Università
Ca'Foscari
Venezia

Laurea Triennale in Informatica

Sicurezza e Performance delle Piattaforme Online

Uno studio sulla sicurezza e performance delle piattaforme online

Relatore

Professore Pietro Ferrara

Laureando

Andrea D'Attero

Matricola: 881470

Anno Accademico

2021/2022

“You must always be aware of your limitations and also of your best qualities.” - Garry Kasparov

Ringraziamenti

Durante lo sviluppo del progetto ho ricevuto un costante supporto e aiuto dal mio relatore e dai miei collaboratori.

Vorrei pertanto iniziare ponendo i miei ringraziamenti verso il Professore Pietro Ferrara e il Dottore Giovanni Stevanato.

Il progetto su cui questa tesi è scritta ha costituito una valida opportunità per dimostrare le mie abilità acquisite durante il corso del triennio universitario.

Vorrei continuare ringraziando inoltre i miei genitori Anna ed Eugenio i quali sono stati di incredibile sostegno morale e hanno costituito parte integrante della mia performance scolastica.

Abstract

La presente tesi rappresenta uno studio approfondito sulla sicurezza e performance delle piattaforme online andando ad analizzare, in particolare, le applicazioni web dedicate al gioco degli scacchi.

Le diverse tematiche discusse riguardano l'implementazione di sistemi di autenticazione, gioco degli scacchi multiplayer e videoconferenza online.

L'obiettivo finale è analizzare lo sviluppo di applicazioni web e definire delle best-practices riguardo alla performance e alla sicurezza delle piattaforme online, illustrando le diverse problematiche che si incontrano nella costruzione di questi applicativi e confrontando la varietà di soluzioni disponibili.

Contenuti

Introduzione

Con l'inizio del nuovo secolo il mondo dell'informatica ha potuto osservare una grande rivoluzione: si è passati infatti dalla web 1.0 dove l'utente aveva limitate possibilità di interazione con il Web alla web 2.0 dove egli diventa parte attiva.

Con Web 2.0 si intende definire l'insieme di applicazioni online che permettono l'interazione tra il sito-utente (blog, chat, Wiki, Youtube, Facebook, Gmail, ecc.) come mezzi di comunicazione di massa. Oggi infatti si può comunicare in modo facile con ogni angolo del mondo disponendo di informazioni di qualsiasi natura in tempo reale.

Infatti il paradigma di interazione tra l'utente e l'applicazione Web si è trasformato dalla semplice visualizzazione di pagine Web alla possibilità di effettuare richieste alle quali si ottiene in cambio delle risposte.

Grazie al web 2.0, l'utente diventa parte attiva egli infatti può ora, oltre a consultare il sito di una azienda di suo interesse, recensire un prodotto, acquistarlo o restituirlo proprio come se si trovasse personalmente di fronte al prodotto con la comodità di essere contemporaneamente in un altro luogo.

Questa nuova modalità viene agevolata anche attraverso la nascita di nuovi strumenti informatici come ad esempio gli smartphone o dei palmari aumentando la platea degli utilizzatori. Tale rivoluzione ha previsto dunque la nascita di nuovi servizi online offerti da Aziende attraverso piattaforme sviluppate con diversi stack tecnologici per la creazione di nuovi software sempre più evoluti e adattabili alle varie esigenze delle Aziende e capaci di adattarsi alle nuove esigenze di queste. Questo nuovo modo di interagire col mondo del web ha comportato la necessità di adottare nuovi standard di sicurezza visto che ci espone ad attacchi di hacker noto anche come Cybercrime. Vengono alla luce proprio in questo periodo, per sopperire a questa necessità, diverse nuove tecnologie, standard e regolamentazioni nonché nuove organizzazioni che si occupano della loro imposizione.

Nello stesso periodo della rivoluzione web 2.0 e proprio in conseguenza di tale fenomeno nasce la necessità di regolamentare e tutelare in modo più puntuale la privacy dell'utente utilizzatore, nonché la sicurezza dei suoi dati personali. Questo evento ha portato la comunità europea nel 2016 a emanare un regolamento (679/2016) che obbliga le Aziende fornitrici di servizi online di avere, prima dell'utilizzo dei loro servizi, il consenso dell'utente riguardo all'uso dei cookies.

La maggiore interazione tra utente e web ha portato con sé anche un più alto rischio di attacchi informatici che possono portare al furto di dati sensibili come l'identità o dei dati di pagamento dell'utente, determinando una nuova coscienza di cybersecurity.

Negli ultimi anni, il numero di attacchi a piattaforme online è aumentato, infatti, secondo quanto riportato in un articolo della PurpleSec [2], una delle aziende di consulenza sulla sicurezza più accreditate, nel periodo della pandemia, i cyber crimini sono aumentati del 600% rispetto agli anni precedenti e riguardano perlopiù phishing, ransomware, malware e DoS.

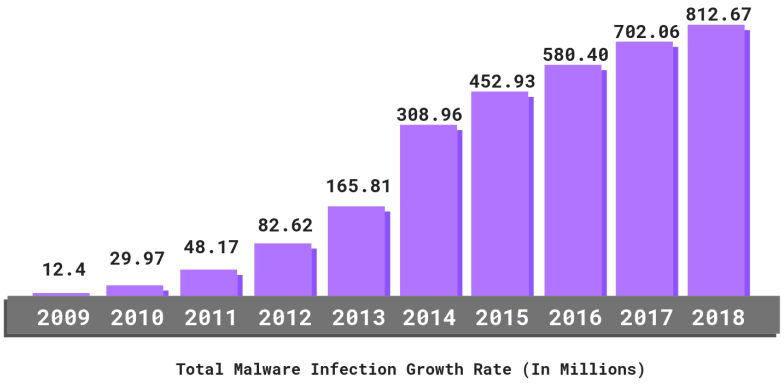

Figura 1.1: Grafico di crescita delle infezioni malware.

A tale riguardo possiamo citare una dichiarazione della IBM [1] la quale afferma che le Aziende dovrebbero destinare almeno il 10% del loro budget per lo sviluppo di nuove procedure di cybersecurity, risulta quindi evidente l'importanza della sicurezza durante lo sviluppo di applicazioni.

Notoriamente un tipico attacco alla sicurezza di un'applicazione è il DDoS (Distributed Denial of Service) il quale consiste nell'invio di estreme quantità di dati, provenienti da computer infettati da virus, verso i server che ospitano l'applicazione web. Ad oggi, non esiste un'effettiva protezione da questo attacco, tuttavia, esistono tecniche per la mitigazione, le quali si occupano di aumentare le prestazioni dell'applicazione stessa in modo che essa possa elaborare dati più velocemente rendendo l'attacco completamente inutile.

Per esporre concretamente in termini di sicurezza e performance ciò che è necessario svolgere, ai fini di proteggersi da eventuali cyber attacchi, è necessario ricorrere ad un esempio di applicazione web, nello specifico, andremo a discutere di piattaforme online dedicate al gioco degli scacchi e alla comunicazione ipermediale.

Il gioco degli scacchi

Gli scacchi sono un gioco strategico svolto su una tavola quadrata detta scacchiera composta da otto righe (numerate da 1 a 8) e otto colonne (contrassegnate dalle lettere dalla A all' H) per un totale di 64 caselle, di colore alternato solitamente il bianco o nero.

Ogni giocatore possiede 16 pezzi di colore diverso composti da:

1. un re che nel gioco è il pezzo più importante;
2. una regina che nel gioco è il pezzo più potente;
3. due alfieri;
4. due cavalli;
5. due torri;
6. otto pedoni.

Ogni casella può essere occupata da un solo pezzo il quale può "catturare" il pezzo avversario andando a occupare la sua casella. L'obiettivo finale del gioco è dare scacco matto, ovvero il giocatore deve portare il Re dell'avversario nella condizione di non poter più effettuare alcuna mossa e cadere in questo modo sotto cattura. Ogni differente pezzo si muove nella scacchiera in modo diverso e, ad eccezione del cavallo, non può scavalcare gli altri pezzi. Non possono inoltre mai fermarsi nella casella occupata da un altro pezzo del proprio colore ma possono prendere il posto del pezzo dell'avversario che in questo modo esce di scena definitivamente cadendo catturato.

1. I pedoni possono avanzare solo in avanti e nel caso vengano mossi per la prima volta sia ha la possibilità di farli avanzare di 2 caselle,
2. l'alfiere può muoversi a piacere nelle diagonali,
3. la torre può spostarsi su qualunque altra casella della stessa riga o colonna,
4. la regina può muoversi in qualsiasi direzione e di quante caselle vuole ma si deve fermare se incontra un altro pezzo del suo colore
5. il re può spostarsi di una casella in qualsiasi direzione ammissibile.

Esistono anche mosse speciali quali:

- Arrocco (castling): questa mossa si può effettuare alla condizione che tra il re ed una delle due torri vi siano solo caselle libere e né la torre né il re siano ancora mai stati mossi allora, il re si può spostare di due caselle verso la torre e la torre viene riposizionata tra la casella di partenza e quella di arrivo del re.

- En passant: è una mossa che può essere eseguita soltanto dai pedoni e avviene quando un pedone, muovendosi per la prima volta di due passi, finisce esattamente accanto (sulla stessa traversa e su colonna adiacente) ad un pedone avversario, nella mossa successiva quest'ultimo può catturarlo come se si fosse mosso di un passo solo. Nel caso in cui non venga effettuata la mossa en passant si perde il diritto di eseguirla successivamente.

All'inizio della partita il giocatore con i pezzi di colore bianco effettuerà la prima mossa, dopo di che, in maniera alternata, i due giocatori andranno a spostare i propri pezzi sulla scacchiera.

Piattaforme esistenti

Per il gioco degli scacchi online esistono già diverse piattaforme, le più famose sono: lichess e chess.com le quali sono costruite con stack tecnologici completamente diversi.

Le tecnologie adottate da lichess si dividono in:

- Scala: linguaggio di programmazione utilizzato per la costruzione del lato backend e generazione di pagine dinamiche.
- HTML: linguaggio di markup per la costruzione di pagine statiche.
- CSS: linguaggio utilizzato per definire lo stile delle pagine web.
- JavaScript: per lo scambio di richieste e risposte tra client e server.
- JQuery: per la manipolazione del documento HTML.
- Nginx: utilizzato come reverse-proxy con funzionalità di load balancing.

Le tecnologie adottate da chess.com si dividono in:

- Ruby: utilizzato per la costruzione del lato backend.
- PHP: utilizzato per generare pagine dinamiche.
- JavaScript: per lo scambio di richieste e risposte tra client e server.
- Cloudflare: utilizzato come reverse-proxy con funzionalità di load-balancing e CDN.

Riguardo alle applicazioni di videoconferenza, le più diffuse al momento sono Zoom e Google Meet le quali sfruttano la tecnologia WebRTC per la comunicazione audio-video multiutente che verrà discussa nel Capitolo 2.

Capitolo 1: Autenticazione

Ogni qualvolta un soggetto accede ad un computer o entra in rete il sistema gli chiede di autenticarsi.

L'autenticazione è quindi una parte fondamentale dell'applicazione e permette di identificare e definire chi è l'utente e quali operazioni è autorizzato a svolgere fornendogli i relativi permessi, infatti, l'utente può accedere a informazioni diverse a seconda dei privilegi di accesso che possiede definendo i livelli di sicurezza e di riservatezza, impedendo dunque attacchi da parte di "ospiti" malintenzionati che tentano di rubare dati sensibili.

Esistono diversi sistemi di autenticazioni nelle applicazioni web, tuttavia, l'architettura di progetto di un sistema di autenticazione è comune e si basa sul modello client-server nel seguente modo:

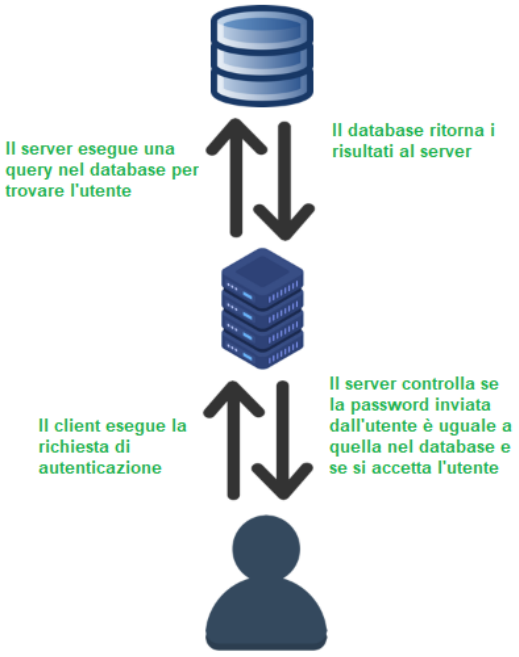

Figura 1.2: Schema di autenticazione.

I sistemi di autenticazione per applicazioni web risultano essere molto complicati da implementare a causa di due problemi principali:

- la registrazione delle password nel database;

- controllo lato server della correttezza password.

1.1 Vulnerabilità nella registrazione delle password

Le applicazioni web che necessitano dell'uso di autenticazione degli utenti devono necessariamente ricorrere alla registrazione delle password generate al momento della creazione dell'account.

La salvaguardia delle password è stata, negli anni, fonte di diverse problematiche, le quali hanno permesso di diffondere sul web interi dataset non cifrati permettendo così a persone malintenzionate l'utilizzo inappropriato dell'applicazione stessa.

L'azienda Adobe nel 2013 è stata fonte di scandalo quando è stato rivelato che le password degli utenti venivano salvate dopo aver applicato un algoritmo di crittografia il quale risultava insufficiente nel garantire la sicurezza.

Un'altra azienda che ha subito forti critiche è stata DreamHost, la quale 11 anni fa salvava le password degli utenti senza nemmeno utilizzare algoritmi di cifratura.

Il NIST (National Institute of Standards and Technology), è un'agenzia governativa degli USA che si occupa di diversi aspetti della standardizzazione di tecnologie, tra le quali troviamo anche l'autenticazione. Questo ente governativo ha definito quali criteri guida da utilizzare al momento della creazione di una password.

Nel documento "NIST Special Publication 800-63", vengono definite le linee guida per l'autenticazione digitale sulla rete.

1.1.1 Approccio 1: Password in chiaro

Il più semplice e più vulnerabile approccio per registrare le password consiste nel salvare il contenuto in chiaro. Questo offre facilità nel controllo lato server della correttezza in quanto viene assicurato che per qualsiasi password inserita il controllo impiegherà la stessa quantità di tempo, impedendo così, ad utenti malintenzionati di analizzare il tempo di risposta e sfruttare attacchi di tipo bruteforce per decifrare la password.

La principale problematica causata da questa metodologia di registrazione delle password è causata dal fatto che in caso di data breach della base di dati ogni account utente è

immediatamente vulnerabile, inoltre, uno studio condotto dalla Google nel 2019 [3] su un campione di 3000 americani di età compresa tra 16 e 50 anni ha mostrato che il 52% delle persone utilizzano la stessa password per diversi account.

1.1.2 Approccio 2: Cifrare le password

Un approccio più sofisticato, tuttavia ancora estremamente vulnerabile, consiste nel cifrare le password tramite algoritmi a chiave simmetrica prima di salvarle nel database.

I problemi che derivano dalla cifratura delle password sono:

1. nel caso in cui la chiave di cifratura utilizzata per mascherare le password venga rivelata, l'intera sicurezza del database sarebbe comunque compromessa;
2. un malintenzionato con accesso al database può constatare che utenti diversi con lo stesso tipo di password cifrata avranno probabilmente una password comune e pertanto debole.

1.1.3 Approccio 3: Hash delle password

L'hashing delle password si rivela la tecnica migliore, anche se non completamente sufficiente, per la registrazione delle password all'interno di un database.

L'hashing è una procedura matematica che consente di ottenere da un insieme di dati un valore univoco totalmente indipendente dall'input originale.

Il problema principale del hashing è che è soggetto alle cosiddette "collisioni" ovvero input diversi posso portare allo stesso hash in output, infatti, gli algoritmi per la produzione di hash sfruttano un'operazione matematica chiamata modulo tipicamente descritta dall'operatore %. Un esempio semplice per illustrare la vulnerabilità delle collisioni è il seguente:

supponiamo di avere un utente che utilizza come password la parola "cane", sfruttando un algoritmo di hashing che converte le lettere in numeri e per ogni lettera esegue l'operazione di modulo con valore del modulo 5 otteniamo 3140 tuttavia anche le password "cade", "caxe", "mane", "wane" ecc. hanno lo stesso valore dopo aver eseguito l'algoritmo di hashing ciò significa che se un utente malintenzionato eseguisse un attacco di brute force

avrebbe maggiori possibilità di autenticarsi, in quanto, a diverse password corrisponde lo stesso hash.

Un'altra vulnerabilità di questo approccio consiste nella scelta dell'algoritmo, infatti, per la costruzione di hash esistono diversi algoritmi che negli anni, con l'aumento delle capacità computazionali dei computer, sono diventati vulnerabili.

Un algoritmo di hashing si dice vulnerabile quando genera un hash di dimensioni piccole e/o è soggetto ad un numero elevato di collisioni.

Esistono infatti le cosiddette "rainbow tables" ovvero tabelle di hash di password pre-calcolate per algoritmi di hashing deboli come MD5, SHA1 le quali permettono, per questi algoritmi, di indovinare il contenuto originale che ha generato l'hash.

Per risolvere il problema delle collisioni e delle rainbow tables viene utilizzato il metodo chiamato "hashing and salting" che consiste nell'associare un "salt" ovvero un valore generato casualmente diverso per ogni utente, alla password, questo garantisce che password simili o identiche siano salvate nel database in maniera completamente diversa, inoltre, rende il brute forcing estremamente complicato, in quanto, richiede di provare ogni possibile password assieme ad ogni possibile salt, infine, risolve il problema delle rainbow tables in quanto esse sono costruite senza considerare i possibili salt.

Quando si effettua hashing delle password è bene essere a conoscenza dei diversi algoritmi che possono essere impiegati per riuscire a calcolare l'effettivo hash finale.

Gli algoritmi di hashing più diffusi sono:

- MD5: algoritmo sviluppato da Ronald Rivest nel 1991, data una stringa in input di lunghezza arbitraria genera una nuova stringa di dimensione massima 128 bit, tuttavia, al giorno d'oggi, l'algoritmo di hashing MD5 risulta vulnerabile a causa della piccola dimensione dell'output.
- SHA-1: algoritmo sviluppato dalla NSA nel 1995, utilizza lo stesso principio di funzionamento del MD5 tuttavia produce stringhe in output di dimensione massima 160 bit, attualmente risulta vulnerabile a causa della piccola dimensione della stringa prodotta in output, nel 2017 la Google è riuscita a realizzare la prima tecnica per generare attacchi per collisione.

- SHA-2: è la versione migliorata dell'algoritmo SHA-1, produce hash di diverse dimensioni (configurabili): 224, 256, 384 0 512 bit.
- SHA-3: è la versione migliorata dell'algoritmo SHA-2, tuttavia, al momento risulta inutilizzato grazie alla buona sicurezza offerta dal suo predecessore.

1.1.4 Approccio 4: Autorizzazione esterna (OAuth)

La OAuth è un protocollo di autorizzazione che fornisce una delega sull'autenticazione dell'utente ad un servizio esterno come Google, Facebook, Twitter e altri.

Il protocollo OAuth è stato creato per interagire principalmente attraverso HTTP, il suo funzionamento si basa sullo scambio di token di accesso che vengono inviati da servizi di autorizzazione di terze parti.

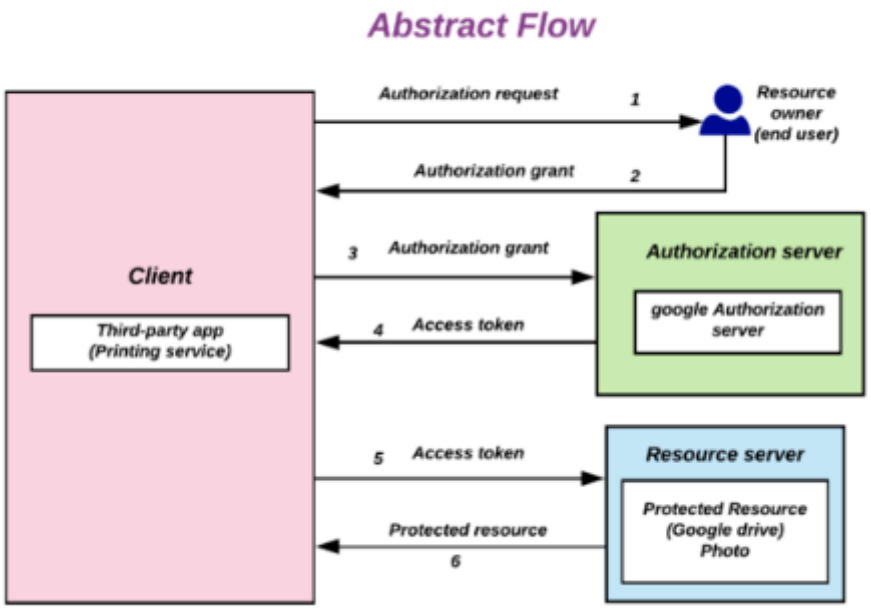

Figura 1.3: Diagramma di flusso dell'autenticazione OAuth.

Solitamente OAuth sfrutta due tipi diversi di token:

- **Access token**: viene utilizzato per eseguire delle richieste al server fornendo autorizzazione.

- **Refresh token**: viene utilizzato per richiedere un nuovo access token quando questo scade senza dover richiedere all'utente nuovamente l'autorizzazione.

Attualmente il sistema di autorizzazione OAuth risulta essere il più sicuro, in quanto, non necessità di password che possono essere violate e delega il processo di autenticazione a servizi esterni. Rimane comunque importante nascondere i tokens e utilizzarli solo per accedere a risorse interne o fornite dal server che offre il servizio OAuth.

1.2 Sistemi di Autenticazione

Esistono principalmente tre approcci diversi per l'autenticazione:

- Autenticazione tramite token: utilizza una semplice stringa contenente dei dati codificati (solitamente in Base64) che identificano univocamente l'utente che ha eseguito il login.
- Autenticazione tramite cookie: utilizza i cookies ovvero stringhe di testo che vengono salvate dal browser ed utilizzate per identificare l'utente.
- Autenticazione tramite cookie e token: questo approccio si basa sull'utilizzo combinato di token e cookies. Questo tipo di autenticazione prevede che il token sia inserito all'interno di un unico cookie.

Ogni sistema di autenticazione possiede dei vantaggi e degli svantaggi di cui andremo a discutere di seguito.

1.2.1 Autenticazione tramite token

Questo tipo di autenticazione consiste appunto nello scambio tra il server ed il client di oggetti chiamati "token".

I token contengono solitamente un id utente il quale viene inviato dal client al server ad ogni richiesta al fine di identificare il soggetto che accede al sistema e verificare i suoi permessi.

Durante lo sviluppo di un sistema di autenticazione tramite token è necessario definire:

1. come il token è costruito;
2. come e dove salvare il token.

Per rispondere a queste domande è necessario ricorrere a degli standard, attualmente, il più diffuso è il JSON Web Token (JWT). Questo standard permette di costruire un token da un json, ovvero un oggetto JavaScript, contenente tutte le informazioni necessarie all'autenticazione dell'utente e ulteriori dati che non possono essere modificati dopo che il token è stato generato.

Un JSON Web Token consiste di tre parti:

1. header: consiste nel tipo di token e dall'algoritmo di firma utilizzato per generare il token stesso.
2. payload: contiene i dati del token.
3. signature: non è altro che l'hash del header e payload e serve per verificare che il messaggio non sia stato compromesso.

L'header ed il payload sono codificati secondo un algoritmo solitamente il Base64 e concatenati con la signature da con un punto producendo una stringa del seguente tipo:

```
header.payload.signature
```

I token vengono firmati non cifrati, infatti, il contenuto di un token è facile da decodificare (basta ad esempio utilizzare il sito jwt.io).

Quando si utilizza un sistema di autenticazione basato sui token è necessario seguire delle best-practices, in quanto, questo approccio presenta delle vulnerabilità che devono essere evitate, le più importanti sono:

1. La chiave utilizzata per firmare il token deve essere mantenuta segreta.

2. Evitare di aggiungere dati sensibili al payload del token, in quanto, a causa del fatto che il token può essere decodificato facilmente, i dati sensibili sarebbero facili da ottenere da utenti malevoli.

3. I token devono avere una data di scadenza e devono esistere meccanismi per l'aggiornamento del token.

4. Utilizzare l'HTTPS, bisogna assolutamente evitare di scambiare token su connessioni non sicure.

Un’altra buona pratica, poco utilizzata, può essere creare un secondo token che garantisce che il token dell’autenticazione sia stato generato proprio dal server proprietario.

L’approccio di autenticazione attraverso l’uso di token offre diversi vantaggi:

1. Stateless: ovvero il server non ha bisogno di mantenere in memoria informazioni di stato in quanto tutto ciò che è necessario all’autenticazione è contenuto nel token stesso ed è salvato dal client.

2. I token possono essere generati facilmente da ovunque: il processo di creazione del token è separato dalla verifica dello stesso, infatti è possibile che questo venga verificato anche da un server diverso da quello che lo ha generato.

3. Accurato controllo dell’accesso: All'interno del token è possibile definire i ruoli, le autorizzazioni e le risorse a cui l'utente può accedere.

4. Flessibilità: il token non è necessariamente legato al browser ma può essere inserito in singole richieste.

5. Semplificano il processo di autenticazione: i token di autenticazione assicurano che gli utenti non debbano reinserire le proprie credenziali di accesso ogni volta che visitano un sito Web.

6. Immunità da attacchi CSRF (Cross-site request forgery)

Il principale svantaggio dell’autenticazione tramite token consiste nella difficoltà di salvare il token in maniera sicura, infatti se questo non è protetto adeguatamente esso risulta molto più facile da rubare rispetto ai cookie a causa della vulnerabilità XSS che verrà discussa nel Capitolo 3.

Esistono diversi modi mettere in sicurezza il token:

1. Utilizzo dei refresh token: in questo caso al posto di salvare l'access token viene salvato il refresh token il quale anche se rubato non permette l’accesso diretto all’account.
2. Utilizzo dei cookie: il token viene salvato all’interno di un cookie il quale è di default protetto dal browser.

1.2.2 Autenticazione tramite cookie

Questo tipo di autenticazione consiste nello scambio tra client e server di oggetti chiamati “cookies” che contengono un insieme di informazioni che identificano l’utente.

Il cookie viene generato dal server dopo un corretto login da parte dell’utente e viene inviato ad esso sotto forma di risposta HTTP, la quale deve avere:

- un header particolare chiamato SET-COOKIE;
- un insieme di attributi;
- una data di scadenza del cookie.

Quando il client riceve la risposta dal server il browser estrae il cookie in formato json e lo salva nell’oggetto document.cookie, in seguito, ad ogni nuova richiesta dal client, il browser includerà nella richiesta il cookie.

Esistono due tipi principali di cookie:

- stateful cookies: contengono un id che punta ad un record nel database che contiene le informazione della sessione, riassumiamo i vantaggi e gli svantaggi nella seguente tabella:

Vantaggi	Svantaggi
non hanno limiti sulla dimensione delle informazioni salvate	aumento di latenza in quanto ad ogni richiesta è necessario eseguire una chiamata al database o alla cache per leggere le informazioni di sessione
risulta facile eliminare i dati della sessione in quanto basta eliminare il record dal database o dalla cache	può risultare difficile la scalare il sistema di autenticazione a causa della dipendenza dal database quando il numero di utenti aumenta

- stateless cookies: contengono tutte le informazioni utili alla sessione e risiedono sul client, necessitano di essere crittografati o almeno firmati per evitarne la manomissione. riassumiamo i vantaggi e gli svantaggi nella seguente tabella:

Vantaggi	Svantaggi

facili da implementare e facili da scalare, inoltre	limitato numero di informazioni in quanto i browser offrono capacità massima di 4 KB per il client
riducono la latenza, in quanto non è necessario effettuare chiamate al database	difficoltà nel riuscire a revocare una sessione, poiché non esiste alcun record nel database da eliminare è necessario sfruttare approcci diversi

Nel caso dei cookie stateless è fondamentale che in applicazioni che sfruttano più web server questi dispongano delle stesse chiavi utilizzate per crittografare o firmare il cookie, poiché, qualora così non fosse, se un cookie generato dal server A fosse ricevuto in una richiesta del client dal server B esso non sarebbe in grado di decifrarne il contenuto (nel caso di cookie crittografati) o stabilirne la validità (nel caso di cookie firmati) e andrebbe ad invalidare la richiesta anche se questa risulta corretta.

L'approccio di autenticazione attraverso l'uso di cookie offre diversi vantaggi:

1. HttpOnly: I cookie possono essere marcati come HttpOnly, questo garantisce che essi non possano essere letti dal client, impedendo così attacchi XSS.
2. Facile sviluppo: Non richiedono l'implementazione di nessun codice lato client in quanto vengono gestiti interamente dal browser.

Gli svantaggi principali dell'autenticazione attraverso cookie sono:

1. Domain-based: I cookie appartengono al dominio del server e non possono (e non devono) essere utilizzati da servizi esterni.
2. CSRF: Vulnerabilità ad attacchi di tipo Cross-site Request Forgery (CSRF) discussa nel Capitolo 3.
3. No API: Non sono adatti all'uso nel caso di costruzione di servizi API.

È importante conoscere ed applicare le best practices per i sistemi di autenticazione che fanno uso di cookies, di seguito esponiamo le principali misure da adottare:

1. Impostare i flag secure e httpOnly a true nel cookie in quanto garantiscono che il cookie verrà trasmesso solamente su connessioni cifrate e che esso non possa essere letto o manipolato da API presenti nel client (ad esempio attraverso l'uso di JavaScript).

2. Nel caso vengano utilizzati dei cookie stateless è necessario cifrare e firmare il contenuto del cookie.

L'approccio di autenticazione tramite cookie, soffre di una vulnerabilità di alta rilevanza chiamata Cross-site Request Forgery (CSRF) che verrà discussa nel capitolo 3.

1.2.3 Autenticazione tramite cookie e token

Negli ultimi anni è stato sviluppato un nuovo approccio all'autenticazione web che prevede l'uso combinato dei cookie con i token e permette di:

- semplificare il salvataggio del token nel lato client mantenendo la sicurezza.
- risolve il problema della vulnerabilità XSS dei token e CSRF dei cookie.

Questo tipo di autenticazione prevede che al corretto login dell'utente il server generi un token e lo invii in risposta al client con l'header SET-COOKIE, quando il client riceverà la risposta il browser automaticamente imposterà il valore di document.cookie al token ricevuto.

Capitolo 2:

Implementazione della Piattaforma

In questo capitolo ci concentreremo sui requisiti richiesti nell'implementazione della piattaforma discuteremo dell'architettura software e dell'effettiva costruzione dell'applicazione finale.

I requisiti progettuali sono i seguenti:

- Implementazione di un sistema di autenticazione;
- Implementazione del gioco multiplayer;
- Implementazione dei tornei;
- Implementazione di una web chat.

2.1 Architettura Progettuale

L'architettura software consiste nella scelta delle tecnologie da utilizzare per la costruzione della piattaforma finale.

In questa parte definiamo le librerie, framework, linguaggi di programmazione, API utilizzate ed il modo con cui queste componenti interagiscono tra loro.

2.1.1 Scegliere le tecnologie

Per lo sviluppo delle applicazioni web esiste una grande varietà di tecnologie utilizzabili, gli stack software (insiemi di componenti per lo sviluppo di piattaforme) principali più diffusi sono:

- MEAN: prevede come tecnologie MongoDB/MySQL per il database, Node e Express per il lato backend e Angular per il frontend, offre buone prestazioni, possiede una grande community, tuttavia, di base, permette soltanto la creazione di un'applicazione web, è quindi necessario ricorrere ad altre tecnologie (come ad esempio Ionic o Electron) per riuscire ad esportare l'applicazione web in applicazione per telefoni

mobili o computer, inoltre richiede un processo di apprendimento di Angular piuttosto lungo.

- MERN: prevede come tecnologie MongoDB/MySQL per il database, Node e Express per il lato backend e React per il frontend, offre buone prestazioni, possiede una grande community, permette di costruire applicazioni cross-platform (web, telefoni e computer) inoltre la curva di apprendimento di React è inferiore a quella di Angular.

- MEVN: prevede come tecnologie MongoDB/MySQL per il database, Node e Express per il lato backend e Vue per il frontend, offre buone prestazioni, possiede una community in crescita, permette di costruire applicazioni cross-platform (web, telefoni e computer), inoltre, la curva di apprendimento risulta migliore di React in quanto non richiede eccessiva conoscenza del linguaggio JavaScript.

- MEFN: prevede come tecnologie MongoDB/MySQL per il database, Node e Express per il lato backend e Flutter per il lato frontend, offre le migliori prestazioni rispetto agli altri stack in quanto le applicazioni sono native ciò significa che sono ottimizzate per il dispositivo specifico, possiede una community in crescita, permette di costruire applicazioni cross-platform (web, telefoni e computer), tuttavia, la curva di apprendimento è maggiore rispetto allo stack MERN e non è dotata di una grandissima quantità di librerie.

- LAMP: prevede come tecnologie Linux come sistema operativo, Apache come server web, MySQL per il database, PHP/Perl/Python per il lato backend, offre ottime prestazioni (migliori dello stack MERN, MEAN e MEVN), permette di costruire soltanto applicazioni web, è quindi necessario riscrivere più volte l'applicazione nel qual caso la si voglia esportare su dispositivi mobili o computer, possiede la più grande community rispetto agli altri stack, tuttavia la curva di apprendimento e sviluppo è molto alta, richiede una approfondita conoscenza di Apache.

Come precedentemente introdotto, a causa della grande vastità degli stack software presenti nello sviluppo web è necessaria un'attenta analisi per capire quale sia il più adatto allo sviluppo della propria piattaforma.

I criteri più importanti da considerare nella scelta dello stack di tecnologie sono:

- **Facilità di apprendimento**: è importante selezionare tecnologie veloci da apprendere per evitare di allungare la fase di sviluppo dell'applicazione e permettere ai collaboratori di entrare velocemente nella fase di produzione.
- **Rapidità di sviluppo**: una tecnologia scelta deve essere facile da utilizzare e permettere un rapido sviluppo.
- **Community**: la grandezza della community è importante quando si sceglie una tecnologia, in quanto, in caso di difficoltà nello sviluppo, può essere fondamentale chiedere informazioni a sviluppatori più esperti.
- **Performance**: lo stack scelto deve mantenere buone performance, pressoché equivalenti a quelle degli altri software stack disponibili.
- **Ecosistema**: presenza di librerie, framework e API preesistenti che possono semplificare lo sviluppo.
- **Flessibilità e scalabilità**: necessarie per il continuo sviluppo dell'applicazione.
- **Requisiti di progetto**: uno stack deve avere la possibilità di rispettare tutti i requisiti di progetto (esistono stack come ad esempio MEAN, MERN e MEVN che talvolta non permettono facilmente l'interazione con funzioni base del dispositivo).

2.1.2 Tecnologie adottate

Lo stack tecnologico selezionato per lo sviluppo del progetto è stato il MERN. Le motivazioni che hanno portato a scegliere questo insieme di tecnologie sono state le seguenti:

- **Facilità di apprendimento**: nonostante lo stack MEVN abbia una curva di apprendimento migliore rispetto agli altri, abbiamo deciso di utilizzare lo stack MERN in quanto l'esperienza passata acquisita in progetti già sviluppati con questo insieme di tecnologie ha permesso di risparmiare tempo nell'apprendimento di nuovi stack.
- **Rapidità di sviluppo**: lo stack MERN attraverso JavaScript permette velocità di sviluppo.

- **Community**: le tecnologie nello stack selezionato hanno le più grandi community rispetto alle tecnologie degli altri stack.
- **Performance**: MERN offre ottime prestazioni anche se inferiori allo stack MEFN.
- **Ecosistema**: ampio numero di librerie, framework e API a disposizione.
- **Requisiti di progetto**: lo stack MERN rispetta a pieno tutti i requisiti di progetto.

Per la gestione delle connessioni socket durante la partita è stato utilizzato l'approccio che prevede l'uso dei socket ed in particolare abbiamo utilizzato la libreria socket.io in quanto rispetto ai websocket tradizionali che si basano sullo scambio di messaggi stringhe, socket.io offre invece la possibilità di inviare eventi contenenti oggetti json, rendendo lo sviluppo della comunicazione a lato backend più semplice.

Per la gestione delle videochiamate abbiamo utilizzato la libreria PeerJs che permette attraverso la tecnologia open source WebRTC di creare connessioni peer-to-peer con scambio audio/video.

Riguardo all'analisi del gioco abbiamo utilizzato la libreria stockfish.js che permette al browser di interfacciarsi direttamente con Stockfish, attualmente la più potente intelligenza artificiale per il gioco degli scacchi.

Nello sviluppo della parte di analisi delle partite è stato necessario ricorrere alla libreria http-proxy-middleware per riuscire a far funzionare il motore lato client ed eliminare il carico computazionale lato server.

Per implementare il gioco e le sue regole è stato fatto uso della libreria chess.js la quale implementa tutte le funzionalità necessarie per le partite.

La tecnologia utilizzata per la distribuzione e replicazione del web server finale è Docker ed in particolare si farà uso del docker-compose per separare in diversi container le varie immagini di cui l'applicazione finale usufruisce.

Una tecnologia interessante, tuttavia non utilizzata, che permette migliore scalabilità dei socket è Redis, un key-value store che funge da cache e permette di migliorare le prestazioni

eliminando la maggior parte delle richieste in lettura al database, permettendo, nel nostro caso, di far comunicare socket connessi in server diversi.

2.1.3 System Design

Il system design consiste nel definire i componenti hardware e software finali da utilizzare al fine di offrire ottime prestazioni anche quando il sistema è sottoposto ad un numero alto di richieste e trasferimento di dati, mantenendo scalabilità e manutenzione facili ed economicamente vantaggiosi.

Il system design si divide in:

- **scalabilità verticale**: consiste nel miglioramento dell'infrastruttura attuale senza necessità di comprare nuove apparecchiature hardware e/o software, esempi di scalabilità verticale sono: acquisto di nuova RAM, CPU e HDD. La scalabilità verticale tuttavia raggiunge dei limiti definiti dalla capacità di espandibilità delle macchine stesse.
- **scalabilità orizzontale**: consiste nell'acquisto di nuove apparecchiature e permette la divisione del carico computazionale tra i nuovi sistemi acquisiti, tuttavia risulta molto costosa.

Esistono diverse componenti hardware utilizzabili nel system design quali:

- **Load balancer**: permette di distribuire il traffico dell'applicazione attraverso diversi server, esistono due tipi principali di load balancer:
    - hardware load balancer: componente hardware tipicamente costosa ma estremamente efficiente.
    - software load balancer: semplice reverse proxy che funge da load balancer e offre diversi algoritmi per la distribuzione del traffico:
        - Round Robin: questo algoritmo distribuisce il traffico in maniera equa inviando le richieste ricevute in ordine ai server, ad esempio, supponendo di disporre di tre server: la prima richiesta viene elaborata dal primo server, la seconda richiesta viene elaborata dal secondo

server, la terza richiesta viene elaborata dal terzo server e dalla quarta richiesta l'algoritmo ricomincia da capo facendola elaborare dal primo server e così via.

        - Least Loaded: questo algoritmo prevede che le richieste ricevute siano elaborate dal server che ha il minore carico di lavoro al momento.

        - Session Based: tutte le richieste provenienti da una sessione vengono elaborate dallo stesso server che ha creato la sessione, questo previene la possibilità che un server possa ricevere una richiesta senza disporre delle conoscenze per elaborarla correttamente.

        - Hashing: tutte le richieste provenienti dallo stesso indirizzo ip verranno elaborate dallo stesso server che ha accettato la connessione inizialmente.

    Esempi di load balancer sono Nginx, Apache e HAProxy.

- Cache: è una memoria volatile estremamente veloce utilizzata per mantenere i dati che vengono letti più spesso, offre velocità maggiori rispetto all'interrogazione diretta di un database o hard-disk, un esempio di cache è Redis. Solitamente le cache grazie al principio di Pareto, che afferma che circa il 20% delle cause provoca l'80% degli effetti, cercano di mantenere in memoria il 20% dei dati rispetto all'intero dataset. Gli algoritmi utilizzati per decidere quali dati devono essere mantenuti in memoria sono:

    - Least Recently Used (LRU): elimina i dati che sono stati utilizzati meno di recente.

    - Most Recently Used (MRU): elimina i dati che sono stati utilizzati più di recente.

    - First In First Out (FIFO): i dati vengono rimossi in base all'ordine in cui sono stati aggiunti, indipendentemente dalla frequenza o dal numero di accessi precedenti.

    - Last In First Out (LIFO): i dati aggiunti più di recente vengono rimossi prima, indipendentemente dalla frequenza o dal numero di accessi precedenti.

    - Least Frequently Used (LFU): elimina i dati che hanno avuto il minor numero di accessi in lettura/scrittura.
    - Random Replacement (RR): i dati vengono eliminati secondo un criterio aleatorio.

- **Web Server**: è un server che ospita un'applicazione web e si occupa di risolvere richieste ricevute dai client.
- **CDN**: il Content Delivery Network anche conosciuto come Content Distribution Network è un sistema distribuito di computer collegati in rete che si occupa di distribuire contenuti come immagini e video, offrono migliori prestazioni rispetto all'uso di web server, in quanto, essendo distribuiti si trovano dislocati in diversi continenti e grazie a ciò permettono di servire le richieste dei client localmente.

Quando si sviluppa il design di un sistema è bene considerare dei pattern già esistenti e provati per essere efficaci, gli schemi più diffusi sono:

- **Monolithic**: tutta la logica è gestita da un singolo componente, risulta ottima per carichi di lavoro bassi, è molto economica, tuttavia insicura in quanto non replicata pertanto se la singola componente dovesse guastarsi l'intero sistema smetterebbe di funzionare.
- **Layered**: i componenti sono raggruppati in strati logici, ogni richiesta passa attraverso i livelli eseguendo alcune attività specifiche che dipendono dal sistema stesso.
- **Service Oriented Architecture**: in questo modello ogni componente è in realtà un servizio che esegue alcune funzionalità specifiche, risulta spesso costoso tuttavia sicuro e molto performante.
- **Microservices Architecture**: consente di creare un'applicazione come raccolta di piccoli servizi indipendenti chiamati micro servizi, è il design pattern più popolare al momento grazie ai bassi costi e l'alta affidabilità e performance.
- **Client-server**: l'applicazione viene gestita interamente da un server a cui i client fanno affidamento per la risoluzione di richieste, è estremamente semplice e molto

utilizzato offre buone performance ma deve essere mantenuto e aggiornato con la crescita delle richieste.

- **Peer-to-peer**: il carico dei dati è distribuito tra i client, risulta molto performante su sistemi di condivisione file ma talvolta insicuro sotto vari aspetti come ad esempio l'impossibilità di verificare l'integrità dei file trasmessi.

Durante lo sviluppo del progetto abbiamo deciso di utilizzare una architettura ibrida composta dal modello client-server per l'autenticazione ed il gioco e dall'architettura peer-to-peer per la gestione della webchat.

Il design ideale di sistema per il progetto è rappresentato nella figura 2.1. Le parti contrassegnate in blu indicano che i componenti devono essere replicati, mentre quelle in rosso indicano che al momento i componenti non sono stati implementati.

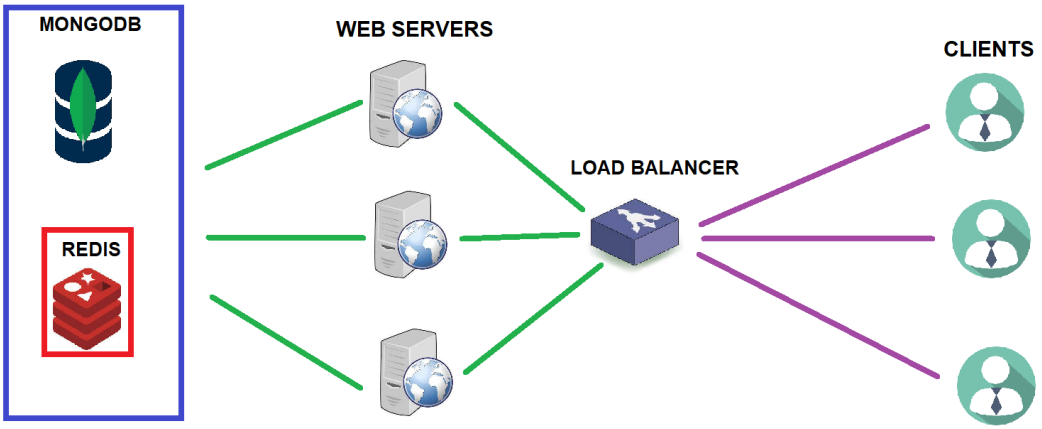

Figura 2.1: Diagramma ideale dell'architettura di sistema.

2.2 Sistema di autenticazione

In questa parte presentiamo le motivazioni che ci hanno portato a scegliere quale sistema di autenticazione utilizzare e come questo è stato sviluppato nella nostra piattaforma.

2.2.1 Sign-in e Sign-up

Come sistema di autenticazione abbiamo deciso di utilizzare il tradizionale approccio che richiede la email e la password senza ricorrere a servizi esterni.

Al momento della creazione di un nuovo profilo il browser invia una richiesta HTTP al server alla route /sign-up il quale quando la riceve esegue una callback che si occupa di:

1. Estrarre i campi username, email e password dal body della richiesta;
2. Controllare che non sia presente un utente con gli stessi username e/o email;
3. Generare un nuovo id univoco per l'utente;
4. Effettuare l'hash della password;
5. Creare un nuovo record sulla collection User contenente userId, username, email e hash della password;
6. Creare un nuovo profilo di gioco contenente userId ed il punteggio elo.

Prima di salvare il nuovo utente nel database la password viene sottoposta ad un algoritmo di hashing offerto dalla libreria bcryptjs la quale genera una stringa composta dall'hash della password originale a cui viene concatenato un salt casuale di lunghezza 10. Questa stringa infine viene sottoposta nuovamente all'hashing ed infine viene salvata sul database nel campo password.

Quando l'utente effettua il login il browser invia un richiesta alla route /sign-in, questa, una volta ricevuta dal server, esegue una callback che si occupa di:

1. Estrarre i campi username, email e password dal body della richiesta;
2. Eseguire una query per trovare l'utente a cui corrisponde la email presente nella richiesta;
3. Comparare attraverso la libreria bcryptjs la password contenuta nella richiesta inviata dal client con quella salvata nel database;
4. Se le password corrispondono il server genera un nuovo token e lo invia al client.

2.2.2 Gestione del token

A seguito della richiesta di login il server invia una risposta all'applicazione client la quale estrae il token dalla risposta e lo salva tramite uso di localStorage.

Normalmente questa è una pratica da evitare in quanto espone il token ad attacchi XSS, tuttavia grazie all'utilizzo del framework React l'applicazione è immune a questo tipo di vulnerabilità (vedremo in dettaglio il perchè nel Capitolo 3).

Quando il token viene salvato su localStorage, esso viene incluso in ogni successiva richiesta al server mediante l'header Authorization.

Ad ogni successiva richiesta ricevuta dal server, questo si occuperà di estrarre il token dalla richiesta e verificare che sia valido ed in tale caso la richiesta verrà processata correttamente altrimenti essa verrà ignorata.

2.3 Gestione della partita

Introduciamo ora come è stata implementata all'interno dell'applicazione web la parte di gioco degli scacchi tra più giocatori.

2.3.1 Creazione della partita

Un utente correttamente autenticato quando accede alla pagina home dell'applicazione ha la possibilità di creare una nuova partita impostando:

- la durata dell'incontro;
- la classificazione ovvero se il punteggio dei giocatori a fine partita verrà modificato in caso di vittoria e sconfitta;
- l'incremento di tempo per ogni mossa eseguita.

Una volta definita e confermata la creazione della nuova partita il browser invierà una richiesta alla route /games/create che andrà ad eseguire una callback del server che si occupa di:

1. Decidere il colore dei pezzi del giocatore che ha creato la partita;

2. Creare la nuova partita e aggiungerla al database;

3. Inviare come risposta il record appena creato nel database.

Quando il client che ha creato il match riceve la risposta dal server senza alcun errore esso verrà redirezionato alla pagina /games/:gameId.

2.3.2 Inizio della partita

Al primo accesso dell'utente nella pagina identificata dalla route /games/:gameId viene istanziata una connessione di tipo WebSocket con il server.

Dopo che la connessione è stata stabilita tra server e client quest'ultimo invia una richiesta alla route /games/:gameId/play la quale eseguirà una callback che si occupa di:

1. Impostare la partita come iniziata modificando il valore del campo isStarted a true solo nel caso in cui entrambi i giocatori siano entrati;

2. Impostare il timestamp di inizio partita;

3. Inviare come risposta i dati della partita.

In seguito a questo il client invierà tramite socket un evento "join" comunicando l'effettiva partecipazione alla partita.

Successivamente al momento della connessione alla partita del secondo giocatore il server invierà un evento "start" ad entrambi i socket nella stanza che andrà ad attivare il timer del giocatore con i pezzi di colore bianco.

2.3.3 Svolgimento della partita

Durante lo svolgimento della partita, quando uno dei giocatori effettua il movimento di un pezzo, viene inviato un evento "move" al server, il quale contiene: il token del giocatore, l'id della partita, la mossa effettiva giocata rappresentata da una stringa codificata secondo notazione algebrica.

Quando il server riceve un evento "move" esegue una callback che si occupa di:

- Verificare che il token sia valido, ovvero che non sia scaduto e che appartenga ad un utente.
- Verificare che il token appartenga effettivamente all'utente che può eseguire la mossa di gioco (questo per evitare che utenti malintenzionati possano modificare la posizione di gioco senza essere effettivamente in possesso delle facoltà per farlo).
- Verificare attraverso chess.js che la mossa sia una mossa valida e nel qual caso lo sia eseguirla.
- Inviare la mossa eseguita dal giocatore al suo avversario attraverso i socket.
- Diminuire il tempo di gioco del giocatore che ha compiuto la mossa in base al tempo trascorso dall'ultima mossa eseguita dall'avversario e la mossa appena eseguita dal giocatore stesso.
- Aumentare il tempo di gioco del giocatore in base al campo timeIncrement definito durante la creazione della partita.
- Avviare il timer di game over per l'avversario

2.3.4 Fine della partita

Nel gioco online degli scacchi esistono diversi modi in cui la partita può terminare: timeout, resa, sconfitta, pareggio per stalemate (avviene quando il giocatore si trova nella condizione di non poter muovere alcun pezzo seppur non è in situazione di scacco) e pareggio per accordo.

La gestione dei vari casi possibili del fine partita avviene tramite invio dal lato server al client di specifici eventi:

- **Evento win**: inviato dal server al client per indicare che la partita è stata vinta.
- **Evento lose**: inviato dal server al client per indicare che la partita è stata persa.
- **Evento surrender**: inviato dal server al client per indicare che la partita è stata vinta per resa.

- **Evento accepted-draw**: inviato dal server al client per indicare che la partita è terminata in parità per accordo di entrambi i giocatori.

La vittoria per timeout avviene quando un utente esaurisce tutto il tempo disponibile per eseguire una mossa o quando, essendo uscito dalla partita, non ritorna in gioco nei 60 secondi successivi.

Il server dispone di un timeout che ha durata equivalente al tempo di gioco rimanente al giocatore che deve compiere la mossa.

Il timeout viene avviato all'inizio della partita per il giocatore bianco e si alterna tra i giocatori ogni volta che questi compiono una mossa. Quando il timeout termina allora il giocatore che è di turno perde ed il server invia un evento win all'avversario ed un evento lose al giocatore perdente.

Se ad uno dei giocatori durante la partita cade la connessione e non riesce a ristabilirla nei successivi 60 secondi allora perde la partita ed il server invia un evento win all'avversario ed un evento lose al giocatore perdente.

La vittoria per resa avviene quando uno dei due giocatori spontaneamente decide di arrendersi, in questo caso, il socket del client invia un evento di tipo "surrender" il quale attiverà una callback lato server che si occupa di:

1. Terminare la partita per vittoria per resa impostando i campi:
    a. **winnerId**: al valore dell'id del giocatore avversario.
    b. **timestamps**: aggiungendo il timestamp dell'evento appena ricevuto alla lista dei timestamp.
    c. **hasEnded**: verrà impostato a true per indicare che la partita è finita.
    d. **reason**: verrà impostato a "Surrender".
2. Comunicare all'avversario la resa.

La fine partita per pareggio ottenuto tramite accordo avviene quando uno dei due giocatori decide di offrire una patta. Al verificarsi di questa situazione, il socket del client invia un evento di tipo "offer-draw", il quale esegue una callback lato server che invia un evento

all’avversario di tipo “offer-draw”, in questo caso l’avversario può rispondere in due maniere diverse:

- Accettando l’offerta di patta: il socket dell’avversario quindi invierà un evento di tipo “offer-draw” ed il server risponderà inviando ad entrambi i giocatori l’evento “accepted-draw”, inoltre imposterà i seguenti campi del record relativo alla partita corrente:
    - timestamps: aggiungendo il timestamp dell’evento appena ricevuto alla lista dei timestamp.
    - hasEnded: verrà impostato a true per indicare che la partita è finita.
    - reason: verrà impostato a “Draw”.
- Rifiutando l’offerta di patta: in questo caso non viene inviato nessun evento, infatti, basta che il giocatore muova un pezzo per rifiutare l’offerta di pareggio, inoltre, non sarà possibile offrire un pareggio per almeno le prossime 10 mosse eseguite dai giocatori.

2.3.5 Post Partita

A seguito della fine della partita, nel caso in cui questa fosse classificata, il punteggio dei giocatori deve essere modificato in base all'esito finale (vittoria, sconfitta e pareggio) e allo standard utilizzato per definire la variazione dell'elo.

Negli anni si sono susseguiti diversi sistemi per il calcolo del punteggio dei giocatori al termine della partita di scacchi. Attualmente, il sistema più utilizzato è stato definito dal fisico Arpad Emrick Elo nel 1960 e sfrutta un approccio statistico basato sulle probabilità di vittoria e sconfitta dei giocatori.

Di seguito illustriamo tramite pseudocodice il funzionamento dell'algoritmo di calcolo del elo:

$$\text{DiffA} = (\text{RatB-RatA})/400$$
$$\text{ExpectedScoreA} = \frac{1}{1 + 10^{\text{DiffA}}}$$
$$\text{RatA} = \text{RatA} + \text{k} \cdot (\text{outcome - ExpectedScoreA})$$

Dove:

- RatA: punteggio elo attuale del giocatore A.
- RatB: punteggio elo attuale del giocatore B.
- k: costante definita dallo standard utilizzato.
- outcome: 1 se la partita è terminata con una vittoria, 0.5 se la partita è terminata in pareggio e 0 se la partita è terminata con una sconfitta.
- ExpectedScoreA: indica la probabilità di vittoria del giocatore A.

Per definire la costante k nel calcolo del punteggio elo si può ricorrere a diversi standard quali:

- Standard USCF:

Valore di k	conditions
32	elo del giocatore < 2100
24	elo del giocatore compreso tra 2100 e

	2400
16	elo del giocatore > 2400

- Standard FIDE:

Valore di k	conditions
40	per un giocatore nuovo nell'elenco di valutazione fino al completamento degli eventi con un totale di 30 partite e per tutti i giocatori fino al compimento del 18° compleanno, a condizione che la valutazione rimanga inferiore a 2300
20	per i giocatori che sono sempre stati classificati sotto 2400
10	per i giocatori con una valutazione pubblicata di almeno 2400 e almeno 30 partite giocate in eventi precedenti. Successivamente rimane permanentemente a 10

Siccome al momento della registrazione dell'utente l'applicazione non raccoglie i dati anagrafici abbiamo deciso di utilizzare lo standard USCF, in quanto lo standard FIDE prevede l'uso della metrica dell'età dei giocatori.

2.4 Web Chat

In questa parte discutiamo riguardo a come è stata implementata la funzionalità della web chat.

2.4.1 Creazione della stanza

Un utente correttamente autenticato ha la possibilità attraverso la pagina home di creare delle stanze utilizzate per la comunicazione audio video con altri utenti.

Al momento della creazione della stanza è possibile decidere se questa abbia accesso pubblico o privato, la differenza tra queste due tipologie consiste nella possibilità degli utenti di accedere alla stanza liberamente nel caso questa sia ad accesso pubblico, o su concessione dell'admin nel caso in cui la stanza sia impostata con accesso privato.

Una volta confermata la creazione della stanza il browser invia una richiesta HTTP alla route /rooms/create la quale esegue una callback che si occupa di:

1. Creare un id univoco per la stanza;
2. Impostare l'utente che ha creato la stanza come admin;
3. Creare la nuova stanza nel database;
4. Restituire il record nel database della stanza all'utente.

Quando l'utente riceve la risposta dal server esso viene redirezionato alla pagina effettiva della stanza creata.

2.4.2 Accesso alla stanza

Un utente autenticato per accedere ad una stanza deve utilizzare il path /rooms/:roomId, una volta entrato non avrà immediatamente accesso ma dovrà aspettare nella waiting room che il server comunichi la possibilità di accedere.

L'accesso alla stanza dipende interamente dalla tipologia di stanza:

- Pubblica: quando un utente si dirige all'indirizzo della stanza il browser invia una richiesta HTTP al server richiedendo l'accesso al quale il server risponde dando la possibilità all'utente di accedere.
- Privata: quando un utente si dirige all'indirizzo della stanza il browser invia una richiesta HTTP al server richiedendo l'accesso, il server inoltra la richiesta all'admin nella stanza il quale può accettare o rifiutare la richiesta:
    - Nel caso l'admin accetti la richiesta viene garantito all'utente che ha richiesto l'accesso la possibilità di entrare nella stanza.

- Nel caso l’admin rifiuti la richiesta l’utente che ha richiesto l’accesso viene redirezionato alla pagina principale.

2.4.3 Comunicazione tra i client

Una volta ottenuto l’accesso alla stanza il client apre una connessione socket attraverso la libreria PeerJs con il server peer, se l'apertura della connessione ha successo il client apre una ulteriore connessione di tipo websocket utilizzata per ricevere ed inviare eventi tra i client.

Stabilita la connessione websocket con il server, il client invia l’evento “join-room”, a cui il server risponderà inviando, in broadcast, a tutti i client nella stanza un nuovo evento di tipo “user-connected” che notifica agli utenti la connessione di un nuovo peer.

Quando la connessione peer viene accettata il peer-server la redireziona verso gli altri peer nella stanza.

In seguito alla connessione del client con i peer, per tutti i membri della stanza verrà eseguita una callback lato client che avvia lo scambio dati audio-video con l’utente che si è appena connesso.

Ad ogni nuova azione sulla scacchiera viene inviato un evento al server per mezzo della connessione websocket stabilita in precedenza.

Gli eventi che vengono scambiati tra i client nella stanza sono i seguenti:

- **user-connected**: indica che un nuovo utente si è connesso.
- **user-disconnected**: indica che un utente si è disconnesso.
- **admin-mute**: l’admin della stanza disattiva al client la possibilità di poter comunicare attraverso l’audio.
- **board-update**: viene inviato dall’admin ai client e permette di impostare la posizione di gioco.
- **toggle-move**: viene inviato dall’admin verso uno specifico client e fornisce la possibilità all’utente scelto di effettuare una mossa nella scacchiera.

- **toggle-mute**: viene inviato da un client verso tutti gli altri client nella stanza e permette di abilitare/disabilitare il microfono.
- **toggle-camera**: viene inviato da un client verso tutti gli altri client nella stanza e permette di abilitare/disabilitare la videocamera.
- **toggle-board**: viene inviato dall'admin verso uno specifico client e fornisce la possibilità all'utente scelto di muovere i pezzi della scacchiera.
- **move**: rappresenta una mossa eseguita da uno dei partecipanti alla stanza che ha permesso di muovere i pezzi nella scacchiera.
- **joined-room**: indica che l'utente è entrato nella stanza correttamente.
- **ban**: viene utilizzato per espellere un utente dalla stanza.

Capitolo 3: Sicurezza

In questo capitolo verranno illustrate le problematiche che emergono durante lo sviluppo delle applicazioni web ed in particolare le vulnerabilità a cui le piattaforme online possono essere soggette.

Le principali fonti di falle di sicurezza sono:

- Presenza di bug nel programma sviluppato: la cui origine è determinata dallo sviluppatore del programma stesso come, ad esempio, il Buffer overflow che consiste nella scrittura di un maggior numero di dati all'interno di un buffer rispetto a quanto esso può effettivamente contenere.
- Presenza di bug nelle dipendenze utilizzate: la cui origine è determinata da librerie o altri strumenti impiegati dall'applicazione stessa per raggiungere un fine.
- Inadeguata architettura di sistema: l'architettura della piattaforma può essere inadeguata ed esporre l'applicazione a vulnerabilità causate dall'inefficienza nell'elaborazione di dati.

Riguardo alle prime due tipologie di vulnerabilità è solitamente sempre possibile e nella maggior parte dei casi facile riuscire a riconoscere e sistemare le falle di sicurezza.

Nel caso di un'inadeguata architettura di sistema può risultare estremamente complicato riuscire a sistemare la vulnerabilità in quanto si rende necessario effettuare un intero check delle componenti di sistema e sostituire o riparare la componente stessa esponendo la piattaforma ad un periodo di downtime.3.1 Sicurezza Database

La sicurezza a livello database si occupa di definire accuratamente i privilegi e le credenziali di accesso dei diversi utenti che possono accedere al database e le regole per mantenere l'integrità dei dati.

Le principali fonti di vulnerabilità a livello database si dividono in:

1. **Inappropriata configurazione del database**: utilizzo di credenziali di autenticazione troppo deboli, scelta inadeguata di provider e/o regione.

2. **Vulnerabilità delle query**: SQL Injection e scarsa ottimizzazione delle query.

3.1.1 Configurazione del database

Prima di iniziare con lo sviluppo effettivo del database è necessario configurare appropriatamente il DBMS.

Durante lo sviluppo abbiamo deciso di utilizzare un'istanza cloud del database MongoDB, questo ha richiesto al momento della creazione del cluster la configurazione di alcuni parametri quali:

- **Cloud Provider & Region**: indica il provider (Amazon AWS, Google Cloud o Microsoft Azure) e la regione dove verranno mantenuti i dati.
- **Cluster Tier**: indica la tipologia di Cluster adottata e si divide in:
    - dedicata: il cluster viene configurato su una macchina unica, questo offre alte prestazioni, tuttavia richiede alti costi.
    - condivisa: il cluster viene condiviso su una macchina che contiene istanze di altri cluster, questo non comporta costi tuttavia si ha a disposizione un numero limitato di risorse.

Come Cloud Provider & Region abbiamo scelto Amazon AWS in Irlanda il quale data la vicinanza con l'Italia offre minore latenza e rientra nel piano gratuito offerto da MongoDB Cloud.

Come Cluster Tier abbiamo selezionato Shared il quale offre la memoria RAM condivisa con le altre istanze e 512MB di spazio dedicato allo storage.

Dopo aver creato il cluster è necessario configurare la modalità di accesso in lettura e scrittura al database definendo:

- **Metodo di autenticazione**: indica il modo selezionato con cui autenticarsi, MongoDB offre due metodi principali:
    - Username and Password: viene creato un utente con i permessi di lettura e scrittura al database identificato da username e password.

    - Certificato: viene utilizzato un certificato trasmesso all'inizio della connessione con il database, MongoDB utilizza certificati X.509 che offrono l'autenticazione senza password e permettono a qualsiasi utente fornito di apposito certificato di accedere con i privilegi di admin al database.

- **Whitelist degli Indirizzi IP**: viene utilizzata per limitare l'accesso al database, consentendo solo a determinati indirizzi di poter interagire con il cluster.

- **Database Access**: definisce i vari utenti ed i relativi permessi di cui dispongono per accedere al database.

Come metodo di autenticazione essendo in fase di sviluppo e non produzione abbiamo scelto l'approccio Username and Password definendo un utente admin con permessi di accesso in lettura e scrittura al database, in un contesto di produzione è preferibile utilizzare l'autenticazione basata su certificato, in quanto offre maggiore sicurezza e semplifica il processo di login.

Riguardo al whitelisting degli indirizzi IP essendo in fase di sviluppo abbiamo deciso di permettere l'accesso da qualunque indirizzo, tuttavia, in un contesto di produzione questa risulta essere una pratica sconsigliata a cui è preferibile invece permettere l'accesso al cluster solo agli indirizzi IP delle componenti che interagiscono effettivamente con il database, ad esempio i web server.

Il database access è stato configurato attraverso l'uso di un unico utente admin il quale dispone di tutti i permessi per la manipolazione della base di dati. Solitamente questa metodologia è sconsigliata in produzione, risulta invece più appropriato definire diversi utenti con diverse tipologie di permessi, ad esempio creando un utente per la lettura dei dati dal database ed un'altro utente per la scrittura dei dati nel database. Utilizzando questo approccio si evita che un utente malintenzionato riesca a manomettere la query e a farle eseguire azioni che non è pensata di compiere.

3.1.2 Vulnerabilità delle query

Pur essendo MongoDB un database NoSQL risulta vulnerabile al SQL Injection, infatti, in MongoDB, è possibile iniettare codice all'interno di query che non sono state appropriatamente formattate.

Un attacco SQL Injection avviene quando l'input proveniente dall'utente non viene rigorosamente controllato, ed in questo caso, può permettere nei database di inserire codice malevolo all'interno delle query.

In MongoDB le iniezioni di codice avvengono in maniera diversa e risultano molto meno potenti rispetto alle tradizionali SQL Injection presenti nei database relazionali. Infatti, non esiste modo di iniettare codice, all'interno di una query, che possa alterarne completamente il significato, questo è garantito da MongoDB attraverso due meccanismi:

- Utilizzo dei BSON: le query sono rappresentate tramite oggetti Binary JSON e non in formato stringa.
- Assenza di parsing: le query non vengono sottoposte al parsing che rappresenta la fondamentale vulnerabilità dei linguaggi SQL, in quanto, viene richiesto di interpretare il significato della query stessa, la quale può essere alterata dall'input dell'utente.

Un esempio di SQL Injection in MongoDB è il seguente:

```
User.findOne({
      "name" : req.body.name,
      "password" : req.body.password
}, callback);
```

Normalmente quello che avviene al momento dell'esecuzione della query consiste nella ricerca all'interno della tabella User di un unico utente identificato dalle stringhe name e password, tuttavia, se il campo name non è sottoposto a delle rigorose regole di sanitizzazione, potrebbe essere inserito come valore `{ $ne: 1 }`. Al verificarsi di tale circostanza il significato della query viene alterato e verrà restituito un risultato che non tiene

in considerazione il valore del campo password (in particolare viene restituito un utente il cui campo name è diverso dal valore “1”).

Un’altro modo in cui una SQL Injection può avvenire in MongoDB è attraverso l’uso della funzione $where all’interno della query, la quale, permette di filtrare i dati attraverso l’uso di una funzione javascript.

Un esempio di questo è il seguente:

```
db.collection.find({ $where: function() {

      return (this.product == $productData)

}});
```

Il contenuto di $productData potrebbe essere il seguente: `'m'; sleep (10000)` questo comporta l’esecuzione remota di codice.

Per risolvere la vulnerabilità SQL Injection è consigliato fare uso di librerie come mongo-sanitize che si occupano della sanitizzazione dell’input, oppure utilizzare il driver della libreria mongoose e definire le collezioni come degli schemi. Al fine di rendere lo sviluppo dell’applicazione più veloce e mantenere il codice scalabile abbiamo preferito utilizzare la seconda opzione.

3.2 Sicurezza Web

Durante lo sviluppo di un’applicazione web si può incorrere a diverse problematiche di sicurezza che sono, nella maggior parte dei casi, legate al funzionamento del browser, ed in minore parte, all’architettura di sistema dell’applicazione.

Le principali fonti di vulnerabilità a livello dell’applicazione web sono:

1. **Insufficiente controllo dell'input**: questa condizione può causare l’inclusione di script malevoli che possono rubare informazioni dell’utente.
2. **Scarso controllo della fonte dell’input**: questa condizione può far eseguire azioni ad un utente senza che esso abbia l’effettiva intenzione di compierle.

3. **Mancanza di monitoraggio delle dipendenze**: l'applicazione web sviluppata necessità dell'uso di librerie esterne le quali possono essere soggette a vulnerabilità, una mancanza di controllo e aggiornamento delle dipendenze può abilitare attacchi dannosi all'applicazione.
4. **Mancanza di crittografia**: la crittografia non viene utilizzata, questo può permettere ad utenti malevoli nella stessa rete di controllare e leggere il traffico e addirittura manipolarlo.
5. **Inadeguata architettura di sistema**: un sistema poco performante può essere sottoposto a grossi carichi di richieste, questo può rallentare il servizio o addirittura renderlo inutilizzabile.

3.2.1 XSS

Il Cross-site Scripting è tuttora una delle vulnerabilità più comuni nel web e avviene quando l'applicazione web non effettua un controllo adeguato sull'input ricevuto, questo, permette ad utenti malevoli di inserire codice, solitamente JavaScript, all'interno di una pagina web che potrà essere eseguito da altri utenti al momento della visualizzazione della pagina stessa.

Esistono diverse tipologie di attacchi XSS quali:

1. **Reflected**: avviene quando l'input di un utente viene immediatamente inserito nella pagina. Il metodo più comune di attacco XSS Reflected avviene quando i parametri contenuti nell'url di una pagina web vengono inseriti all'interno della pagina stessa ad esempio supponiamo di avere il seguente url:

   https://xssvulnerable.com/home?message=<script>alert();</script>

   Al momento del caricamento della pagina il parametro "message" verrà letto e inserito all'interno del codice HTML, in seguito il browser eseguirà tutti gli script contenuti nella pagina ed essendo il valore del parametro "message" interpretabile come uno script anch'esso verrà eseguito.

2. **Stored**: avviene quando l'input di un utente viene salvato sul database o comunque dal server. Il metodo più comune di attacco XSS Stored avviene quando l'utente malevolo compila un form ed il contenuto viene salvato dal server e mostrato ad ogni successiva richiesta di visualizzazione.

Esistono diversi metodi per proteggere l'applicazione web da attacchi XSS quali:

1. **Encoding dell'output**: l'input ricevuto dall'utente prima di essere visualizzato all'interno della pagina viene sottoposto al escaping che consiste nel sostituire tutti i caratteri che possono essere interpretati come codice con caratteri equivalenti ma non interpretabili, ad esempio i caratteri: &, <, >, " e ' vengono trasformati rispettivamente in: &, <, >, " e ' i quali vengono interpretati dai parser HTML e mostrati come se fossero caratteri normali;
2. **Sanitizzazione dell'input**: utilizzare librerie come HTML sanitization che si occupano di convertire l'input ed impedire l'attacco XSS;

3. **Disabilitare gli script**: consiste nell'impedire l'esecuzione di script all'interno di una pagina;
4. **Utilizzare framework immuni a XSS.**

Durante lo sviluppo dell'applicazione abbiamo deciso di utilizzare due diverse tecniche per proteggere l'applicazione da attacchi XSS:

- A livello backend abbiamo utilizzato la dipendenza "helmet" che mette in sicurezza l'applicazione da diverse vulnerabilità, tra cui XSS, impostando alcuni speciali header HTTP nelle richieste scambiate tra client e server.
- A livello frontend abbiamo utilizzato React che è immune ad XSS, infatti, esegue automaticamente l'escaping di ogni variabile utilizzata nelle componenti delle pagine.

3.2.2 CSRF

Il Cross-site request forgery è ad oggi una delle vulnerabilità più comuni nel web e avviene quando un utente interagisce con un sito malevolo il quale esegue delle richieste spacciandosi per l'utente originale verso un altro sito.

Un esempio di attacco CSRF è il seguente:

1. L'utente si connette ad un sito dannoso e compila un form.
2. Una volta che il form è compilato il sito dannoso invia una richiesta ad un altro sito vulnerabile a CSRF su cui l'utente è autenticato.
3. Siccome la richiesta è inviata dal browser questo inserirà il cookie di autenticazione all'interno della richiesta.
4. Il sito vulnerabile a CSRF eseguirà la richiesta credendo questa sia stata effettivamente eseguita dall'utente originale.

Esistono diversi metodi per proteggere l'applicazione web da attacchi XSS quali:

1. **Synchronizer Token Pattern**: il server ad ogni richiesta genera un token unico, casuale e segreto chiamato STP che viene ricevuto dal client e inserito in ogni form HTML.

Quando l'utente compila il form, il browser inserisce all'interno della richiesta il token STP che viene convalidato dal server.

2. **Cookie-to-header token**: il server quando viene visitato crea un cookie di sessione contenente un token di verifica e lo invia al client, il browser inserirà il cookie in ogni successiva richiesta.

   Quando il server riceve una richiesta legge il cookie e ne estrae il token effettuando una convalida.

3. **SameSite Cookie**: il cookie di autenticazione viene inserito solo nelle richieste provenienti dal sito stesso e non da altri siti esterni.

Durante lo sviluppo della nostra applicazione non è stato necessario utilizzare nessuna di queste misure, visto che React è immune ad XSS è possibile salvare il token all'interno del localStorage il quale essendo domain specific non permette a siti esterni l'accesso in lettura e scrittura rendendo inefficace l'attacco CSRF.

3.2.3 Vulnerabilità delle dipendenze

Le dipendenze utilizzate dall'applicazione web per il suo funzionamento spesso celano delle vulnerabilità che se non accuratamente analizzate possono dare vita ad attacchi in grado di minare la sicurezza degli utenti o il servizio offerto stesso.

Quando si sviluppa un'applicazione in Node viene creato, al momento dell'inizializzazione del progetto, attraverso il comando `npm init` il file package.json che contiene dei metadati utili per definire la versione dell'applicazione e delle sue dipendenze. Ad ogni successiva installazione, attraverso il comando `npm install <dependency_name>` viene aggiunta una nuova clausola all'interno del file package.json che definisce la versione installata della dipendenza.

Al fine di mantenere l'applicazione sicura è necessario effettuare degli audit delle dipendenze installate e qualora vengano mostrate vulnerabilità è fondamentale effettuare un aggiornamento immediato delle dipendenze.

Per realizzare un audit Node mette a disposizione il comando `npm audit` il quale si occupa di effettuare una scansione delle dipendenze installate leggendo il file package.json ed effettuare una ricerca per ogni dipendenza se la versione attualmente utilizzata è vulnerabile, in tal caso, offre dei suggerimenti su cosa fare per eliminare le vulnerabilità.

Un altro strumento esterno utile per la rilevazione ed eliminazione delle falle di sicurezza all'interno di un'applicazione web è snyk.io il quale non solo controlla le dipendenze ma esegue scansioni tramite uso di analisi statica sul codice del programma sviluppato, evidenziando eventuali problematiche.

3.2.4 Crittografia

La comunicazione tra client e server deve essere necessariamente crittografata, in quanto, se così non fosse, attraverso l'utilizzo di programmi come Wireshark e Mitmproxy sarebbe possibile effettuare attacchi di tipo sniffing & spoofing, questi, consentono di leggere e modificare il traffico all'interno di una rete, pertanto, sarebbe possibile per utenti malevoli ottenere le credenziali di accesso alla piattaforma web di altre persone che condividono la stessa connessione ad Internet.

Per ottenere una connessione sicura tra client e server nelle applicazioni web è necessario ricorrere al protocollo HTTPS il quale sfruttando la crittografia offerta da SSL o TLS riesce ad impedire la lettura dei pacchetti trasmessi.

I protocolli di crittografia SSL e TLS funzionano nella seguente maniera:

1. Negoziazione tra i due comunicanti su l'algoritmo di crittografia da utilizzare.
2. Scambio delle chiavi attraverso il protocollo RSA e autenticazione.
3. Cifratura simmetrica e autenticazione dei messaggi.

Per utilizzare il protocollo di crittografia è necessario essere in possesso di un certificato rilasciato da una Certificate Authority, ovvero un soggetto pubblico o privato in grado di emettere un certificato digitale tramite una procedura di certificazione che segue standard internazionali.

Una volta ottenuto il certificato è necessario installarlo sul server per usufruire della connessione crittografata.

Durante lo sviluppo dell'applicazione abbiamo generato il certificato attraverso OpenSSL ottenendo i file key.pem e cert.pem. In seguito, abbiamo installato il certificato all'interno dell'applicazione web modificando il file app.js nella seguente maniera:

```
const server = https.createServer({

    key: fs.readFileSync(path.join(__dirname, 'key.pem')),

    cert: fs.readFileSync(path.join(__dirname, 'cert.pem'))

}, app);
```

3.2.5 DoS e DDoS

Un inadeguato design di sistema può compromettere le performance del servizio offerto ed esporre la piattaforma a maggiori rischi di Denial of Service.

Il DoS (Denial of Service) è un attacco ad un sistema informatico che si distingue in diverse forme:

1. **DoS da singolo host**: il DoS da singolo host è un insieme di attacchi, potenzialmente rintracciabili, atti a disturbare il servizio offerto, le varie tipologie di DoS da singolo host sono:
    1.1. Syn-flood: consiste nel sommergere un server di pacchetti SYN che hanno lo scopo di aprire tante connessioni, occupando così tutte le risorse del server per gestirle.
    1.2. Smurf: consiste nell'invio di un numero moderato di pacchetti che, tuttavia, andranno a moltiplicarsi prima di raggiungere la destinazione finale.
    1.3. RUDY: bersaglia le applicazioni web inviando richieste POST con contenuto variabile, mantenendo così tutte le sessioni aperte con il server interrotte.
2. **DoS distribuito**: è un attacco che sfrutta molte macchine infettate da virus definite "zombie" le tecniche utilizzate per disturbare il servizio dell'applicazione sono identiche a quelle del DoS a singolo host, tuttavia risultano molto amplificate attraverso il numero variabile di macchine infettate.

Non esiste una effettiva soluzione generale al problema del DDoS in quanto non è arginabile tramite policy statiche tuttavia esistono diversi meccanismi per mitigarne l'effetto:

1. **Load balancing**: utilizzare un load balancer e distribuire il traffico su più server può rendere il sistema più robusto ad attacchi DoS.
2. **Impostare accuratamente il firewall**: è molto importante ridurre il numero di porte aperte ai soli servizi accessibili all'esterno e possibilmente bloccare gli indirizzi ip segnalati come dannosi.
3. **Utilizzare servizi di protezione cloud**: servizi di protezione da attacchi DoS come Cloudflare permettono di:
    a. Mantenere il servizio attivo quando l'applicazione è sotto attacco, attraverso il caching delle pagine web (anche dinamiche).
    b. Dirottare il traffico proveniente dal DoS verso i cosiddetti "blackhole" che hanno lo scopo di eliminare i pacchetti.
4. **Distribuzione e replicazione della infrastruttura**: la distribuzione e replicazione dell'infrastruttura permette di:
    a. Arginare l'attacco DoS verso un sottoinsieme delle infrastrutture utilizzate dall'applicazione web.
    b. Distribuire il traffico attraverso le varie infrastrutture rendendo il servizio più resistente al DoS.

Capitolo 4: Performance e Scalabilità

In questo capitolo discutiamo sulle performance e sulla scalabilità dell'applicazione web, andremo a vedere per i diversi servizi offerti: quale architettura risulta più appropriata da utilizzare, come scalare l'infrastruttura ed infine metteremo a confronto le performance dei diversi modelli di rete.

4.1 Architettura di rete

All'inizio dello sviluppo di un'applicazione è fondamentale definire l'architettura di rete di cui la piattaforma finale farà uso.

Durante lo sviluppo del nostro progetto abbiamo deciso di utilizzare due particolari architetture di rete per riuscire a soddisfare i requisiti imposti dal progetto. In particolare, abbiamo utilizzato le architetture: client-server e peer-to-peer.

I fattori da tenere in considerazione durante la scelta di un'architettura di rete sono i seguenti:

1. **Performance**: deve offrire alta velocità di comunicazione tra i client.
2. **Scalabilità**: deve essere facile da estendere.
3. **Rapidità di sviluppo**: deve essere facile da sviluppare.
4. **Costi**: deve avere costi ridotti.
5. **Sicurezza**: i dati trasmessi devono essere protetti.

4.1.1 Architettura client-server

L'architettura di rete client-server prevede che la comunicazione tra diversi client passi attraverso un server il quale si occupa di scambiare i messaggi tra le parti, pertanto, il server gestisce le risorse di cui gli utenti fanno uso.

L'architettura client-server presenta dei vantaggi e svantaggi riassunti nella seguente tabella:

Vantaggi	Svantaggi
Ottime performance che migliorano più grande la rete diventa	Difficile scalabilità: può risultare complicato estendere l'infrastruttura di rete, in quanto richiede di aggiungere e configurare nuovi server.
Maggiore sicurezza in quanto il server si occupa di controllare i dati trasmessi e convalidarli, inoltre la comunicazione tra le due parti è indiretta, questo garantisce l'anonimato.	Richiede maggiore tempo per lo sviluppo a causa di dover necessariamente configurare i server e definire i protocolli utilizzati dall'applicazione.
	Risulta molto costosa a causa della necessità di acquistare nuovi server.

L'architettura client-server risulta adatta ad applicazioni che offrono servizi specialmente nei modelli request-response.

Per motivi di sicurezza e necessità di performance è stata utilizzata l'architettura client-server per la gestione del gioco online.

4.1.2 Architettura peer-to-peer

L'architettura di rete peer-to-peer permette la comunicazione diretta tra i client detti "peer".

L'architettura peer-to-peer presenta dei vantaggi e svantaggi riassunti nella seguente tabella:

Vantaggi	Svantaggi
La rete è facilmente scalabile e non richiede l'aggiunta di nessuna componente.	A seconda della tipologia di servizio le performance possono migliorare o peggiorare con l'aumento dei peer.
Lo sviluppo risulta rapido in quanto basta definire il protocollo di comunicazione ed integrarlo nell'applicazione.	La sicurezza dei dati è ottenibile attraverso la cifratura end-to-end, tuttavia non essendoci nessun server nella comunicazione le operazioni eseguite all'interno della rete non vengono convalidate, questo rende impossibile creare servizi sicuri.
L'architettura non presenta costi, in quanto	

non è necessario acquistare nessun dispositivo.	

L'architettura peer-to-peer risulta ottima per lo scambio di dati, ed in particolare, nei sistemi di condivisione di file permette di aumentare le performance con l'aumento dei peer all'interno della rete. Tuttavia, nei sistemi di condivisione audio-video real time, la velocità offerta tende a diminuire con l'aumento dei peer, questo poiché ogni utente deve inviare gli stessi dati verso più connessioni, pertanto, in caso di comunicazione tra tanti utenti, è consigliato adottare un sistema ibrido client-server e peer-to-peer che permette l'upgrade da connessione diretta tra gli utenti a connessione indiretta tramite uso di server.

Per motivi di rapidità di sviluppo e costo abbiamo deciso di utilizzare l'architettura peer-to-peer per la comunicazione audio-video realtime attraverso la libreria WebRTC che permette, qualora necessario, l'upgrade ad una comunicazione client-server.

4.2 Scalabilità

Con l'aumentare del numero di utenti che usufruiscono dell'applicazione sviluppata aumenta anche il numero di richieste al server e di conseguenza il carico computazionale da gestire, per questo risulta importante riuscire ad estendere l'infrastruttura della piattaforma, in modo da riuscire a sopportare la crescita della utenza.

4.2.1 Dislocazione geografica

Una piattaforma può arricchirsi di utenti provenienti da diverse parti del mondo, questo, nel caso in cui il servizio sia in esecuzione in un unico centro, può causare latenza nella comunicazione tra client e server a seconda della posizione geografica dell'utente.

Per risolvere la problematica di latenza dovuta alla distanza tra il client ed il server è necessario e sufficiente replicare la piattaforma in diversi centri con posizioni geografiche diverse.

4.2.2 Espansione della infrastruttura attuale

Quando una infrastruttura informatica riceve un incremento delle richieste da parte dei client diventa necessario aumentare le performance, questo può essere ottenuto in due maniere diverse:

1. Aumentando il numero di server all'interno della stessa infrastruttura andando quindi a suddividere il carico di richieste tra più computer, questo però comporta maggiori costi in quanto richiede l'acquisto di interi nuovi dispositivi;
2. Migliorando le componenti hardware/software dei server attuali, questo comporta minori costi rispetto alla prima opzione tuttavia può causare dei downtime dovuti all'installazione e configurazione delle nuovi componenti all'interno delle macchine.

Conclusione

Il progetto su cui questa tesi è stata redatta ha dato modo di sperimentarmi sulla pianificazione e sviluppo delle applicazioni web, permettendomi di ottenere una più completa conoscenza di tecnologie utilizzate a livello industriale.

La tesi riassume in sé i vari aspetti da considerare durante lo sviluppo di una piattaforma web evidenziando le pratiche migliori, le difficoltà e le soluzioni nell'implementazione di funzionalità.

Abbiamo visto quanto semplice risulta commettere errori che possono introdurre vulnerabilità e quanto complesso deve essere il processo di analisi del codice e manutenzione dell'applicazione al fine di garantire un servizio ottimale e sicuro.

Il mondo del web è in costante evoluzione grazie al continuo sviluppo di nuove tecnologie in grado di soddisfare i bisogni della società.

Ci siamo soffermati molto sulla rivoluzione attuale portata dalla Web 2.0 che ancora tutt'oggi presenta novità di impatto globale. Tuttavia, è bene realizzare anche quale sarà il futuro dell'Internet come lo conosciamo, infatti, negli ultimi anni, si è sentito molto parlare della Web 3.0 composta da BlockChain e intelligenza artificiale. Al momento risulta difficile comprendere a pieno quali saranno gli effetti che queste due tecnologie avranno sul nostro pianeta e sulla società.

Bibliografia

[1] Ashley Waters

Top 50 Cybersecurity Statistics, Figures and Facts, 2022

[2] PurpleSec

Cyber Security Statistics The Ultimate List Of Stats Data, & Trends For 2022

[3] *Harris Poll Online Security Survey Google, 2019*

[4] Fortinet

What is an Authentication token?

[5] Matt Pollicove

Ultimate Guide to Token-based Authentication, 2021

[6] Auth0

Token Based Authentication Made Easy

[7] Riccardo Focardi

Corso di Sicurezza, Università Ca' Foscari, 2021

[8] Talha Waseem

Web Application: Choosing the Right Technology Stack Before You Start, 2020

[9] Abhijeet Desai

System Design Concepts & Components

[10] Nedim Maric

SQL Injection in MongoDB: Examples and Prevention, 2021

[11] Fabrizio Romano

Corso di Laboratorio di Amministrazione di Sistema, Università Ca' Foscari 2021

[12] Andrew S. Tanenbaum, David J. Wetherall

Reti di calcolatori quinta edizione 2018

[13] *Chess Rules: A Quick Summary of the Rules of Chess*