

Relazione PW

Gruppo 3

Frontend:

Introduzione creazione del frontend:

Nel progetto è stata inizialmente realizzata un'autenticazione di un utente generico che accede a una pagina web di un negozio online. La homepage web è stata sviluppata in modo semplice, poiché non rappresentava il focus principale del lavoro.

La parte più importante da implementare è stata la sezione amministrativa del progetto: la dashboard per la gestione dei log di sistema.

Per realizzare la parte front end, abbiamo utilizzato html, css ed un leggero ausilio di Javascript (libreria Chart.js in particolar modo) per rendere la pagina più interattiva.

Parte Aliahna:

Nel progetto mi sono occupata di una parte del front end dell'applicazione, cioè l'interfaccia con cui l'utente interagisce. Ho realizzato la homepage del piccolo e-commerce (aiutandomi con diversi template), con una barra di ricerca, i collegamenti ai contatti, al carrello e alla pagina di login, e una griglia di prodotti con foto, nome e prezzo. La grafica è pensata per essere semplice e leggibile, con uno stile scuro e moderno. La ricerca è dinamica: mentre l'utente digita, i prodotti vengono filtrati in tempo reale.

L'aggiunta dei prodotti al carrello è solo simulata tramite un messaggio, visto che non rientra nello scopo reale della nostra applicazione.

Oltre alla homepage, ho realizzato la pagina di login e di registrazione. Le due form risiedono su due Routes differite (/login e /register). I campi sono tutti obbligatori e controllati dal browser e dal backend, in modo da evitare invii di dati errati. È proprio da queste pagine che nasceranno diversi degli eventi da registrare nel sistema di logging, ad esempio i tentativi di login non riusciti, le registrazioni di nuovi utenti, il rilevamento di input malevoli e di attacchi brute force.

In generale, l'obiettivo è stato costruire un'interfaccia semplice da usare, ma allo stesso tempo pronta a dialogare con la parte di sicurezza che verrà implementata nel server.

Parte di Kevin:

All'interno del progetto mi sono occupato di vari punti dell'applicativo per ciò che concerne il Front End, ponendo quindi particolare attenzione alla struttura grafica, alla leggibilità e alla coerenza dell'interfaccia.

La pagina che ha richiesto più tempo è stata sicuramente il file logs.html che rappresenta la Dashboard Log di Sicurezza (Security Logs Dashboard), un pannello di amministrazione dei

logs. La pagina è una vera e propria suite di monitoraggio, che combina una legenda espandibile con conteggio per tutti i tipi di eventi, un sistema di filtri granulari e filtri rapidi per l'interrogazione dei dati, e una sezione Alert di Sicurezza che evidenzia in modo critico attacchi come Brute Force e Malicious Input (SQL/XSS), raggruppando i tentativi per IP. In aggiunta, sono presenti statistiche chiave e tre grafici Chart.js per l'analisi visiva (Distribuzione Log, Top IP, Attività Oraria), completando il tutto con una tabella dettagliata degli ultimi 100 eventi con formattazione colorata a seconda del tipo di log.

Il secondo file che ho creato è quello della pagina *Contatti*, combinando un layout informativo con un modulo di invio messaggi funzionante. La pagina è divisa in una sezione info-section che mostra chiaramente Email, Telefono e Indirizzo in un layout a griglia con icone, e una form-section che contiene il form di contatto obbligatorio. È inclusa la logica per mostrare i messaggi di feedback (ad esempio, conferma di invio) e un avviso di sicurezza finale che garantisce che i dati sono protetti e che gli input malevoli sono monitorati.

Ho creato due pagine di errore, la prima proviene dal file *404.html*. La pagina è focalizzata su un container centrale che mostra il codice '404' in grande e un'icona animata in arancione come colore di accento. Ho fornito chiare opzioni di navigazione tramite pulsanti primari e secondari (Torna alla Home e Pagina Precedente) e ho aggiunto un box di suggerimenti dinamici che cambia l'invito al login a seconda dello stato di autenticazione dell'utente.

Nel file *500.html*, ho implementato la pagina di errore 500 (Server Error), riutilizzando il layout centrale e responsive della 404, ma ho cambiato l'identità visiva e il contenuto per riflettere la gravità del problema. Ho sostituito il colore d'accento con il rosso e usato un'icona di server animata con un effetto 'tremolio'. Il messaggio informa l'utente che il problema è stato registrato e offre azioni appropriate come tornare alla Home o Riprovare con un box di istruzioni specifiche, confermando la coerenza del design di gestione degli errori.

Ho creato la pagina Privacy Policy in HTML, utilizzando le classi CSS già definite (policy-container, policy-header) per mantenere la coerenza del tema scuro e offrire un layout ottimizzato per la lettura. Il contenuto è un documento legale completo e conforme al GDPR, che dettaglia il Titolare del Trattamento, i dati raccolti (inclusa la criptazione delle password con bcrypt), la finalità e la base giuridica. La pagina elenca inoltre i diritti dell'utente, specifica l'uso di soli cookie tecnici e fornisce i contatti per i reclami al Garante, con un link 'Torna alla Registrazione' in apertura e la data di 'Ultimo aggiornamento' in chiusura.

Ho definito poi un foglio di stile CSS completo con un tema scuro (dark mode) e un colore principale arancione/ambra; ho stilizzato tutti gli elementi dell'interfaccia, come i form, i bottoni animati e le allerte, e ho incluso un layout e-commerce responsive con una barra di navigazione fissa in alto e una griglia di prodotti che si adatta a tutti gli schermi.

Poi per quanto riguarda l'area 'amministrativa' del sito (tipo Account, Log, Privacy, Contatti). Ho usato un tema scuro con colori di sfondo scuri e testo chiaro e ho sfruttato al massimo le variabili CSS (come --primary) per rendere il tutto facile da personalizzare. Ho definito uno stile base coerente per tutti i contenitori, le intestazioni e i pulsanti (inclusi i form, i bottoni di salvataggio/pericolo e le notifiche). Poi, ho creato layout specifici: pagine semplici e leggibili per Privacy/Termini, una sezione Account con griglie dati e moduli, e una vera e propria

Dashboard Log complessa, dotata di schede statistiche, filtri avanzati, avvisi e tabelle per la visualizzazione dei dati in modo professionale.

Backend:

Introduzione creazione del backend:

Dopo un'attenta riflessione sulle tecnologie disponibili, abbiamo deciso di utilizzare python come linguaggio per il backend. In particolare, il framework Flask è stata la nostra scelta tra tutti quelli disponibili (avevamo anche considerato Django e FastApi, ma sono risultati troppo complessi e distanti dalle nostre competenze).

Siamo partiti con una semplice implementazione di Flask per capirne il funzionamento base, successivamente abbiamo individuato i task da svolgere per giungere al completamento del progetto nella sua totalità.

Abbiamo deciso di rendere il backend dell'applicativo il più modulare possibile, in questo modo abbiamo lavorato su parti differenti, ma interconnesse tra loro, in modo autonomo. GitHub è stato fondamentale per riuscire a coordinarci sulla scrittura del codice.

Parte Matteo:

Mi sono occupato della realizzazione della parte applicativa necessaria all'interazione tra frontend e backend, in particolar modo della creazione delle varie route per interagire con l'applicazione. Per fare ciò, come detto nell'introduzione, ho utilizzato il framework Flask e tutti i relativi moduli che mette a disposizione per facilitare al programmatore la creazione di un sistema sicuro e facilmente espandibile.

Come punto di partenza, ho realizzato le prime funzioni per gestire le chiamate a determinati endpoint: schermata di login, di registrazione, homepage del sito, sistema di gestione dell'account (accessibile da ogni singolo utente), dashboard di gestione dei logs e pagina contatti.

All'avvio dell'applicazione, viene creato un utente admin predefinito (questa cosa viene gestita mediante un attributo `'is_admin'` sul database) con il quale si può accedere alla dashboard dei logs (gli utenti non admin ovviamente non possono utilizzare questa sezione). È stata anche data all'utente la possibilità di cambiare password ed eliminare l'account (utilizzando una pagina apposta), come conforme alla normativa GDPR.

Il sistema di autenticazione è stato implementato tramite un modulo già disponibile in Flask (`flask_login`) in grado di gestire le sessioni ed il salvataggio delle credenziali in modo sicuro sul database (ho utilizzato `bcrypt` come algoritmo di hash).

È stata data la possibilità di registrare nuovi utenti sul sistema (vengono applicati filtri per scoraggiare l'uso di password deboli).

Dopo aver implementato il sistema di login e l'interazione con il database (della creazione di quest'ultimo si è occupato Daniel), è stato realizzato il sistema per catturare i log ed inviarli ad altri moduli dell'applicativo (anche questi realizzati da Daniel), in modo tale che potessero essere categorizzati, analizzati ed inviati al database in modo consono alla visualizzazione che avverrà in seguito nella dashboard.

Nei log vengo analizzati sia gli eventi malevoli che quelli semplicemente informativi (login e richiesta di pagine).

Il sistema è in grado di rilevare diversi tipi di attività sospette, abbiamo implementato l'individuazione dei tentativi di bruteforce e dell'inserimento di caratteri malevoli che potrebbero portare all'esecuzione di vulnerabilità note (SQL injection, XSS, Command Injection, Path Traversal).

Gli input malevoli vengono rilevati nelle pagine di registrazione e di login, ma anche in una sezione creata appositamente (/contatti) per far inserire all'utente input malevolo e loggarlo di conseguenza: si potrebbe dire che questa pagina funge da Honeygot.

La dashboard di gestione dei log è stata realizzata partendo da uno scheletro creato da Kevin.

Dopo aver ricevuto i file dal mio compagno, ho semplicemente implementato in modo attivo le funzionalità che aveva creato staticamente nell'html: sistema di filtraggio per i log, schede di warning per segnalare eventi sospetti, grafici per visualizzare meglio le problematiche che affliggono il sistema ed un indice per riuscire ad identificare in modo rapido e comprensibile le varie categorie di log.

Per finire: ho implementato delle spunte obbligatorie durante la fase di registrazione sul sito, riguardanti normative sulla privacy e termini e condizioni d'uso; ho aggiunto le pagine di errore 404 e 500 (create da Kevin) che vengono utilizzate in modo automatico da Flask grazie a delle funzioni già implementate di base nel framework.

Parte Daniel:

Mi sono occupato della parte di gestione del database, utilizzando SQLAlchemy, un ORM (Object-Relational Mapping), scelto per la mia familiarità col mezzo avendolo usato durante il periodo di tirocinio ma anche per la sua compatibilità con Flask. Ho creato, come richiesto dalle specifiche, una tabella per gli utenti per gestire gli account e una tabella per i log visibili dall'amministratore.

Mi sono occupato anche della gestione della validazione dei dati inseriti, filtrando possibili input SQL nella registrazione degli account (anche se a livello pratico, per la gestione effettiva del database via ORM sarebbero validate anche eventuali query malevole tramite prepared statements, ma comunque inserito per best practice) e filtrando possibili attacchi comuni, loggando tutti i fallimenti nel database. Ho anche sviluppato la parte di validazione della password, impostando filtri di utilizzo comune per impedire la creazione di account con password estremamente banali.

Ho anche gestito la parte di analisi dei log a livello backend, per compiere questa operazione ho dovuto implementare diverse funzioni interagenti tra loro.

Per facilitare a Matteo l'utilizzo della mia parte di applicativo, gli ho comunicato le funzioni principali da utilizzare (sono state inserite delle doc strings per facilitare la comprensione delle singole funzioni, specificando il tipo di dato di input e di return).

Fondamentale è stato l'utilizzo delle regex per effettuare il filtraggio e la categorizzazione degli input malevoli. Il modulo validator.py si occupa proprio di gestire in modo consono questo meccanismo.