PROGETTO DI BASI DI DATI

CdL in Informatica Umanistica Unipi

ANNO ACCADEMICO 2020/2021

Francesco Bertoli Matricola: 590178
Federico Boggia Matricola: 535711

1. Abstract

Per la realizzazione del progetto di Basi di Dati, di seguito denominato “Binatomy CMS”, è stato utilizzato il pattern architetturale **MVC (Model View Controller)**.
La struttura del progetto è suddivisa in tre tipologie di componenti che insieme costituiscono l’applicazione: i *model*, che si occupano di interrogare il database con apposite query, le *view*, pagine web che contengono l’interfaccia grafica e presentano gli output, e i *controller*, il cui compito è far interagire model e view.

Come sistema di accesso al database è stato scelto **PDO** (PHP Data Object).
La scelta di usare PDO è stata dettata dal fatto che è dotato di una sola interfaccia capace di comunicare con diversi DBMS. Nell’ottica di un futuro riutilizzo e ampliamento di Binatomy CMS, questo rende agevole il passaggio a sistemi diversi da MySQL.

Tutte le query utilizzate vengono trattate sotto forma di *prepared statements*. Questa scelta è dovuta in primo luogo al fatto che i prepared statements riducono il tempo di parsing della query da parte del server, poiché la preparazione della query avviene una volta sola; inoltre i prepared statements sono efficaci nella protezione contro attacchi di *SQL Injection*: essendo le query pre-compilate, l’input utente non può interferire con il codice della query.

Sempre per la gestione della sicurezza dell’applicazione è stato adottato l’utilizzo della libreria *HTMLPurifier*[1] che, sanificando il contenuto dei post, protegge il sistema da eventuali attacchi *XSS.* Inoltre gli input vengono controllati e validati da funzioni implementate appositamente per questo compito.

Per ottimizzare le performance e la resa visiva dell’applicazione è stato implementato un sistema di *rendering* delle views, salvate in formato .phtml, che vengono così generate separatamente e dinamicamente attraverso un sistema di *templating.* Sono state adottate diverse soluzioni per l’ottimizzazione delle performance come l’adozione del formato *.webp*[2] per le immagini e l’uso di *pagination*.

[1] http://htmlpurifier.org/
[2] https://developers.google.com/speed/webp/docs/riff_container

Per quanto riguarda i nomi dei file e la struttura del progetto sono state utilizzate delle convenzioni:

1. I nomi delle tabelle mySQL sono tutti minuscoli e in inglese;

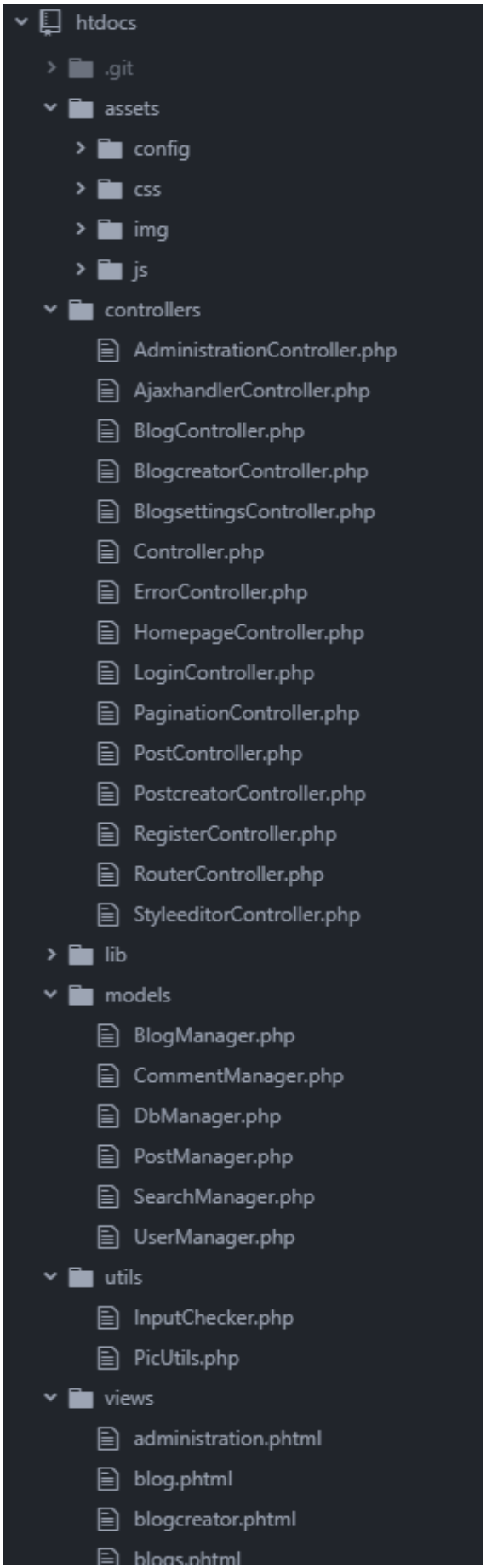
2. I controller e i model utilizzano la scrittura *camel case*: BlogController (controller relativo alla pagina blog), UserManager (model che gestisce le interrogazioni del database relative alla tabella utenti), ecc.

3. I nomi delle viste (views) sono tutti minuscoli (ad es. homepage.phtml)

Tale sistema di nomenclatura è alla base del funzionamento del metodo *autoloadFunction()* contenuto in index.php: tale metodo è registrato come funzione autoload di PHP (*spl_autoload_register()*): si occupa di caricare automaticamente le classi istanziate nel programma, senza dover scrivere all'inizio di ogni file una lunga lista di "include" o "require".

I controller sono tutti derivati dal controller astratto (Controller.php) dal quale ereditano proprietà e metodi fondamentali al funzionamento dell'applicazione, quali ad esempio le funzioni di redirect, di rendering delle viste e di gestione delle eccezioni. Inoltre per la gestione dei controller è stato implementato un RouterController che riceve un indirizzo URL, lo processa e chiama il controller appropriato in base all'informazione fornitagli. Questo consente in seguito al controller di processare la richiesta e mostrare la vista relativa.

Tale procedimento ha consentito, attraverso le giuste impostazioni del file .htaccess, di implementare un sistema di *pretty urls*: durante l'utilizzo del sito non è mai visibile la struttura delle directory e dei file nel web server. Gli URL sono formati con una logica più *user-friendly*:



Esempio:

www.dominio.com/blog/IlMioBlogPreferito

2. Progettazione Concettuale

2.1 Sistema degli URL

Il sistema degli URL è realizzato grazie al reindirizzamento di tutte le richieste da parte dell'utente verso un singolo file (index.php), che costituisce quindi l'*entry-point* del sito. Questo è stato possibile configurando opportunamente il web server.
Il file di entry point effettua tre operazioni fondamentali:

1. **Dichiarazione e implementazione della funzione di autoload**: viene registrata una funzione di autoload programmata per verificare se il nome della classe istanziata appartiene a una classe model, controller o utility
2. **Connessione al database**: viene effettuata la connessione al database, mediante un metodo della classe DbManager
3. **Istanziamento del router**: viene istanziato un oggetto RouterController, al quale viene passato l'URL richiesto dall'utente. Si procede quindi al rendering della vista corrispondente alla richiesta dell'utente.

2.2 Controller Astratto

La classe definisce tre proprietà. La prima è un array con dei dati, che viene utilizzato per memorizzare i dati recuperati dai modelli. Questo array viene in seguito passato ad una vista, che mostrerà i dati all'utente. In questo modo, i dati tra il modello e la vista vengono passati senza interrompere l'architettura MVC.
La seconda proprietà è il nome della view di cui effettuare il rendering.
L'ultima proprietà è l'intestazione HTML della pagina da visualizzare: contiene un titolo e una descrizione.
A tali proprietà viene assegnato un valore dai controller derivati.

La classe ha quattro metodi principali. Il primo è il metodo con cui il controller elabora i parametri. Ogni controller implementa a sua volta il suddetto metodo da solo, per questo motivo il metodo è *abstract*. Il secondo metodo si occupa di renderizzare una vista all'utente, il terzo metodo aggiunge un semplice metodo di reindirizzamento: indirizza un utente a un'altra pagina e termina lo script corrente. L'ultimo metodo controlla se un utente è autenticato nel sistema.

2.3 Router

Il router determina quale controller deve essere chiamato in base all'indirizzo URL. Crea un'istanza del controller chiamato e la memorizza in una proprietà.

Il controller RouterController riceve un URL dall'utente, lo elabora e chiama un controller annidato in base alle informazioni fornite (ad esempio, BlogController). Entrambi questi controller hanno una view, ma mentre quella del router è un template con il layout del sito web (header, footer, etc.), la view del controller annidato è un template basato sul tipo di controller, ovvero sul nome stesso del controller (blog, post, commenti, etc.).

2.3 Pagination

Per ottimizzare il caricamento dei contenuti delle pagine è stato implementato un sistema di *pagination* con un proprio controller: questo si occupa di gestire il rendering all'interno delle viste di blog, post e commenti, permettendo quindi il caricamento graduale dei diversi elementi all'interno delle pagine.

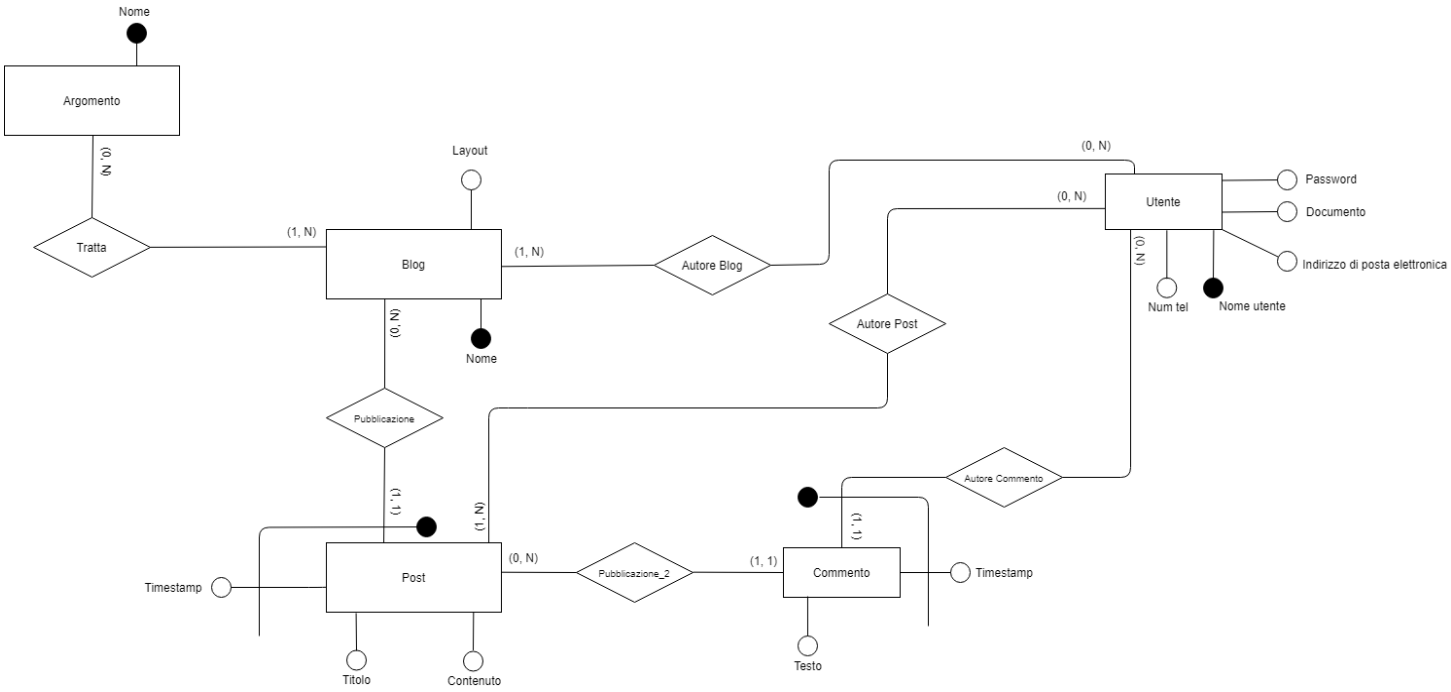
Il sistema di pagination è implementato, così come altre funzionalità del sito, attraverso tecnologia AJAX.

2.4 Ajax

Binatomy CMS fa utilizzo della tecnologia **AJAX** (*Asynchronous JavaScript and XML*). Diverse componenti dell'applicazione, attraverso lo scambio dinamico di dati tra client e server in background, sono in grado di implementare funzionalità come:

- **Controllo e validazione dinamica di vari campi di input utente:** in alcune form del sito (come quella per registrarsi o per creare un nuovo blog), l'utente viene avvisato in tempo reale se alcuni dati da lui inseriti sono già presenti nel database (ad esempio lo username che vuole scegliere o il nome del blog che intende creare)
- **Pagination:** i blog visualizzati nella homepage e i post visualizzati nei blog vengono caricati a piccoli gruppi, in modo da migliorare la performance. Questo è realizzato mediante richieste AJAX attivate dallo scroll utente
- **Caricamento degli sfondi**: il caricamento degli sfondi di blog e post è realizzato sempre con AJAX

3. Schema E-R



Dizionario delle entità

Entità	Descrizione	Attributi	Identificatore
Blog	Sequenza di post	Nome, Layout	Nome
Utente	Utenti che hanno effettuato il processo di registrazione	Nome utente, Password, Indirizzo di posta elettronica, Documento, Numero di telefono	Nome utente
Post	Contenuto ipertestuale e/o multimediale	Titolo, TimeStamp, Contenuto	TimeStamp, Blog
Commento	Commenti testuali relativi ai post	Testo, TimeStamp	Utente, Timestamp
Argomento	Etichette univoche riferite ai blog	Nome	Nome

Dizionario delle relazioni

Relazione	Descrizione	Componenti	Attributi
Autore Blog	Associa un blog a un utente registrato	Utente Registrato, Blog	
Autore Post	Associa un post a un utente registrato	Utente Registrato, Post	
Autore Commento	Associa un commento a un utente registrato	Utente Registrato, Commento	
Pubblicazione	Associa un post a un blog	Post, Blog	
Pubblicazione_2	Associa un commento a un post	Commento, Post	
Tratta	Associa un argomento a un blog	Argomento, Blog	

Schema logico

- Blog(Nome, Layout)
- Utente(NomeUtente, Password, Documento, Numero di Telefono, Indirizzo di posta elettronica)
- Post(TimeStamp, NomeBlog, Titolo, Contenuto)
- Commento(Timestamp, NomeUtente, TimestampPost, NomeBlogPost, Testo)
- Argomento(Nome)
- Autore Blog (NomeBlog, Autore)
- Autore Post(Nome utente, TimestampPost, NomeBlogPost)
- Tratta(NomeBlog, NomeArgomento)

Vincoli di integrità			
1	Autore Blog(NomeBlog)	→	Blog(Nome)
2	Post(NomeBlog)	→	Blog(Nome)
3	Autore Post(NomeBlog, TimestampPost)	→	Post(NomeBlog, Timestamp)
4	Tratta(NomeArgomento)	→	Argomento(Nome)
5	Tratta(NomeBlog)	→	Blog(Nome)
6	Commento(NomeBlogPost, TimestampPost)	→	Post(NomeBlog, Timestamp)
7	Commento(NomeUtente)	→	Utente(NomeUtente)

Business rules

Regole di vincolo
(1) Ogni utente registrato può essere autore al massimo di 5 blog.
(2) È sufficiente che un utente registrato sia coautore di un blog per pubblicare post all'interno di esso.

4. Implementazione

4.1 URLS e sistema di routing?

La realizzazione dei *pretty urls*, come accennato in precedenza, ha richiesto alcuni passaggi, di seguito presentati.
Il file *.htaccess* di configurazione del web server è programmato per reindirizzare tutte le richieste verso index.php: questo permette di evitare la gestione degli URL di default e di poterne implementare una manualmente.
Il file *index.php* effettua tre operazioni fondamentali:

1. **Dichiarazione e implementazione della funzione di autoload**: attraverso il metodo di PHP spl_autoload_register() viene impostata come funzione di autoload la funzione autoloadFunction($class). La funzione viene chiamata quando nell'applicazione viene istanziato un oggetto di una classe non caricata. È programmata per verificare, attraverso espressioni regolari, se il nome della classe istanziata contiene "Controller" o "Model": a seconda del caso, la classe viene caricata con una istruzione di require(). Nel caso la classe non contenga nessuna delle due parole, viene interpretata come una classe di utility, pertanto viene eseguito il require() puntando alla cartella *utils.*
2. **Connessione dal database**: viene effettuata la connessione al database, mediante il metodo connect() della classe DbManager
3. **Istanziamento di RouterController**: viene istanziato un oggetto RouterController, al quale viene passato l'URL richiesto dall'utente attraverso il metodo main(). Si procede quindi al rendering della vista corrispondente.

4.2 CRUD wrapper e connessione

Per comunicare con il database e integrare tale comunicazione nel codice è stato implementato un wrapper *CRUD* (Create, Read, Update, Delete), in modo da fornire uno strumento comune a tutte le componenti dell'applicazione per interagire con il database.
Il wrapper è stato implementato su PDO e fa parte del livello logico, motivo per il quale per gestirlo è stata creato un modello.
La connessione al database viene archiviata non appena viene stabilita, attraverso una classe disponibile ovunque nell'applicazione, per fare ciò sono stati implementati nel modello DbManager membri *static*. Tutti i metodi e le proprietà nella classe sono *static*, il che ha una corrispondenza con il fatto che DbManager è una classe di utilità.

In DbManager viene creata un'istanza PDO per stabilire la connessione al database. L'istanza accetta le impostazioni di connessione come parametro.

Le impostazioni di connessione sono un array associativo nel quale vengono usati le costanti della classe PDO come chiavi.
Il comando di inizializzazione è *"SET NAMES utf8"*. Imposta la codifica in modo che i caratteri internazionali vengano visualizzati correttamente.

Il metodo di connessione crea un'istanza PDO, che utilizza i parametri per la connessione al database (*host, nome utente, password* e *nome database*). Questa viene memorizzata nella proprietà statica *$connection*.

Nel model DbManager sono state definiti i metodi che consentono l'invio delle query al database.
I metodi principali di DbManager sono:

1. queryOne(): restituisce un singolo record
2. queryAll(): restuisce tutti i record interessati;
3. query(): generica

4.3 Assets

I file che non sono parte integrante del sistema model, view, controller, sono raccolti nella cartella assets.

- In assets/config è presente un file .json dove sono conservati i parametri degli stili di default che vengono caricati dinamicamente e proposti all'utente al momento della creazione del proprio blog.
- In assets/css sono conservati diversi fogli di stile, applicati alle diverse viste, in modo da permettere una modularizzazione dei diversi stili e un riutilizzo degli stessi
- In assets/js sono conservati i file javascript. Per la gestione degli eventi è stata utilizzata la libreria *jQuery*[3] e la relativa documentazione.

4.4 Utils

Oltre ad assets e mvc sono presenti dei file php che svolgono funzionalità ausiliari al funzionamento dell'applicazione. Queste funzionalità sono gestite da due file: *InputChecker.php* e *PicUtils.php.* Il primo gestisce e controlla gli input dell'utente, anche attraverso l'uso della libreria HTMLPurifier, definendo metodi di validazione, controllo, sanificazione, escaping e normalizzazione degli input. Metodi che vengono poi utilizzati da diverse componenti dell'applicazione.

Il secondo file gestisce il caricamento e il mantenimento delle immagini all'interno dell'applicazione, occupandosi anche della conversione delle immagini in formato .webp.

[3] https://api.jquery.com/

Le **immagini di sfondo** sono salvate con il nome del blog cui si riferiscono e **una stringa di timestamp**: questo serve a evitare che, quando un utente cambia immagine di sfondo del proprio blog, quest'ultima non si aggiorni a causa della cache del browser.
Con l'aggiunta della stringa timestamp a ogni cambio di sfondo l'immagine di background cambia nome, forzando il browser a riscaricarla.

5. Sicurezza e integrità

Vengono di seguito descritte le misure di sicurezza adottate in Binatomy CMS, sia lato back-end che front-end.

5.1 Prepared Statements

L'applicazione si serve di varie query, definite nelle classi model.
Ogni model ha all'interno una serie di metodi relativi ad azioni sul database (ad es. getBlogPosts, getLayout, createBlog ecc.). Tali metodi conservano il codice della query da usare, redatto utilizzando il **placeholder "*?*"** al posto dell'input utente, e ricevono in ingresso gli input da inserire nella query.
La query viene dunque passata alla classe DbManager insieme a un array contenente l'input.
All'interno di **DbManager la query viene eseguita come prepared statement**.
Così facendo l'input utente non viene mai a contatto con il codice della query, e viene inserito solo una volta che la query è stata compilata come prepared statement.

5.2 Prevenzione con AJAX di nomi duplicati

Molti campi di input vengono controllati ancora prima di essere inviati al server.
È il caso, ad esempio, del form di registrazione: quando l'utente digita del testo all'interno dei campi *username*, *document*, *mail* e *phone* viene innescata una richiesta AJAX che interroga il server se le stringhe inserite dall'utente sono già state utilizzate da altri utenti. In caso affermativo, non viene abilitato il bottone "Invia".

5.3 Sanificazione del contenuto HTML dei post

Binatomy CMS consente agli utenti di inserire codice HTML all'interno dei propri post allo scopo di ampliare le possibilità di formattazione, grazie all'utilizzo dell'editor di rich text WYSIWYG *TinyMCE*.
Questo espone al rischio potenziale di iniezione di codice Javascript malevolo all'interno del contenuto del post.

Misure di sicurezza:

- **Front-end**: TinyMCE filtra in automatico il contenuto del post attraverso l'escaping dei caratteri sospetti
- **Back-end**: il contenuto del post viene comunque filtrato nuovamente nel codice PHP attraverso la libreria *HTMLPurifier*, allo scopo di sanificare eventuali **richieste post contraffatte inviate aggirando i controlli front-end.**

5.4 Validazione della personalizzazione del blog

Binatomy CMS permette agli utenti di **personalizzare il proprio blog** attraverso la pagina *Personalizza Blog*. Tutti gli elementi del layout scelti dall'utente (colori, font, ecc.) vengono codificati **in un'unica stringa JSON** che viene inviata al server e inviata al database, per poi essere utilizzata dalle viste per modificare il CSS delle pagine del blog.
Tutto ciò espone al rischio che l'utente malintenzionato modifichi la stringa inviata al server effettuando un attacco di **CSS Injection**.
Misura di sicurezza:

- **Back-end**: attraverso la funzione di parsing checkPOSTRequest() di StyleeditorController, la stringa JSON contenente il layout viene validata doppiamente. Dapprima si controlla che i campi della stringa coincidano con quelli "sicuri" definiti all'interno di una variabile; in seguito si controlla che ogni singolo valore sia ben formato (ad. es. i valori dei colori vengono sottoposti a un controllo isHexColor(), i nomi dei font sono controllati con una funzione isStringWithSpaces() ecc.)

5.5 Registrazione e autenticazione degli utenti: hashing

Al momento della registrazione dell'utente, viene salvato nel database lo hash della password modificata con un salt. L'algoritmo usato per lo hashing è *bcrypt*, nativamente implementato da PHP con le funzioni password_hash() e password_verify().
È stato scelto di utilizzare l'hashing per evitare di salvare le password in chiaro nel database, pratica dannosa nel caso in cui il database venisse esposto.

Il salting della password è necessario per evitare che, in caso di esposizione del database, si possa risalire a quest'ultima attraverso dizionari di hash di possibili password.

5.6 Filtro delle richieste POST

L'implementazione PHP elabora le richieste POST attraverso campi predefiniti, in maniera da evitare l'inserimento di nuove variabili non previste da parte di utenti malintenzionati.

5.7 Utilizzo di reCAPTCHA

Nella pagina di registrazione è stato implementato un controllo CAPTCHA sfruttando reCAPTCHA di Google, al fine di bloccare eventuali tentativi di registrazione ripetuti.

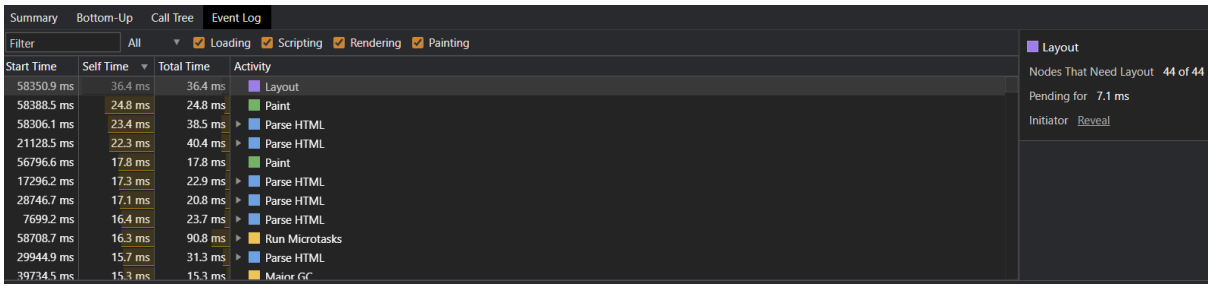
6. Analisi di prestazione

La maggior parte delle pagine richiede un tempo inferiore ai 500 ms per il caricamento completo, il caricamento viene però ottimizzato e frammentato nelle pagine che utilizzano la pagination, permettendo così caricamenti inferiori ai 200-300 ms. Il contenuto del DOM viene caricato in media in 80 ms.
Accedere a un blog richiede in genere meno di 50 ms, gli stessi tempi vengono registrati per l'accesso ad un post.

Per ottimizzare la gestione della memoria del server si è deciso di limitare il numero di blog per utente a 5.

Nell'immagine seguente è riportato il funzionamento del sito nell'arco di circa un minuto nel quale sono state effettuate operazioni di accesso, creazione, modifica e cancellazione di blog e post.

Dal log degli eventi è possibile evincere come l'attività che singolarmente impegna maggiormente la memoria è il Layout



7. Funzionamento

In allegato alla presente relazione vi sono alcuni screenshot che illustrano il funzionamento dell'applicazione.

Sitografia

1. http://anantgarg.com/2009/03/13/write-your-own-php-mvc-framework-part-1/ https://owasp.org/www-community/vulnerabilities/
2. https://owasp.org/www-project-top-ten/
3. https://web.archive.org/web/20121117113421/http://framework.zend.com/manual/1.12/en/coding-standard.php-file-formatting.html#coding-standard.php-file-formatting.general
4. https://www.php.net/manual/en
5. https://www.ict.social/php/mvc
6. https://serverfault.com/questions/88919/faster-option-redirect-via-php-to-php-or-apache-mod-rewrite-redirect-to-php
7. https://stackoverflow.com/questions/5701747/should-i-close-my-php-tags?noredirect=1&lq=1

8. https://www.html.it/pag/370900/il-pattern-mvc-model-view-controller/
9. https://www.w3schools.com/css/css_rwd_viewport.asp

Bibliografia

Learning PHP and MySQL (English Edition) 2006

di Michele E. Davis (Autore), Jon A. Phillips (Autore) http://web-algarve.com/books/MySQL%20&%20PHP/Pro%20PHP%20MVC.pdf