Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Tracciamento e analisi del comportamento degli utenti su piattaforme web di incentivazione

Tesi di laurea triennale

Relatore

Prof. Lamberto Ballan

Laureando

Filippo Tonini 2008080

Anno Accademico 2022-2023

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage svolto presso l'azienda Reward Srl. Lo stage è stato svolto verso la conclusione del percorso di studi della laurea triennale in Informatica, ha avuto una durata di circa trecento ore.
Lo scopo del progetto di stage è raccogliere ed analizzare i dati sul comportamento degli utenti nelle piattaforme già esistenti dell'azienda. Il progetto prevede lo sviluppo di un package Laravel da installare nelle piattaforme di Reward e di un server che riceva i dati raccolti dal package, li elabori e li renda consultabili tramite una dashboard. Sia le metriche ritenute più significative sia le tecnologie da usare (ad eccezione di Laravel) sono scelte liberamente dallo studente, ma discusse e valutate con il tutor aziendale.
L'elaborato riporterà il contesto aziendale dove è stato svolto lo stage, le attività svolte, ed una valutazione sul lavoro effettuato.

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Lamberto Ballan, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Vorrei anche ringraziare tutti gli altri professori che mi hanno seguito durante il percorso universitario, in quanto mi hanno dato le consocenze necessarie per scrivere questo documento.

Infine vorrei ringraziare il tutor aziendale Mattia De Nadai e tutti i colleghi si Reward che hanno trovato il tempo di aiutarmi durante lo stage.

Padova, Dicembre 2023 Filippo Tonini 2008080

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'azienda

L'azienda Reward s.r.l. realizza prodotti, servizi e piattaforme per il marketing one-to-one, fidelizzazione dei clienti, incentive e incremento delle performance. L'azienda mira a far raggiungere ad altre aziende i loro obiettivi di vendita tramite strategie di incentivazione.
Alcune tipi di piattaforme realizzate dall'azienda sono Incentiway, Game&Gain e Shop-Square che sono rispettivamente piattaforme di: incentivazione, gamification, e credito prepagato digitale. Reward è in grado di creare dei siti personalizzati, su richiesta di un cliente, partendo da un template di un certo tipo di piattaforma. L'azienda segue poi i clienti nell'aggiornamento e manutenzione dei siti creati.
Reward fa parte del gruppo epipoli, che ha importanza internazionale nell'industria Fintech, in particolare nel mercato dei servizi prepagati e dei sistemi di engagement.

Figura 1.1: Reward s.r.l.

1.2 L'offerta di stage

L'azienda ha bisogno di misurare l'esperienza degli utenti per individuare delle possibili migliorie da effettuare nelle piattaforme. Da tale necessità è nata la proposta di stage. Il progetto prevede di sviluppare una piattaforma di tracciamento del comportamento degli utenti sulle piattaforme già esistenti di Reward. I dati sul comportamento degli utenti saranno raccolti tramite un pacchetto plug and play, che potrà essere installato nelle piattaforme in modo non invasivo. I dati saranno poi passati a un server che eseguirà l'elaborazione e l'esposizione dei dati sul web sotto forma di dashboard.
Lo studente dovrà apprendere i metodi e le tecnologie utilizzate allo state dell'arte per il tracciamento degli utenti, in modo da poter effettuare le scelte tecnologiche e architetturali autonomamente, ed infine realizzare la piattaforma. Il tutor aziendale

seguirà da vicino tale percorso anche se non parteciperà direttamente nello sviluppo.

1.3 Struttura del documento

Il secondo capitolo descriverà attività e obiettivi dello stage.

Il terzo capitolo descriverà le principali tecnologie utlizzate.

Il quarto capitolo descriverà l'applicativo Matomo, parte cruciale del progetto.

Il quinto capitolo descriverà lo sviluppo del modulo di acquisizione dei dati, ovvero il pacchetto Laravel.

Il sesto capitolo descriverà lo sviluppo del modulo di raccolta dei dati, in particolare l'applicazione Django.

Nel settimo capitolo verranno discusse le conclusioni del progetto.

Capitolo 2

Descrizione dello stage

2.1 Introduzione al progetto

L'obiettivo del progetto è sviluppare una piattaforma per analizzare il comportamento degli utenti sulle altre piattaforme di Reward. Tramite questa l'azienda deve poter individuare i problemi degli utenti nella navigazione dei loro siti web, usando metriche come tempi di load delle pagine o heatmap.
Per poter fare ciò il prodotto sviluppato sarà composto da due parti:

- **Un package** da poter installare nelle piattaforme dell'azienda, dato che queste usano il framework Laravel il pacchetto sarà sviluppato per tale framework. Il pacchetto raccoglierà informazioni sugli utenti e le invierà al server. Il pacchetto dovrà essere di installazione semplice e veloce, idealmente plug and play.
- **Un server** che riceverà i dati emessi dal pacchetto, gli elaborerà e gli esporrà sul web. Per la realizzazione di quest'ultimo la scelta delle tecnologie da usare è libera. La possibilità di reperire determinati tipi di dati è fondamentale per il raggiungimento dei risultati del progetto, l'elaborazione invece è apprezzata ma non necessaria.

2.2 Obiettivi formativi

Gli obiettivi formativi dell'attività di stage sono i seguenti:

- Apprendere i metodi e le tecnologie usate nel contesto aziendale.
- Apprendere principi, metodi e tecnologie usati allo stato dell'arte nell'ambito di web user analytics e recommender systems.

2.3 Requisiti

Nei primi giorni di stage sono stati eseguiti degli incontri con il tutor aziendale per definire in modo dettagliato i requisiti funzionali. Da notare che la piattaforma non sarà pubblica quindi l'attore "utente" sarà un dipendente dell'azienda Reward.
A seguito sono riportati i requisiti che saranno identificati nel seguente modo:

`R[Priorità]-[Identificativo]`

Dove:

- **R** indica che si tratta di un requisito;
- **Priorità** può assumere i seguenti valori:
  - **O**: requisito obbligatorio;
  - **D**: requisito desiderabile;
  - **F**: requisito facoltativo.
- **Identificativo**: numero progressivo che identifica il requisito in forma gerarchica, strutturato come segue:

`[codicePadre].[codiceFiglio]`

Codice	Descrizione
RO-1	L'utente deve poter visualizzare una mappa di calore dei movimenti del cursore per una pagina a scelta
RO-2	L'utente deve poter abilitare la raccolta dati riguardanti mappe di calore dei movimenti del cursore in una qualsiasi pagina di una piattaforma
RO-3	L'utente deve poter visualizzare una mappa di calore dei click per una pagina a scelta
RO-4	L'utente deve poter abilitare la raccolta dati riguardanti mappe di calore dei click in una qualsiasi pagina di una piattaforma
RO-5	L'utente deve poter visualizzare una mappa di calore dello scorrimento per una pagina a scelta
RO-6	L'utente deve poter abilitare la raccolta dati riguardanti mappe di calore dello scorrimento in una qualsiasi pagina di una piattaforma
RO-7	L'utente deve poter visualizzare i dati sui tempi di caricamento medi delle pagine delle varie piattaforme
RO-9	L'utente deve poter visualizzare il browser e sistema operativo dei visitatori di una piattaforma
RO-10	L'utente deve poter visualizzare informazioni sulla finestra dei visitatori di una piattaforma
RO-11	L'utente deve poter identificare in modo anonimo il visitatore di una piattaforma
RO-12	L'utente deve poter visualizzare le statistiche sulle pagine visitate dai visitatori di una piattaforma

RO-12	L'utente deve poter gestire i siti e le pagine su cui vengono tracciati gli utenti
RD-13	L'utente deve poter installare installare il pacchetto di raccolta dati tramite PHP composer
RD-14	L'utente deve poter visualizzare un riepilogo dell'e-commerceper una piattaforma (se applicabile)
RD-15	L'utente deve poter richiedere dei prodotti da raccomandare per un certo visitatore (se applicabile)

Tabella 2.1: Requisiti

2.4 Pianificazione

La durata complessiva dello stage è stata di 9 settimane di lavoro di cui due in part-time e il resto full-time per un totale di circa trecento ore.

Con l'azienda è stata concordata la seguente pianificazione:

Durata in ore	Settimana	Descrizione
24	1	◇ Incontro con persone coinvolte nel progetto per discutere i requisiti e le richieste relativamente al sistema da sviluppare; ◇ Verifica credenziali e strumenti di lavoro assegnati; ◇ Presa visione dell'infrastruttura esistente; ◇ Formazione sulle tecnologie aziendali utilizzate; ◇ Configurazione ambiente di sviluppo;

24	2	◇ Definizione architettura modulo acquisizione dati; ◇ Definizione architettura piattaforma monitoraggio e reportistica; ◇ Definizione tecnologie e librerie da utilizzare nel progetto;
76	3, 4	◇ Implementazione del modulo di acquisizione dati;
38	5	◇ Configurazione di un test di acquisizione dati su piattaforma esistente;
76	6, 7	◇ Implementazione piattaforma di monitoraggio e reportistica;
76	8, 9	◇ Collaudo piattaforma di monitoraggio e reportistica;
Totale ore:		**314**

Tabella 2.2: Pianificazione delle attività

Nella durata dello stage la pianificazione ha seguito alcune modifiche a seguito di una maggiore comprensione del problema, le modifiche saranno riportate nelle conclusioni. Ci sono stati confronti giornalieri e riunioni settimanali con il tutor aziendale per monitorare il progresso. In aggiunta è stato fatto un incontro a inizio e fine stage per definire gli obbiettivi e successivamente verificarne il raggiungimento.

Capitolo 3

Tecnologie e strumenti

In questo capitolo vengono presentate le tecnologie utilizzate durante lo stage, si è scelto di utilizzare solo tecnologie open-source.

3.1 Applicativi

3.1.1 Matomo

Matomo è un applicativo che permette di raccogliere informazioni sui siti web indicati tramite degli snippet di codice javascript. Le feature più interessanti per questo progetto sono:

- Raccoglimento dei dati sui visitatori come browser, dispositivo, azioni nel sito.
- Raccoglimento di dati e-commerce come acquisti, aggiornamenti del carrello, visualizzazione di prodotti.
- Tempi di caricamento e performance.

Sono disponibili anche plugin che aggiungono funzionalità come heatmap o A/B testing, però queste non sono open-source quindi non verranno utilizzate.

Figura 3.1: Matomo logo

3.1.2 Nginx

nginx è un software open source che permette tra le varie funzionalità di esporre la propria piattaforma sul web. L'applicazione verrà utilizzata nel server che si occuperà della raccolta, visualizzazione ed elaborazione dei dati.

Figura 3.2: Nginx logo

3.2 Framework

3.2.1 Laravel

Laravel è un framework PHP usato per creare applicazioni full-stack, ed offre soluzioni già pronte per problemi comuni come autenticazione o testing. Per questo progetto le feature da notare sono:

- *Eloquent ORM* che permette a laravel di riportare nel database oggetti e classi usati nel codice, in tal modo il programmatore non dovrà mai interagire direttamente col databse.
- elegante architettura *MVC* che permette al programmatore di gestire rotte, controller e modelli in modo semplice, veloce e intuitivo.
- possibilità di sviluppare e integrare pacchetti in modo immediato.

Figura 3.3: Laravel logo

3.2.2 Django

Django è un framework python full-stack, veloce e sicuro. Alcune feature da notare sono:

- similmente a Laravel ha il suo sistema di *ORM*.
- utilizza una variante di MVC chiamata *MTV*.
- *Template* HTML che facilitano lo sviluppo del front-end.

◇ Scalabile e veloce, in grado di gestire grandi quantità di dati.

Figura 3.4: Django logo

Vold dashboard

Volt è un toolkit per Django basato su bootstrap 5 che introduce una serie di template html e componenti da poter usare nel front-end.

3.3 Librerie

3.3.1 heatmap.js

heatmap.js è una leggera libreria javascript che permette di visualizzare una serie di punti sotto forma di heatmap, è veloce, facile da usare, in grado di gestire oltre 40.000 punti. La libreria disegna la mappa di calore sull'elemento del dom indicato, tramite le canvas html.

Figura 3.5: heatmap.js logo

3.3.2 Django REST framework

Django REST framework è un toolkit che permette di esporre API sul web. Le API si possono implementare facilmente col sistema di url e views di django.

Figura 3.6: Django rest framework logo

3.3.3 Bootstrap

Bootstrap è un toolkit per rendere più immediato lo styling del frontend. La feature principale sono le classi css già pronte da utilizzare nel proprio html.

Figura 3.7: bootstrap logo

3.3.4 scikit-surprise

scikit-surprise è una libreria python che mette a disposizione diversi algoritmi e funzionalità per la creazione di recommender systems.

Figura 3.8: scikit-surprise logo

Capitolo 4

Matomo

In questo capitolo parleremo del software Matomo e perché è stato scelto. L'applicativo è stato usato in concomitanza con quanto sviluppato dallo studente ed è stato fondamentale per il raggiungimento degli obbiettivi del progetto.

4.1 Soluzioni custom contro soluzioni all-in-one

Il tracciamento del comportamento degli utenti sul web è un problema comune tra aziende con piattaforme web, per questo esistono già numerosi prodotti sviluppati da terzi che offrono tale servizio. Detto ciò una soluzione già pronta potrebbe non soddisfare le esigenze di ogni specifico caso e quindi potrebbe essere necessario sviluppare del software ad hoc. A seguito verranno discussi pro e contro delle due soluzioni.
A favore di soluzioni all-in-one:

- ◇ Minori costi e tempi di sviluppo: generalmente queste piattaforme sono integrate nel proprio sito semplicemente aggiungendo uno snippet di codice, rendendo il raccoglimento dati quasi immediato.
- ◇ Grande vastità di funzionalità: sono disponibili numerose metriche, dalle informazioni sui visitatori a statistiche e-commerce.
- ◇ Allo stato dell'arte: i prodotti sono stati ampiamente collaudati e sono stati realizzati da sviluppatori specializzati nell'ambito del tracciamento degli utenti.

A sfavore di tali soluzioni:

- ◇ Fin troppo elaborati: per le finalità specifiche del progetto molte funzionalità potrebbero essere non necessarie e potrebbero causare difficoltà nell'uso o rallentamenti.
- ◇ Scatola chiusa: non è detto che sia garantito la proprietà e l'accesso ai dati grezzi,
- ◇ Solo in parte gratuiti: alcune funzionalità possono essere nascoste dietro piani a pagamento, oppure potrebbe essere necessario un pagamento proporzionale al traffico dati.

Prima di prendere la decisione su quale approccio adottare sono state ricercate diverse soluzioni all-in-one, col fine di trovare una soluzione che non presenti gli svantaggi descritti sopra:

Google Analytics

Google analytics è la prima scelta per molte aziende ma per le esigenze specifiche di questo progetto presenta i seguenti problemi:

- ◇ Raw data: l'accesso a dati grezzi via query SQL o API è solo disponibile nel piano a pagamento.
- ◇ Data sampling: per ridurre i costi solo un sottoinsieme dei dati raccolti viene analizzato da Google.
- ◇ Features non necessarie: ci sono un grande numero di feature non necessarie per il progetto che renderebbero l'utilizzo della piattaforma meno immediato.
- ◇ Heatmaps: la piattaforma non da la possibilità di visualizzare heatmap delle proprie pagine.

Google Analytics offre però il vantaggio di poter essere integrabile con molti strumenti, alcuni esempi sono:Google Tag Manager, BigQuery e Google Ads. Le funzionalità aggiuntive però spesso comportano un aumento dei costi.

Hotjar

Hotjar è un altra piattaforma molto conosciuta in particolare per le Heatmaps, però ha i seguenti svantaggi:

- ◇ Raw data: l'accesso a dati grezzi via query SQL o API non è disponibile.
- ◇ Data sampling: per ridurre i costi solo un sottoinsieme dei dati raccolti viene analizzato da Hotjar.
- ◇ Statistiche sulle performance: Hotjar non colleziona dati sulle performance delle pagine.

Matomo

Matomo è una alternativa a Google analytics, è open source nella versione on-premise ovvero installando l'applicativo su un server proprio. Lo svantaggio è:

- ◇ Heatmaps: le heatmap sono un plugin a pagamento.

Alcune soluzioni mettono a disposizione anche un **Tag Manager**. Questo serve ad avere un posto unico, gestito solitamente tramite interfaccia grafica, dove vengono gestite tutte le metriche raccolte nel sito. Praticamente significa che nella propria piattaforma verrà solo aggiunto uno snippet di codice del Tag Manager, tramite questo snippet la piattaforma raccoglierà i dati corrispondenti ai tag impostati. Ogni tag è una misura raccolta e inviata alla piattaforma di analitica. In concomitanza con i tag spesso viene offerta la possibilità di creare degli eventi personalizzati da collegare ai tag. Sia Google analytics che Matomo offrono un sistema di tag e trigger.

Dato che nessuna delle soluzioni all-in-one soddisfa pienamente i requisiti del progetto è stato scelto di utilizzare una soluzione ibrida, sfruttando Matomo, in quanto è la

piattaforma che si avvicina di più alle esigenze del progetto. Per alcune feature verrà usato Matomo le altre saranno sviluppate dallo studente.
Questa soluzione non dovrà in alcun modo intaccare l'esperienza utente, la piattaforma Matomo dovrà quindi essere nascosta e gestita automaticamente, mentre l'utente adopererà solo la piattaforma creata ad hoc per il progetto.

4.2 Features

A seguito verranno discusse le feature e caratteristiche di Matomo che sono state fondamentali per il riuscimento del progetto.

Self hosting

A differenza di molte altre piattaforme di web analytics Matomo mette a disposizione il codice per un installazione su un proprio server. Questa versione che Matomo chiama "On-premise" è gratuita ad eccezione di alcuni plugin a pagamento come heatmaps o A/B testing. Per questo progetto usare Matomo on-premise è molto vantaggioso in quanto permette di avere sia l'applicazione sviluppata dallo studente che Matomo sulla stessa macchina, facilitando la comunicazione tra i due.

Accesso ai dati grezzi

Matomo dà la possibilità di accedere a tutti i dati grezzi tramite API, nel caso di self hosting l'accesso è ancora più immediato potendo direttamente accedere al database SQL. Questa feature sarà fondamentale per poter estrarre ed eseguire una ulteriore elaborazione dei dati.

Reports

Matomo mette a disposizione numerosi grafici e metriche, le più interessanti per il progetto sono:

- Dispositivi, browser e dimensioni;
- Durata delle visite;
- Azioni dei visitatori;
- Statistiche sulle performance;
- Visitatori ricorrenti e unici;

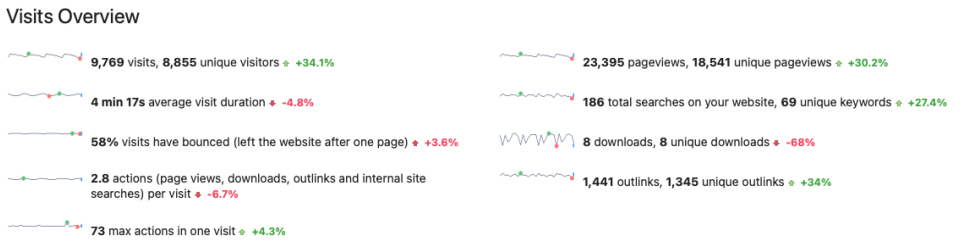

Figura 4.1: visitors demo dashboard

Con setup aggiuntivo si può accedere ai report e-commerce, che mostrano statistiche come:

- ◇ Guadagni totali;
- ◇ Carrelli abbandonati o completati;
- ◇ Prodotti più acquistati;

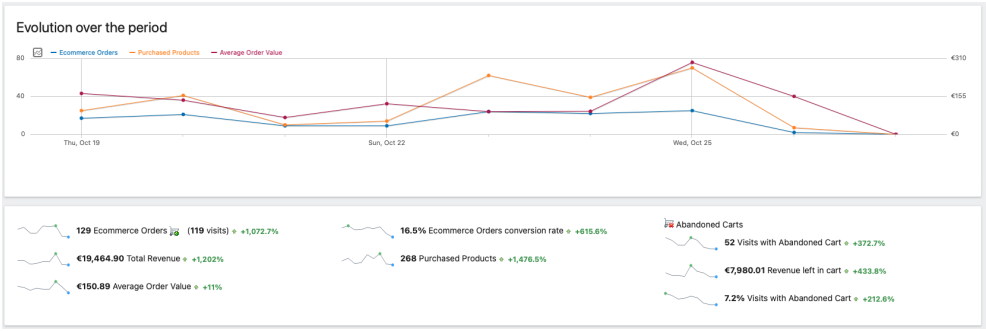

Figura 4.2: e-commerce dashboard di ShopSquare

Esportazione reports

Matomo permette di esportare i report dall'interfaccia grafica in vari formati come CSV, XML o TSV. Ancora più interessante per il progetto è la possibilità di inserire un widget Matomo contenente un report nella propria piattaforma. Viene messo a disposizione del codice HTML già pronto per ogni report presente nella piattaforma di Matomo. Per il nostro progetto questa feature renderà molto più semplice riportare le dashboard di Matomo nella piattaforma che verrà creata, in questo modo l'utente non dovrà accedere a Matomo.

4.3 Utilizzo

A seguire verrà discusso più concretamente come usare le feature descritte precedentemente, e come in generale implementarle.

Tracciamento visitatore

Per tracciare le visite è necessario inviare al server Matomo dati a ogni accesso ad una determinata pagina, questo viene eseguito tramite del codice javascript che deve essere inserito nel tag "head" di ogni pagina.
Matomo identifica i visitatori unici tramite i cookies, se essi sono disabilitati cerca di identificare gli utenti tramite una combinazione di dati come indirizzo IP, browser, sistema operativo. Questo modo di identificare i visitatori non è perfetto in quanto, lo stesso visitatore che accede da due dispositivi diversi, sarà trattato come due visitatori diversi.
Per risolvere questo problema Matomo mette a disposizione una funzionalità chiamata User ID. Questa permette di identificare i visitatori in modo più accurato nei siti con un sistema di autenticazione. Matomo utilizza l'identificativo interno dell'utente del sito, per esempio l'attributo id del database, per identificare gli utenti indipendentemente dal dispositivo. Naturalmente per permettere ciò bisogna fornire a Matomo l'identificativo degli utenti nel momento della loro vista. Nel capitolo successivo (5) verrà descritta l'integrazione di tale funzionalità nel progetto di stage, e il codice di tracciamento dei visitatori nel suo complesso.

Tracciamento e-commerce

Per il tracciamento completo delle azioni e-commerce è necessario mandare a Matomo i seguenti dati:

- articoli presenti nel carrello dopo ogni aggiornamento di questo,
- visualizzazione di un prodotto,
- visualizzazione di una categoria,
- conferma dell'acquisto con articoli acquistati.

Per ogni prodotto è necessario fornire almeno un identificativo e se possibile anche dati aggiuntivi quali: nome, categoria, marca, prezzo e quantità. Nel caso della conferma dell'acquisto e l'aggiornamento del carrello è necessario fornire anche il prezzo totale del carrello.
Ciascuna di queste azioni deve essere monitorata e inviata a Matomo, questo viene fatto tramite javascript. Come è stato deciso di implementarli verrà discusso nel capitolo successivo (5).

Gestione siti e permessi

La piattaforma Matomo permette di avere diversi utenti che hanno accesso alle dashboard dei siti. Per ogni sito è possibile modificare i permessi di un utente tra:

- None: nessun permesso,
- View: solo lettura,
- Write: lettura e scrittura,
- Superuser: lettura, scrittura e funzionalità di gestione utenti e siti.

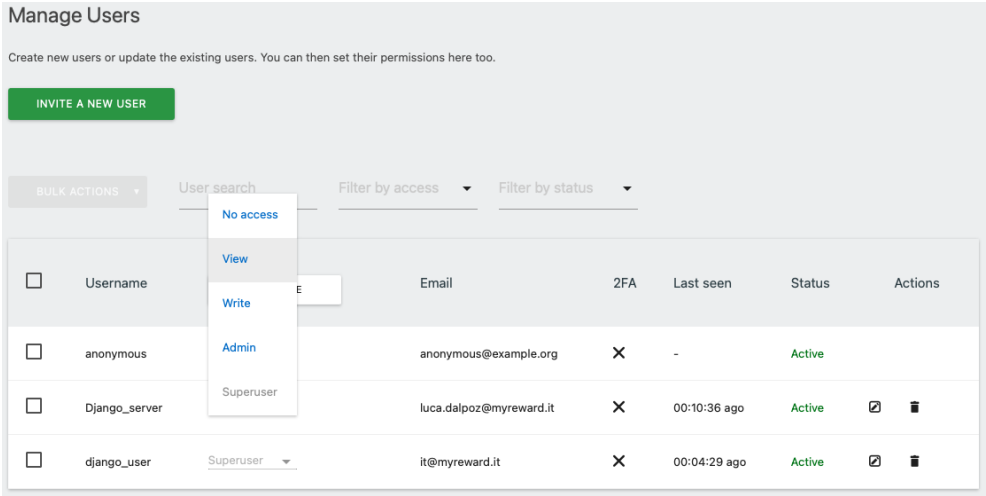

Figura 4.3: pagina di impostazione permessi di Matomo

Per ogni istanza Matomo esiste sempre un superuser, che sarà colui che l'ha creata. Ogni sito ha anche un utente chiamato "Anonymous" ovvero l'utente generico, che non ha eseguito l'accesso.
Un utente può creare una chiave chiamata "token auth" da usare nelle chiamate API, il token avrà gli stessi permessi dell'utente che l'ha creato, alcune chiamate possono essere fatte solo con token aventi un certo livello di permessi.
Per questioni di sicurezza non si possono impostare dei permessi di default per tutti i siti per un utente. Ciò significa che alla creazione di un nuovo sito bisogna impostare i permessi dell'utente in questione, dato che l'utente non interagirà con l'interfaccia Matomo questo sarà fatto automaticamente tramite chiamate API nell'applicazione Django (Più informazioni si trovano nel capitolo 6).

API e query SQL

Matomo permette di fare chiamate API alla propria instanza Matomo, sia nel caso di cloud hosting che self hosting. Tali chiamate permettono di ricevere i dati grezzi raccolti da Matomo in formati come CSV, JSON, XML. Inoltre ci sono funzioni di management dei siti come l'aggiunta di un nuovo sito o l'impostazione dei permessi.
Alcuni parametri che saranno spesso utilizzati sono:

- ◇ idSite: identifica il sito, è assegnato automaticamente da Matomo.
- ◇ period & date: identificano il periodo e la data di cui si vogliono avere i dati.
- ◇ format: formato in cui sono restituiti i dati.
- ◇ token_auth: token di autenticazione di chi fa la richiesta, il token ha permessi equivalenti ai permessi dell'utente che l'ha creato; alcune chiamate necessitano di un preciso livello di permessi.

Widgetize Dashboards

Come discusso precedentemente Matomo da la possibilità di integrare le dashboards nel proprio sito tramite uno snippet html. Questo avviene tramite una chiamata all'istanza Matomo avente diversi parametri, questi hanno la funzione di specificare e personalizare il report visualizzato, ad esempio si può inserire il periodo del report o il tipo di grafico.
Di default l'utente anonimo non ha permessi di visualizzazione. Quindi il widget non visualizzerà nessun report, a meno che non vengano impostati i permessi corretti. C'è anche la possibilità di aggiungere un token di autenticazione come parametro nella chiamata a Matomo, se l'utente ha permessi di visualizzazione il report sarà visualizzato correttamente. Nel capitolo riguardante il Modulo di raccolta dati (6), verrà discusso più approfonditamente come è stato deciso di risolvere questo problema.

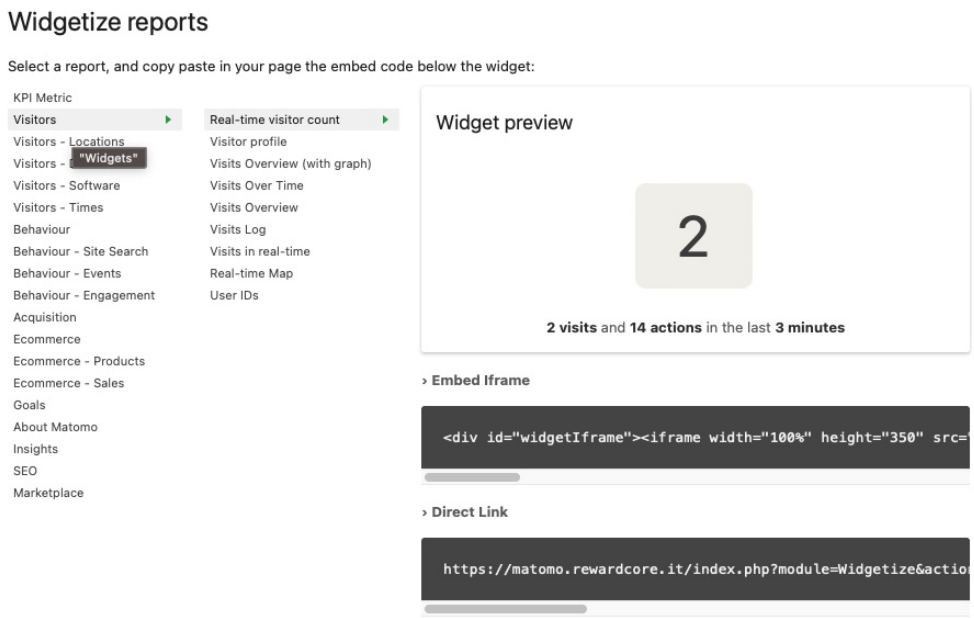

Figura 4.4: pagina di esportazione dei widget

Capitolo 5

Modulo acquisizione dati

Nel capitolo che segue si parlerà del package Laravel sviluppato durante lo stage. Verrà descritto lo scopo del modulo e le sue funzionalità principali insieme a problemi e limitazioni. Inoltre verrà descritta l'architettura e decisioni progettuali prese per il pacchetto. Infine verrà fornita una descrizione approfondita e dettagliata delle sue componenti, e il processo di implementazione.

5.1 Scopo del modulo

Il modulo di acquisizione dati dovrà individuare le informazioni sul comportamento degli utenti, raccoglierle e inviarle al server. Il modulo consisterà in un pacchetto Laravel da installare facilmente tramite composer, in tal modo potrà essere velocemente integrato con le piattaforme dell'azienda. A questo fine il pacchetto deve essere più disaccoppiato possibile dal resto della piattaforma. Non è possibile che i due siano completamente indipendenti, dunque l'obbiettivo è quello di massimizzare tale indipendenza; questo perché alcuni dati sono reperibili solo interrogando direttamente la piattaforma. Il pacchetto quindi sarà solo in parte plug and play, e necessiterà di configurazione aggiuntiva per abilitare alcune funzionalità.
Come accennato nel capitolo precedente (4) non tutti i requisiti possono essere soddisfatti da Matomo, infatti mancano le heatmaps che sono un plugin a pagamento. Per questo motivo le funzionalità riguardanti le heatmaps saranno sviluppate completamente dallo studente. Questo significa che il modulo di raccolta dati, ovvero il server, sarà composto da due parti, l'applicazione Django sviluppata ad hoc che tra le altre cose si occuperà delle heatmaps e l'applicazione Matomo. Il Modulo di acquisizione dei dati dovrà quindi gestire la comunicazione con entrambe le parti del server.
A seguito verranno descritte le tre principali funzionalità del modulo di acquisizione, ognuna di queste richiederà un configurazione diversa per essere attivata:

Tracciamento visitatore

Il modulo si occuperà di abilitare il tracciamento di base di Matomo, dunque l'acquisizione e invio di dati sui visitatori e sulle pagine. Per l'acquisizione di questo tipi di dati non è necessario avere accesso a dati specifici del sito (a parte user ID che è opzionale). L'integrazione di questa funzionalità in un nuovo sito è assolutamente non invasiva e non richiede alcun tipo di configurazione.
I dati che il pacchetto dovrà recuperare e fornire a Matomo saranno:

- **user ID** (se l'utente ha eseguito l'accesso) ovvero l'identificativo dell'utente nel database della piattaforma Reward questo permette un tracciamento più preciso dei visitatori unici come spiegato nel capitolo 4; Questo dato andrà recuperato dal database del server Laravel o dal corrispettivo modello.

- **site ID** ovvero l'identificativo Matomo da cui provengono i dati, nel nostro caso il pacchetto dovrà essere installato su diversi siti quindi questa informazione dovrà essere compilata dinamicamente; per recuperare l'id dinamicamente si è usato una rotta API esposta da Matomo, che dato l'URL del sito restituisce il site ID.

Nel paragrafo 5.3 verrà descritto precisamente come ciò sarà implementato.

Tracciamento ecommerce

Il modulo dovrà permettere, con ulteriore configurazione, di tracciare le azioni e-commercee inviarle a Matomo. Per questo tipo di dati è necessario essere a conoscenza dei prodotti, del carrello e di azioni effettuate dall'utente in ambito ecommerce. Dunque per abilitare questa funzionalità bisorrà intervenire sul sito in questione; il pacchetto e il sito, quindi, non saranno completamente indipendenti.
I tipi di azioni e-commerce Matomo sono categorizzate in: Aggiornamento del carrello, Completamento ordine, Visualizzazione del prodotto, Visualizzazione di una categoria. Bisogna specificare che Matomo non memorizza i prodotti nel carrello di un certo utente, quindi ad ogni aggiornamento del carrello bisogna inviare nuovamente tutti i prodotti del carrello; stesso per il completamento dell'ordine, all'acquisito bisogna mandare insieme all'informazione di completamento anche tutti i prodotti acquistati. Una volta inviati tutti i prodotti del carrello bisogna anche comunicare il totale del carrello.
I dati che il pacchetto dovrà fornire a Matomo saranno quindi:

- **Carrello**: il pacchetto ad ogni aggiornamento del carrello o completamento dell'ordine, deve reperire dal server Laravel l'intero carrello dell'utente ed inviarlo a Matomo, insieme al prezzo totale.

- **Prodotti**: qualvolta ci sia bisogno di inviare informazioni a Matomo riguardanti un prodotto, quindi in caso di aggiornamento o acquisito del carrello, e visualizzazione di un prodotto, è necessario fornire almeno un identificativo e se possibile anche dati aggiuntivi quali: nome, categoria, marca, prezzo e quantità.

Heatmaps

Il modulo dovrà acquisire i dati sui movimenti del cursore, sulla posizione dei click e sullo scorrimento della pagina ed inviarli al server. Queste tre funzionalità sono separate e possono essere abilitate indipendentemente l'una dall'altra.

5.2 Progettazione

Per la progettazione del modulo i vincoli solo:

- ◇ il codice di tracciamento e delle heatmaps che dovrà essere in javascript, in quanto questo è l'unico modo di acquisire alcune informazioni del comportamento dell'utente tramite browser, inoltre è anche il linguaggio in cui Matomo fornisce il suo codice di tracking.
- ◇ il pacchetto dovrà seguire l'architettura per i pacchetti proposta da Laravel e le convenzioni di Reward in materia.

I package seguono l'architettura l'MVC di tutti i progetti Laravel.

Model

Nel nostro caso non ci sarà bisogno di creare modelli, in quanto i necessari sono già presenti in package creati dall'azienda.
In particolare, i dati da estrarre dalla piattaforma sono: i dati sull'utente, come il suo identificativo o il suo carrello; i dati sui prodotti, come la categoria o il prezzo. Questi dati sono necessari per permettere a Matomo un tracciamento completo. Reward ha sviluppato i package Laravel: "siteusers" e "products" che gestiscono rispettivamente gli utenti e i prodotti. Il primo pacchetto offre una facade che sarà sfruttata per la comunicazione col pacchetto, per il secondo invece la comunicazione avverrà direttamente con uno o più modelli.

View

Per il progetto non sarà necessario creare nuove views, però in quelle già esistenti verrà inserito del codice javascript.
Il codice avrà le funzionalità di acquisire dei dati del comportamento dell'utente, richiedere i dati aggiuntivi alla piattaforma Laravel, e inviare quanto acquisito al modulo di raccolta dati.
Per permettere questa comunicazione tra server Laravel e browser il pacchetto dovrà esporre delle rotte che saranno chiamate dal codice javascript tramite ajax.
Alcuni eventi, in particolare le azioni e-commerce eseguite dall'utente dovranno causare l'invio di dati al modulo di raccolta. Per questo saranno gli eventi javascript come, click di un bottone o invio di un form, che avvieranno il processo di acquisizione ed invio dei dati.
Questo significa che nelle views già esistenti, che in laravel corrispondono a file blade, dovranno essere inserite delle classi per permettere al codice javascript di rimanere in ascolto su certi eventi. Una volta installato il pacchetto in una nuova piattaforma, sarà quindi necessario indicare i tag HTML dove rimanere in ascolto per gli eventi e-commerce. La parte del pacchetto riguardante l'e-commercequindi non sarà plug and play e richiederà di intervenire sulla piattaforma Reward in questione.

Controller

Verrà creato un controller che gestirà la comunicazione tra il codice di tracciamento javascript e i modelli menzionati sopra, questo gestirà quindi le rotte e reperirà i dati dai modelli.

Pubblicazione del pacchetto nel progetto Laravel

Le rotte e il codice javascript del pacchetto devono essere riportati all'interno del progetto base prima di essere eseguiti.
Per quanto riguarda le rotte sono pubblicate tramite i "Service Providers", questi sono dei file all'interno del pacchetto che contengono informazioni sulla pubblicazione nel progetto principale. In questo caso il service provider si occuperà di pubblicare le rotte e le configurazioni nel progetto.
Nel caso del codice javascript il processo è simile i file js vengono pubblicati e poi condensati in un unico file fornito al browser, questo è possibile tramite un package open source chiamato Laravel mix.

5.3 Implementazione delle feature

A seguito verrà discussa in dettaglio l'implementazione del pacchetto, che è costituito dalle seguenti componenti:

- ◇ *tracking.js* che gestirà l'acquisizione ed invio dei dati gestiti da Matomo.
- ◇ *create-heatmap.js* che gestirà l'acquisizione dei dati sulle heatmap e il loro invio all'applicazione Django.
- ◇ *trackingcontroller.php* e *web.php*, rispettivamente controller e rotte per gestire le richieste sull'acquisizione dei dati specifici della piattaforma.
- ◇ *UsertrackingServiceProvider.php*

5.3.1 tracking.js

Il file javascript si occupa del tracciamento Matomo.
Al caricamento della pagina viene inizializzata la variabile globale *_paq*, di cui a seguito vedremo l'uso, e vengono chiamate le funzioni di inizializzazione di Matomo e di tracciamento della Visualizzazione di una categoria.

```
$(async function () {

    //_paq init
    var _paq = window._paq = window._paq || [];

    await initMatomo();  //calls matomo initializer for page & checks if the page is that of a category & if user is logged
    trackCategoryView(); //check if category is being viewed
```

Figura 5.1: javascript codice principale

inizializzazione di Matomo

```
async function initMatomo() {
    let user = await getUserId()
    if (user) {
        _paq.push(['setUserId', user]);
    }
    getSiteId().then((siteid) => {
        if (siteid) {
            _paq.push(['trackPageView']);
            _paq.push(['enableLinkTracking']);
            (function () {
                var u = '//matomo.rewardcore.it/';
                _paq.push(['setTrackerUrl', u + 'matomo.php']);
                _paq.push(['setSiteId', siteid.toString()]);
                var d = document, g = d.createElement('script'), s = d.getElementsByTagName('script')[0];
                g.async = true; g.src = u + 'matomo.js'; s.parentNode.insertBefore(g, s);
            })();
        }
    });
}
```

Figura 5.2: funzione javascript initMatomo()

La funzione inserisce il codice javascript di Matomo, ovvero lo script *matomo.js*, e configura il tracking per la pagina inserendo diverse funzioni nell'array *_paq*. L'esecuzione delle funzioni dell'array e l'invio dei dati viene gestito da Matomo.
La funzione è asincrona in quanto ha chiamate asincrone ad altre funzioni.
Per l'esecuzione delle funzioni Matomo la sintassi che deve essere usata è:

```
_paq.push([<funzione>, <parametri>])
```

La documentazione di tutte le funzioni e parametri di ciascuna è reperibile al seguente link: https://developer.matomo.org/api-reference/tracking-javascript
L'inizializzazione chiama la funzione *getSiteId()*:

```
async function getSiteId() {
    let id = null
    const url = 'https://matomo.rewardcore.it/?module=API&method=SitesManager.getSitesIdFromSiteUrl&url='
        + encodeURIComponent('https://' + window.location.host)
        + '&token_auth=********************&format=JSON';
    const options = {
        method: "GET"
    }
    res = await fetch(url, options)
    json_res = await res.json()
    id = json_res[0].idsite
    return id;

}
```

Figura 5.3: funzione javascript getSiteId()

La funzione richiede all'istanza Matomo l'id del sito stesso a partire dall'URL utilizzando una chiamata API, per questo motivo la funzione è stata resa asincrona. Al seguente link si può trovare una lista completa delle API disponibili: https://developer.matomo.org/api-reference/reporting-api

L'inizializzazione chiama anche la funzione *getUserId()*:

```
async function getUserId() {
    let user = null;
    await $.ajax({
        url: '/it/tracking/getUserId',
        method: 'POST',
        dataType: 'json',
        success: function (result) {
            user = result.user
        },
        error: function (error) {
        },
    });
    return user;
}
```

Figura 5.4: funzione javascript getUserId()

La funzione sfrutta un rotta del package */it/tracking/getUserId* che sarà discussa successivamente. Anche questa funzione è asincrona.
Da notare che lo user id sarà fornito a Matomo tramite la funzione *setUserId* che dovrà essere chiamata prima di ogni funzione di tracking come *trackPageView*, *trackEcommerceOrder* o *trackEcommerceCartUpdate*

tracciamento della visualizzazione della categoria

Una volta completata l'inizializzazione di Matomo viene chiamata la funzione per tracciare la visualizzazione di una categoria.

```
function trackCategoryView() {
    //track category view
    //check if there's a category on page then track. Add class to div with category name inside.
    if ($('.track_categoryview').length) {
        let category = $('.track_categoryview').text();
        _paq = window._paq = window._paq || [];
        // Push Category View Data to Matomo - Fill category dynamically
        _paq.push(['setEcommerceView',
            false, // Product name is not applicable for a category view.
            false, // Product SKU is not applicable for a category view.
            category, // (Optional) Product category, or array of up to 5 categories
        ]);
        if (user) { _paq.push(['setUserId', user]) }
        _paq.push(['trackPageView']);
    }
}
```

Figura 5.5: funzione javascript trackCategoryView()

Per permettere questo tipo di tracciamento è necessario inserire la classe apposita nell'elemento HTML contenente la categoria. Infatti la funzione controlla che la classe di tracciamento delle categorie sia presente nel file HTML, se essa è presente segnala la visualizzazione della categoria in questione a Matomo.

tracciamento della visualizzazione di un prodotto

La funzione serve a tracciare la visualizzazione di un prodotto, da notare che nel caso di Reward la visualizzazione del prodotto corrisponde con la scelta di un taglio nella pagina, per questo l'evento su cui rimanere in ascolto è il cambio del tag *select*. Anche in questo caso quindi andrà aggiunta la classe al tag *select* della pagina HTML dei prodotti.

```
//track product view
//add class "track_productview" on the select product version tag
$('.track_productview').on('change', function () {
    let id = this.value;
    if (id) {
        $.ajax({
            url: '/it/tracking/getProductData',
            type: 'POST',
            dataType: 'json',
            contentType: 'application/json',
            data: JSON.stringify({
                id: id,
            }),
            success: function (result) {
                _paq = window._paq = window._paq || [];
                _paq.push(['setEcommerceView',
                    result.product.sku, // (Required) productSKU
                    result.product.name, // (Optional) productName
                    result.product.category, // (Optional) categoryName
                    result.product.price, // (Recommended) price
                ]);
                if (user) { _paq.push(['setUserId', user]) }
                _paq.push(['trackPageView']);
            },
            error: function (error) {
            },
        });
    }
});
```

Figura 5.6: funzione javascript trackProductView()

Per tracciare la visualizzazione di un prodotto bisogna fornire a Matomo i dati del prodotto che saranno recuperati dalla piattaforma Laravel tramite la rotta */it/tracking/getProductData*. Questi vengono richiesti tramite una chiamata ajax al server e, in caso di successo, vengono inseriti nell'array di Matomo come parametri della funzione *setEcommerceView*.

tracciamento dell'aggiornamento del carrello

La funzione è simile a quella precedente, in questo caso viene usata la classe HTML per rimanere in ascolto su tutti i bottoni che comportano un aggiornamento del carrello. Non è importante che sia un aggiunta o una rimozione dal carrello, in quanto ogni aggiornamento viene inviato tutto il nuovo carrello.

```
//track cart updates
//add class "track_cartupdate" to the buttons
$('.track_cartupdate').on('click', function () {
    setTimeout(() => {
        $.ajax({
            url: '/it/tracking/getCartData',
            method: 'POST',
            dataType: 'json',
            success: function (result) {
                _paq = window._paq = window._paq || [];
                result.cart_data.forEach(item => {
                    _paq.push(['addEcommerceItem',
                        item[0], // (Required) productSKU
                        item[1], // (Optional) productName
                        item[2], // (Optional) categoryName
                        item[3], // (Recommended) price
                        item[4]// (Optional, defaults to 1) quantity
                    ]);
                })
                if (user) { _paq.push(['setUserId', user]) }
                _paq.push(['trackEcommerceCartUpdate', result.cart_total]);
            },
            error: function (error) {
            },
        });
    }, 100);
});
```

Figura 5.7: funzione javascript trackCartUpdates()

La rotta del pacchtto usata per recuperare l'intero carrello è */it/tracking/getCartData*. Questi dati sono poi inseriti tramite un ciclo nell'array di Matomo come parametri della funzione *addEcommerceItem*.
Infine viene chiamato *trackEcommerceCartUpdate* per segnalare l'aggiornamento del carrello con i prodotti inseriti tramite *addEcommerceItem* e comunicare il nuovo totale del carrello.

tracciamento del completamento dell'ordine

Come nelle precedenti, la funzione rimane in ascolto sull'invio del form di completamento dell'ordine.

```
//track submit order
//add class "track_submitorder" to the billing-form form tag
$('.track_cartsubmit').on('submit', function () {
    let codTrans = $(this).find('input[name="codTrans"]').val();
    $.ajax({
        url: '/it/tracking/getCartData',
        type: 'POST',
        dataType: 'json',
        success: function (result) {
            _paq = window._paq = window._paq || [];
            result.cart_data.forEach(item => {
                _paq.push(['addEcommerceItem',
                    item[0], // (Required) productSKU
                    item[1], // (Optional) productName
                    item[2], // (Optional) categoryName
                    item[3], // (Recommended) price
                    item[4]// (Optional, defaults to 1) quantity
                ]);
            })
            if (user) { _paq.push(['setUserId', user]) }
            _paq = window._paq = window._paq || [];
            _paq.push(['trackEcommerceOrder',
                codTrans, // (Required) orderId
                result.cart_total, // (Required) grandTotal (revenue)
            ]);
        },
        error: function (error) {
        },
    });
});
```

Figura 5.8: funzione javascript trackSubmitOrder()

La funzione è molto simile alla precedente, con eccezione della funzione Matomo chiamata che ora sarà *trackEcommerceOrder* e prenderà come parametro anche un identificativo dell'ordine, in questo caso il codice della transazione.

5.3.2 create-heatmap.js

Il file javascript si occupa della gestione delle heatmaps.
Per ogni tipo di heatmap andrà inserito, nella pagina di interesse, un tag *div* di dimensioni pari alle dimensioni del contenuto della pagina (quindi trascurando padding laterale), se questo è presente i dati sulla posizione verranno immagazzinati in un array e inviati periodicamente all'applicazione Django del server. Per i tre diversi tipi di heatmap bisogna:

- ridimensionare il container perché abbia le stesse dimensioni del contenuto.

```
$('#moveHeatmapContainer, #clickHeatmapContainer, #scrollHeatmapContainer').height($(document).height());
```

Figura 5.9: javscript resize dei container delle heatmaps

- creare l'evento di click/movimento/scorrimento che aggiunga le informazioni dell'evento all'array di punti.

```
$('#clickHeatmapContainer').on('click', function (e) {
  let x  = (e.pageX - $('#moveHeatmapContainer').offset().left)/ $('#clickHeatmapContainer').width();
  let y = e.pageY / $('#clickHeatmapContainer').height();
  if( x>0 && y>0){
    mouseClickArray.push([x, y]);
  }
})
```

Figura 5.10: javascript rilevazione click per heatmap

- ◇ eseguire l'invio periodico dei dati al server, insieme ai dati verrà inviato anche il tipo di heatmap, l'url del sito di provenienza e le dimensioni dello schermo; questi dati torneranno utile nell'applicazione Django e verranno discussi nel capitolo 6.

```
//click, move, scroll heatmaps config
if (document.getElementById('clickHeatmapContainer')) {
  var mouseClickArray = [];
  setInterval(() => {
    if (mouseClickArray.length) {
      $.ajax({
        url: 'https://usertracking.rewardcore.it/heatmaps/updateheatmap',
        method: 'POST',
        contentType: 'application/json',
        data: JSON.stringify({
          site_url: window.location.href,
          data: mouseClickArray,
          type: 'click',
          screenWidth: window.innerWidth
        }),
        success: function (result) {
          mouseClickArray = [];
        },
        error: function (error) {
        }
      })
    }
  }, 5000)
}
```

Figura 5.11: invio dati delle heatmap al server

A parte piccoli accorgimenti il codice per i tre tipi di heatmap è molto simile.

5.3.3 trackingcontroller.php e web.php

Per permettere al codice descritto sopra di recuperare dalla piattaforma i dati su utenti e prodotti, sono state create delle rotte nel file *web.php*, che sono poi gestite dal file *trackingcontroller.php*.

web.php

Nel file sono indicate le rotte, il loro metodo, i middleware usati, le funzioni del controller che gestiscono la rotta.

```php
<?php

use Reward\Usertracking\Http\Controllers\TrackingController;
use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::group(['prefix' => '{locale}/tracking', 'middleware' => ['web', 'security']], function () {
    Route::post('/getCartData', [TrackingController::class, 'getCartData'])->name('TrackingController.getCartData');
    Route::post('/getProductData', [TrackingController::class, 'getProductData'])->name('TrackingController.getProductData');
    Route::post('/getUserId', [TrackingController::class, 'getUserId'])->name('TrackingController.getUserId');
});

Route::group(['prefix' => 'api/'], function(){
    Route::get('/getAllProducts', [TrackingController::class, 'getAllProducts'])->name('TrackingController.getAllProducts');
});
```

Figura 5.12: file delle rotte del pacchetto web.php

Vengono creati due gruppi di rotte, che differiscono per prefisso e middleware.
Il primo gruppo contiene le rotte utilizzate dal javascript del pacchetto descritto precedentemente, hanno del middleware sviluppato da Reward che è utilizzato in diverse rotte delle piattaforme.
Il secondo gruppo contiene una sola rotta API utile per il sitema di raccomandazioni dell'applicazione Django, questo gruppo ha controlli di sicurezza middleware meno stringenti.
Le funzioni del controller hanno lo stesso nome delle rotte stesse, a seguito verranno discusse una per volta.

trackingcontroller.php

Il controller è costituito da una classe ereditata da *BaseController*, una classe messa a disposizione da laravel precisamente a questo scopo. A seguito verranno descritte le funzioni del controller.

La funzione **getCartData()** sfrutta la facade *Auth* per recuperare il carrello corrispondente all'utente che effettua la richiesta. Il carrello viene poi inserito in un array tramite un ciclo. Infine il carrello e il suo totale vengono convertiti in formato *json* e restituiti al richiedente.

```php
class TrackingController extends BaseController
{

    public function getCartData()
    {
        $cart = Auth::guard('client')->user()->getCart();
        $cartData = array();
        if ($cart->items->count() > 0) {
            foreach ($cart->items as $item) {
                array_push($cartData, array(
                    $item->product->external_ref,
                    $item->product->name,
                    $item->product->category,
                    $item->product->getPrice(),
                    $item->quantity,
                ));
            }
            $total = $cart->getTotalPrice();
        } else {
            $total = 0;
        }
        return json_encode([
            'cart_data' => $cartData,
            'cart_total' => $total
        ]);
    }
```

Figura 5.13: funzione php getCartData()

La funzione **getProductData()** richiede le informazioni dal modello *CatalogProduct* che fa parte di un altro pacchetto Laravel realizzato da Reward. Successivamente inserisce i dati in formato *json*.

```php
    public function getProductData(Request $request)
    {
        $id = $request->id;
        $product = CatalogProduct::where('id', $id)->first();
        return json_encode([
            'product' => [
                'sku' => $product->external_ref,
                'name' => $product->name,
                'category' => $product->category,
                'price' => $product->getPrice(),
            ]
        ]);
    }
```

Figura 5.14: funzione javascript getUserId()

La funzione **getUserId()** interroga la facade *Auth* sull'id dell'utente che ha inviato la richiesta. Se l'utente non ha eseguito l'accesso viene ritornato *null*. L'informazione viene restituita tramite *json*.

```php
public function getUserId()
{
    $userId = Auth::guard('client')->user()->id;
    return json_encode(['user' => $userId]);
}
```

Figura 5.15: funzione php getUserId()

La funzione **getAllProducts()** è costituita da una query al modello *CatalogProduct* che restituisce alcuni dati di tutti i prodotti "LIVE" ovvero attivi in quella piattaforma in quel momento. I dati sono ritornati in formato *json*.

```php
public function getAllProducts(){
    return json_encode(CatalogProduct::select('external_ref', 'name', 'category', 'brand', 'reward_price')->where('state', 'LIVE')->get());
}
```

Figura 5.16: funzione php getAllProducts()

Le funzioni hanno lo scopo di reperire le raccomandazioni dei prodotti che saranno determinate dall'applicazione Django descritta in 6. Altre parti della piattaforma potranno richiedere questi dati chiamando la funzione del controller. In particolare le funzioni effettuano delle chiamate API all'applicazione Django del server. Queste restituiscono gli idientificativi dei prodotti in formato *json*.

```php
    public function getRecommendationsForUser($n, $user_id){
        $postdata = array(
            'site_url' => sprintf(
                "%s://%s",
                isset($_SERVER['HTTPS']) && $_SERVER['HTTPS'] != 'off' ? 'https' : 'http',
                $_SERVER['SERVER_NAME']
            ),
            'user_id' => $user_id,
            'n' => $n,
        );
        $url =  "http://usertracking.rewardcore.it/recommendations/recommend";
        $curl = curl_init($url);
        curl_setopt($curl, CURLOPT_HEADER, false);
        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($curl, CURLOPT_POST, true);
        curl_setopt($curl, CURLOPT_POSTFIELDS, $postdata);
        $json_response = curl_exec($curl);
        $status = curl_getinfo($curl, CURLINFO_HTTP_CODE);
        curl_close($curl);
        return $json_response;
    }

    public function getRecommendationsForItem($n, $item_id){
        $postdata = array(
            'site_url' => sprintf(
                "%s://%s",
                isset($_SERVER['HTTPS']) && $_SERVER['HTTPS'] != 'off' ? 'https' : 'http',
                $_SERVER['SERVER_NAME']
            ),
            'item_id' => $item_id,
            'n' => $n,
        );
        $url =  "http://usertracking.rewardcore.it/recommendations/recommend-related-items";
        $curl = curl_init($url);
        curl_setopt($curl, CURLOPT_HEADER, false);
        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($curl, CURLOPT_POST, true);
        curl_setopt($curl, CURLOPT_POSTFIELDS, $postdata);
        $json_response = curl_exec($curl);
        $status = curl_getinfo($curl, CURLINFO_HTTP_CODE);
        curl_close($curl);
        return $json_response;
    }
}
```

Figura 5.17: funzioni php per le raccomandazioni dei prodotti

5.3.4 UsertrackingServiceProvider

A seguito verrà riportato il service provider del pacchetto, questo si occupa semplicemente di pubblicare i file nel progetto base Laravel.

```php
class UsertrackingServiceProvider extends ServiceProvider
{
    public function boot(Dispatcher $events)
    {
        $this->loadRoutesFrom(__DIR__.'/routes/web.php');
        $this->mergeConfigFrom(__DIR__.'/config/usertracking.php', 'usertracking');

        $this->app->register('Reward\Usertracking\UsertrackingRouteServiceProvider');

        $this->publishes([
            __DIR__.'/config/usertracking.php' => config_path('usertracking.php'),
            __DIR__.'/resources/js' => resource_path('js/packages'),
        ]);
    }

    public function register() {}
}
```

Figura 5.18: file php trackingControllerServiceProvider()

La classe eredita *ServiceProvider*, una classe di Laravel creata per questo scopo. Il metodo *boot* viene eseguito da Laravel come inizializzazione del pacchetto. Nella funzione vengono caricate le rotte e configurazioni, viene registrato il service provider e vengono pubblicate le rotte ed il codice javascript.

Capitolo 6

Modulo raccolta dati

Nel seguente capitolo verrà descritto il modulo di raccolta dati, questo è composto da un applicazione Django e dall'applicazione Matomo.
Nel capitolo verrà discusso prevalentemente l'applicazione Django in quanto è la parte effettivamente sviluppata dallo studente, informazioni su Matomo e le sue funzionalità si trovano nel capitolo apposito 4.

6.1 Scopo del Modulo

Lo scopo del modulo è quello di ricevere, elaborare e mostrare i dati non gestiti da Matomo. Inoltre, per raggruppare in un unica piattaforma tutte le informazioni, verranno riportate anche le dashboard di Matomo; in questo modo l'utente non dovrebbe mai dover accedere a Matomo ma poter fare tutto dalla piattaforma Django.

Heatmaps

L'applicazione Django dovrà gestire i dati sulle heatmap ricevuti dal pacchetto Laravel. In particolare dovrà:

- essere in grado di processare e inserire i punti in arrivo nel database,
- essere in grado di reperirli e generare una heatmap a partire da questi punti.

Per ogni pagina di ogni sito potranno essere visualizzate le tre heatmap, una per i click, una per i movimenti del cursore, una per lo scorrimento della pagina.

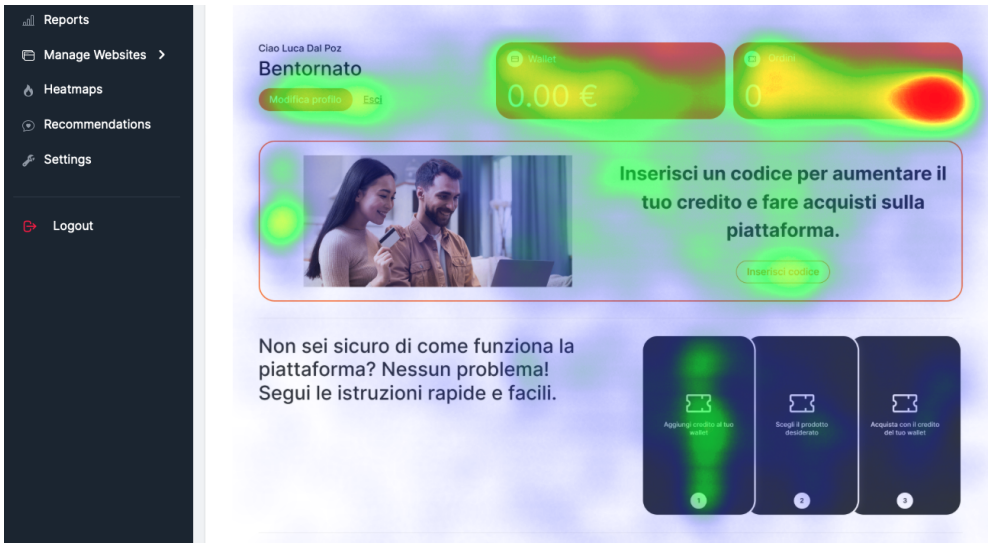

Figura 6.1: Pagina della heatmap dei movimenti del cursore

Gestione di pagine e siti

Per permettere la visualizzazione dei dati, come heatmap o reports di un sito, questo andrà prima inserito come nuovo sito nell'applicazione Django, questa comunicherà con Matomo e provocherà la creazione di un nuovo sito sull'istanza Matomo.
Per ogni sito potranno essere inserite delle pagine. Queste serviranno per la visualizzazione delle heatmap.

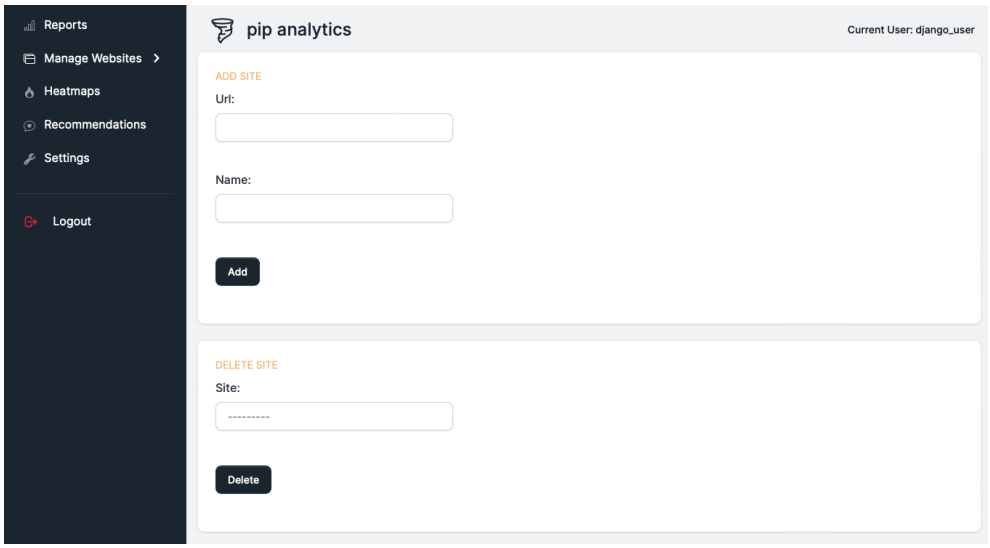

Figura 6.2: Pagina della gestione dei siti

Dashboards di Matomo

Per ogni sito registrato nell'applicazione Django sarà possibile visualizzare i report più importanti di Matomo, in particolare sarà visualizzato: riepilogo dei visitatori, riepilogo e-commercee tempo medio caricamento pagine.

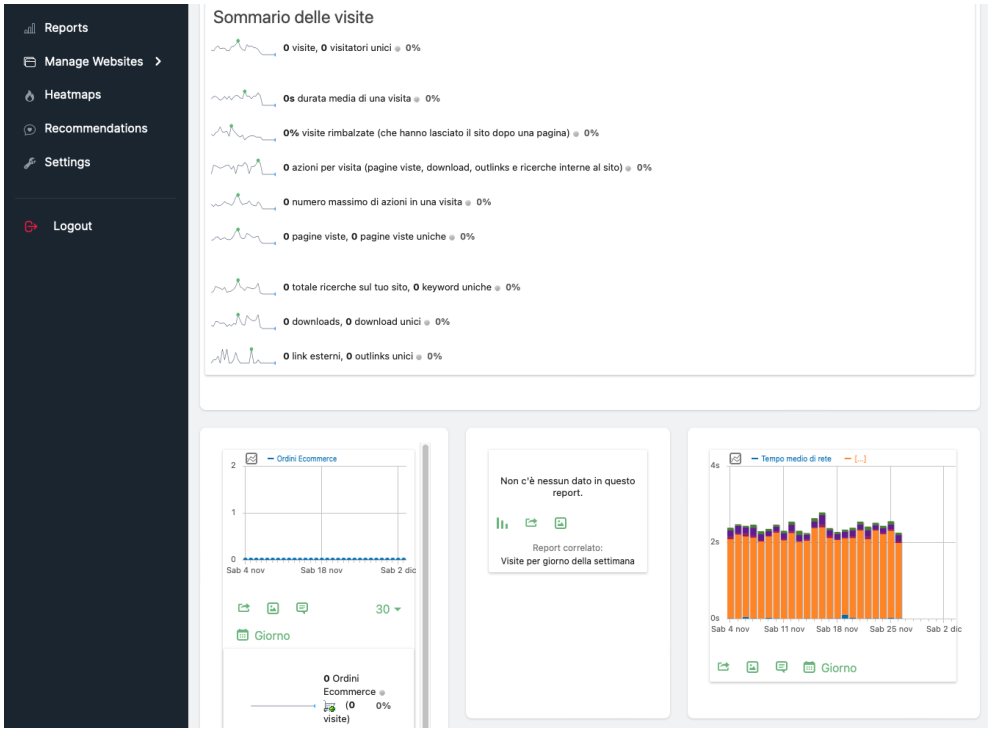

Figura 6.3: Pagina dei report Matomo

Recommender systems

Per ogni sito registrato nell'applicazione Django sarà possibile addestrare un modello di machine learning per raccomandazioni di prodotti. Questo modello sarà addestrato all'inserimento del sito nella piattaforma, sarà poi periodicamente riaddestrato in caso di aggiornamento dei prodotti a acquisizione di nuovi dati sugli utenti.

6.2 Progettazione

I diversi ambiti di funzionalità descritti sopra saranno raggruppati in quelle che Django chiama "app" che saranno più indipendenti possibili l'una dall'altra. L'applicazione avrà comunque un architettura monolitica in quanto ci saranno comunque dipendenze, soprattutto a livello di database.
Django adotta l'architettura "Model View Template" che è una variante di MVC dove:

- il **model** è responsabile per la logica e connessione col database,

- la **view** accetta le richieste e invia le risposte (similmente a un controller in MVC),
- il **template** si occupa della presentazione, solitamente tramite file html.

Oltre a view, modelli e template in una app sono presenti:

- **urls** che elenca metodo, middleware e funzione corrispondente nella view per ogni rotta dell'app.
- **migrations** che sono sostanzialmente delle query sul database fatte per riflettere i cambiamenti effettuati sui modelli. I file di migration sono generati automaticamente da Django.
- **file statici** che sono file da servire al browser come immagini o file javascript. I file statici di ogni app sono collezionati per poi essere serviti sul web tramite il comando collect static.

Per il progetto verranno create tre app una per gestione dei siti, una per le heatmaps, una per le raccomandazioni. Ognuna avrà le sue view, modelli, template, urls, migrations e file statici. In più oltre alle app create dallo studente sono presenti l'applicazione "authentication" e "home" che sono rispettivamente sviluppati da Django e da Volt. Queste si occupano dell'autenticazione, pannello admin e schermata di home. Oltre a queste app saranno presenti i file comuni tra le applicazioni, ovvero i file "core" di Django, come impostazioni e inizializzazione.

Tema Django

Per velocizzare lo sviluppo del presentation layer dell'applicazione si è scelto di utilizzare un admin dashboard open source chiamata "Volt Django". Questa è sostanzialmente un template per un'applicazione Django che mette a disposizione una serie di pagine e componenti HTML già pronte basate su bootstrap. Per sviluppare le pagine del progetto si è partiti da questi template che poi sono stati modificati in base alle esigenze del progetto.

6.3 Implementazione delle feature

A seguito verrà descritta l'implementazione per ogni app.

6.3.1 Gestione di pagine e siti

models

I modelli principali sono per questa app sono *Site* e *Page*: Il primo contiene le informazioni sul sito come url e nome.

```python
class Site(models.Model):
    url = models.URLField(primary_key=True)
    name = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    def __str__(self):
        return self.name
```

Figura 6.4: modello Site

Nel caso in cui serva l'id di Matomo del sito è stato creato un metodo *getMatomoId()* per reperire l'id tramite chiamata API.

```python
def getMatomoId(url):
    token = '944db8434ac54b13babd8ddf8f3702c2'
    matomo_url = 'https://matomo.rewardcore.it/'
    payload = {'module':'API',
               'method':'SitesManager.getSitesIdFromSiteUrl',
               'url': url,
               'token_auth': token,
               'format':'json'
               }
    res = requests.get(matomo_url, params=payload)
    try:
        return res.json()[0]['idsite']
    except IndexError:
        print('error fetching site id')
        return None
```

Figura 6.5: funzione python getMatomoId()

Il modello *Page* invece contiene informazioni sulla pagina come: sito di riferimento, percorso, screenshot della pagina, nome.

```python
class Page(models.Model):
    base = models.ForeignKey(Site, on_delete=models.CASCADE)
    path = models.CharField(max_length=100)
    shot = models.ImageField(upload_to='page_shots')
    name = models.CharField(max_length=50)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    def __str__(self):
        return self.name
```

Figura 6.6: modello Page

urls

A seguito sono riportate le rotte dell'applicazione.

```
urlpatterns = [
    path('managesites', views.manageSites),
    path('addsite', views.addSite, name="add-site"),
    path('deletesite', views.deleteSite, name="delete-site"),
    path('managepages', views.managePages),
    path('addpage', views.addPage, name="add-page"),
    path('deletepage', views.deletePage, name="delete-page"),
    path('reports/<path:url>', views.Visits),
    path('reports/', views.Visits),
]
```

Figura 6.7: file di rotte per pagine e siti, urls.py

Le rotte sono molteplici e ad ognuna corrisponde una funzione nel file views diversa. Verranno descritte solo le funzioni esemplari, che introducono concetti o metodologie nuove.

views:manageSites

La funzione si occupa di mostrare la pagina per aggiungere e rimuovere un sito, da notare l'utilizzo dei form Django. Il form vuoto viene creato tramite costruttore il costruttore di *SiteForm*, che verrà riportato successivamente, l'oggetto form viene passato come contesto del template HTML.

```
def manageSites(request):
    add_form = SiteForm()
    delete_form = DeleteSiteForm()

    return render(request, "sites/manage-sites.html", {'add_form': add_form, 'delete_form': delete_form})
```

Figura 6.8: funzione python view:manageSites()

forms:SiteForm

Le classi dei form sono create nell'apposito file *forms.py* e, per i form creati a partire da un modello, ereditano dalla classe *models.ModelForm* fornita da Django. Il framework infatti permette la creazione automatica di un form a partire da un modello, in questo caso il Modello è *Site* e i campi dati del form sono tutti gli attributi del modello. C'è inoltre la possibilità di personalizzare i campi dati dei form come fatto nella sezione widget tramite delle classi bootstrap.

```python
class SiteForm(forms.ModelForm):
    class Meta:
        model = Site
        fields = '__all__'
        widgets = {
            'url': forms.TextInput(attrs={
                'class': 'form-control'
            }),
            'name': forms.TextInput(attrs={
                'class': 'form-control'
            }),
        }
```

Figura 6.9: classe del form per aggiungere un sito

views:addSite

I dati ricevuti tramite POST vengono inseriti in un oggetto *SiteForm* che viene poi validato. Se questo è valido la funzione procede a creare il sito nella piattaforma Matomo tramite richiesta API, inoltre per permettere la visualizzazione dei widget di quel determinato sito è necessario assegnare i permessi esclusivamente di visualizzazione ad uno user. La gestione delle rotte API e la direttiva "api_view" sono fornite dal "Django Rest Framework" menzionato in 3.

```python
def addSite(request):
    if request.method == 'POST':
        add_form = SiteForm(request.POST, request.FILES)
        delete_form = DeleteSiteForm()
        if add_form.is_valid():
            #adds site to matomo
            addsite_token = '********************'
            addsite_url = 'https://matomo.rewardcore.it'
            addsite_payload = {'module':'API',
                        'method':'SitesManager.addSite',
                        'siteName': add_form.cleaned_data.get('name'),
                        'urls': add_form.cleaned_data.get('url'),
                        'ecommerce': 1,
                        'format':'json',
                        'token_auth': addsite_token
                        }
            addsite_res = requests.get(addsite_url, params=addsite_payload)
            site_id = addsite_res.json()['value']
            #sets django_user permissions to 'view' in matomo
            setaccess_token = '**********************'
            setaccess_url = 'https://matomo.rewardcore.it'
            setaccess_payload = {'module':'API',
                        'method':'UsersManager.setUserAccess',
                        'userLogin': 'Django_server',
                        'access':'view',
                        'idSites': site_id,
                        'format':'json',
                        'token_auth': setaccess_token
                        }
            setaccess_res = requests.get(setaccess_url, params=setaccess_payload)
```

Figura 6.10: prima parte funzione python view:addSite()

Il nuovo sito viene salvato tramite la funzione *save()* chiamata sull'oggetto form. Il sito viene poi inserito nella lista delle trusted origins dell'applicazione django, questo permette di eseguire CORS, la policy altrimenti bloccherebbe la ricezione dei dati sulle heatmap provenienti dal pacchetto Laravel installato nel sito.
Infine viene mostrato il messaggio di successo e viene creato il ed addestrato il corrispondente modello di recommendations di cui si discuterà a seguito.

```
        #adds site to db
        site = add_form.save()
        site.trustOrigin()
        sweetify.success(request, 'Nice! new site saved')
        #make recommendation models

        predictor = RecommendationsPredictor.objects.create(site = site)
        if not predictor.setup():
            sweetify.warning(request, 'site added but there was an error in the recommendation model creation')
    else:
        sweetify.error(request, 'something went horribly wrong')
    return render(request, "sites/manage-sites.html", {'add_form': add_form, 'delete_form': delete_form})
else:
    return HttpResponse(status=400)
```

Figura 6.11: seconda parte funzione python view:addSite()

Utilizzando metodi analoghi agli esempi menzionati, sono stati implementati nell app anche: la cancellazione dei siti, la creazione e cancellazione delle pagine, la visualizzazione delle dashboards di Matomo.

6.3.2 Heatmaps

models

Il modello dell'app è il seguente:

```
class HeatMapData(models.Model):
    page = models.ForeignKey('sites.Page', on_delete=models.CASCADE, null=True)
    x = models.DecimalField(max_digits=10, decimal_places=9, null=False)
    y = models.DecimalField(max_digits=10, decimal_places=9, null=False)
    value = models.BigIntegerField()
    type = models.CharField(max_length=50)
    viewport = models.CharField(max_length=50)
    created_at = models.DateTimeField(auto_now_add=True)
    def __str__(self):
        return self.site_url
```

Figura 6.12: modello HeatmapData

Ogni oggetto è un punto della heatmap che contiene:

- ◇ *page*: pagina di riferimento.
- ◇ *x, y*: Coordinate, ogni punto è un range che avrà come valore i dati registrati in quell'intervallo.
- ◇ *value*: quantità di punti registrati nell'intervallo di coordinate x,y.
- ◇ *type*: tipo di heatmap, click, movimento del cursore o scorrimento.
- ◇ *viewport*: dimensione della finestra da cui provengono i dati.

urls

Le rotte dell'app sono le seguenti.

```
urlpatterns = [
    path('updateheatmap', views.updateHeatmap),
    path('<path:url>/<str:viewport>/<str:type>', views.heatmaps),
    path('', views.heatmaps),
]
```

Figura 6.13: file rotte per le heatmaps, urls.py

La prima è una rotta API utilizzata dal pacchetto Laravel per inviare dati all'applicazione Django, le altre sono usate per visualizzare le heatmaps.

views:updateHeatmap

La funzione è gestisce le richieste API aventi metodo POST:

```
@api_view(['POST'])
def updateHeatmap(request):

    data = request.data
    site_url = data['site_url']
    type = data['type']
    screenWidth = data['screenWidth']

    if screenWidth >= 1400:
        viewport = 'desktop'
    elif screenWidth >= 1000:
        viewport = 'tablet'
    elif screenWidth >= 580:
        viewport = 'smartphone'

    for point in data['data']:
        x_rounded = round(point[0], 2)
        y_rounded = round(point[1], 2)
        if type == 'scroll':
            x_rounded = 0

        hm_entry = HeatMapData.objects.annotate(full_url=Concat('page__base__url','page__path')).filter(
            full_url=site_url, x=x_rounded, y=y_rounded, type=type, viewport=viewport).first()
        if not hm_entry:
            page = Page.objects.annotate(full_url=Concat('base__url','path')).filter(full_url = site_url).first()
            if not page:
                return HttpResponse(status=400)
            hm_entry = HeatMapData.objects.create(page=page, x=x_rounded, y=y_rounded, type=type, viewport=viewport, value=0)
        hm_entry.value += 1
        hm_entry.save()

    return Response(status=status.HTTP_200_OK)
```

Figura 6.14: funzione python views:updateHeatmap()

La funzione:

1. Trasforma la larghezza in pixel dello schermo (*screenWidth*) in una delle tre viewport,

2. Arrotonda le coordinate al range più vicino,

3. Se il punto già esiste aumenta il suo valore se no ne crea uno nuovo.

views:heatmaps

La funzione si occupa di fornire i dati sulle heatmap e renderizzare il template.

```python
def heatmaps(request, url = None, viewport = None, type = 'click'):

    if url and viewport:
        heatmapdata = HeatMapData.objects.annotate(full_url=Concat('page__base__url','page__path'))
            .filter(full_url=url, viewport=viewport, type=type)
        data = serializers.serialize('json', heatmapdata)
        current = Page.objects.annotate(full_url=Concat('base__url','path'), full_name=Concat('base__name',V(': '),'name'))
            .get(full_url=url)
    else:
        current = None
        data = None

    pages = Page.objects.annotate(full_url=Concat('base__url','path'), full_name=Concat('base__name',V(': '),'name'))

    context = {
        'currentpage': current,
        'allpages': pages,
        'heatmapdata': data,
        'type': type
    }

    template = loader.get_template("heatmaps/heatmaps.html")

    return HttpResponse(template.render(context, request))
```

Figura 6.15: funzione python views:heatmaps()

La funzione:

1. Reperisce tutti i dati di una certa heatmap, che è identificata dal tipo, l'url completo della pagina, viewport.
2. Trasforma i dati in json,
3. Reperisce i dati sulle altre heatmap disponibili,
4. Renderizza il template con i dati reperiti come contesto.

heatmaps.html

Nella pagina viene inserito un container nel quale poi verrà disegnata la heatmap dalla libreria *heatmap.js*. Al suo interno ci sarà lo screenshot della pagina in questione.

```html
<div id="heatmapContainer" class="p-0">
  <img src="{{currentpage.shot.url}}" class="rounded"/>
</div>
```

Figura 6.16: container html per disegnare le heatmap

I dati sui punti delle heatmap saranno passati allo script tramite il seguente tag:

```html
<script src="{% static 'create-heatmap.js' %}" id="createheatmap-script"
  data-points="{{ heatmapdata }}" data-type="{{ type }}">
</script>
```

Figura 6.17: tag html per includere lo script create-heatmap.js

create-heatmap.js

Lo script chiamato dal template è il seguente:

```
function getDataPoints() {
    return $('#createheatmap-script').data('points');
}

let data = getDataPoints()

$(function () {

    if (document.getElementById('heatmapContainer')) {
        if ($('#createheatmap-script').data('type') == 'scroll') {
            var heatmapConfig = {
                container: document.getElementById('heatmapContainer'),
                radius: document.getElementById('heatmapContainer').clientWidth,
                maxOpacity: 0.5
            }
        } else {
            var heatmapConfig = {
                container: document.getElementById('heatmapContainer'),
                radius: 40
            }
        }
        var heatmap = h337.create(heatmapConfig);
    }

    if ($('#createheatmap-script').data('type') == 'scroll') {
        data.forEach(element => {
            let x = parseInt(document.getElementById('heatmapContainer').clientWidth / 2)
            let y = parseInt(Number(element.fields.y) * document.getElementById('heatmapContainer').clientHeight)
            heatmap.addData({ x: x, y: y, value: parseInt(Number(element.fields.value)) })
        });
    } else {
        data.forEach(element => {
            let x = parseInt(Number(element.fields.x) * document.getElementById('heatmapContainer').clientWidth)
            let y = parseInt(Number(element.fields.y) * document.getElementById('heatmapContainer').clientHeight)
            heatmap.addData({ x: x, y: y, value: parseInt(Number(element.fields.value))})
        });
    }
```

Figura 6.18: file create-heatmap.js

Il codice:

1. Prende i dati dall'attributo del tag html,
2. Crea la heatmap nell'apposito container,
3. Inserisce i punti nella heatmap.

Come si può notare dal caso in cui la heatmap sia di scorrimento della pagina la creazione e l'inserimento dei punti sono leggermente diversi.

6.3.3 Recommender systems

note sul set di dati

Per quanto riguarda la tecnica di raccomandazioni "collaborative filtering" il train set sarà costituito dai dati e-commerce degli utenti della piattaforma, questi dati sono raccolti da Matomo dunque al momento dello sviluppo di questa app saranno pochi. Per il tempo ristretto di stage, quindi, non è stato possibile recuperare un dataset abbastanza ampio per permettere di prendere decisioni su iperparametri e algoritmo migliore per il modello di machine learning. Per questo motivo quello che segue è un prototipo che andrà aggiustato e rivisto dall'azienda in futuro.

Collaborative filtering

train

La funzione serve ad addestrare il modello di collaborative filtering:

```python
def train_cf(self):

    url = self.site.url

    # data fetch
    products_dict = Product.objects.filter(site=self.site).values('sku')
    products_df = pd.DataFrame(products_dict)
    # get site id
    matomoid = Site.getMatomoId(url)
    if not matomoid:
        return False
    # get ecommerce data from db
    dbconn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
        database="Matomo"
    )
    query = f"SELECT HEX(idvisitor) as idvisitor, s.name as sku, l.quantity as quantity
        FROM matomo_log_conversion_item l JOIN matomo_log_action s ON l.idaction_sku = s.idaction
        WHERE l.idsite = {str(matomoid)}"
    orders_df = pd.read_sql(query, dbconn)
```

Figura 6.19: prima parte dell'addestramento, collaborative filtering

La funzione reperisce i dati necessari per l'addestramento quali: prodotti del sito di interesse e ordini del sito tramite una query diretta al database di Matomo.

```python
    # data preparation
    merged_df = pd.merge(orders_df, products_df, on='sku')
    max_qty = orders_df['quantity'].max()
    data = Dataset.load_from_df(
        merged_df[['idvisitor', 'sku', 'quantity']], reader=Reader(rating_scale=(0, max_qty)))
    trainset = data.build_full_trainset()

    # training
    algo = SVD(n_epochs=10)
    model = algo.fit(trainset)

    # save model to file
    filename = url.replace("/", "$").replace(":", "#") + ".pickle"
    dump.dump(file_name=f'apps/recommendations/cf_models/{filename}', algo=model, verbose=1)
    return True
```

Figura 6.20: seconda parte dell'addestramento, collaborative filtering

I dati vengono preparati e viene applicato l'algoritmo SVD, fornito da Scikit-surprise, per individuare la matrice dei prodotti affini agli utenti. Il modello viene poi salvato su un file che sarà utilizzato per fare le previsioni.

predict

A seguito è riportata la funzione per prevedere lo score dei prodotti per un utente.

```
def predict_cf(self, uid, n):
    url = self.site.url
    scores = []

    filename = url.replace("/", "$").replace(":", "#") + ".pickle"
    loadedmodel = dump.load(
        file_name=f'apps/recommendations/cf_models/{filename}')[1]

    for prod in Product.objects.all().values('sku'):
        iid = prod['sku']
        scores.append([iid, loadedmodel.predict(uid, iid).est])

    scores.sort(key=lambda x: x[1], reverse=True)

    # decode url filename: filename.replace('#',':').replace('$','_')
    df = pd.DataFrame(scores[:n])
    df.columns = ['sku', 'sim_scores']
    df['sim_scores'] = minmax_scale(df[['sim_scores']])
    return df
```

Figura 6.21: codice per previsioni tramite collaborative filtering

Viene caricato il modello, successivamente per ogni prodotto viene stimato uno score per l'utente, questi vengono ordinati e restituiti.

content based filtering

train

La funzione addestra il modello di content based filtering:

```python
def train_cbf(self):
    url = self.site.url

    # data fetch
    products_dict = Product.objects.filter(site=self.site).values()
    content_df = pd.DataFrame(products_dict)

    content_df['Content'] = content_df.apply(
        lambda row: ' '.join(row.dropna().astype(str)), axis=1)

    # Use TF-IDF vectorizer to convert content into a matrix of TF-IDF features
    tfidf_vectorizer = TfidfVectorizer()
    content_matrix = tfidf_vectorizer.fit_transform(content_df['Content'])

    content_similarity = linear_kernel(content_matrix, content_matrix)

    # save model to file
    filename = url.replace("/", "$").replace(":", "#") + ".pickle"
    jldump(value=content_similarity,
        filename=f'apps/recommendations/cbf_models/{filename}')
    return True
```

Figura 6.22: codice di addestramento, content based filtering

La funzione:

1. Reperisce i dati sui prodotti per il sito,

2. Calcola la similarità tra i prodotti tramite TF-IDF e linear kernel

3. Salva il modello su file

predict

A seguito è riportata la funzione per prevedere lo score dei prodotti, in questo caso viene fornito un prodotto da cui calcolare quelli simili.

```python
def predict_cbf_foritem(self, iid, n, products_df = pd.DataFrame(), content_similarity = None):
    if products_df.empty:
        # data fetch
        products_dict = Product.objects.all().values('sku')
        products_df = pd.DataFrame(products_dict)

    if not content_similarity.any():
        filename = self.site.url.replace("/", "$").replace(":", "#") + ".pickle"
        content_similarity = jlload(filename=f'apps/recommendations/cbf_models/{filename}')

    #get similar products from similarity scores
    try:
        index = products_df[products_df['sku'] == iid].index[0]
    except IndexError:
        print('product ordered by the user is not present in products dataframe, try updating products')
        return False

    similarity_scores = list(enumerate(content_similarity[index]))
    sorted_products = sorted(similarity_scores, key=lambda x: x[1], reverse = True)[1:n+1]
    #put them in dataframe with sku
    products_indices = [i[0] for i in sorted_products]
    scores = [i[1] for i in sorted_products]
    top_products_df = pd.DataFrame(products_df.iloc[products_indices]['sku'])
    top_products_df['sim_scores'] = scores

    return top_products_df
```

Figura 6.23: codice per previsioni tramite content based filtering

Viene caricato il modello, vengono stimati i prodotti simili,vengono ordinati e restituiti. Nel caso in cui la previsione voglia essere fatta a partire da un utente questo può essere fatto anche tramite content based filtering, semplicemente per ogni prodotto comprato dall'utente si prevedono quelli simili.

Utilizzo

Il sistema di raccomandazioni può essere utilizzato in due modi:

- Nel caso in cui si vogliano avere delle **raccomandazioni per l'utente**, questi due modelli possono essere usati in concomitanza, in modo da avere un sistema di raccomandazioni più preciso o per evitare problemi come il cold start del collaborative filtering. Le raccomandazioni dunque saranno effettuate a partire da i risultati di entrambi i modelli. In questo caso si parla di un "Hybrid recommender system".

- Invece nel caso in cui si vogliano avere le **raccomandazioni a partire dal prodotto** si può utilizzare solo il content based filtering.

Capitolo 7

Conclusioni

7.1 Consuntivo finale

Durata in ore	Settimana	Descrizione
24	1	◇ Incontro con persone coinvolte nel progetto per discutere i requisiti e le richieste relativamente al sistema da sviluppare; ◇ Verifica credenziali e strumenti di lavoro assegnati; ◇ Presa visione dell'infrastruttura esistente; ◇ Formazione sulle tecnologie aziendali utilizzate; ◇ Configurazione ambiente di sviluppo;
24	2	◇ Configurazione ambiente di sviluppo; ◇ Definizione architettura modulo acquisizione dati; ◇ Definizione architettura piattaforma monitoraggio e reportistica; ◇ Definizione tecnologie e librerie da utilizzare nel progetto;

38	3	◇ Definizione tecnologie e librerie da utilizzare nel progetto; ◇ Implementazione del modulo di acquisizione dati;
38	4	◇ Implementazione del modulo di acquisizione dati;
38	5	◇ Configurazione di un test di acquisizione dati su piattaforma esistente;
76	6, 7	◇ Implementazione piattaforma di monitoraggio e reportistica;
76	8, 9	◇ Implementazione piattaforma di monitoraggio e reportistica; ◇ Collaudo piattaforma di monitoraggio e reportistica;
Totale ore:		**314**

Tabella 7.1: Consuntivo finale

Rispetto alla pianificazione iniziale le ore ore e le attività sono state rispettate, nella configurazione dell'ambiente di sviluppo ci sono stati dei problemi che hanno ritardato leggermente alcune attività.

7.2 Raggiungimento degli obiettivi

Sono stati soddisfatti tutti i requisiti obbligatori del progetto.
Per quanto riguarda quelli desiderabili non è stato possibile rendere il pacchetto installabile esclusivamente tramite PHP composer a causa dei motivi menzionati nel capitolo 5. Oltre all'installazione tramite composer è necessario effettuare ulteriori modifiche per abilitare tutte le funzionalità del pacchetto.
A seguito è riportata la lista dei requisiti Implementati:

Requisito	Stato
RO-1	Soddisfatto
RO-2	Soddisfatto
RO-3	Soddisfatto
RO-4	Soddisfatto
RO-5	Soddisfatto
RO-6	Soddisfatto
RO-7	Soddisfatto
RO-9	Soddisfatto
RO-10	Soddisfatto
RO-11	Soddisfatto
RO-12	Soddisfatto
RO-12	Soddisfatto
RD-13	Non soddisfatto
RD-14	Soddisfatto
RD-15	Soddisfatto

Tabella 7.2: Tracciamento dei requisiti

Il grado di raggiungimento degli obbiettivi come percentuale dei requisiti implementati è:

Requisiti	Completamento
Obbligatori	100%
Desiderabili	66%
Facoltativi	-

Tabella 7.3: Completamento dei requisiti

7.3 Conoscenze acquisite e valutazione personale

Durante il periodo di stage ho avuto molte opportunità di acquisire conoscenze sia in ambito informatico che in ambito aziendale.

Ho potuto sperimentare in un contesto reale i principi e le tecniche apprese nei corsi universitari che mi ha permesso di comprendere più a fondo il loro utilizzo e la loro importanza.
Durante il progetto ho avuto anche modo di approfondire il tracciamento degli utenti sul web, acquisendo nuove conoscenze pratiche, tecnologie e limitazioni di dell'ambito.
Un altra esperienza nuova sono stati i rapporti con i colleghi in ambito di lavoro, reputo queste esperienze molto importanti in quanto possono drasticamente migliorare l'atmosfera dell'ufficio e permettere un processo di sviluppo più piacevole e conseguentemente più riuscito.
Nonostante non sia il primo progetto che realizzo è sicuramente quello più importante; nel senso che, dovendo avere un'applicazione non personale e dover essere utilizzato da molti utenti, la pressione e le responsabilità sono state più alte rispetto a progetti passati.

Glossario

A/B testing *A/B testing* è un metodo di testare due versioni di un sito per vedere quale è migliore dal punto di vista della user experience. 7

API L'*Application Programming Interface API* è un intermediario software tramite cui due applicazioni possono comunicare tra di loro. Quando un'applicazione fa una chiamata API, invia ad un'altra applicazione una richiesta che viene ricevuta e processata. 9, 12, 21, 37

architettura monolitica l'*architettura monolitica* si ha quando un applicazione è composta da funzionalità contenute in un unico blocco (detto "monolite"). 32

composer *composer* è un manager di pacchetti per php. 5, 17, 44

CORS *Cross-origin resource sharing (CORS)* è un sistema di sicurezza che gestisce la condivisione di risorse tra domini diversi. 35

facade *facade* è un design pattern che definisce un interfaccia di alto livello semplificata che rende il sottosistema più facile da usare. 19

framework un *framework* è un architettura logica di supporto, ovvero una struttura base che facilita lo sviluppo software. 3, 8

heatmap *Heatmap* è una mappa di calore che mette in evidenza alcune aree di un immagine tramite colori più caldi dove i dati registrati sono maggiori. 3, 7, 9, 17, 31

HTML *HyperText Markup Language* è un linguaggio di markup che permette di impaginare e formattare pagine web attraverso dei *tag*. 8, 14, 19, 22, 33

javascript *javascript* è un linguaggio di scripting eseguito lato client che viene utilizzato per codificare degli script rendere interattive le pagine web. 9, 14, 19, 20, 25

PHP *php* è un linguaggio di scripting eseguito lato server per creare pagine web dinamicamente. 5, 8, 44

plug and play *plug&play* è una modalità di installazione di nuovi elementi che funzionano senza alcun intervento di impostazione o configurazione da parte dell'utente. 1, 3, 17, 19

SQL *structured Query Language (SQL)* è un linguaggio che permette di gestire i dati di un database tramite query, ovvero interrogando il database. 12, 13

Bibliografia

Siti web consultati

Bootstrap. URL: https://getbootstrap.com.

Django. URL: https://www.djangoproject.com.

Django documentation. URL: https://docs.djangoproject.com/en/4.2/.

Django Rest Framework. URL: https://www.django-rest-framework.org.

Google Analytics. URL: https://marketingplatform.google.com/intl/it/about/analytics/.

heatmap.js. URL: https://www.patrick-wied.at/static/heatmapjs/.

Hotjar. URL: https://www.hotjar.com.

Laravel. URL: https://laravel.com.

Matomo. URL: https://matomo.org.

Matomo documentation. URL: https://developer.matomo.org/api-reference/reporting-api.

Scikit-surprise. URL: https://surpriselib.com.

Volt admin panel. URL: https://themesberg.com/templates/django/free.