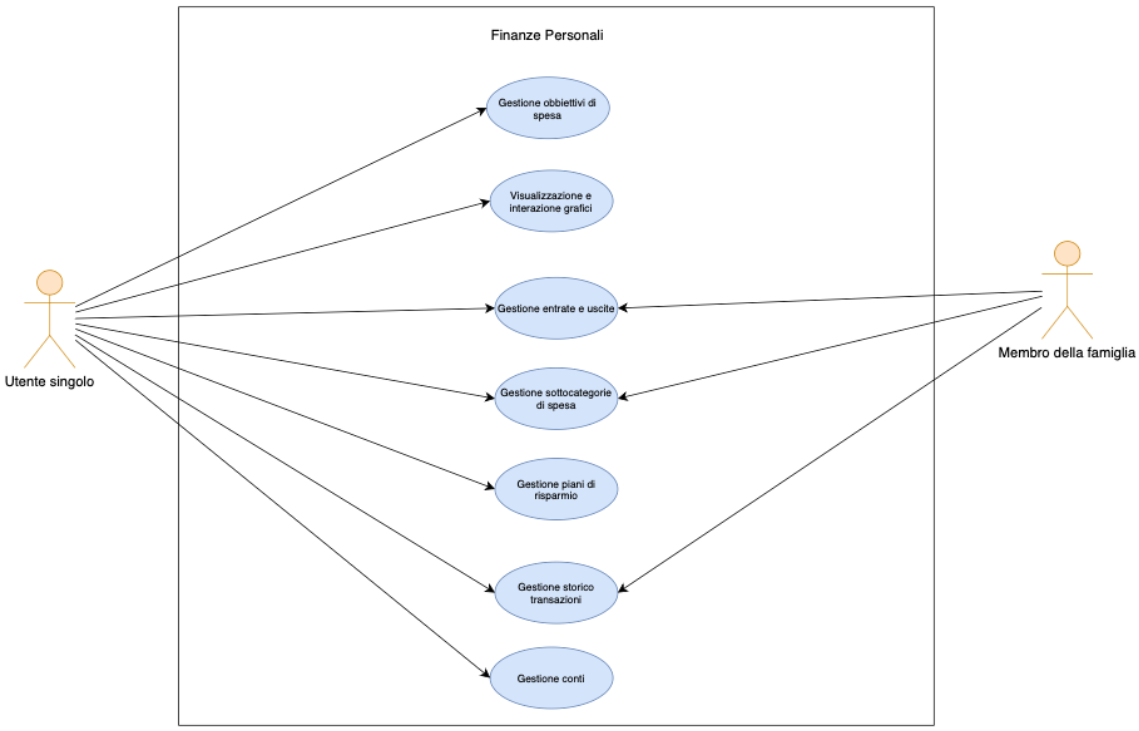

Figura 3.3: Casi d'uso relativi al package "Finanze personali"

sezione del budgeting famigliare. In particolare, i casi d'uso inclusi in questo package sono i seguenti:

- *Creazione famiglia*: questo caso d'uso descrive l'interazione del membro della famiglia con il sistema per la creazione di una famiglia. Accedendo alla sezione relativa al budgeting famigliare, l'utente può creare una nuova famiglia specificando il nome. In caso di esito positivo il sistema rende disponibile nella sezione di accesso alle famiglie quella appena creata.
- *Gestione accesso gruppo famigliare*: questo caso d'uso descrive l'interazione del membro della famiglia con il sistema per l'accesso ad una famiglia. Accedendo alla sezione relativa al budgeting famigliare, l'utente può entrare in una famiglia inserendo il codice fornito da un altro membro. In caso di esito positivo il sistema rende disponibile, nella sezione di accesso alle famiglie, quella in cui l'utente è appena entrato.
- *Gestione conti condivisi*: questo caso d'uso descrive l'interazione del membro della famiglia con il sistema per la gestione di un conto condiviso. Accedendo alla sezione relativa al budgeting famigliare, l'utente può modificare o eliminare un conto già creato da lui o da un altro membro della famiglia, oppure crearne uno nuovo. In caso di creazione il sistema procede ad intestare il conto a tutti i membri della famiglia, rendendolo disponibile per la registrazione delle transazioni da parte di ogni componente.
- *Gestione piani di risparmio condivisi*: questo caso d'uso descrive l'interazione del membro della famiglia con il sistema per la gestione dei piani risparmo condivisi. Accedendo alla sezione dedicata ad una famiglia l'utente può modificare o eliminare un piano risparmio già esistente , oppure può crearne uno nuovo specificando obbiettivo, conto di riferimento e scadenza. In caso di creazione, il sistema

attribuisce il piano di risparmio a tutti i membri che hanno intestato il conto di riferimento. Ogni volta che un utente singolo o un membro della famiglia registra una nuova transazione il sistema controlla se esiste un piano risparmio relativo al conto della transazione appena inserita. In caso affermativo il sistema aggiorna il progresso attuale del piano cambiando dinamicamente la sua rappresentazine grafica.

La Figura 3.4 mostra il diagramma dei casi d'uso relativi a questo package.

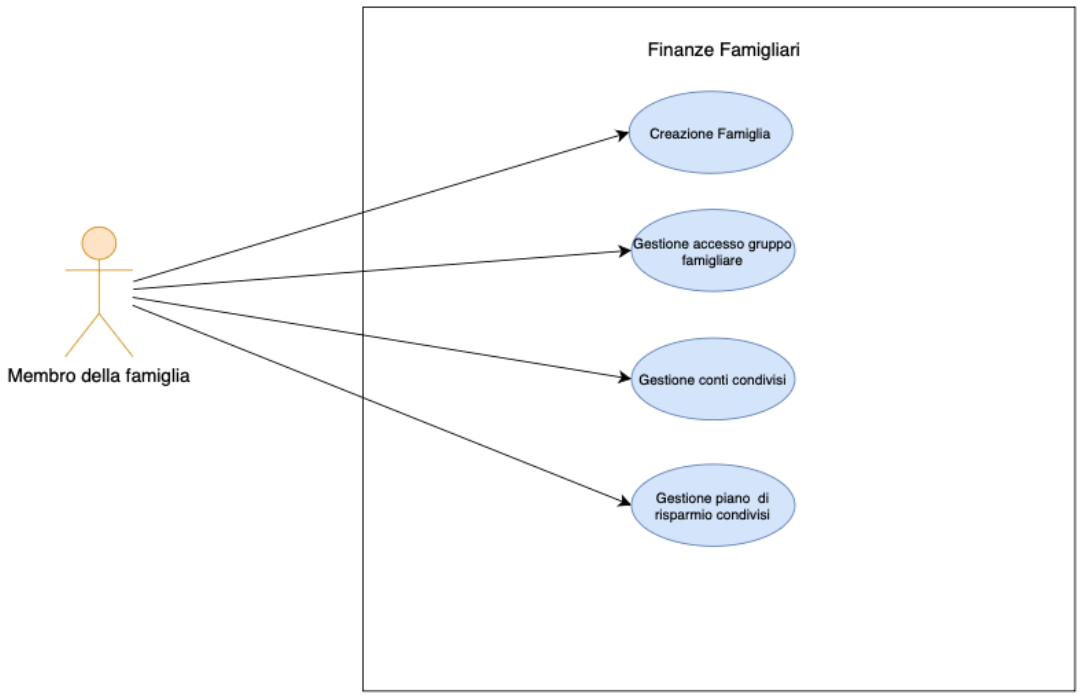

Figura 3.4: Casi d'uso relativi al package "Finanze famigliari"

- *Sfide e premi*: questo package include i casi d'uso relativi alla gestione delle sfide di risparmio e all'acquisizione di premi. In particolare, esso prevede i seguenti casi d'usp:
  - *Gestione premi*: questo caso d'uso descrive l'interazione del membro della famiglia con il sistema per l'acquisto dei premi messi a disposizione dalla piattaforma. L'utente dalla dashboard accede alle impostazioni ed entra nella sottosezione relativa all'acquisto dei premi. Nel caso in cui l'utente possegga abbastanza monete, il sistema gli assegna il premio selezionato.
  - *Gestione sfide di risparmio*: questo caso d'uso descrive l'interazione del membro della famiglia con il sistema per la gestione delle sfide di risparmio. Entrando nella sezione relativa a una famiglia, l'utente può modificare o eliminare una sfida che lo riguarda, oppure creare una nuova sfida specificando lo sfidante (un altro membro della famiglia), la categoria di spesa e la data di scadenza. Ogni volta che un utente singolo o un membro della famiglia inserisce una transazione, il sistema verifica se esiste una sfida per la categoria della transazione appena inserita. In caso affermativo, il sistema aggiorna il progresso attuale della sfida, modificando dinamicamente la sua rappresentazione grafica. Al termine di ogni giornata, il sistema verifica se ci sono sfide concluse e, in caso positivo, assegna delle monete all'utente vincitore.

La Figura 3.5 mostra il diagramma dei casi d'uso relativi a questo package.

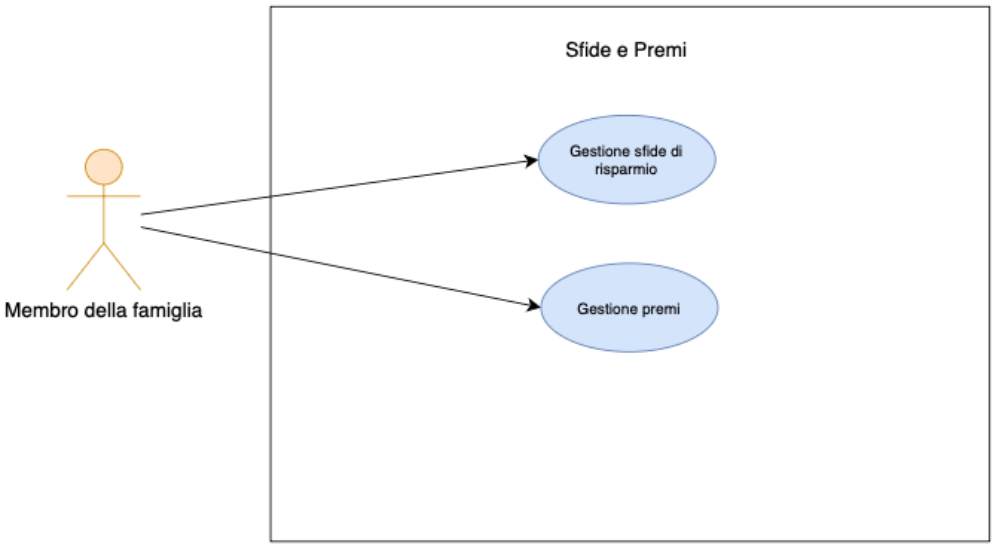

Figura 3.5: Casi d'uso relativi al package "Sfide e premi"

- *Investimenti*: questo package include i casi d'uso relativi al monitoraggio degli investimenti finanziari. In particolare, esso prevede i seguenti casi d'uso:
  - *Acquisto azioni*: questo caso d'uso descrive l'interazione dell'utente singolo con il sistema per l'acquisto di azioni. Entrando nella sezione relativa agli investimenti, l'utente può aggiornare o registrare una posizione specificando il ticker dell'azienda (simbolo univoco che identifica una specifica azione, titolo o strumento finanziario in borsa), il numero di azioni e il conto relativo. Il sistema, dopo aver verificato la disponibilità di liquidità del conto specificato, registra l'aquisto e aggiorna il portafoglio utente.
  - *Vendita azioni*: questo caso d'uso descrive l'interazione dell'utente singolo con il sistema per la vendita di azioni. Entrando nella sezione relativa agli investimenti, l'utente può registrare una vendita specificando il ticker dell'azienda e il numero di azioni. Il sistema registra la vendita e aggiorna il portafoglio utente e la liquidità del conto relativo alla posizione venduta.
  - *Visualizzazione andamento portafoglio*: questo caso d'uso descrive l'interazione dell'utente singolo con il sistema per la visualizzazione dell'andamento del proprio portafoglio di investimenti. Entrando nella sezione relativa agli investimenti, l'utente può visualizzare una panoramica dettagliata delle proprie azioni e delle relative performance. I dettagli di ogni performance includono il valore corrente, i guadagni o le perdite realizzate, il prezzo medio di carico (PMC) e l'andamento nel tempo attraverso un grafico. Ogni giorno il sistema aggiorna dinamicamente i dati selezionando il prezzo di chiusura del giorno precedente relativo agli strumenti finanziari che l'utente ha nel portafoglio.

La Figura 3.6 mostra il diagramma dei casi d'uso relativi a questo package.

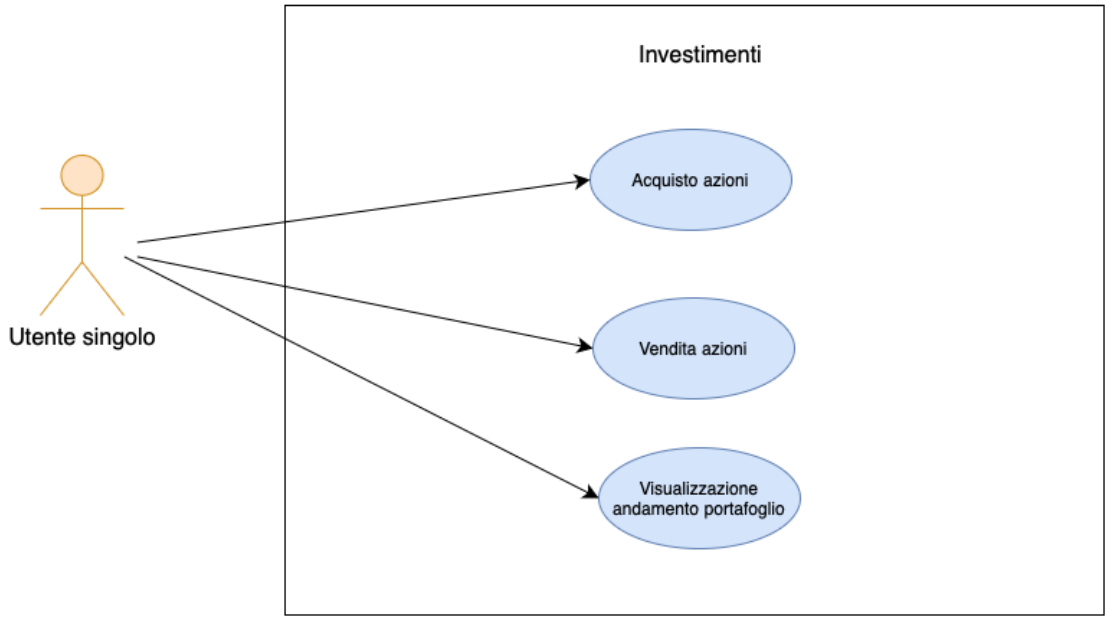

Figura 3.6: Casi d'uso relativi al package "Investimenti"

3.3 Matrice di mapping

Per verificare la corretta modellazione dei casi d'uso utilizziamo la matrice di mapping. Quest'ultima è uno strumento che permette di associare requisiti e casi d'uso e aiuta a tracciare la copertura dei requisiti e a verificare che tutte le funzionaltà siano implementate correttamente. In Figura 3.7 è riportata la matrice di mapping relativa al nostro progetto.

Matrice di Mapping	Gestione accesso piattaforma	Gestione credenziali utente	Gestione obbiettivi di spesa	Gestione Entrate e Uscite	Visualizzazione e interazione grafici	Gestione Piani di Risparmio	Gestione sottocategorie di spesa	Gestione storico transazioni	Gestione conti	Creazione famiglia	Gestione accesso gruppo famigliare	Gestione conti condivisi	Gestione piani di risparmio condivisi	Gestione premi	Gestione sfide di risparmio	Acquisto azioni	Vendiata azioni	Visualizzazione andamento portafoglio
creazione account	x																	
autenticazione utente	x																	
gestione password		x																
crud entrate e uscite				x				x										
crud sottocategorie di spesa							x											
crud spese future				x				x										
visualizzazione grafici					x													
crud piani risparmio						x							x					
crud conti									x			x						
monitoraggio investimenti																		x
crud investimenti																x	x	
crud spese ricorrenti				x														
visualizzazione storico transazioni								x										
crud obiettivi di spesa			x															
accesso famiglia										x	x							
monitoraggio spese famigliari								x										
condivisione conti												x						
partecipazione sfide di risparmio															x			
crud sfide di risparmio															x			
acquisto premi														x				

Figura 3.7: Matrice di Mapping relativa al nostro progetto

CAPITOLO 4

Progettazione

In questo capitolo verrà descritto il processo di progettazione del sistema, esso è suddiviso in diverse fasi. Inizialmente, saranno identificate e descritte le entità principali, che rappresentano i concetti più importanti del sistema. Successivamente, si procederà con la traduzione di queste entità in un modello entità-relazione (E-R) e con la trasformazione di quest'ultimo nel modello relazionale, che permette di strutturare le entità in tabelle per l'implementazione del database. Infine, si descriverà il mapping del modello relazionale in modelli Django.

4.1 Premessa

La progettazione rappresenta la fase successiva all'analisi dei requisiti nel processo di sviluppo di un progetto software. Durante questa fase i requisiti raccolti vengono tradotti in soluzioni funzionali. La progettazione si suddivide principalmente in due fasi:

- *Progettazione concettuale*: durante questa fase, vengono definite le entità principali del sistema e le loro interazioni in modo astratto, senza entrare nei dettagli implementativi.
- *Progettazione logica*: una volta definite le entità principali a livello concettuale, si procede con la definizione della struttura del sistema in modo più dettagliato. Durante questa fase vengono convertite le entità e le relazioni nel modello logico "entità-relazione".

4.2 Entità principali

La fase di progettazione inizia con l'individuazione delle entità principali, queste ultime rappresentano gli elementi di base del sistema e i concetti principali su cui si basa l'intera applicazione. Queste entità sono state identificate analizzando i requisiti, con l'obiettivo di tradurre le esigenze degli utenti e degli stakeholder in componenti concreti. Nel nostro progetto le entità principali individuate sono:

- *Utente*: questa entità rappresenta la persona che utilizza i servizi della piattaforma. Sulla base dell'analisi degli stakeholder esterni, durante la progettazione concettuale, l'entità utente è stata caratterizzata da una generalizzazione per distinguere due sotto-entità, ovvero l'utente individuale e il membro di famiglia. Entrambe condividevano attributi comuni, ma la prima presentava anche un attributo aggiuntivo, ovvero le monete dell'account. Successivamente, durante la fase di progettazione logica, la generalizzazione

è stata eliminata tramite un accorpamento delle entità figlie nell'entità genitore. La struttura finale dell'entità utente (la cui rappresentazione è riportata in Figura 4.1) presenta diversi attributi, in particolare:

- *id*: l'id funge da chiave primaria; esso viene utilizzato per riferirsi all'utente in modo preciso all'interno delle relazioni con altre entità;
- *username*: lo username rappresenta il nome univoco scelto dall'utente per accedere alla piattaforma;
- *email*: l'email indica l'indirizzo email associato all'account dell'utente che quest'ultimo utilizza per l'autenticazione;
- *password*: la password è l'ulteriore campo che un utente utilizza per autenticarsi all'interno della piattaforma;
- *nome*: questo attributo rappresenta il nome dell'utente;
- *cognome*: questo attributo rappresenta il cognome dell'utente;
- *data di nascita*: questo attributo rappresenta la data di nascita dell'utente;
- *indirizzo*: questo attributo rappresenta l'indirizzo di residenza dell'utente;
- *telefono*: questo attributo rappresenta il numero di telefono dell'utente;
- *sesso*: questo attributo rappresenta il sesso dell'utente;
- *data registrazione*: questo attributo rappresenta la data in cui l'utente si è iscritto alla piattaforma;
- *monete account*: questo attributo rappresenta la quantità di monete dell'utente che quest'ultimo ha ottenuto attraverso le sfide.

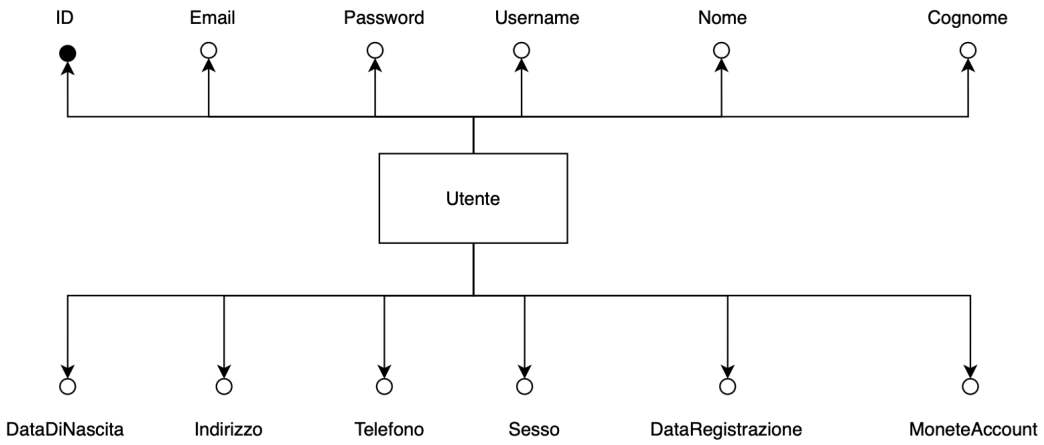

Figura 4.1: Entità Utente

- *Famiglia*: questa entità rappresenta un gruppo di utenti iscritti alla piattafforma che condividono risorse finanziarie e obbiettivi economici. La struttura finale dell'entità famiglia (la cui rappresentazione è riportata in Figura 4.2) presenta diversi attributi; in particolare:
  - *id Famiglia*: l'id funge da chiave primaria, identificando univocamente ogni famiglia nel sistema;

- *numero partecipanti*: questo attributo indica il numero di utenti associati alla famiglia;
- *data creazione*: questo attributo rappresenta la data di creazione della famiglia;
- *nome famiglia*: questo attributo rappresenta il nome della famiglia deciso dall'utente al momento della creazione.

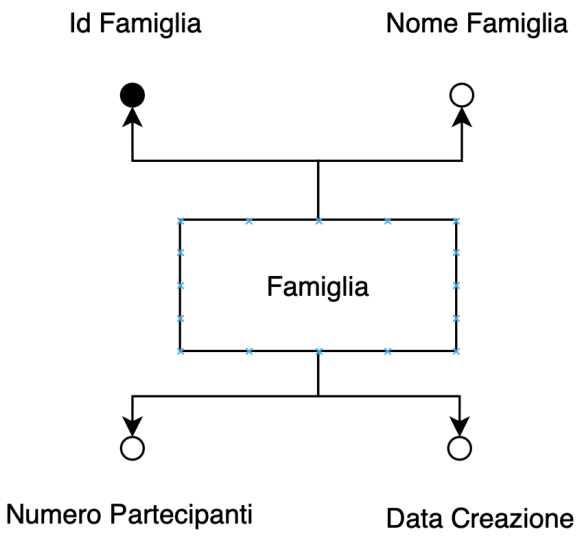

Figura 4.2: Entità Famiglia

- *Conto*: questa entità rappresenta i conti che gli utenti possono gestire all'interno della piattaforma. Un conto può appartenere a un utente individuale o a più utenti che fanno parte della stessa famiglia. Esso funge da punto centrale per la registrazione e la gestione delle transazioni. Durante la progettazione concettuale l'entità conto è stata caratterizzata da una generalizzazione per distinguere i diversi tipi di conto; questi ultimi sono:
  - *conto corrente*: conto dove l'utente può registrare transazioni normali, transazioni future o transazioni periodiche, sia in entrata che in uscita;
  - *conto di investimento*: conto dove l'utente può registrare operazioni di investimento;
  - *conto risparmio*: conto dove l'utente può impostare un piano di risparmio personalizzato.

  Successivamente, durante la fase di progettazione logica, la generalizzazione è stata eliminata tramite un accorpamento delle entità figlie nell'entità genitore. La struttura finale dell'entità conto (la cui rappresentazione è riportata in Figura 4.3) presenta diversi attributi, in particolare:
  - *id conto*: l'id funge da chiave primaria, identificando univocamente ogni conto nel sistema;
  - *saldo*: questo attributo rappresenta il valore complessivo delle risorse finanziarie associate al conto, includendo sia la liquidità disponibile sia il valore complessivo degli strumenti finanziari (nel caso dei conti di investimento);

- *liquidità*: questo attributo rappresenta la quantità di fondi disponibili nel conto sotto forma di liquidità immediatamente accessibile; nel caso dei conti di investimento, la liquidità consente di effettuare nuove operazioni di investimento; nel caso, invece, dei conti correnti o conti risparmio, la liqudità corrisponde al saldo.
- *nome conto*: questo attributo indica il nome del conto;
- *tipo conto*: questo attributo indica il tipo di conto.

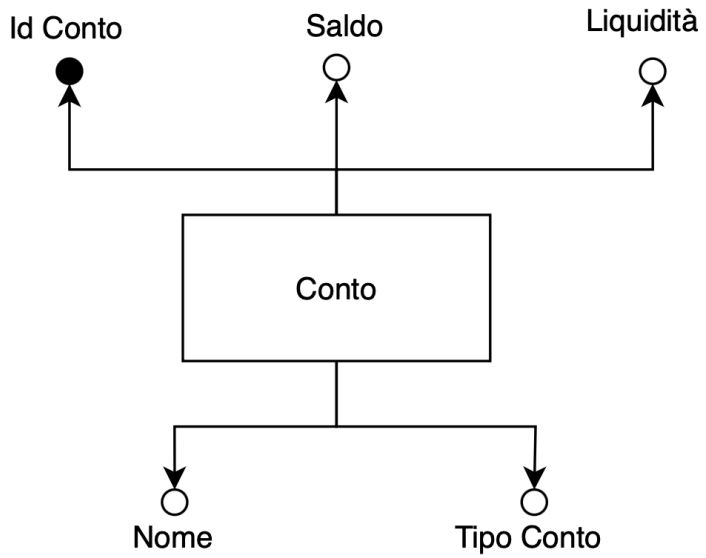

Figura 4.3: Entità Conto

- *Premio*: questa entità rappresenta i premi che gli utenti possono acquistare utilizzando le monete dell'account accumulate attraverso le sfide. Attualmente i premi che la piattaforma mette a disposizione sono buoni Amazon, tuttavia; nella progettazione di questa entità si è tenuto conto della possibilità di introdurre buoni di altre piattaforme. La struttura finale dell'entità premio (la cui rappresentazione è riportata in Figura 4.4) presenta diversi attributi; in particolare:
  - *id premio*: l'id funge da chiave primaria, identificando univocamente ogni premio nel sistema;
  - *valore buono*: questo attributo rappresenta il valore del buono;
  - *costo*: questo attributo indica la quantità di monete richieste per riscattare il premio associato;
  - *nome azienda*: questo attributo specifica il nome della piattaforma o dell'azienda presso cui il buono può essere utilizzato;
  - *nome premio*: questo attributo indica il nome del premio.

- *Obbiettivo spesa*: questa entità rappresenta gli obbiettivi di spesa che gli utenti possono definire per monitorare e gestire le loro finanze in modo efficace. Questi obbiettivi permettono agli utenti di fissare limiti di spesa per determinate categorie. La struttura finale dell'entità obbiettivo spesa (la cui rappresentazione è riportata in Figura 4.5) presenta diversi attributi; in particolare:

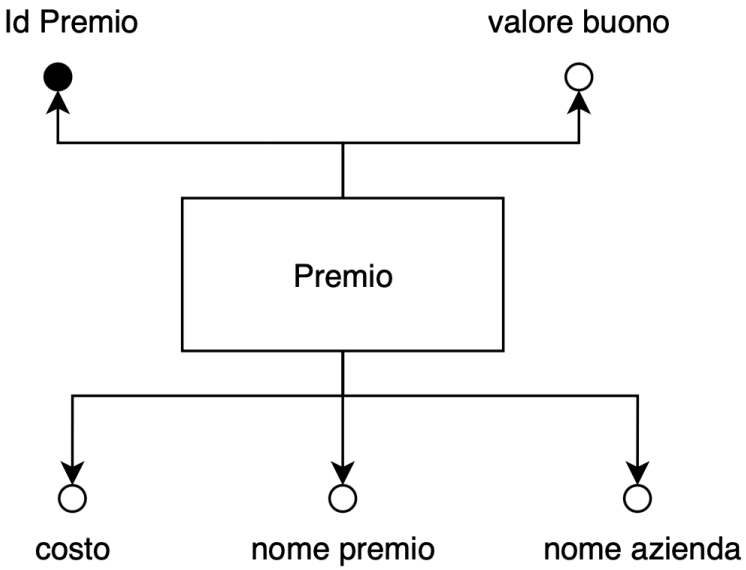

Figura 4.4: Entità Premio

- *id obbiettivo*: l'id funge da chiave primaria, identificando univocamente ogni obbiettivo nel sistema;
- *importo obbiettivo*: questo attributo rappresenta l'importo massimo di spesa che l'utente intende raggiungere;
- *importo speso*: questo attributo rappresenta la somma totale di denaro spesa dall'utente nella categoria specifica dell'obbiettivo; essa è calcolata a partire dalla data di creazione dell'obbiettivo fino al momento attuale;
- *percentuale completamento*: questo attributo indica il rapporto tra l'importo speso e l'importo obbiettivo;
- *tipo*: questo attributo indica la periodicità dell'obbiettivo di spesa;
- *data creazione*: questo attributo indica la data in cui l'utente ha creato l'obbiettivo;
- *data scadenza*: questo attributo indica la data di scadenza dell'obbiettivo; essa è calcolata in base alla data di creazione e alla periodicità dell'obbiettivo.

- *Piano risparmio*: questa entità rappresenta i piani di risparmio creati dagli utenti. Questi piani aiutano gli utenti a raggiungere specifici traguardi finanziari entro un periodo di tempo determinato. La struttura finale dell'entità (la cui rappresentazione è riportata in Figura 4.6) presenta diversi attributi; in particolare:
  - *id piano*: l'id funge da chiave primaria, identificando univocamente ogni piano nel sistema;
  - *obbiettivo*: questo attributo rappresenta il quantitativo di denaro che l'utente desidera avere sul conto specificato entro la data di scadenza prefissata;
  - *data scadenza*: questo attributo rappresenta la data di scadenza del piano risparmio;
  - *data creazione*: questo attributo rappresenta la data in cui l'utente ha impostato il piano di risparmio;

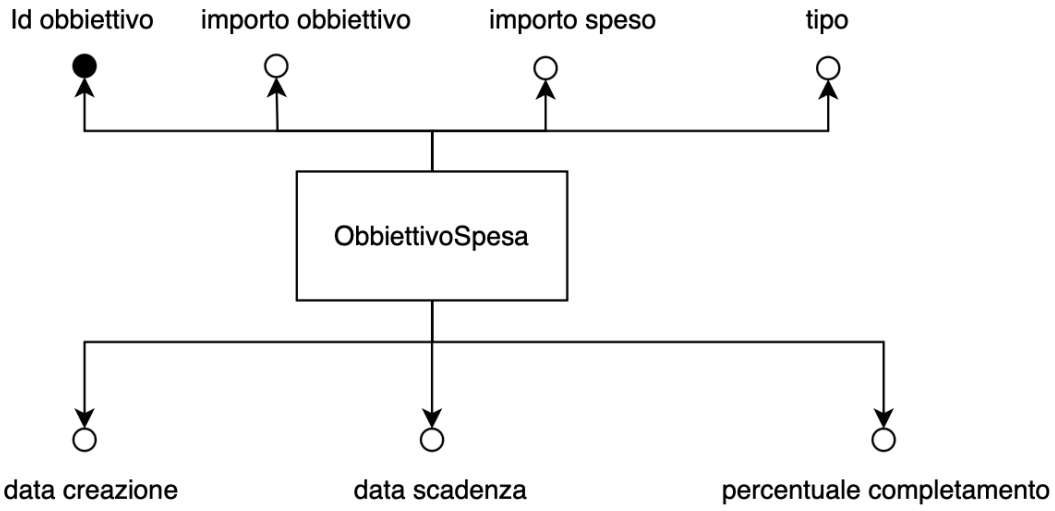

Figura 4.5: Entità Obbiettivo Spesa

– *percentuale completamento*: questo attributo indica il progresso verso il raggiungimento dell'obiettivo; tale progresso è espresso come percentuale; essa è basata sul rapporto tra l'importo attuale sul conto di riferimento e l'importo obbiettivo.

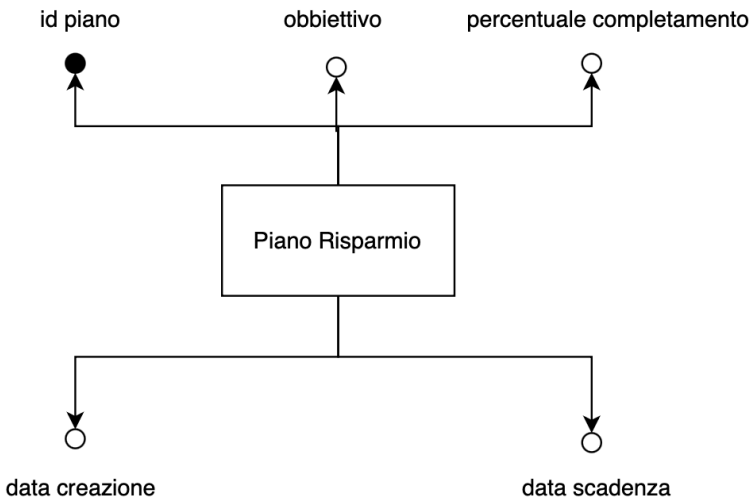

Figura 4.6: Entità Piano Risparmio

- *Categoria Spesa*: questa entità rappresenta una categoria di spesa che il sistema mette a disposizione; queste categorie possono essere usate dagli utenti per classificare le transazioni. Le categorie aiutano a organizzare e monitorare le spese fornendo una

visione chiara delle aree di spesa. La struttura finale dell'entità Categoria Spesa (la cui rappresentazione è riportata in Figura 4.7) presenta diversi attributi; in particolare:

- *id categoria*: l'id funge da chiave primaria, identificando univocamente ogni categoria nel sistema;
- *nome categoria*: questo attributo indica il nome della categoria di spesa.

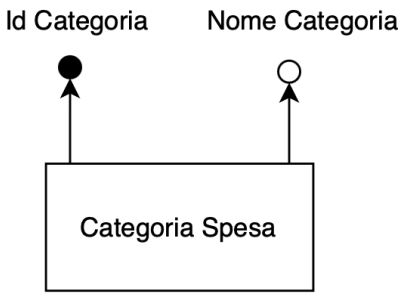

Figura 4.7: Entità Categoria Spesa

- *Sottocategoria spesa*: questa entità rappresenta le suddivisioni più specifiche all'interno delle categorie di spesa principali, consentendo una classificazione dettagliata delle spese. Oltre alle sottocategorie predefinite fornite dal sistema, l'utente ha la possibilità di crearne di nuove. La struttura finale dell'entità Sottocategoria spesa (la cui rappresentazione è riportata in Figura 4.8) presenta diversi attributi; in particolare:
  - *id sottocategoria*: l'id funge da chiave primaria;
  - *nome sottocategoria*: questo attributo indica il nome della sottocategoria;
  - *personalizzata*: questo attributo indica se la sottocategoria è stata fornita dal sistema oppure se essa è stata creata da un utente;
  - *data creazione*: questo attributo indica la data di creazione della sottocategoria.

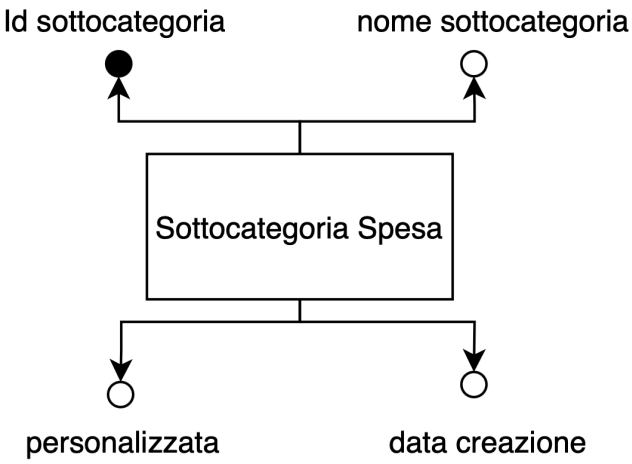

Figura 4.8: Entità sottocategoria spesa

- *Transazione*: questa entità rappresenta qualsiasi movimento finanziario registrato all'interno della piattaforma; questo include entrate, uscite, investimenti e trasferimenti tra conti. Durante la progettazione concettuale questa entità è stata caratterizzata da una generalizzazione per distinguere il tipo di transazione; in particolare una transazione può essere:
  - *una transazione singola*: una transazione singola è una normale transazione di entrata o uscita;
  - *transazione periodica*: una transazione periodica è una transazione che si ripete nel tempo;
  - *una transazione futura*: una transazione futura è una transazione che verrà contabilizzata in una data diversa da quella odierna;
  - *una trasferimento tra conti*: un trasferimento tra conti è una transazione che sposta del denaro da un conto ad un altro (i conti devono appartenere allo stesso utente oppure a persone della stessa famiglia);
  - *un investimento*: un investimento è una transazione che descrive un'operazione di investimento nei mercati finanziari.

  Durante la fase di progettazione logica, la generalizzazione è stata eliminata tramite un accorpamento delle entità figlie nell'entità genitore. La struttura finale dell'entità transazione (la cui rappresentazione è riportata in Figura 4.9) presenta diversi attributi; in particolare:
  - *id transazione*: l'id funge da chiave primaria, identificando univocamente ogni transazione nel sistema;
  - *importo*: questo attributo rappresenta l'importo della transazione;
  - *descrizione*: questo attributo indica una descrizione della transazione;
  - *data*: questo attributo indica la data in cui la transazione è stata contabilizzata;
  - *tipo rinnovo*: questo attributo indica il tipo di periodicità, applicabile nel caso di transazioni periodiche;
  - *prossimo rinnovo*: questo attributo indica la data del prossimo rinnovo, nel caso di transazioni periodiche;
  - *eseguita*: questo attributo indica se la transazione è stata contabilizzata;
  - *ticker*: questo attributo indica il simbolo del titolo azionario, esso è utilizzato per transazioni di tipo investimento;
  - *numero di azioni*: questo attributo specifica il numero di azioni acquistate o vendute, nel caso di una transazione di investimento;
  - *prezzo azione*: questo attributo indica il prezzo per singola azione, esso è utilizzato per transazioni di tipo investimento;
  - *tipo transazione*: questo attributo indica il tipo di transazione.

- *sfida famigliare*: questa entità rappresenta una competizione tra due membri di una famiglia all'interno della piattaforma. Una sfida consiste in un utente che sfida un altro membro a chi spende meno in una determinata categoria di spesa durante un certo periodo di tempo stabilito. Questo tipo di sfida stimola sia la collaborazione che una sana competizione per il raggiungimento di obiettivi finanziari individuali, incentivando la gestione consapevole delle finanze di tutta la famiglia. La struttura finale dell'entità sfida famigliare (la cui rappresentazione è riportata in Figura 4.10) presenta diversi attributi; in particolare:

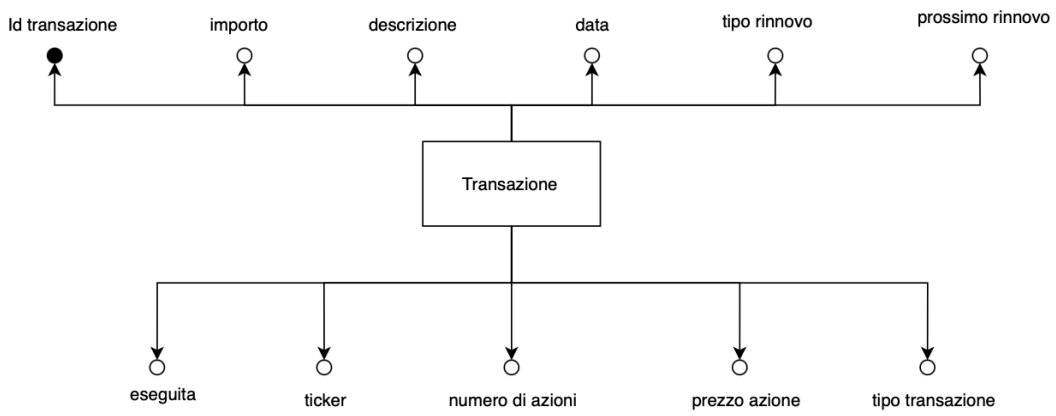

Figura 4.9: Entità Transazione

  - *id sfida*: l'id funge da chiave primaria, identificando univocamente ogni sfida nel sistema;
  - *data creazione*: questo attributo indica la data di creazione della sfida;
  - *conclusa*: questo attributo indica se la sfida si è conclusa;
  - *descrizione*: questo attributo indica una descrizione della sfida;
  - *vincitore*: questo attributo indica, in caso di sfida conclusa, l'utente che ha vinto la sfida;
  - *data scadenza*: questo attributo indica la data di scadenza della sfida;
  - *importo sfidante*: questo attributo rappresenta la somma totale di denaro spesa dall'utente sfidante nella categoria specifica della sfida; essa è calcolata a partire dalla data di creazione della sfida fino al momento attuale;
  - *importo sfidato*: questo attributo rappresenta la somma totale di denaro spesa dall'utente sfidato nella categoria specifica della sfida; essa è calcolata a partire dalla data di creazione della sfida fino al momento attuale;
  - *percentuale sfidante*: questo attributo indica la percentuale del totale speso dall'utente sfidante rispetto alla somma complessiva delle spese nella categoria della sfida;
  - *percentuale sfidato*: questo attributo indica la percentuale del totale speso dall'utente sfidato rispetto alla somma complessiva delle spese nella categoria della sfida.

- *Saldo totale*: questa entità rappresenta la somma complessiva di denaro presente in tutti i conti associati a un utente. Questo valore viene calcolato giornalmente, permettendo di tracciare l'andamento del saldo nel tempo. La struttura finale dell'entità saldo totale (la cui rappresentazione è riportata in Figura 4.11) presenta diversi attributi; in particolare:
  - *id saldo totale*: l'id funge da chiave primaria, identificando ogni saldo univocamente nel sistema;
  - *saldo totale*: questo attributo indica l'importo totale calcolato;
  - *data aggiornamento*: questo attributo indica la data relativa al calcolo.

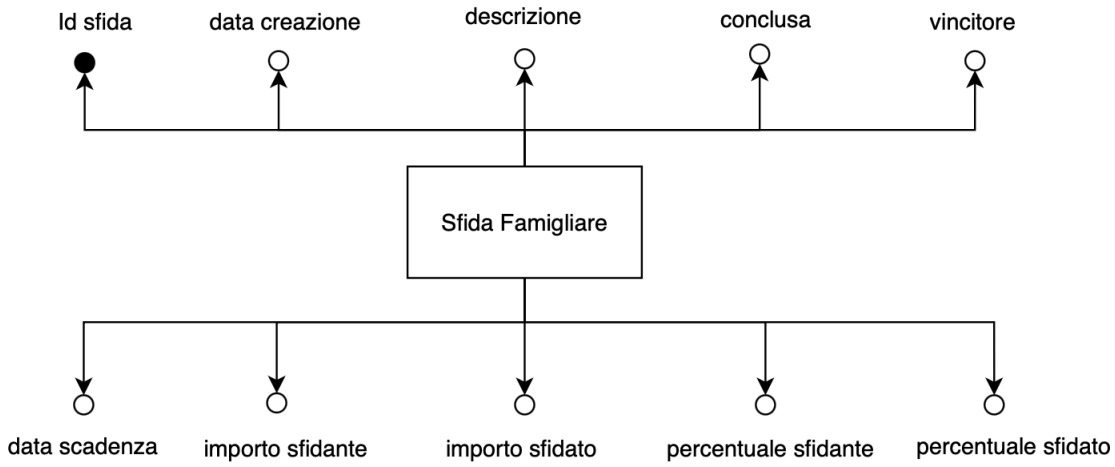

Figura 4.10: Entità Sfida Famigliare

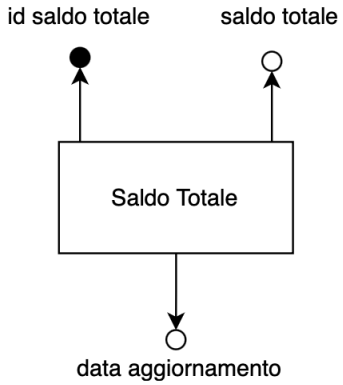

Figura 4.11: Entità Saldo Totale

- *Posizione aperta*: questa entità rappresenta una posizione di investimento di un utente all'interno della piattaforma. Essa descrive un investimento specifico in un'azienda. Il valore della posizione viene aggiornato giornalmente; il sistema utilizza il prezzo di chiusura del titolo relativo al giorno precedente. La struttura finale dell'entità Posizione Aperta (la cui rappresentazione è riportata in Figura 4.12) presenta diversi attributi; in particolare:
  - *id posizione*: l'id funge da chiave primaria, identificando univocamente ogni posizione nel sistema;
  - *saldo investito*: questo attributo indica l'importo complessivo che l'utente ha investito nella posizione;
  - *saldo totale*: questo attributo indica il valore attuale della posizione;
  - *pmc*: questo attributo indica il prezzo medio di carico, ovvero il prezzo medio ponderato delle azioni acquistate nella posizione;
  - *differenza*: questo attributo indica la differenza tra il saldo totale e il saldo investito;
  - *ticker*: questo attributo indica il simbolo del titolo azionario a cui la posizione si riferisce;

– *nome azienda*: questo attributo indica il nome dell'azienda a cui la posizione si riferisce;
– *numero azioni*: questo attributo indica il numero di azioni della posizione.

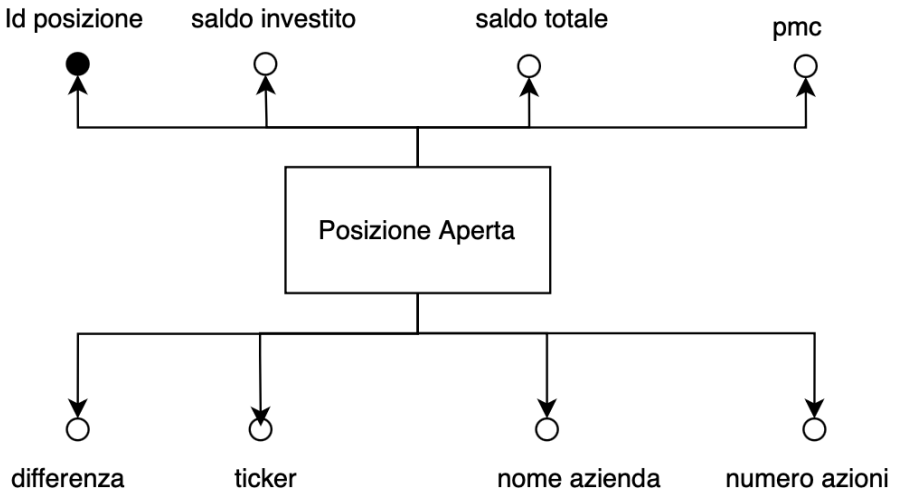

Figura 4.12: Entità Posizione Aperta

- *Saldo totale investimenti*: questa entità rappresenta la somma complessiva del valore corrente di tutte le posizioni di investimento associate a un utente. Questo valore viene calcolato giornalmente, permettendo di tracciare l'andamento del saldo nel tempo. La struttura finale dell'entita Saldo Totale Investimenti (la cui rappresentazione è riportata in Figura 4.13) presenta diversi attributi, in particolare:
  – *id saldo totale investimenti*: l'id funge da chiave primaria, identificando univocamente ogni saldo totale nel sistema;
  – *saldo totale*: questo attributo indica l'importo totale calcolato;
  – *data aggiornamento*: questo attributo indica la data relativa al calcolo.

4.3 Modello Entità - Relazione (E-R)

Le entità descritte nella precedente sottosezione sono connesse tra loro attraverso delle relazioni; queste ultime rappresentano legami logici tra due o più entità. Una istanza di una relazione è una coppia di istanze delle entità partecipanti alla relazione. L'insieme delle entità e delle loro relazioni è rappresentato nel diagramma E-R riportato in Figura 4.14, che sintetizza graficamente il modello concettuale del sistema.

4.4 Traduzione verso il modello relazionale

Il modello relazionale è un approccio per organizzare e rappresentare i dati all'interno di un database utilizzando tabelle per descrivere le informazioni e le relazioni tra esse. La traduzione verso il modello relazionale si basa sul processo di trasformazione del modello

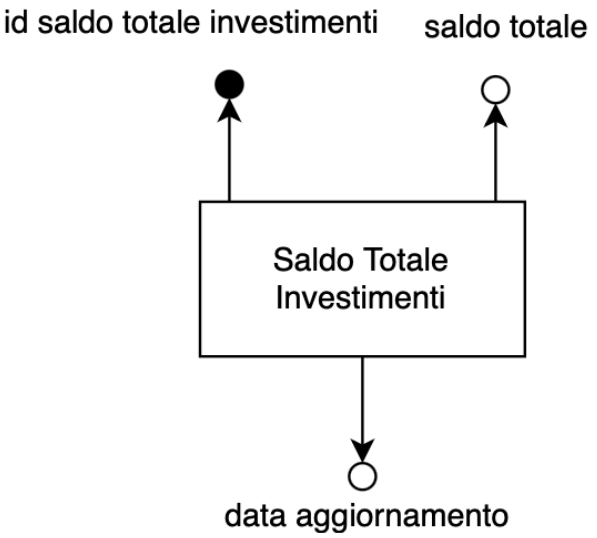

Figura 4.13: Entità Saldo Totale Investimenti

concettuale, espresso tramite diagramma E-R, in una struttura logica basata sul modello relazionale. Questo passaggio consiste nel mappare le entità, le relazioni e i relativi attributi in tabelle.

Durante questo processo, le entità vengono tradotte in tabelle, con i rispettivi attributi rappresentati come colonne. Le relazioni tra entità vengono, invece mappate, tramite chiavi esterne (foreign key), che collegano tabelle diverse per rappresentare i legami tra i dati. Questa trasformazione porta all'aggiunta di nuovi attributi alle entità coinvolte nella relazione e, di conseguenza, nuove colonne alle rispettive tabelle.

In alcuni casi, tuttavia, le relazioni non possono essere rappresentate solo tramite chiavi esterne. Questo accade, ad esempio, per le relazioni molti-a-molti. In tale situazione, la relazione viene trasformata in una nuova entità, con una propria tabella che include le chiavi esterne delle entità coinvolte e gli eventuali attributi specifici della relazione. Nel caso del nostro progetto, la trasformazione delle relazioni ci ha portato a due nuove entità:

- *Intestazione conto*: questa entità rappresenta l'associazione tra un utente e il relativo conto all'interno della piattaforma. La struttura finale dell'entità intestazione conto (la cui rappresentazione è riportata in Figura 4.15) presenta diversi attributi, in particolare:
  - *id intestazione*: l'id funge da chiave primaria, identificando univocamente ogni intestazione nel sistema;
  - *utente*: questo attributo è un riferimento all'utente che detiene il conto, rappresentato attraverso una chiave esterna che collega l'entità Intestazione Conto all'entità Utente;
  - *conto*: questo attributo è un riferimento attraverso una chiave esterna al conto associato all'utente;
  - *data intestazione*: questo attributo indica la data di intestazione.

- *Acquisto premio*: questa entità rappresenta l'acquisto di un premio da parte di un utente all'interno della piattaforma. Ogni acquisto è associato a un premio specifico e viene

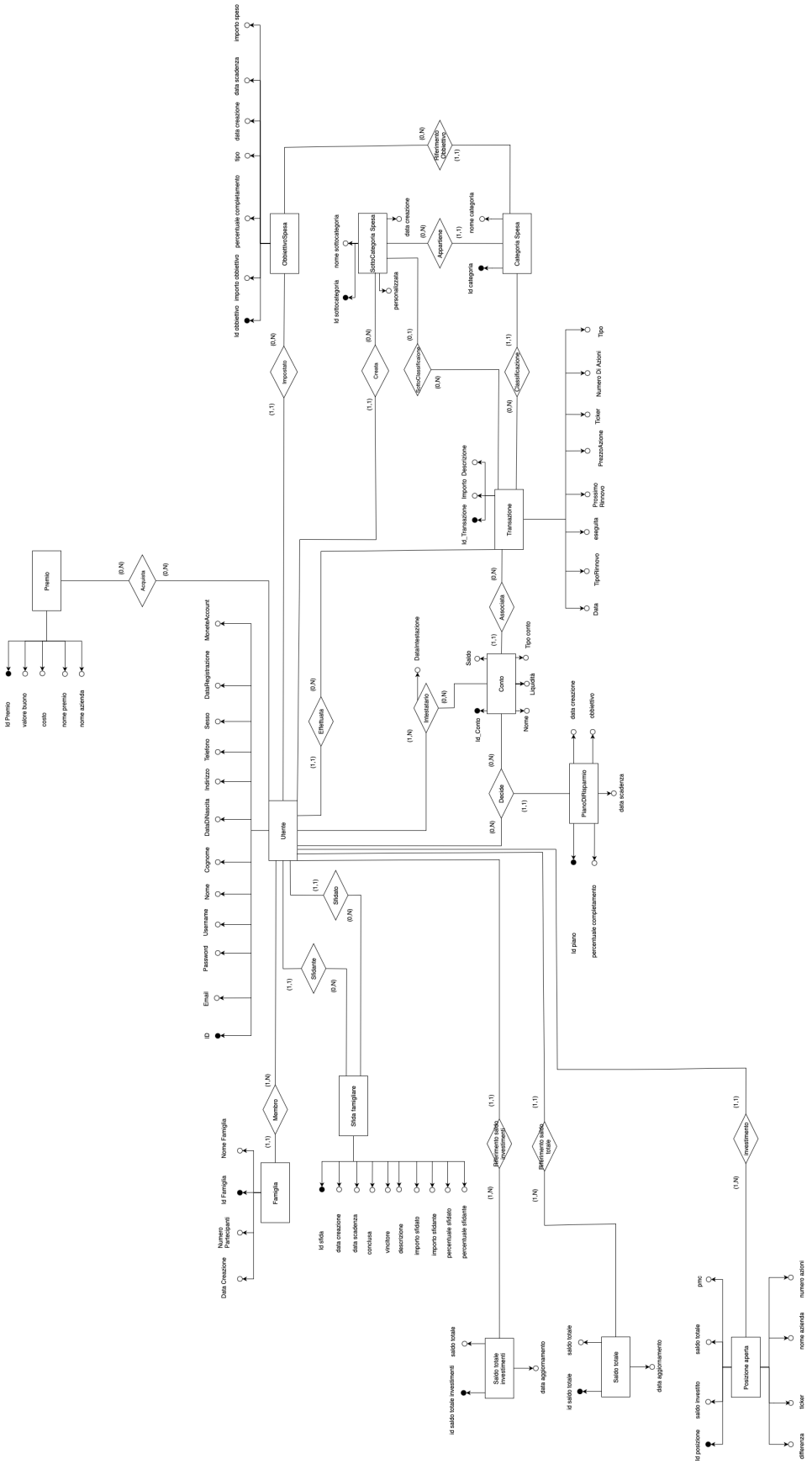

Figura 4.14: Modello E-R del sistema

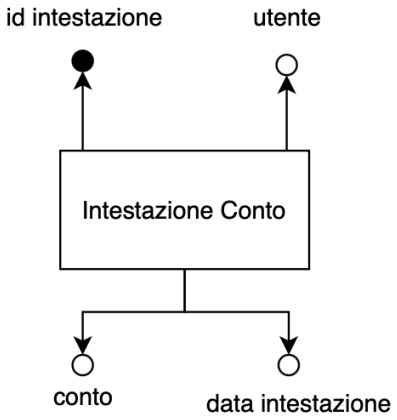

Figura 4.15: Entità Intestazione Conto

registrato con il relativo utente. La struttura finale dell'entità Acquisto Premio (la cui rappresentazione è riportata in Figura 4.16) presenta diversi attributi; in particolare:

- *id acquisto*: l'id funge da chiave primaria, identificando univocamente ogni acquisto nel sistema;
- *premio*: questo attributo rappresenta un riferimento al premio acquistato, rappresentato come una chiave esterna che collega l'entità Acquisto Premio all'entità Premio contenente i dettagli del premio stesso.
- *utente*: questo attributo rappresenta un riferimento all'utente che ha effettuato l'acquisto, rappresentato tramite una chiave esterna che collega l'entità Acquisto Premio all'entità Utente.
- *data acquisto*: questo attributo rappresenta la data in cui l'acquisto è stato effettuato dall'utente.

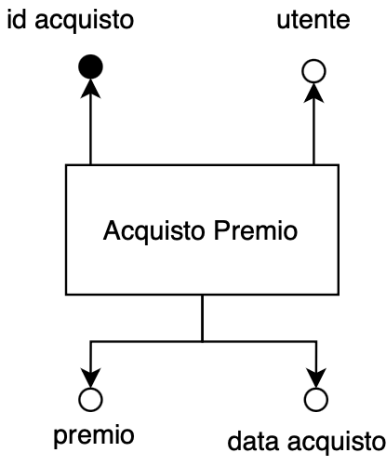

Figura 4.16: Entità Acquisto Premio

4.5 Mappatura del modello relazionale in modelli Django

Per utilizzare il modello relazionale in Django è necessario mappare le tabelle e le relazioni in classi Python chiamate modelli. Ogni modello definisce i campi della tabella come attributi. I principali tipi di attributi utilizzati nei modelli Django sono i seguenti:

- *CharField*: questo campo è utilizzato per rappresentare stringhe di testo di lunghezza limitata. Nel nostro progetto, è stato utilizzato per la maggior parte delle colonne di tipo testuale delle tabelle del modello relazionale.
- *TextField*: questo campo è usato per rappresentare testi di lunghezza variabile. Nel nostro progetto, è stato utilizzato per le tabelle che contengono una colonna dedicata alla descrizione.
- *BooleanField*: questo campo è usato per rappresentare un valore booleano.
- *DateTimeField*: questo campo è usato per rappresentare data e ora. Nel nostro progetto, è stato utilizzato per le tabelle che contengono una colonna dedicata all'indicazione di una data.
- *ForeignKey*: questo campo è utilizzato per creare una relazione tra modelli (tabelle), dove un record fa riferimento a un altro tramite una chiave esterna. Nel nostro progetto, è stato utilizzato per le colonne delle tabelle che rappresentano delle relazioni tra entità.

Un esempio di mapping di una tabella del modello relazionale in un modello Django è illustrato nella Figura 4.17, dove viene mostrata la classe Python che mappa la tabella ottenuta dall'entità Transazione.

```python
class Transazione(models.Model):
    importo = models.DecimalField(max_digits=10, decimal_places=2)
    data = models.DateField()
    eseguita = models.BooleanField(default=True)
    tipo_transazione = models.CharField(
        max_length=20,
        choices=CategoriaTransazione.choices,
        default=CategoriaTransazione.SINGOLA,
    )
    prossimo_rinnovo = models.DateField(null=True, blank=True)
    ticker = models.CharField(max_length=10, null=True, blank=True)
    prezzo_azione = models.FloatField(null=True, blank=True)
    numero_azioni = models.FloatField(null=True, blank=True)
    conto = models.ForeignKey(Conto, on_delete=models.CASCADE,  related_name='conto_partenza')
    utente = models.ForeignKey(Utente, on_delete=models.CASCADE)
    categoria = models.ForeignKey(CategoriaSpesa, null=True, blank=True, on_delete=models.SET_NULL)
    sotto_categoria = models.ForeignKey(SottoCategoriaSpesa, null=True, blank=True, on_delete=models.SET_NULL)
    descrizione = models.TextField(null=True, blank=True)
    tipo_rinnovo = models.CharField(
        max_length=20,
        choices=TipoRinnovo.choices,
        default= None,
        null=True,
        blank=True
    )
    conto_arrivo = models.ForeignKey(Conto, on_delete=models.CASCADE, null= True, blank=True,  related_name='conto_arrivo')
```

Figura 4.17: Classe Python `Transazione`

Django mette a disposizione un suo ORM per utilizzare queste classi. Un ORM (Object-Relational Mapping) è uno strumento che ci permette di interfacciarci e interagire con un database relazionale utilizzando un linguaggio orientato agli oggetti. Ogni classe che rappresenta una tabella nel database è una sottoclasse dalla classe `models.Model` di Django e deve essere definita all'interno dei file `models.py`, che come abbiamo già visto è uno dei file presenti in ogni applicazione Django. Queste classi vengono utilizzate principalmente nelle viste per interagire con il database e nei template per la visualizzazione dei dati.

CAPITOLO 5

Implementazione e manuale utente

In questo capitolo verrà descritta l'implementazione effettiva del progetto. Inizialmente sarà spiegata la sua struttura, organizzata in applicazioni Django, e per ciascuna di esse saranno illustrate le caratteristiche specifiche. Successivamente si procederà con l'analisi di alcune parti particolarmente rilevanti dell'implementazione, come l'utilizzo dei Modelform di Django, l'integrazione di API e la gestione degli utenti. Infine, verrà illustrato il funzionamento della piattaforma web, con contributi grafici che mostreranno le principali funzionalità.

5.1 Struttura del progetto

Il progetto è organizzato in diverse applicazioni Django, ciascuna responsabile di una specifica funzionalità. Questa suddivisione consente una gestione modulare del codice. Ogni applicazione è strutturata per includere i modelli, le viste, i file statici e i template necessari per garantire il corretto funzionamento delle sue specifiche caratteristiche. Di seguito sono riportate le appicazioni del progetto:

- *Users*: questa applicazione gestisce la logica di registrazione e login degli utenti nella piattaforma. Essa contiene i modelli che descrivono l'entità famiglia e l'entità utente.
- *Reward*: questa applicazione gestisce la logica di acquisto dei premi degli utenti nella piattaforma. Essa contiene i modelli che descrivono l'entità premio e l'entità acquisto premio.
- *Home page*: questa applicazione gestisce la pagina iniziale del progetto, fornendo un'introduzione e una panoramica generale delle funzionalità offerte dalla piattaforma.
- *Faq*: questa applicazione gestisce la sezione delle domande frequenti (FAQ) della piattaforma; essa si trova nella home page del sito.
- *Challenges*: questa applicazione gestisce la logica delle sfide tra utenti della stessa famiglia. Essa contiene il modello che descrive l'entità sfida famigliare.
- *Budgeting*: questa applicazione è il cuore della gestione finanziaria personale e familiare della piattaforma. Essa contiene modelli per descrivere l'entità transazione, l'entità piano risparmio, l'entità obbiettivo spesa, l'entità categoria spesa e l'entità sottocategoria spesa.

- *Accounts*: questa applicazione si occupa della gestione dei conti finanziari e delle relative informazioni associate agli utenti. Essa contiene i modelli per descrivere l'entità conto, l'entità intestazioni conto, l'entità saldo totale, l'entità salto totale investimenti e l'entità posizione aperta.

5.2 ModelForm Django

I ModelForm di Django sono uno strumento che semplifica la creazione, la validazione e la gestione dei dati inviati tramite form HTML. Un ModelForm, nello specifico, è una sottoclasse di `forms.Form` che consente di creare un form direttamente collegato a un modello Django. Questa associazione permette a Django di generare automaticamente i campi del form in base agli attributi definiti nel modello corrispondente. Ogni campo del form rispecchia un attributo del modello e, una volta che il form viene validato e inviato, i dati raccolti possono essere utilizzati per creare o aggiornare un'istanza del modello corrispondente.

5.2.1 Struttura di un ModelForm

Un modelform è una classe che comprende le seguenti parti:

- *Classe* `Meta`: questa classe contiene tre attributi, di seguito illustrati:
  - *Attributo* `model`: con questo attributo viene indicato il modello a cui il modelform fa riferimento.
  - *Attributo* `fields`: con questo attributo viene indicata la lista degli attributi del modello che dovranno essere inclusi nel form.
  - *Attributo* `widgets`: questo attributo è un dizionario che consente di personalizzare l'aspetto e il comportamento di ciascun campo nel form. Ogni campo nel form può avere un widget personalizzato, che definisce come esso viene renderizzato nel template HTML.
- *Funzione* *__init__*: questa funzione permette di personalizzare l'inizializzazione del form in modo dinamico. Un esempio comune di utilizzo di essa è quello di definire dinamicamente le opzioni dei campi select attraverso i parametri passati al costruttore.
- *Funzione* `clean`: questa funzione è utilizzata per validare i campi del modulo, ovvero per eseguire controlli aggiuntivi sui dati inseriti dall'utente prima che questi ultimi vengano salvati nel database. Se la validazione fallisce, Django permette di lanciare un'eccezione per segnalare l'errore.

5.2.2 ModelForm del progetto

Ogni modelform è definito all'interno di un file chiamato `forms.py`, che può contenere una o più classi modelform. Nel nostro progetto ogni applicazione Django contiene un file `forms.py` in cui le classi definite variano in base ai modelli che la suddetta applicazione include. Nel complesso del progetto sono stati definiti e utilizzati i seguenti Modelform:

- `NuovoContoForm`: form utilizzato per creare un nuovo conto nella sezione di budgeting personale;
- `NuovoContoFamiglia`: form utilizzato per creare un nuovo conto nella sezione di budgeting famigliare;

- `NuovaTransazioneForm`: form utilizzato per registrare una nuova transazione nella sezione di budgeting personale;
- `NuovaTransazioneFamigliareForm`: form utilizzato per creare una nuova transazione nella sezione di budgeting famigliare;
- `NuovoPianoRisparmio`: form utilizzato per creare un nuovo piano risparmio;
- `NuovoInvestimentoForm`: form utilizzato per registrare una nuova operazione di acquisto di azioni;
- `NuovaVenditaForm`: form utilizzato per registrare una nuova operazione di vendita di azioni;
- `SottocategoriaForm`: form utilizzato per creare una nuova sottocategoria di spesa;
- `NuovaSfidaFamigliareForm`: form utilizzato per creare una nuova sfida di risparmio tra membri della stessa famiglia;
- `NuovoUtenteForm`: form utilizzato per creare un nuovo utente;
- `AuthenticationForm`: form predefinito utilizzato per autenticare un utente tramite l'inserimento di username e password.

Una volta che il form viene compilato e i dati vengono validati tramite la funzione `clean`, questi utlimi vengono inviati al server tramite il metodo POST. Questo metodo invia i dati nel corpo della richiesta HTTP senza esporli nell'URL (come nel caso del metodo GET). Questi dati inviati vengono successivamente utilizzati per aggiornare il database. In Figura 5.1 viene mostrato il codice del modelform `NuovoContoForm`.

5.3 API

Le API (Application Programming Interface) sono strumenti che consentono la comunicazione tra diverse applicazioni software, permettendo ad esse di scambiarsi dati e informazioni. Esse fungono da intermediari facilitando l'interazione tra il frontend e il backend, eseguendo richieste e restituendo risposte in un formato ben definito (ad esempio, in formato JSON). Nel contesto del nostro progetto, le API vengono utilizzate per raccogliere e fornire i dati necessari alla gestione della sezione dedicata agli investimenti. In particolare, sono state utilizzate tre API, di seguito illustrate:

- *Finnhub*: questa API è stata utilizzata per consentire la ricerca di aziende tramite il ticker, fornendo dettagli come il nome completo dell'azienda e il tipo di stock associato. In Figura 5.2 viene illustrata la vista che gestisce la ricerca delle aziende tramite ticker.
- *Alphavantage*: una volta ottenuto il nome completo dell'azienda, tale informazione viene utilizzata da questa API per recuperare il prezzo di chiusura dell'azione in dollari, relativo al giorno precedente. La vista che restituisce il prezzo di chiusura è mostrata in Figura 5.3.
- *ExchangeRate*: dopo aver ottenuto il prezzo di chiusura in dollari dell'azione, questa API viene utilizzata per ottenere il tasso di cambio euro-dollaro corrente. Il sistema utilizza questa informazione per calcolare e restituire il prezzo di chiusura dell'azione in euro. In Figura 5.4 viene mostrata la vista che gestisce la conversione.

```python
class NuovoConto(ModelForm):
    saldo = forms.DecimalField(
            widget=forms.NumberInput(attrs={
                'placeholder': 'Enter start amount',
                'min': '0',
                'step': '0.01' ,
                'class': 'form-control',
            })
        )
    tipo = forms.ChoiceField(
        choices= TipoConto,
        widget=forms.Select(attrs={'class': 'form-control'})
    )
    You, 2 mesi fa | 1 author (You)
    class Meta:
        model = Conto
        fields = ["nome", "tipo", "saldo"]
        widgets = {
            'nome': forms.TextInput(attrs={'placeholder': 'Enter bank name',
                                           'class': 'form-control',
                                           }),
        }

    def __init__(self, *args, **kwargs):
        self.request = kwargs.pop('request', None)
        super(NuovoConto, self).__init__(*args, **kwargs)


    def clean_nome(self):
        nome = self.cleaned_data.get("nome")
        if self.request:
            conti = AccountService.get_conti_utente(UserService.get_utenti_by_user(self.request.user.pk))
            if any(conto.nome.lower() == nome for conto in conti):
                raise ValidationError("This account name is already taken.")

        return nome
```

Figura 5.1: Codice della classe `NuovoContoForm`

```python
def get_company_data(request, company_name):
    api_key = os.getenv('FINHUB_API_KEY')
    url = f'https://finnhub.io/api/v1/search?q={company_name}&token={api_key}'

    try:
        with urllib.request.urlopen(url) as response:
            data = json.loads(response.read().decode())


        filtered_results = [
            company for company in data.get('result', [])
            if 'Common Stock' in company.get('type', '') and '.' not in company['displaySymbol']
        ]
        filtered_results.sort(key=lambda x: x['displaySymbol'])

        return JsonResponse({'result': filtered_results})

    except Exception as e:
        return JsonResponse({'error': str(e)}, status=500)
```

Figura 5.2: Codice della funzione che utilizza l'API *Finnhub*

L'utilizzo combinato di queste tre API consente all'utente di svolgere diverse operazioni legate alla gestione degli investimenti in tempo reale; queste operazioni sono di seguito

```python
@login_required
def get_stock_data(request, symbol):
    api_key = settings.ALPHA_VANTAGE_API_KEY
    url = f'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&apikey={api_key}'

    try:
        with urllib.request.urlopen(url) as response:
            data = json.loads(response.read().decode())
        print(data)
        if 'Error Message' in data:
            return JsonResponse({'error': 'Invalid symbol'}, status=400)

        return JsonResponse(data)

    except urllib.error.URLError as e:
        return JsonResponse({'error': str(e)}, status=500)
```

Figura 5.3: Codice della funzione che utilizza l'API *Alphavantage*

```python
@login_required
def get_latest_price_in_euro(request, symbol):
    stock_data_response = get_stock_data(request, symbol)
    if isinstance(stock_data_response, JsonResponse) and stock_data_response.status_code != 200:
        return stock_data_response

    stock_data = json.loads(stock_data_response.content)
    if 'Time Series (Daily)' not in stock_data:
        return JsonResponse({'error': 'No time series data available for this symbol.'}, status=400)


    latest_date = next(iter(stock_data['Time Series (Daily)']))

    closing_price_usd = float(stock_data['Time Series (Daily)'][latest_date]['4. close'])

    url = 'https://api.exchangerate-api.com/v4/latest/USD'

    try:
        with urllib.request.urlopen(url) as response:
            data = json.loads(response.read().decode())
            exchange_rate = data['rates']['EUR']

    except urllib.error.URLError as e:
        exchange_rate = None

    if exchange_rate is None:
        return JsonResponse({'error': 'Unable to fetch exchange rate'}, status=500)

    closing_price_eur = closing_price_usd * exchange_rate

    response_data = {
        'symbol': symbol,
        'closing_price_usd': f"${closing_price_usd:.2f}",
        'closing_price_eur': f"{closing_price_eur:.2f}"
    }
    return JsonResponse(response_data)
```

Figura 5.4: Codice della funzione che utilizza l'API *ExchangeRate*

illustrate:

- *ricerca azienda*: l'utente può cercare un'azienda tramite il ticker e visualizzare il nome completo dell'azione corrispondente, garantendo una selezione accurata;
- *registrazione operazione di acquisto*: l'utente può registrare un'operazione di acquisto di

azioni, con il sistema che calcola il valore attuale della posizione, basandosi sul prezzo di chiusura più recente;

- *registrazione operazione di vendita*: l'utente può registrare un'operazione di vendita di azioni, con il sistema che calcola automaticamente l'importo ottenuto dalla vendita in base al prezzo di chiusura più recente;
- *monitoraggio portafoglio titolo*: l'utente può monitorare quotidianamente l'andamento del portafoglio grazie al ricalcolo giornaliero del valore delle azioni possedute.

5.4 Classe USER

Nell'implementazione di un'applicazione web, uno degli aspetti fondamentali è la gestione della registrazione. Django mette a disposizione la classe predefinita `USER` che si occupa di gestire gli utenti e le loro credenziali in modo sicuro. Essa fa parte di `django.contrib.auth`, che è un modulo di Django che fornisce un sistema completo di gestione dell'autenticazione degli utenti.

5.4.1 Registrazione dell'utente

Quando un nuovo utente si registra sulla piattaforma, il primo passo è la creazione di un oggetto di tipo `User`. Questo modello consente di memorizzare i dati essenziali per il login dell'utente, poiché esso include campi predefiniti, come username, password, email, first name e last name. Nel nostro progetto, il form `NuovoUtenteForm` raccoglie i dati necessari; una volta validati, questi vengono utilizzati per creare un oggetto `User`.

Nel nostro sistema è, però, necessario raccogliere e memorizzare ulteriori informazioni sull'utente. La soluzione implementata consiste nell'utilizzare l'entità Utente definita nella fase di progettazione, attraverso il suo modello Django corrispondente, includendo, però, un campo aggiuntivo, ovvero un riferimento all'oggetto `User` creato in precedenza.

In Figura 5.5 viene mostrato il codice della vista `registration`, che contiene la logica appena descritta.

L'autenticazione successiva alla registrazione è gestita, come detto in precedenza, con delle funzioni fornite dal modulo `django.contrib.auth`.

5.5 La piattaforma web

Le Figure 5.6, 5.7 e 5.8 mostrano la schermata iniziale della piattaforma; essa fornisce una panoramica delle funzionalità e include la sezione relativa alle FAQ.

Dalla schermata iniziale è possibile procedere con la registrazione o l'accesso alla piattaforma. La Figura 5.9 mostra la schermata relativa alla registrazione.

In caso di registrazione o accesso riuscito, l'utente viene reindirizzato alla dashboard principale, dalla quale è possibile scegliere l'area di interesse; essa è illustrata nella Figura 5.10. Di seguito verrano illustrate le funzionalità di ogni area.

5.5.1 Funzionalità area di budgeting personale

Cliccando sull'area "Personal Budgeting" presente nella dashboard, il sistema reindirizza l'utente alla sezione relativa al budgeting personale; le Figure 5.11 e 5.12 mostrano questa sezione.

Di seguito verranno illustrate le principali funzionalità di questa area della piattaforma:

```python
def registration(request):
    if request.method == "POST":
        form = NuovoUtente(request.POST)
        if form.is_valid():
            username = form.cleaned_data["username"]
            email = form.cleaned_data["email"]
            password = form.cleaned_data["password"]

            user = User.objects.create_user(username=username, email=email, password=password)
            Utente.objects.create(
                user_profile=user,
                username=form.cleaned_data['username'],
                email=form.cleaned_data['email'],
                password=form.cleaned_data['password'],
                nome=form.cleaned_data['nome'],
                cognome=form.cleaned_data['cognome'],
                data_di_nascita=form.cleaned_data['data_di_nascita'],
                indirizzo=form.cleaned_data.get('indirizzo', ''),
                telefono=form.cleaned_data['telefono'],
                sesso=form.cleaned_data['sesso'],
                monete_account=0,
                data_registrazione=timezone.now()
            )

            utente_loggato = authenticate(request, username=username, password=password)
            if utente_loggato is not None:
              auth_login(request, utente_loggato)
            return redirect('dashboard')
    else:
        form = NuovoUtente()

    context = {"form": form}
    return render(request, 'getstarted/registration.html', context)
```

Figura 5.5: Codice della funzione `registration`

Figura 5.6: Schermata iniziale di Budget Nest

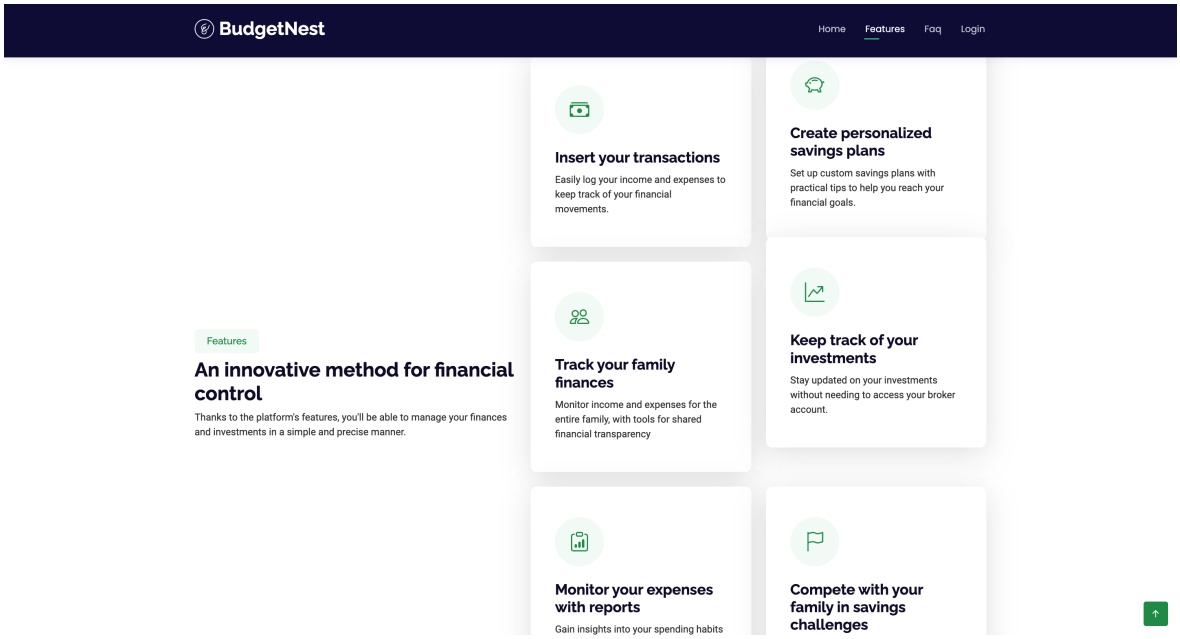

Figura 5.7: Panoramica delle funzionalità nella schermata iniziale di Budget Nest

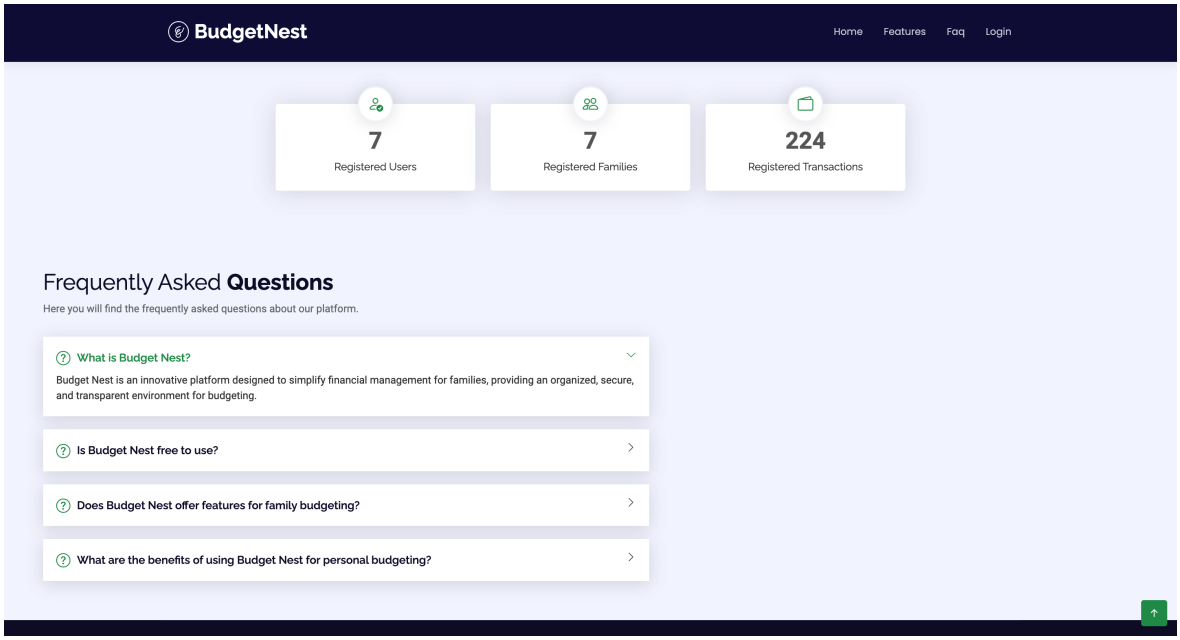

Figura 5.8: Sezione relativa alle FAQ nella schermata iniziale di Budget Nest

- *Gestione conti* : la parte superiore dell'area dedicata al budgeting personale è riservata alla gestione dei conti. Un nuovo conto può essere creato cliccando sul tasto verde apposito; a seguito di ciò si apre un form per l'inserimento dei dati relativi al conto (il form è illustrato nella Figura 5.13).

  In caso di esito positivo nella creazione del conto, il sistema aggiunge dinamicamente (senza necessità di ricaricare la pagina) un nuovo elemento nella sezione relativa alla gestione dei conti. Questo elemento rappresenta il conto appena creato. Ognuno di questi elementi è cliccabile; cliccando su di esso è possibile eliminarlo, modificarne il nome o eliminare una transazione. Nella Figura 5.14 viene mostrato un esempio di tale funzionalità.

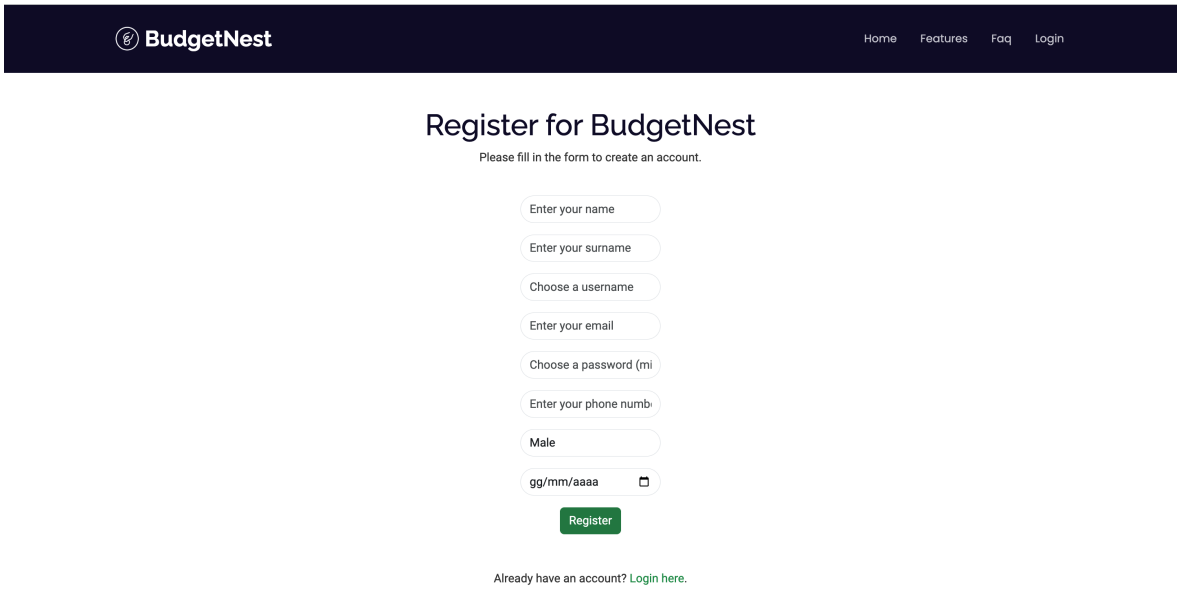

Figura 5.9: Schermata di registrazione della piattaforma Budget Nest

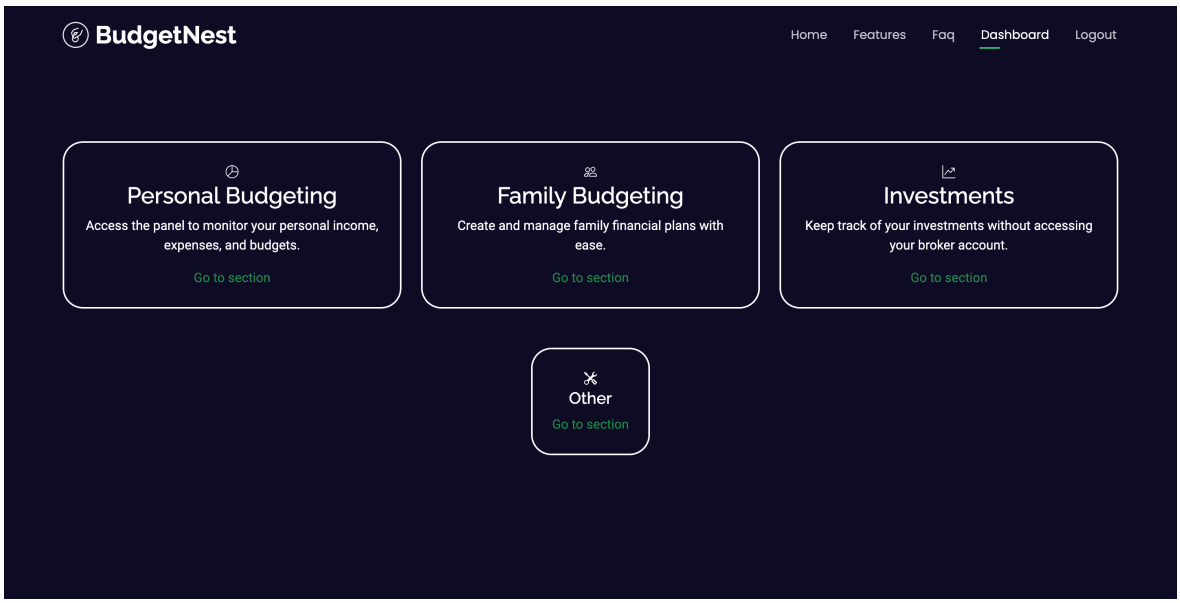

Figura 5.10: Dashboard principale della piattaforma Budget Nest

- *Gestione piani risparmio*: un nuovo piano di risparmio può essere creato cliccando sul tasto "New savings plan" , quest'ultimo apre un form per l'inserimento dei dati relativi al piano di risparmio (il form è illustrato nella Figura 5.15).

  In caso di esito positivo della creazione del piano di risparmio, il sistema aggiunge dinamicamente (senza necessità di ricaricare la pagina) un nuovo elemento nella sezione relativa ai piani risparmio. Ognuno di questi elementi è cliccabile e, cliccando su di esso, è possibile eliminarlo, cambiarne la data di scadenza o cambiarne l'obbiettivo. Nella Figura 5.16 è mostrato un esempio di tale funzionalità.

- *Gestione obbiettivi di spesa*: un nuovo obbiettivo di spesa può essere creato cliccando sul tasto "New spending goal"; quest'ultimo apre un form per l'inserimento dei dati relativi

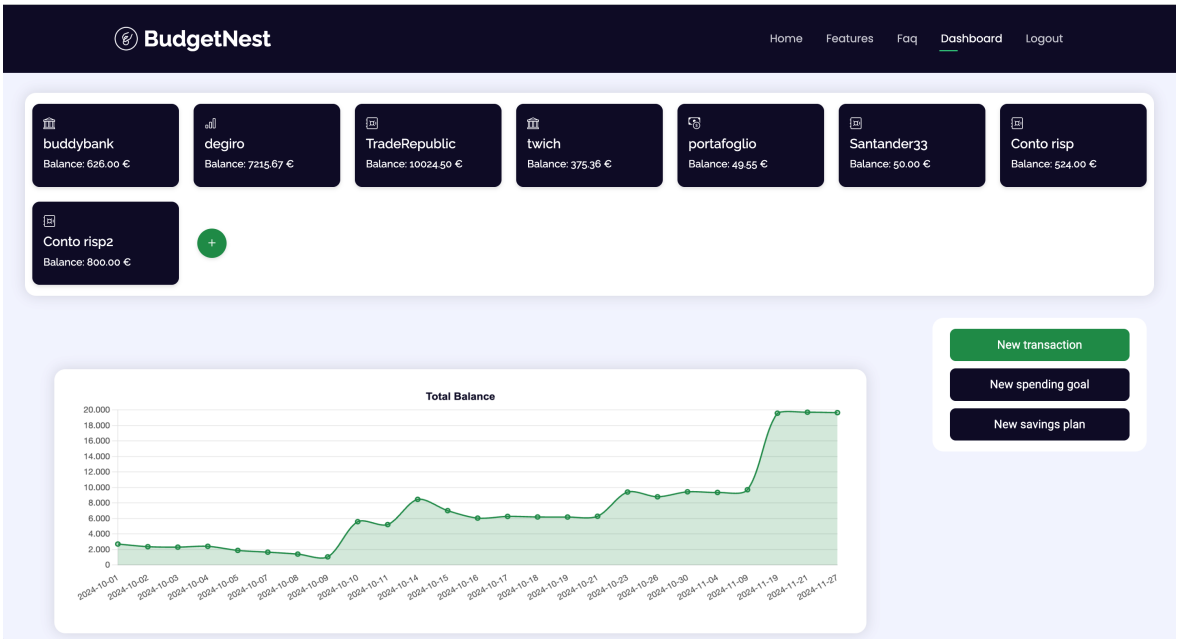

Figura 5.11: Prima parte della sezione relativa al budgeting personale della piattaforma Budget Nest

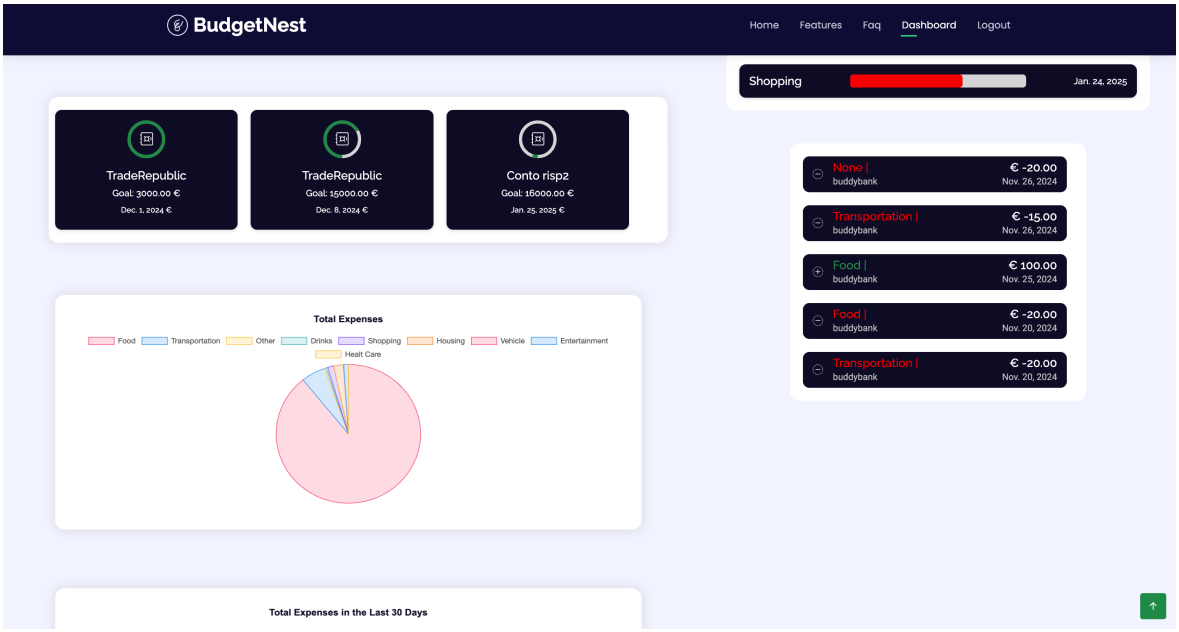

Figura 5.12: Seconda parte della sezione relativa al budgeting personale della piattaforma Budget Nest

all'obbiettivo di spesa (il form è illustrato nella Figura 5.17).

In caso di esito positivo della creazione dell'obbiettivo di spesa, il sistema aggiunge dinamicamente (senza necessità di ricaricare la pagina) un nuovo elemento nella sezione relativa agli obbiettivi di spesa. Ognuno di questi elementi è cliccabile e, cliccando su di esso, è possibile eliminarlo, cambiarne la data di scadenza o cambiarne l'obbiettivo. Nella Figura 5.18 viene mostrato un esempio di tale funzionalità.

- *Gestione transazioni*: una nuova transazione può essere inserita cliccando sul tasto "New transaction"; quest'ultimo apre un form per l'inserimento dei dati relativi alla nuova

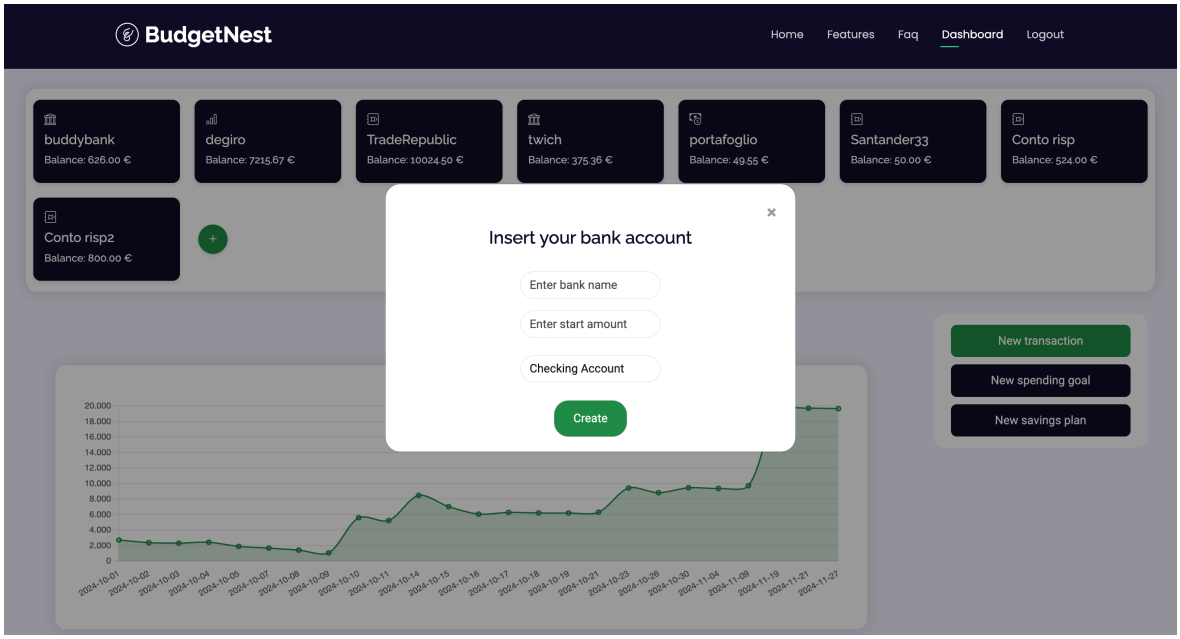

Figura 5.13: Form per la creazione di un nuovo conto

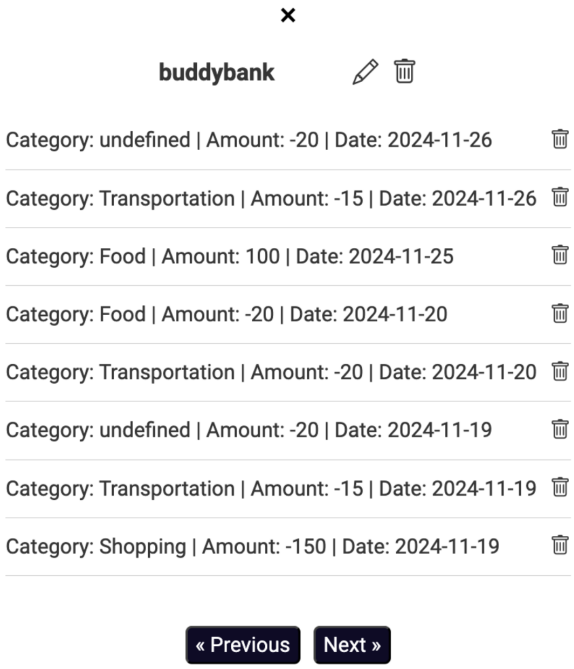

Figura 5.14: Interfaccia di gestione di un conto

transazione (il form è illustrato nella Figura 5.19).

In caso di esito positivo della registrazione della transazione, il sistema aggiorna dinamicamente diversi elementi della dashboard (senza necessità di ricaricare la pagina). Di seguito vengono illustrati gli aggiornamenti che il sistema effettua:

– *Saldo conto*: il saldo del conto relativo alla transazione viene immediatamente aggiornato. Il sistema calcola il nuovo saldo tenendo conto dell'importo della transazione e se si tratta di un'entrata o di un'uscita; automaticamente, esso

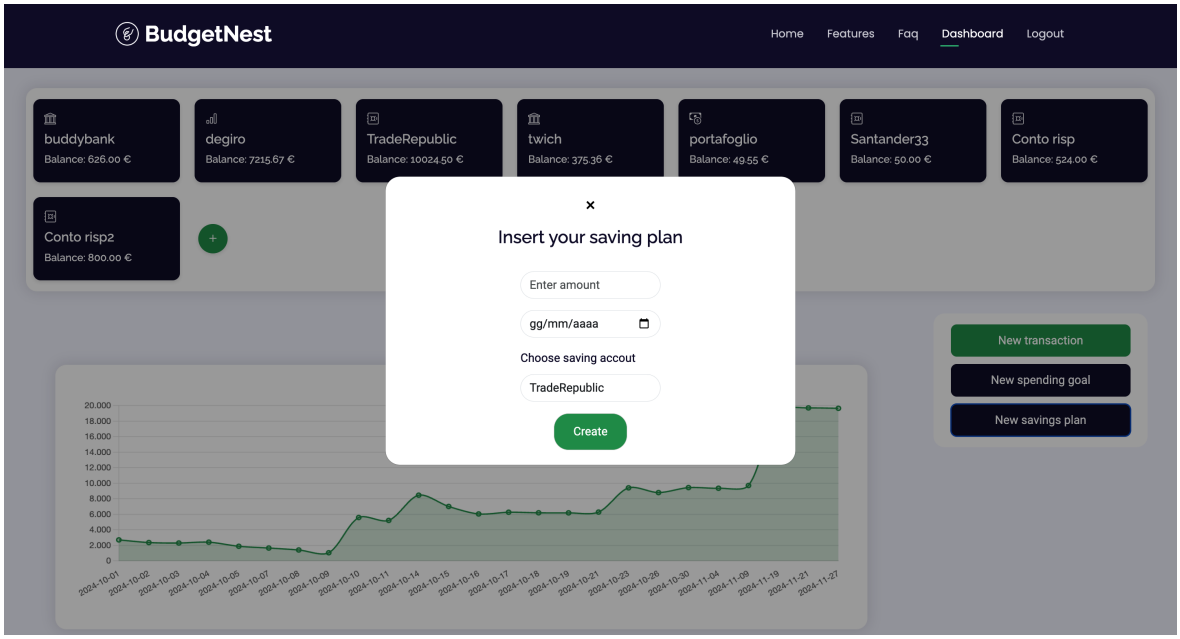

Figura 5.15: Form per la creazione di un nuovo piano di risparmio

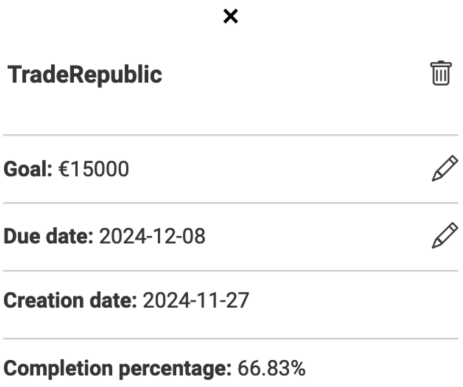

Figura 5.16: Interfaccia di gestione di un piano di risparmio

mostra il risultato aggiornato nella sezione dedicata ala gestione dei conti.

- *Progresso piano risparmio*: se il conto relativo alla transazione è associato a un piano di risparmio, il sistema aggiorna il progresso verso tale piano, ricalcolando la percentuale di completamento.
- *Progresso obbiettivo di spesa*: se la categoria relativa alla transazione è associata ad un obbiettivo di spesa, il sistema aggiorna il progresso verso tale obbiettivo, ricalcolando la percentuale di completamento.
- *Transazioni più recenti*: nella sezione relativa alle transazioni più recenti, il sistema inserisce un nuovo item relativo alla transazione appena registrata.

5.5.2 Funzionalità area di budgeting famigliare

Cliccando sull'area "Family Budgeting" presente nella dashboard, il sistema reindirizza l'utente in un'altra dashboard relativa al budgeting famigliare (essa è illustata nella Figura

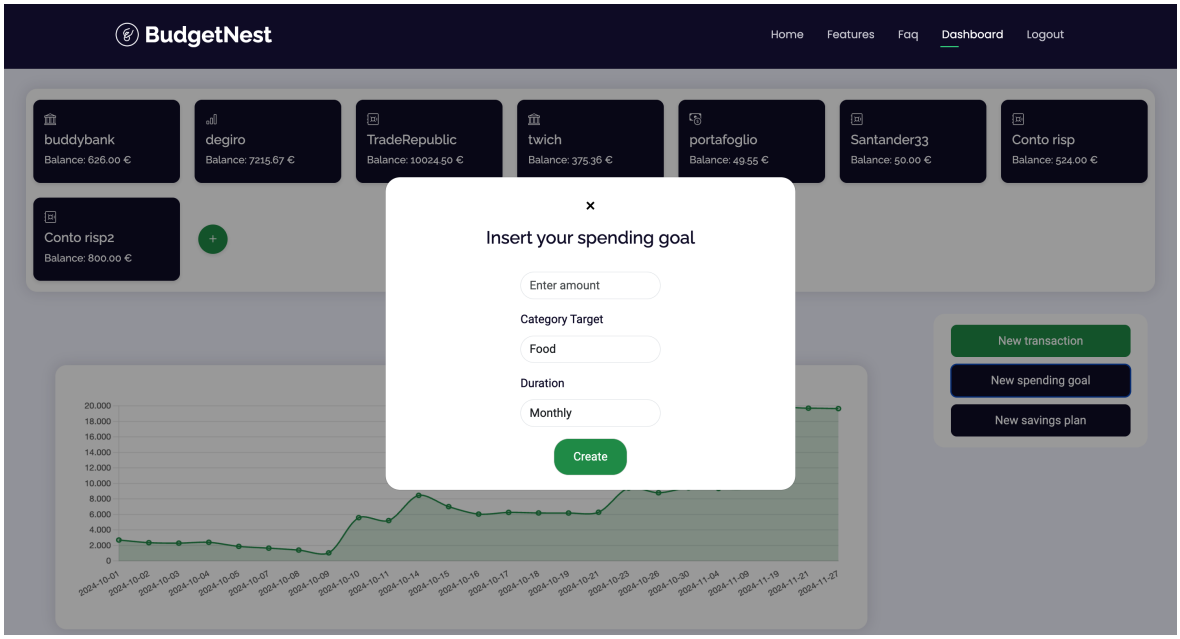

Figura 5.17: Form per la creazione di un nuovo obbiettivo di spesa

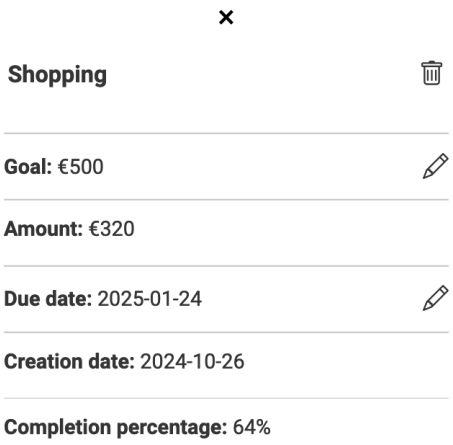

Figura 5.18: Interfaccia di gestione di un obbiettivo di spesa

5.20). In questa dashboard l'utente ha tre opzioni, che sono di seguito illustrate:

- l'utente può creare una nuova famiglia;
- l'utente può entrare in una famiglia inserendo un codice;
- l'utente può accedere all'area delle famiglie di cui è già membro.

Cliccando su una famiglia, il sistema reindirizza l'utente alla sezione dedicata al budgeting familiare per essa. La Figura 5.21 mostra questa sezione.

Di seguito verranno illustrate le principali funzionalità di questa area della piattaforma:

- *Gestione conti condivisi*: in quest'area, l'utente può gestire i conti come nella sezione "Budgeting personale". La differenza principale è che ogni nuovo conto creato in quest'area verrà intestato a tutti i membri della famiglia. Cliccando su un conto, se la

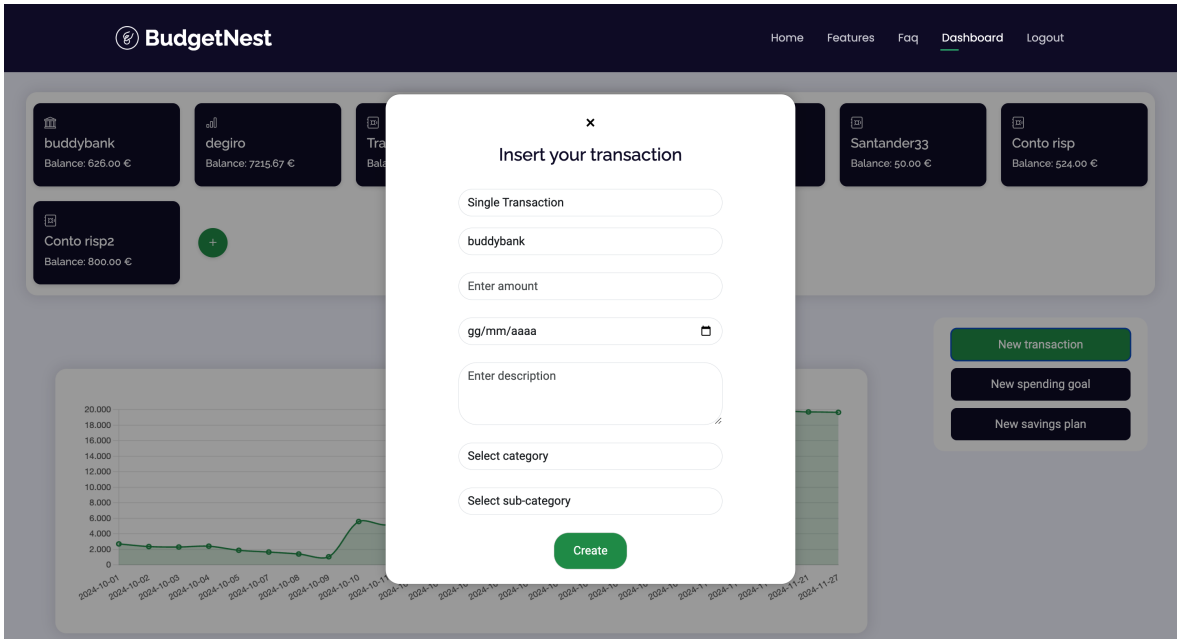

Figura 5.19: Form per l'inserimento di una nuova transazione

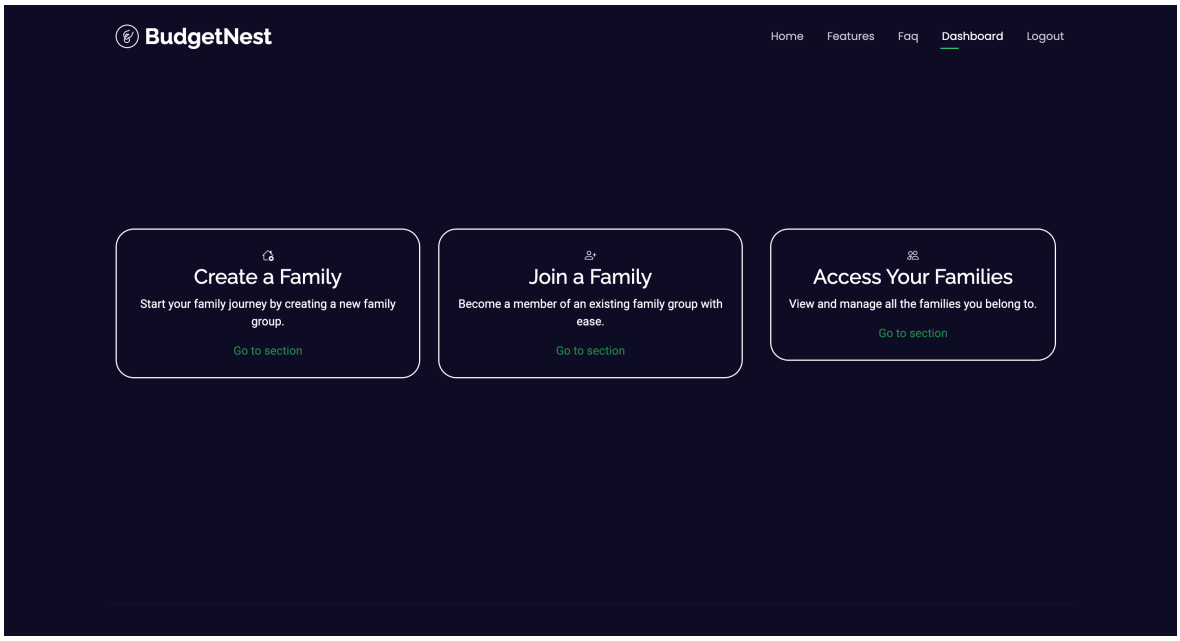

Figura 5.20: Dashboard dell'area "Family Budgeting" della piattaforma Budget Nest

transazione è stata effettuata dall'utente, sarà possibile eliminarla. In caso contrario, il sistema mostrerà il nome del membro della famiglia che ha effettuato la transazione. Un esempio di tale funzionalità è mostrato in Figura 5.22.

- *Gestione piani risparmio*: in quest'area, l'utente può gestire i piani di risparmio come nella sezione "Budgeting personale". La differenza principale è che ogni nuovo piano potrà essere gestito da tutti i membri della famiglia, poiché esso può essere associato solo a conti condivisi.
- *Gestione transazioni*: in quest'area, l'utente può registrare delle transazioni come nella sezione "Budgeting personale". Se la categoria relativa alla transazione è associata ad

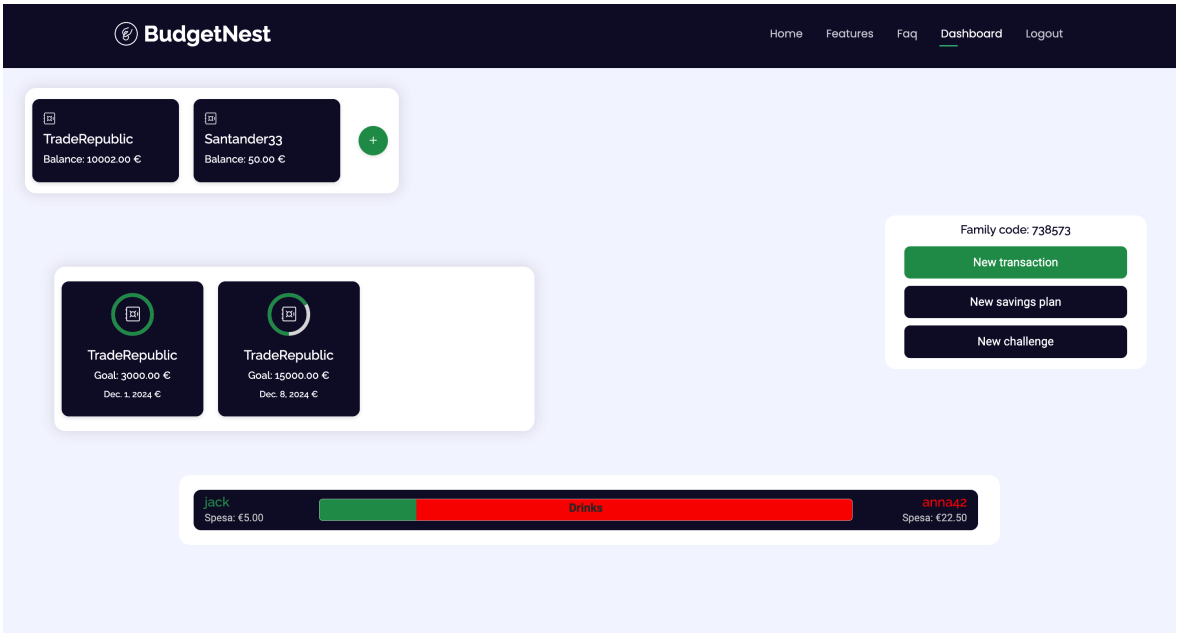

Figura 5.21: Sezione relativa al budgeting famigliare della piattaforma Budget Nest

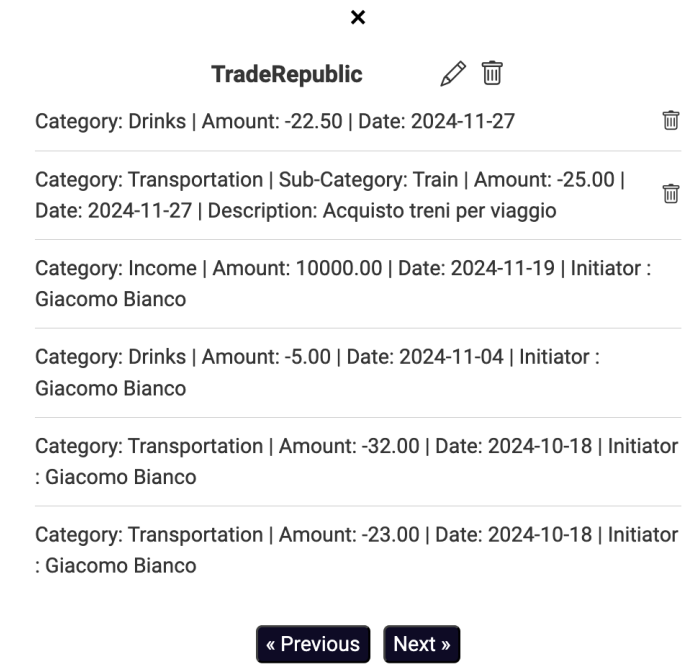

Figura 5.22: Interfaccia di gestione di un conto condiviso

una sfida, il sistema aggiornerà dinamicamente i dati di quest'ultima.

- *Gestione sfide famigliari*: un nuova sfida può essere creata cliccando sul tasto "New challenge"; quest'ultimo apre un form per l'inserimento dei dati relativi alla sfida (il form è illustrato nella Figura 5.23).

  In caso di esito positivo nella creazione della sfida, il sistema aggiunge dinamicamente (senza necessità di ricaricare la pagina) un nuovo elemento nella sezione relativa alle sfide della famiglia. Ognuno di questi elementi è cliccabile e, cliccando su di esso, è

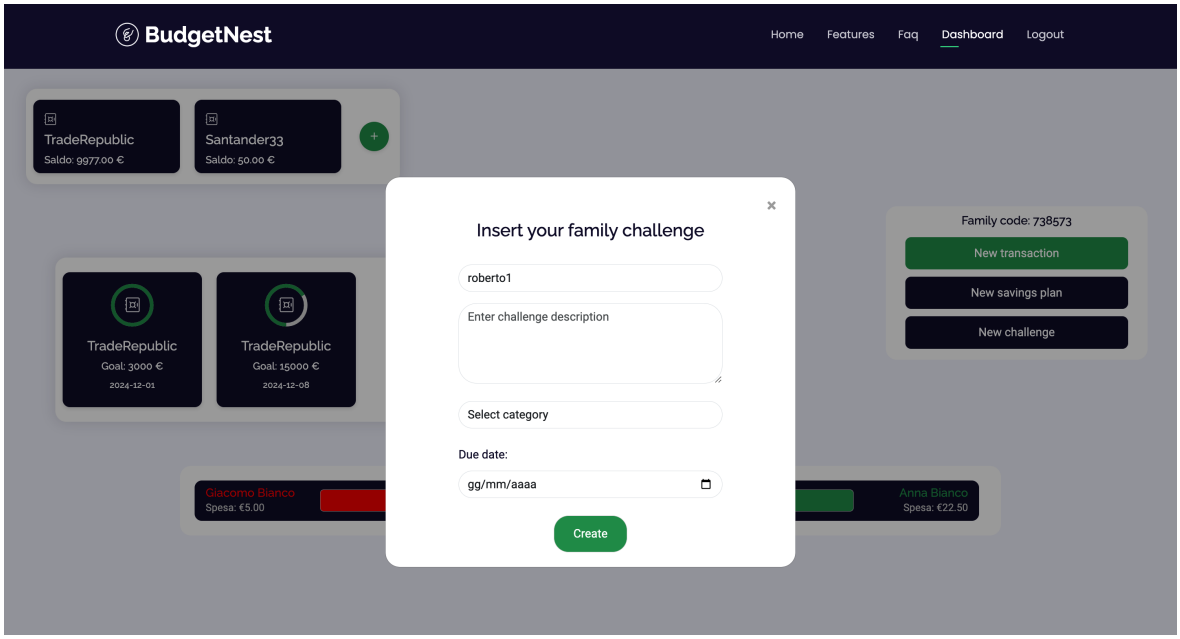

Figura 5.23: Form per la creazione di una nuova sfida

possibile eliminarlo o cambiarne la data di scadenza. Nella Figura 5.24 è mostrato un esempio di tale funzionalità.

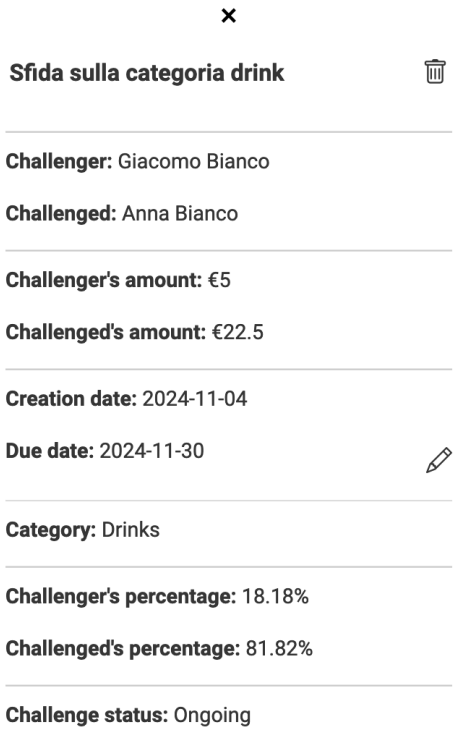

Figura 5.24: Interfaccia di gestione di una sfida famigliare

5.5.3 Funzionalità dell'area di gestione degli investimenti

Cliccando sulla area "Investments" nella dashboard, il sistema reindirizza l'utente alla sezione relativa alla gestione degli investimenti; le Figure 5.25 e 5.26 mostrano questa sezione.

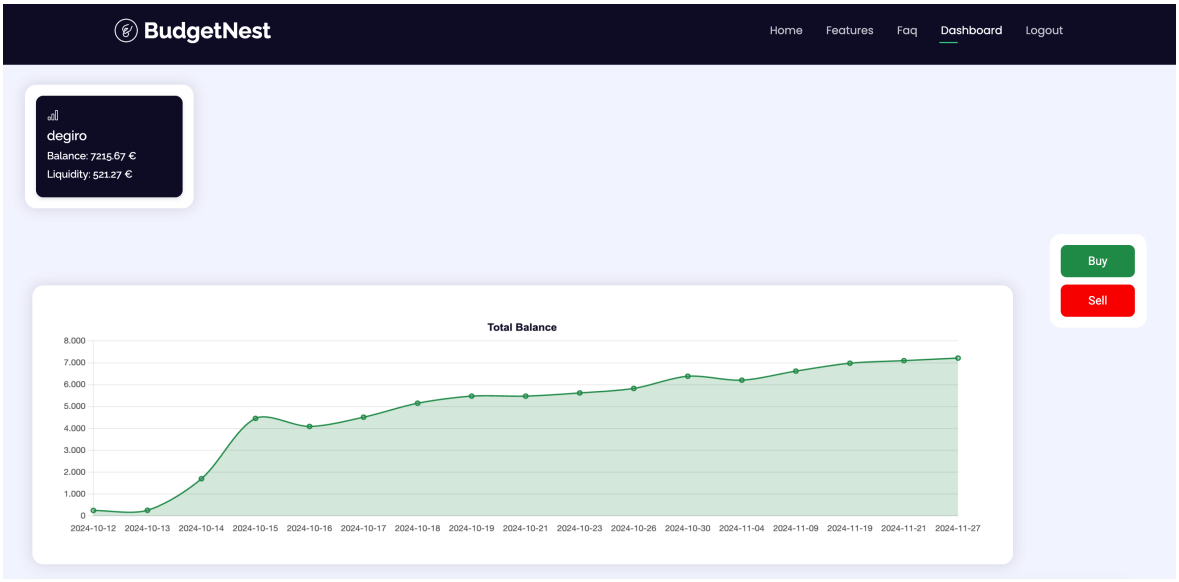

Figura 5.25: Prima parte della sezione relativa alla gestione deli investimenti della piattaforma Budget Nest

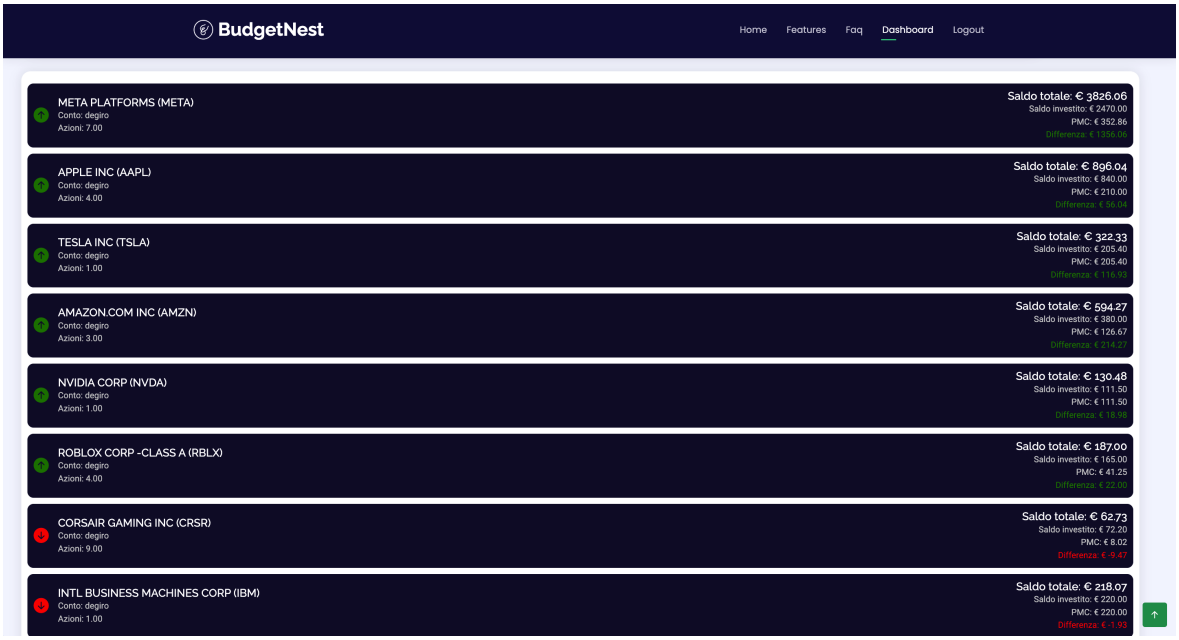

Figura 5.26: Seconda parte della sezione relativa alla gestione deli investimenti della piattaforma Budget Nest

Di seguito verranno illustrate le principali funzionalità di questa area della piattaforma:

- *Registrazione operazione di acquisto*: l'utente può registrare, cliccando il tasto "Buy", un'operazione di acquisto di azioni. Dopo aver cercato l'azienda tramite il ticker e selezionato lo strumento corretto, per registrare l'acquisto sarà necessario che l'utente

indichi il numero di azioni e il prezzo di acquisto. Il form di inserimento è illustrato nella Figura 5.27)

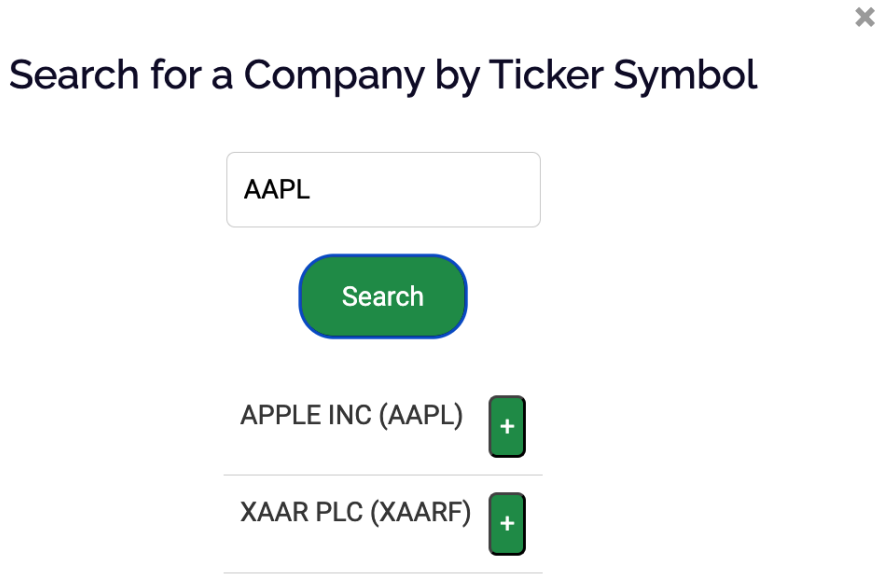

Figura 5.27: Form per la registrazione di un'operazione di acquisto di azioni

In caso di esito positivo dell'operazione, il sistema aggiunge dinamicamente (senza necessità di ricaricare la pagina) un nuova posizione nella sezione relativa al tracciamento delle posizioni. Se esiste già una posizione relativa allo strumento finanziario indicato nell'operazione, il sistema aggiorna i dettagli della posizione esistente invece di crearne una nuova.

- *Registrazione operazione di vendita*: l'utente può registrare, cliccando il tasto "Sell", una operazione di vendita di azioni. Dopo aver selezionato una delle posizioni attive, per registrare la vendita, sarà necessario che l'utente indichi il numero di azioni da vendere. Il form di vendita è illustrato nella Figura 5.28.

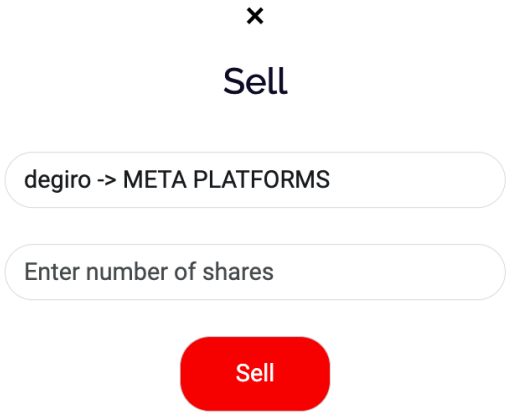

Figura 5.28: Form per la registrazione di un'operazione di vendita di azioni

5.5.4 Funzionalità aggiuntive

Cliccando sull'area "Other" nella dashboard, il sistema reindirizza l'utente a un'area che offre funzionalità aggiuntive (quest'area è illustrata Figura 5.29).

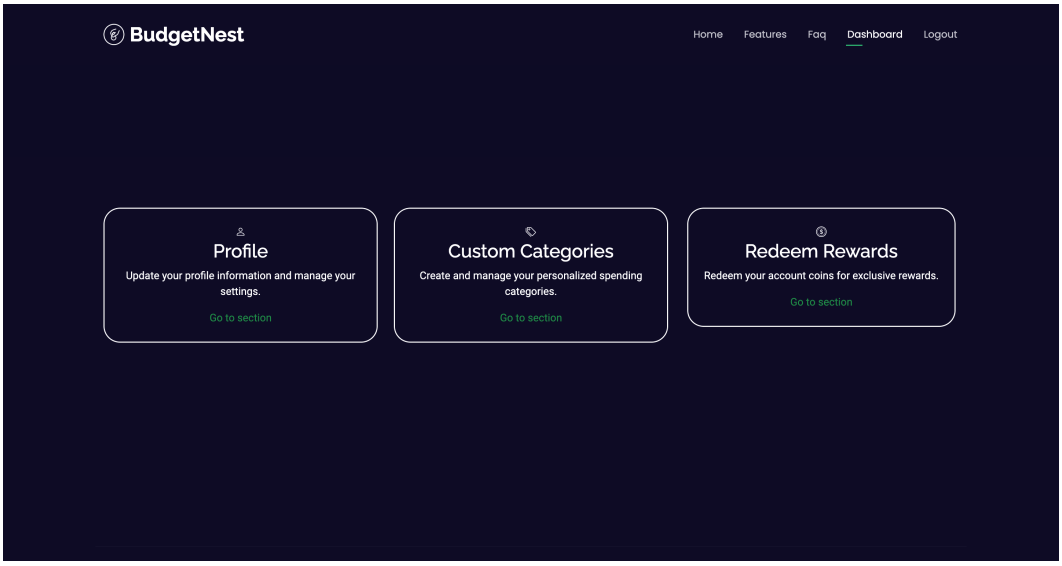

Figura 5.29: Area delle funzionalità aggiuntive della piattaforma Budget Nest

Il sistema mette a disposizione tre funzionalità aggiuntive, che sono di seguito illustrate:

- *Modifica profilo*: cliccando sulla sezione "Profile", il sistema reindirizza l'utente ad una schermata dove quest'ultimo ha la possibilità di modificare i dati del proprio profilo; questa schermata è mostrata nella Figura 5.30.
- *Sottocategorie*: cliccando sulla sezione "Custom Categories", il sistema reindirizza l'utente ad una schermata dove quest'ultimo ha la possibilità di creare delle sottocategorie personalizzate; questa schermata è mostrata nella Figura 5.31.
- *Riscatto premi*: cliccando sulla sezione "Redeem Rewards", il sistema reindirizza l'utente ad una schermata dove quest'ultimo ha la possibilità di riscattare dei premi messi a disposizione dalla piattaforma; questa schermata è mostrata nella Figura 5.32.

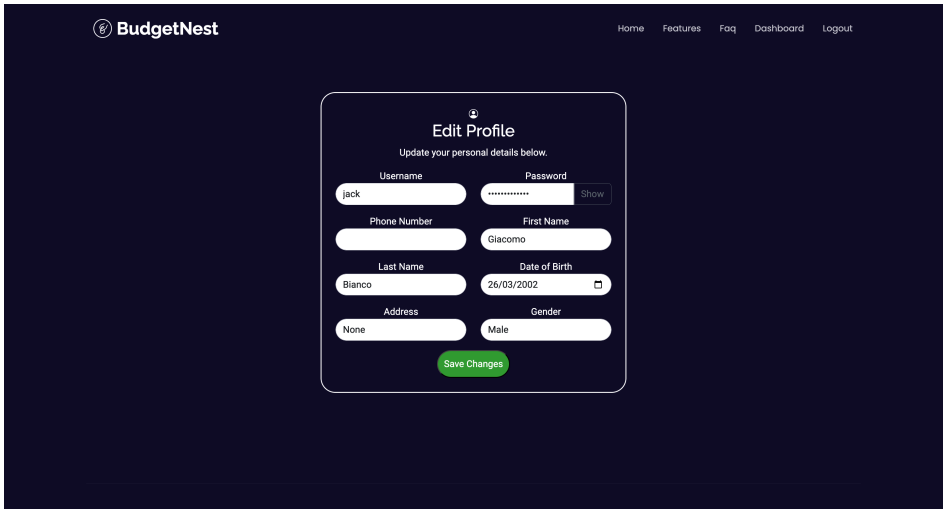

Figura 5.30: Schermata di modifica del profilo

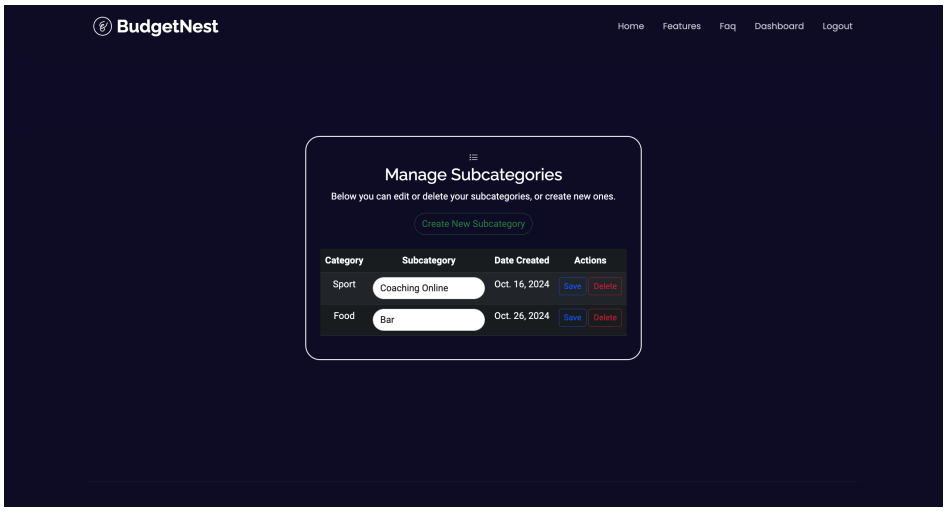

Figura 5.31: Schermata di personalizzazione delle sottocategorie

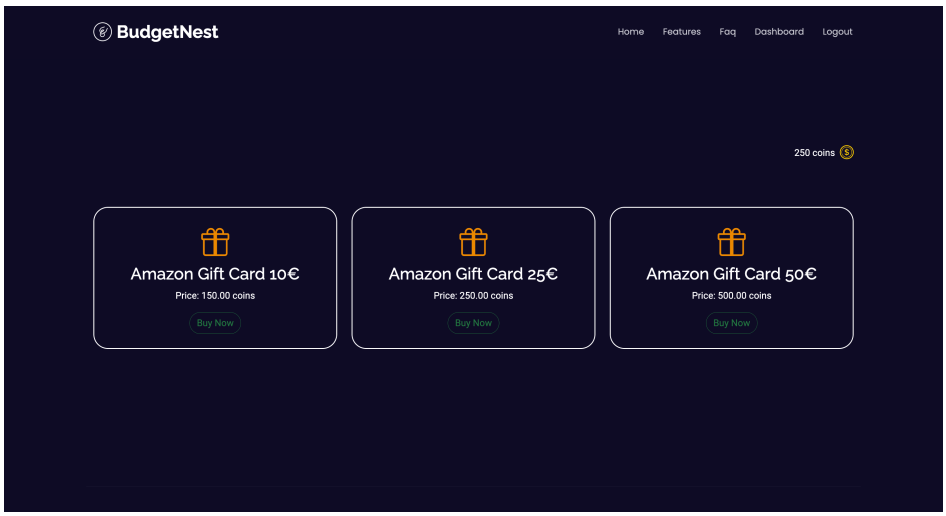

Figura 5.32: Schermata di riscatto dei premi

CAPITOLO 6

Discussione (confronto e SWOT Analysis)

In questo capitolo verrà presentata un'analisi di mercato della piattaforma Budget Nest, oggetto di questa tesi. Nella prima parte verrà effettuato un confronto tra Budget Nest e le principali soluzioni di gestione finanziaria disponibili sul mercato, evidenziando la sua posizione rispetto alle altre piattaforme. Successivamente sarà illustrata l'analisi SWOT del progetto, utile per valutare gli aspetti critici e le potenzialità. Infine, verranno elencati alcuni dei possibili sviluppi futuri della piattaforma.

6.1 Confronto con soluzioni esistenti

Dopo aver analizzato le principali piattaforme di gestione finanziaria disponibili sul mercato e aver presentato le funzionalità di quest'ultima, è ora possibile comprenderne e delinearne la posizione di Budget Nest rispetto a tali soluzioni. Le principali piattaforme attualmente disponibili offrono una vasta gamma di funzionalità, ma presentano alcune lacune che lasciano spazio a miglioramenti. In particolare:

- *PocketGuard*: questa piattaforma si distingue per la sua semplicità nella gestione delle spese quotidiane. Tuttavia, essa manca di funzionalità collaborative che possano coinvolgere più utenti nella gestione di un budget comune.
- *YNAB*: questa piattaforma è considerata uno dei sistemi più avanzati e completi per la gestione del budget. Tuttavia, la sua elevata complessità richiede all'utente tempo e impegno per comprenderne appieno il funzionamento, il che può rappresentare una barriera iniziale per coloro che hanno meno dimestichezza con strumenti tecnologici o una limitata propensione a dedicare tempo alla gestione della finanza personale.
- *PocketSmith*: questa piattaforma è nota per le sue avanzate funzionalità di proiezione finanziaria; tuttavia, rispetto a Budget Nest, essa potrebbe risultare meno intuitiva per chi cerca una soluzione semplice e immediata.

In questo contesto, Budget Nest si presenta come un'alternativa più accessibile e meno sofisticata, puntando sulla semplicità d'uso per gli utenti che cercano un'esperienza di gestione finanziaria immediata, senza la necessità di comprendere concetti complessi prima di utilizzarla.

6.2 Analisi SWOT

L'analisi SWOT è una tecnica utilizzata per identificare i punti di forza, i punti di debolezza, le opportunità e le minacce di un'azienda o di un progetto specifico. SWOT, infatti, è un acronimo che sta per Strengths (punti di forza), Weaknesses (punti di debolezza), Opportunities (opportunità) e Threats (minacce). Ciascuno di questi fattori deve essere analizzato con attenzione per pianificare in modo efficace lo sviluppo di un progetto. La Figura 6.1 presenta alcune domande utili per identificare tali fattori.

Figura 6.1: Domande da porsi in un'analisi SWOT

Di seguito viene illustrata l'analisi SWOT della piattaforma Budget Nest, oggetto di questa tesi:

- *Strengths (punti di forza)*: i punti di forza rappresentano gli aspetti positivi, le risorse interne e le caratteristiche che contribuiscono al successo di un progetto. Di seguito vengono presentati i punti di forza di Budget Nest:
  - *Interfaccia semplice e intuitiva*: la piattaforma è progetatta per essere facilmente utilizzabile da utenti che non hanno esperienza nel campo della gestione finanziaria; infatti, la sua semplicità riduce la curva di apprendimento degli utenti; essa rappresenta quanto tempo e sforzo un utente deve dedicare per diventare competente nell'uso di una nuova tecnologia o piattaforma.
  - *Dinamicità*: indipendentemente dalle azioni intraprese dall'utente, la piattaforma aggiorna i dati in tempo reale senza la necessità di ricaricare la pagina. Ciò garantisce un'esperienza utente fluida e moderna, senza interruzioni.
  - *Gestione famigliare*: la piattaforma va oltre il tradizionale concetto di budgeting personale, offrendo anche funzionalità per la gestione di conti e piani di risparmio condivisi tra più utenti.
  - *Tracciamento degli investimenti*: la piattaforma permette di registrare facilmente le operazioni di acquisto e vendita di azioni, aggiornando automaticamente le posizioni senza bisogno di ricaricare la pagina. L'assenza della necessità di accedere al

broker per monitare il proprio portafoglio titoli consente di evitare errori impulsivi legati alle emozioni.
  - *Sfide famigliari*: la piattaforma integra un sistema di gamification che motiva gli utenti a sfidarsi tra loro e a vincere le sfide per acquistare dei premi; questo sistema crea, di conseguenza, un incentivo al risparmio e alla gestione più consapevole delle finanze.
  - *Completezza*: la piattaforma offre funzionalità per il budgeting personale, il budgeting familiare e il tracciamento degli investimenti, permettendo alla maggior parte dei possibili utenti di soddisfare tutte le principali esigenze finanziarie in un unico luogo, senza la necessità di ricorrere ad ulteriori servizi esterni per altre funzionalità.

- *Weaknesses (Punti di debolezza)*: i punti di debolezza rappresentano gli aspetti interni di un'azienda o progetto che ne limitano le prestazioni, la competitività o l'efficacia. Di seguito vengono presentati i punti di debolezza di Budget Nest:

  - *Limitata integrazione con servizi esterni*: la piattaforma manca di integrazione con conti bancari esterni; tale funzionalità consentirebbe all'utente di semplificare l'inserimento delle transazioni.
  - *Assenza di funzionalità avanzate di reportistica*: la piattaforma non fornisce funzionalità avanzate di reportistica per utenti più esperti, i quali potrebbero desiderare proiezioni a lungo termine o analisi approfondite sull'andamento dei propri investimenti.
  - *Dipendenza dalla connessione internet*: Budget Nest è una piattaforma web, il che implica che gli utenti devono essere sempre connessi a Internet per poterla utilizzare.
  - *Mancanza di un applicazione*: seppur il front-end della piattaforma sia stato sviluppato con Bootstrap, una tecnologia che rende l'interfaccia utente di Budget Nest reattiva e accessibile su dispositivi di dimensioni diverse, essa non offre l'esperienza nativa che un'app dedicata garantisce su dispositivi mobili.

- *Opportunities (Opportunità)*: le opportunità rappresentano fattori esterni positivi che un'azienda o un progetto può sfruttare per migliorare la propria posizione nel mercato. Di seguito vengono presentate le opportunità di Budget Nest:

  - *Crescita dell'interesse per la gestione finanziaria personale e familiare*: le persone sono sempre più interessate a prendere il controllo delle proprie finanze, creando una domanda maggiore per piattaforme come Budget Nest.
  - *Aumento della domanda di soluzioni per proteggere i risparmi*: l'aumento dell'inflazione sta spingendo sempre più persone a cercare modalità per proteggere e far crescere i propri risparmi. Questo crescente interesse verso gli investimenti, sia tra i neofiti che tra gli esperti, rappresenta un'opportunità per piattaforme come Budget Nest.
  - *Espansione delle banche digitali*: l'espansione delle banche digitali sta alimentando la crescita di soluzioni finanziarie online, creando un ambiente favorevole per piattaforme come Budget Nest.

- *Threats (Minacce)*: le minaccie rappresentano fattori esterni che potrebbero ostacolare lo sviluppo o il successo di un'azienda o un progetto. Di seguito vengono presentante le minacce di Budget Nest:

- *Concorrenza crescente*: il settore delle piattaforme di budgeting e gestione finanziaria è sempre più competitivo, con numerose aziende che offrono funzionalità simili.
- *Sicurezza dei dati e minacce informatiche*: le piattaforme di budgeting gestiscono informazioni sensibili; questo le porta ad essere esposte a minacce legate alla sicurezza dei dati.
- *Problemi di scalabilità*: se la piattaforma cresce rapidamente, potrebbero sorgere problemi legati alla gestione di un numero elevato di utenti o alla gestione dei dati.
- *Resistenza al cambiamento*: gli utenti che già utilizzano altre soluzioni potrebbero non essere disposti a migrare verso una nuova piattaforma, principalmente a causa dei dati storici già acquisiti, come le informazioni sulle spese passate. Questo può rappresentare un ostacolo significativo, poiché trasferire tali dati e adattarsi a un nuovo sistema potrebbe risultare molto dispendioso in termini di tempo.

6.3 Possibili sviluppi futuri

Poiché la piattaforma è in continua evoluzione, ci sono diverse aree in cui Budget Nest può essere migliorata per rispondere meglio alle esigenze degli utenti. Di seguito sono illustrate alcune aree di sviluppo concrete, che potrebbero apportare miglioramenti significativi alla piattaforma:

- *Sviluppo di un'app mobile nativa*: si potrebbe pensare di progettare e implementare un'app dedicata per l'utilizzo mobile della piattaforma. Questo offrirebbe un'esperienza utente migliore su schermi di piccole dimensioni.
- *Integrazione con criptovalute*: si potrebbe pensare di integrare nell'app funzionalità per registrare operazioni di acquisto e vendita su strumenti finanziari di nicchia, oltre alle tradizionali azioni e obbligazioni, come ad esempio le criptovalute.
- *Sezione dedicata all'apprendimento*: si potrebbe pensare di introdurre nella piattaforma una sezione dedicata all'apprendimento dei principali concetti del settore finanziario.
- *Introduzione di funzionalità social*: si potrebbe pensare di creare una community dove gli utenti possono condividere consigli, obiettivi e progressi.
- *Integrazione con conti bancari reali*: si potrebbe pensare di implementare una funzionalità che permetta agli utenti di collegare direttamente i propri conti bancari alla piattaforma, in modo da automatizzare la registrazione di entrate e uscite.

CAPITOLO 7

Conclusioni

Nel corso della presente tesi è stata realizzata una piattaforma web concepita per aiutare gli utenti a organizzare e monitorare efficacemente le proprie finanze, con un focus specifico sulla pianificazione delle spese, il monitoraggio degli investimenti e la gestione condivisa delle risorse economiche. Il progetto è nato dall'analisi del panorama attuale delle soluzioni di budgeting e dalla crescente esigenza delle persone di trovare strumenti efficaci per tracciare le proprie spese, individuando aree di miglioramento per realizzare una piattaforma completa, scalabile e intuitiva. Successivamente, è stato analizzato lo sviluppo delle tecnologie web, a partire dalle prime pagine statiche fino all'evoluzione verso soluzioni più complesse, per inquadrare il contesto e le motivazioni che hanno portato alla nascita dei framework. Questo percorso ha fornito le basi per introdurre Django, la tecnologia principale utilizzata per lo sviluppo di questa piattaforma. La scelta di Django ha reso l'intera fase di sviluppo più semplice ed efficiente, grazie agli strumenti offerti e alla versatilità di Python. La piattaforma risultante offre funzionalità per il budgeting personale, familiare e per il monitoraggio degli investimenti, arricchite da un'esperienza utente intuitiva e dinamica. Dal confronto con soluzioni esistenti, è emerso che la nostra piattaforma costituisce un'alternativa valida, sebbene ci siano diverse opportunità di miglioramento. In particolare, potrebbe essere possibile implementare la connessione diretta ai conti bancari, semplificando la registrazione delle transazioni e migliorando l'esperienza utente. Inoltre, lo sviluppo di un'applicazione nativa per dispositivi mobili amplierebbe l'accessibilità della piattaforma e il suo potenziale bacino di utenti, mentre la creazione di una community permetterebbe di condividere strategie ed esperienze. Lo sviluppo di questa piattaforma potrebbe essere il primo passo verso la creazione di una soluzione in grado di supportare concretamente le persone nella gestione delle proprie finanze.

Bibliografia

ATZENI, P., CERI, S., PARABOSCHI, S. e TORLONE, R. (2023), *Basi di dati. Con connect*, McGraw-Hill Education.

BERI, M. (2009), *Sviluppare applicazioni con Django. Guida alla programmazione web*, Apogeo.

BURCH, C. (2010), «Django, a web framework using Python: tutorial presentation», *Journal of Computing Sciences in Colleges*, vol. 25.

CASTELLUCCI, P. (2009), *Dall'ipertesto al Web. Storia culturale dell'informatica*, Apogeo.

CHAN, T. (2024), *Django, Il framework web ad alto livello per Python: La guida definitiva al web developement per siti dinamici e moderni*, Independently published.

CRISCIONE, A. (2006), *Web e storia contemporanea*, Carocci.

DEGNI, R. (2023), *PRO Web Developer - La guida definitiva*, Independently published.

DOSSENA, F. (2022), *Educazione finanziaria for dummies*, Hoepli.

DUCKETT, J. (2017), *HTML e CSS. Progettare e costruire siti web*, Apogeo Education.

FLANAGAN, D. (2021), *Javascript. La guida definitiva. Dalle basi del linguaggio alle tecniche avanzate*, Apogeo.

HORSTMANN, C. S. e NECAISE, R. D. (2009), *Concetti di informatica e fondamenti di Python*, Apogeo Education.

KHALDI, J. (2024), *Database Relazionali: Progettazione Implementazione*, Independently published.

LAFUENTE, A. L. e RIGHI, M. (2006), *Internet e Web 2.0*, UTET University.

MEA, V. D., GASPERO, L. D. e SCAGNETTO, I. (2011), *Programmazione web. Lato server*, Apogeo Education.

PALUMBO, S. (2008), *Progettare Database - Modelli, metodologie e tecniche per l'analisi e la progettazione di basi di dati relazionali*, Lulu.com.

RUBIO, D. (2017), *Beginning Django*, Apress Berkeley.

VENTURI, L. (2024), *Python Avanzato: Tecniche Avanzate e Strumenti Potenti per Sviluppatori Esperti*, Independently published.

Sitografia

- A Guide to Functional Requirements - Nuclino – www.nuclino.com/articles/functional-requirements
- Alpha Vantage: Free Stock APIs in JSON and Excel – www.alphavantage.co
- Bootstrap – www.getbootstrap.com
- Caso d'uso – www.wikipedia.org/wiki/Caso_d%27uso_(informatica)
- Che cos'è AJAX? – www.aws.amazon.com/it/what-is/ajax
- Che cos'è un diagramma entità-relazione? – www.ibm.com/it-it/think/topics/entity-relationship-diagram
- Client-Server Model – www.geeksforgeeks.org/client-server-model
- Come funziona l'architettura M.T.V. di Django? – www.programmareinpython.it/blog/che-cosa-rende-django-speciale-miglior-web-framewo
- Cos'è un framework nella programmazione web – www.aulab.it/notizia/324/cose-un-framework-nella-programmazione-web
- Creating forms from models – www.docs.djangoproject.com/en/5.1/topics/forms/modelforms
- Django: The web framework for perfectionists with deadlines – www.djangoproject.com
- Documentazione di Django – www.docs.djangoproject.com/it/5.1
- ExchangeRate-API - Free and Pro Currency Converter API – www.exchangerate-api.com
- Finnhub Stock APIs - Real-time stock prices – www.finnhub.io
- Framework per applicazioni web – www.wikipedia.org/wiki/Framework_per_applicazioni_web
- Guida Django 1.0 – www.html.it/guide/guida-django-10
- Il pattern MVC – www.html.it/pag/18299/il-pattern-mvc

- Introduzione al Web Framework Django e Primo Progetto – `www.programmareinpython.it/blog/introduzione-al-web-framework-django`
- JavaScript Tutorial – `www.w3schools.com/js`
- Laravel - The PHP Framework For Web Artisans – `www.laravel.com`
- Mainframe: cos'è, come funziona e quando serve – `www.ibm.com/it-it/topics/mainframe`
- Most Popular Backend Frameworks – 2012/2023 – `www.statisticsanddata.org/data/most-popular-backend-frameworks-2012-2023`
- PocketGuard: Budgeting App and Finance Planner – `www.pocketguard.com`
- PocketGuard Review – `www.uk.pcmag.com/personal-finance-apps/153631/pocketguard`
- PocketSmith – The Best Budgeting and Personal Finance Software – `www.pocketsmith.com`
- PocketSmith Review: Forecasting Your Future Finances – `www.moneywise.com/investing/reviews/pocketsmith`
- progettazione di un data-base - Il Prof d'Informatica – `www.ilprofdinformatica.altervista.org/classe5/db_progetto.htm`
- Ruby on Rails — A web-app framework – `www.rubyonrails.org`
- Sistema client/server – `www.wikipedia.org/wiki/Sistema_client/server`
- TheirStack | Technology Lookup and Hiring Signals – `www.theirstack.com/it`
- Welcome to Flask — Flask Documentation (3.1.x) – `www.flask.palletsprojects.com/en/stable`
- Working Structure of Django MTV Architecture | by Rinu Gour – `www.towardsdatascience.com/working-structure-of-django-mtv-architecture-a741c8c64082`
- YNAB – `www.ynab.com`
- YNAB Review – `www.uk.pcmag.com/personal-finance/2470/ynab`

Ringraziamenti

Il mio primo ringraziamento va ai miei genitori, che mi hanno dato la possibilità di intraprendere e portare a termine questo percorso universitario, supportandomi in ogni momento e trovando sempre il modo di sostenermi, anche in situazioni che non comprendevano appieno.

Un altro ringraziamento va ai miei amici: a quelli di sempre, con cui, nonostante la distanza, siamo riusciti a mantenere intatto il nostro legame, e a quelli incontrati lungo questo percorso di studi, con cui ho condiviso momenti indimenticabili, sia dentro che fuori dall'università.

Vorrei infine ringraziare il Prof. Ursino, che mi ha guidato nella stesura della tesi con costante attenzione, disponibilità e rispondendo sempre prontamente ai miei dubbi e alle mie richieste.