

UNIVERSITÀ DEGLI STUDI DI CAMERINO

SCUOLA DI SCIENZE E TECNOLOGIE

Corso di Laurea in Informatica

(classe L-31)



**SISTEMA DI GESTIONE UTENTI
PER IL SERVER HACK TO WIN**

Tesi di laurea triennale

Relatore:

Prof. Fausto Marcantoni

Laureando:

Edoardo Piergentili

ANNO ACCADEMICO 2017-2018

Indice

1	Introduzione.....	3
1.1	Che cos'è Hack To Win.....	3
1.2	Obiettivi della tesi.....	3
1.3	Motivazioni della Scelta degli Strumenti	9
2	Strumenti Utilizzati.....	11
2.1	Linguaggi di Programmazione	11
2.1.1	JavaScript[4]	11
2.1.2	Node.js[5]	11
2.1.3	Nodemailer[7].....	13
2.1.4	HTML[8].....	14
2.1.5	CSS[9]	14
2.1.6	Bootstrap[10].....	15
2.1.7	JQuery[11].....	15
2.1.8	Template Ejs[12].....	15
2.1.9	Express.js[13].....	16
2.1.10	Angular.js[14]	16
2.1.11	Passport.js[15]	16
2.1.12	Socket.IO[16]	16
2.1.13	Tiled Map Editor[17]	17
2.1.14	Redis[18].....	18
2.2	Database	19
2.2.1	RethinkDB[19].....	19
2.2.2	Requisiti del Database	19
2.2.3	Diagramma ER	20
2.2.4	Schema Logico	21
2.2.5	Query.....	21
2.3	CAPTCHA	27
2.3.1	Test di Turing	28
2.3.2	Test di Turing Inverso e CAPTCHA	29
2.3.3	Evoluzione dei CAPTCHA	30
3	Struttura del Sito e Gestione Utenti.....	32
3.1	Pagina di Login.....	33
3.2	Pagina di Password Dimenticata	34

3.3	Pagina di Modifica Password	35
3.4	Pagina di Registrazione.....	37
3.5	Pagina Admin	40
3.5.1	Scheda Utenti.....	41
3.5.2	Scheda Manuali	42
3.5.3	Scheda Email	43
4	Gestione Biblioteca	44
4.1	Descrizione	44
4.2	Finestra Manuali	45
5	Manuale.....	47
5.1	Installazione	47
5.2	Avvio Applicazione	47
5.3	Creazione Mappe.....	47
5.4	Aggiunta Manuali	48
6	Conclusioni e Sviluppi Futuri	50
7	Ringraziamenti.....	51
8	Riferimenti Bibliografici e Sitografici	53
8.1	Bibliografia	53
8.2	Sitografia	53

1 Introduzione

1.1 Che cos'è Hack To Win

Hack To Win è una piattaforma, creata in una precedente tesi di laurea[1], dove gli utenti registrati possono partecipare a dei giochi didattici, chiamati *CTF* (*Capture The Flag*). Nei *CTF* gli utenti dovranno cercare vulnerabilità in sistemi e software messi a disposizione dagli organizzatori della competizione al fine di sfruttarle e di collezionare le varie *flag* (bandiere) nascoste sul sistema bersaglio per passare al livello successivo.

1.2 Obiettivi della tesi

Gli obiettivi che mi sono stati richiesti per la redazione della tesi possono essere riassunti in due punti:

1. Implementare una più accurata gestione degli utenti;
2. Creare una nuova zona del *Capture The Flag* dove l'utente possa trovare dei manuali utili per approfondire argomenti riguardanti la sicurezza informatica, il mondo del networking e dei sistemi operativi.

Per quanto riguarda la gestione degli utenti, la richiesta è stata quella di implementare un sistema di registrazione che risulti più sicuro e funzionale e di gestire il recupero della password dell'utente. La registrazione, oltre a controllare la presenza o meno di username ed email sul database, dovrà contenere un sistema di sicurezza noto come *CAPTCHA* e, qualora fosse andato tutto a buon fine, l'invio di una email contenente un link il quale permetterà all'utente di verificare l'account e, quindi, di accedere alla piattaforma. L'utente smemorato avrà anche la possibilità di recuperare la password: basterà inserire l'email con cui si è registrato ed il sistema, una volta verificata l'effettiva presenza dell'utente all'interno del database, invierà una mail con il link che lo reindirizzerà alla pagina adibita alla modifica della password. Terminato il processo, il sistema invierà una nuova email per notificare l'avvenuta modifica della password. È stato inoltre richiesto di incrementare le funzionalità dell'utente Admin, in modo da gestire completamente gli utenti iscritti. L'Admin, oltre alle funzionalità precedentemente implementate dai miei colleghi, cioè la gestione delle mappe e delle missioni, potrà, quindi:

- Inviare email a uno o più utenti direttamente dalla piattaforma;
- Gestire l'account con cui vengono inviate le email di verifica account, recupero password, ecc.
- Gestire i manuali da inserire/eliminare all'interno del gioco.

Per quanto riguarda, invece, il secondo punto, andrò a creare una nuova mappa all'interno del gioco, che chiamerò Biblioteca, dove l'utente, interagendo con le

varie librerie al suo interno, potrà leggere o scaricare dei manuali caricati dall'amministratore del sistema.

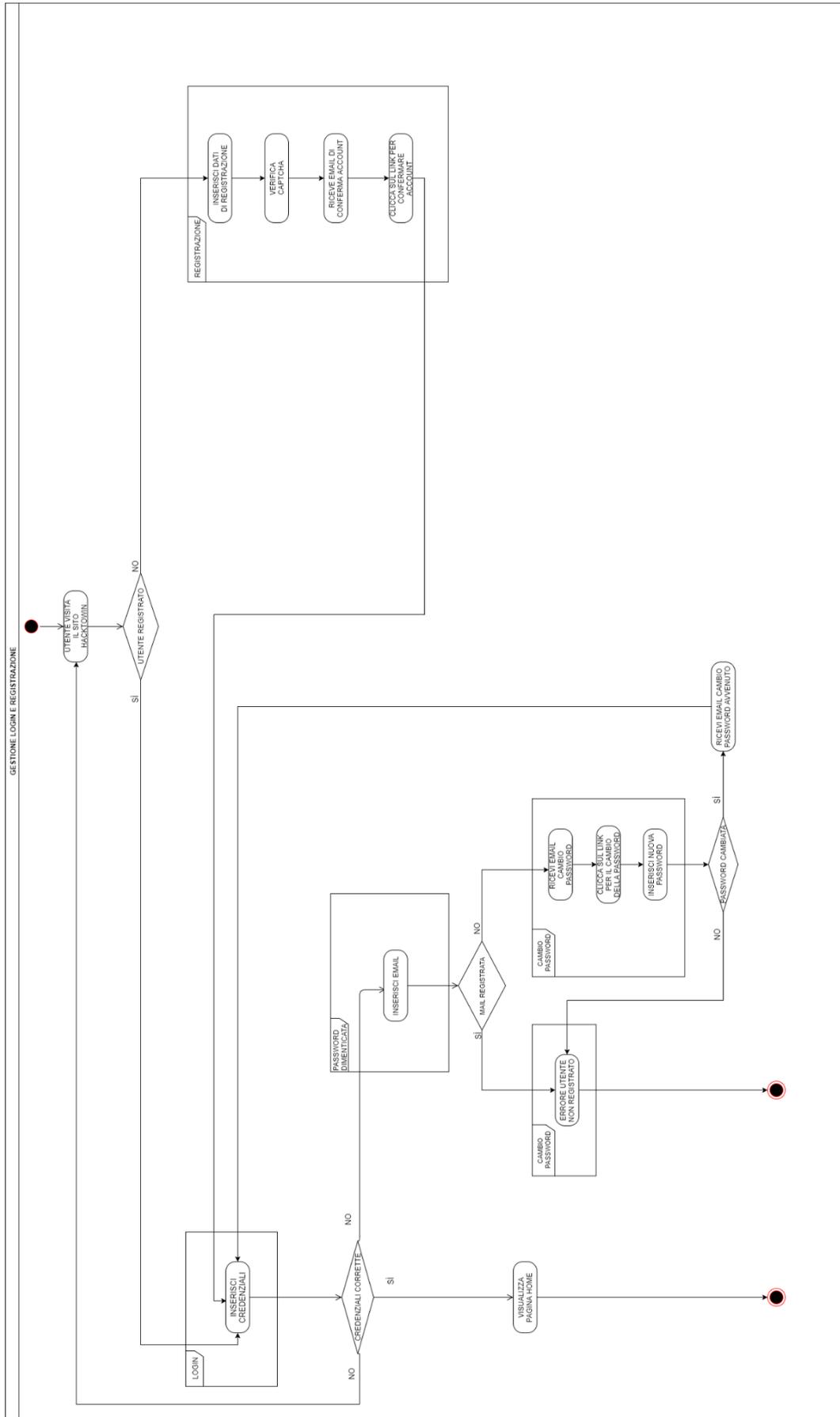


Figura 1 Activity Diagram Login e Registrazione

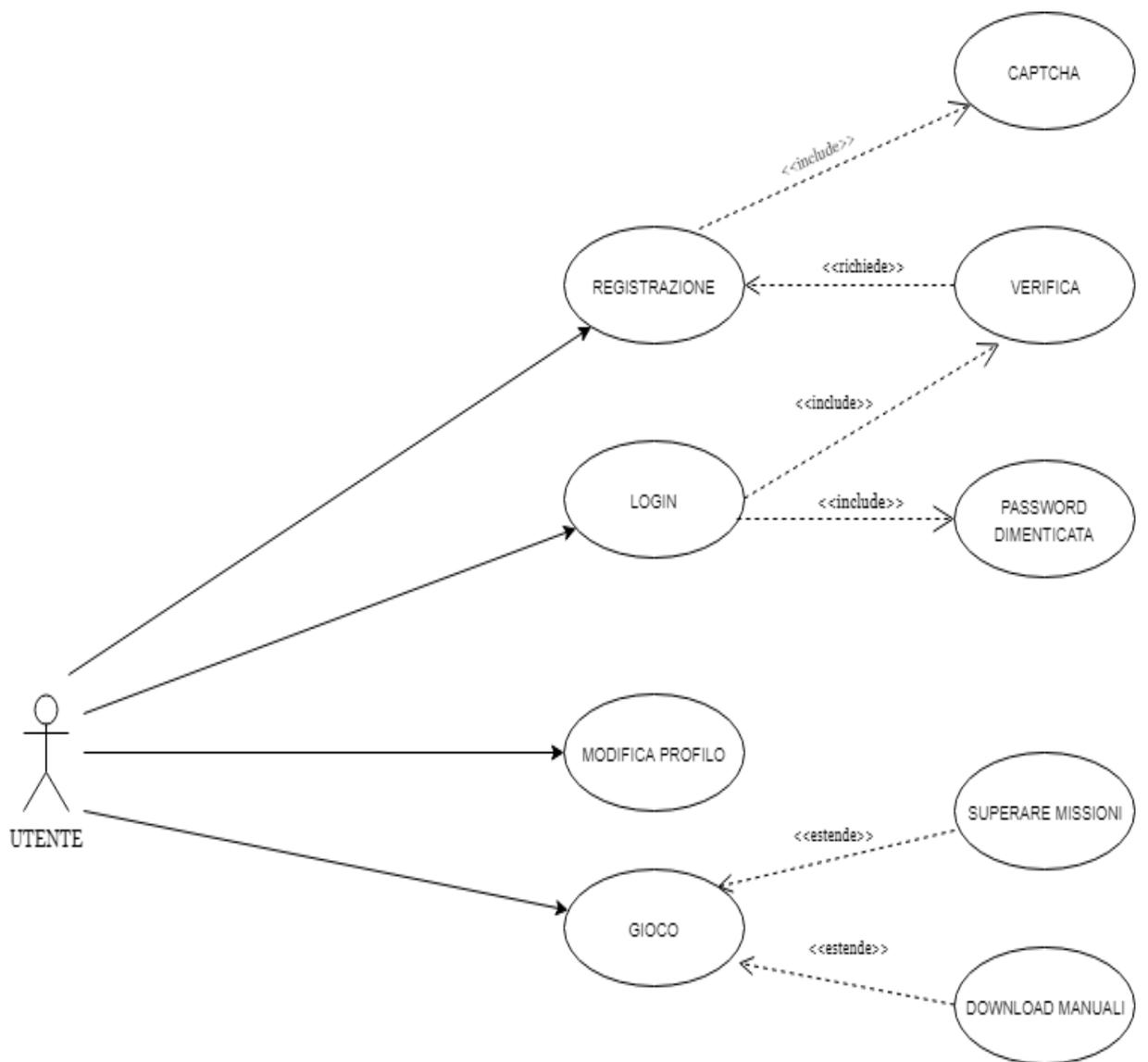


Figura 2 Use Case Diagram Utente

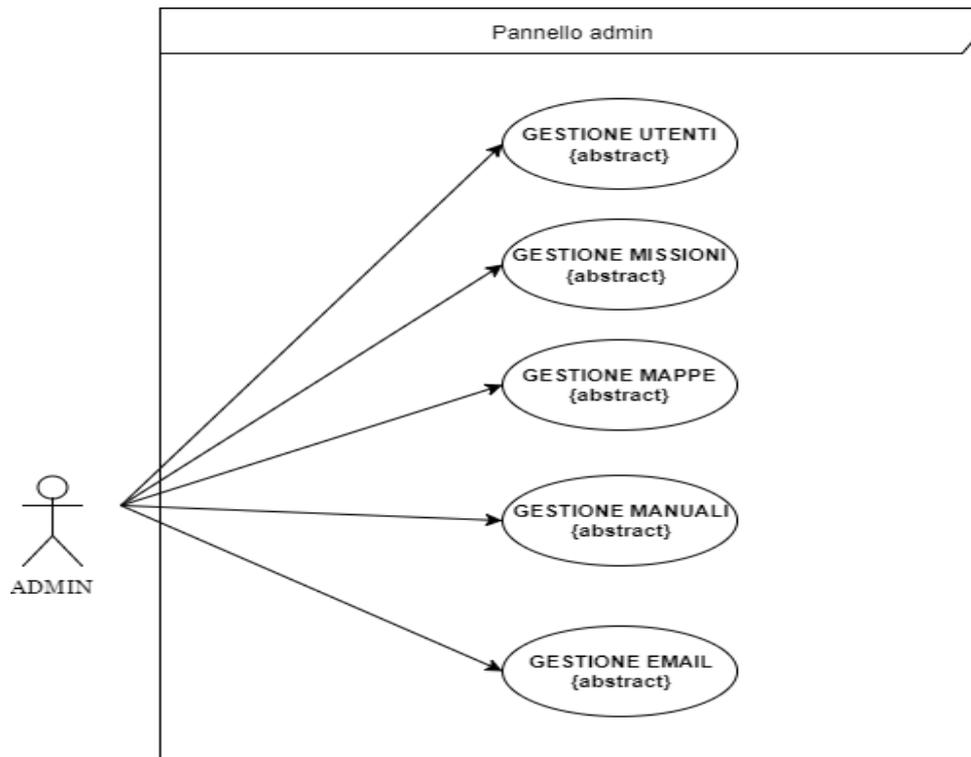


Figura 3 Use Case Admin

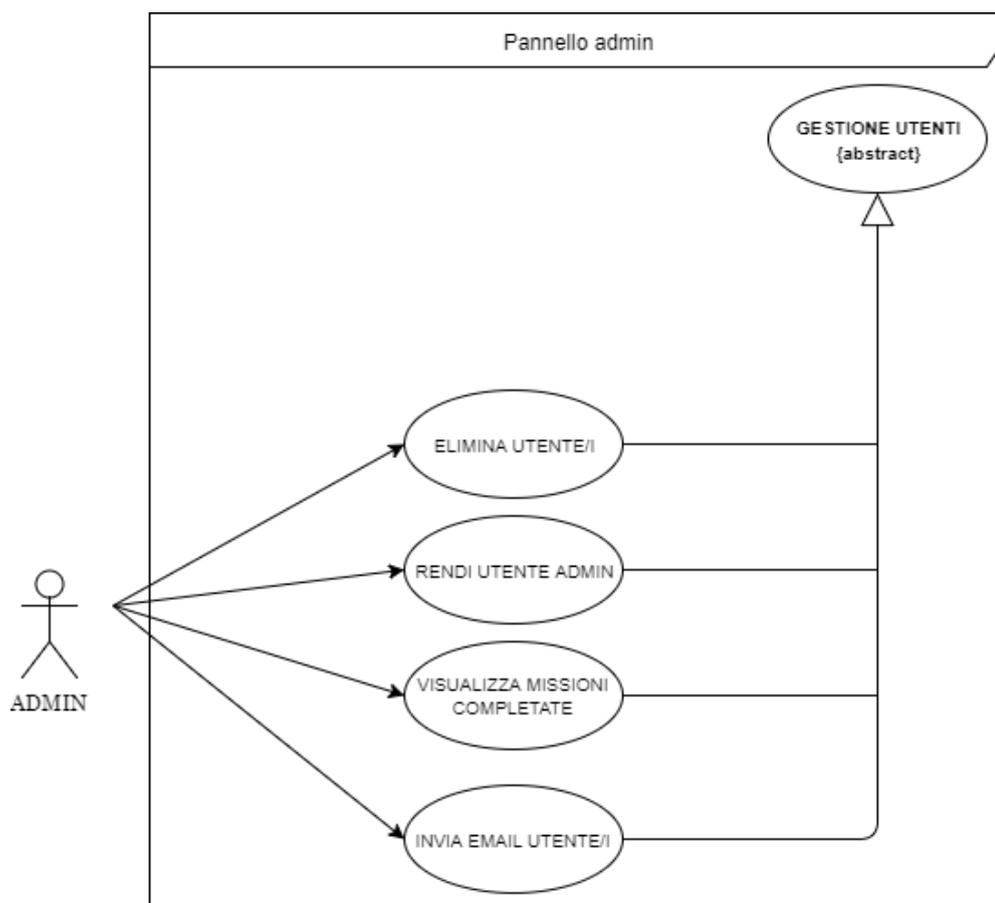


Figura 4 Use Case Diagram Gestione Utenti

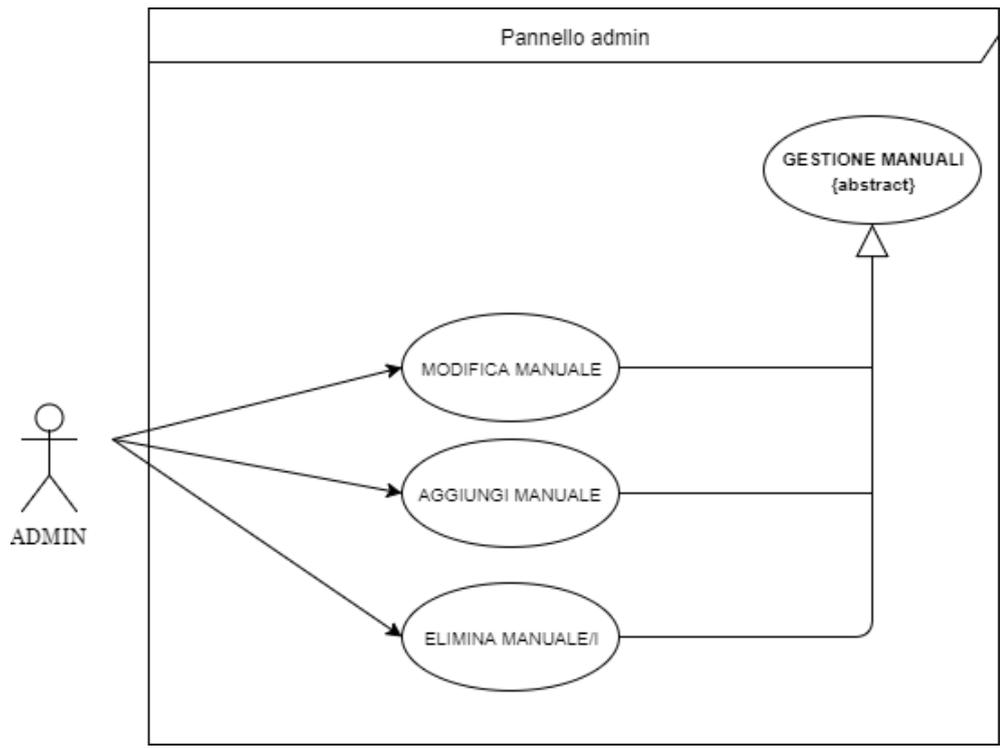


Figura 5 Use Case Diagram Gestione Manuali

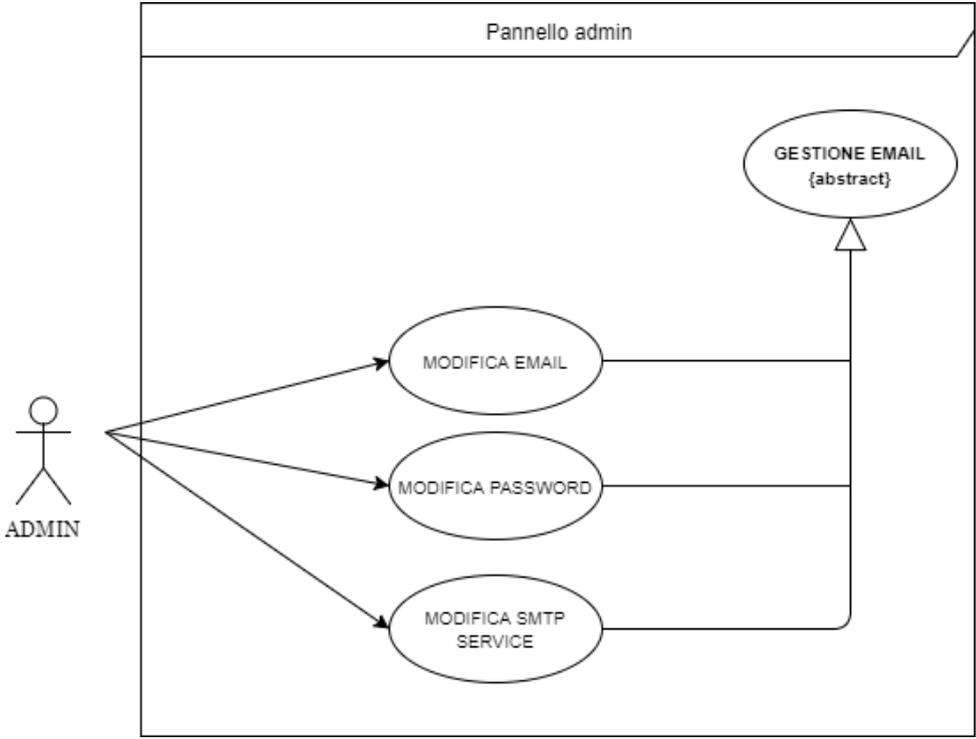


Figura 6 Use Case Diagram Gestione Email

1.3 Motivazioni della Scelta degli Strumenti

In questo paragrafo farò una breve panoramica sul perché ho scelto determinati strumenti per la realizzazione degli obiettivi che mi sono stati prefissati, per poi scendere nel dettaglio nei capitoli successivi.

Seguendo la logica di programmazione utilizzata dai miei colleghi che hanno precedentemente realizzato questa piattaforma, per realizzare le modifiche al Web Server dell'applicazione è stato usato Node.js con l'ausilio del *framework*, cioè un'architettura logica di supporto su cui un software può essere progettato e realizzato, Express.js e il database che contiene i dati dell'applicazione è stato implementato con RethinkDB. Le modifiche *front-end* del sito sono state apportate grazie all'ausilio dell'HTML e CSS, mentre gli effetti grafici più complessi sono stati implementati grazie al framework Bootstrap. Tramite jQuery sono stati implementati i metodi e le funzioni per gestire al meglio le manipolazioni degli elementi e dei loro attributi nelle pagine HTML. Dato il principale utilizzo del linguaggio Node.js nell'applicazione, per quanto riguarda la gestione dell'invio delle mail agli utenti iscritti e la gestione dell'invio delle mail in fase di registrazione e di recupero password, è stato utilizzato un modulo di Node.js, *Nodemailer*, il quale, *customizzato*¹ a seconda delle esigenze richieste, mi ha permesso di creare agevolmente una completa e sicura gestione dei servizi precedentemente citati.

Per la fase di registrazione, oltre all'invio automatico dell'email contenente il link per confermare l'account, generata dal sistema grazie a Nodemailer, per la verifica dell'account, ho implementato anche due sistemi di test *CAPTCHA*: il primo consiste nel riuscire a decifrare una stringa alfanumerica distorta e colorata, mentre il secondo è il test *reCAPTCHA* distribuito gratuitamente da Google, dove l'utente dovrà cliccare su una *checkbox*² con scritto "Io non sono un robot" in modo da garantire al sistema di non essere un bot; qualora il sistema non riuscisse comunque a verificarlo, apparirà un ulteriore test dove bisognerà cliccare su delle determinate immagini. Il primo test è stato implementato ed impostato installando, tramite il gestore dei pacchetti di Node.js, *npm*, il pacchetto *svg-captcha* e salvando la temporaneamente la sessione con l'id univoco dell'immagine *svg* (*Scalable Vector Graphics*, tecnologia in grado di visualizzare oggetti di grafica vettoriale e di gestire immagini scalabili dimensionalmente) generato sul database Redis; mentre il secondo test è stato gestito grazie alle *API* (*Application Programming Interface*) gratuite di Google adattate al linguaggio utilizzato dall'applicazione. Li ho implementati entrambi nell'applicazione per poter imparare le loro funzionalità e per capire se fosse stato possibile riuscire a farli funzionare contemporaneamente senza conflitti. L'invio e la gestione dei dati inseriti nella pagina di Registrazione è stata affidata al

¹ **Customizzare:** Adattare un prodotto, un bene o un servizio, mediante appositi interventi di personalizzazione, alle esigenze e alle aspettative del cliente.

² **Checkbox:** Controllo grafico con cui l'utente può effettuare selezioni multiple.

Template Ejs. Un *template* è un modello predefinito che consente di creare o inserire contenuti di diverso tipo in un documento o in una pagina web. L'autenticazione in fase di login è affidata al *middleware* Passport.js, ossia un insieme di software che fungono da intermediari fra strutture e programmi informatici, mentre la comunicazione tra client e server viene gestita tramite la libreria JavaScript Socket.IO. Il pannello dell'amministratore è stato modificato tramite il framework Angular.js. La creazione della mappa contenente la biblioteca è stata possibile grazie ad un *tool* esterno: Tiled Map Editor.

2 Strumenti Utilizzati

In questo capitolo verranno descritti dettagliatamente gli strumenti che ho indicato nel paragrafo 1.3. Sarà diviso in tre paragrafi: il primo includerà tutti i linguaggi di programmazione coinvolti nella progettazione dell'applicazione, il secondo sarà incentrato sul database che contiene i dati dell'applicazione, mentre nel terzo paragrafo tratterò l'argomento *CAPTCHA*.

2.1 Linguaggi di Programmazione

2.1.1 JavaScript[4]

JavaScript è un linguaggio di *scripting* orientato agli oggetti e agli eventi. Fu originariamente sviluppato da Brendan Eich della *Netscape Communications* con il nome di Mochan e successivamente di LiveScript, ma è stato rinominato "JavaScript" ed è stato formalizzato con una sintassi più vicina a quella di Java. Viene tipicamente utilizzato nella programmazione Web lato client per la creazione di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati, a loro volta, da varie azioni eseguite dall'utente sulla pagina web in uso. Uno dei vantaggi principali di JavaScript è quello di essere un linguaggio interpretato: il codice JavaScript contenuto, ad esempio, in una pagina web viene portato in memoria ed eseguito dall'interprete del browser utilizzato dall'utente. Ciò permette al web server di non essere sovraccaricato a causa delle richieste del client, anche a fronte di script particolarmente complessi. Contrariamente, a seconda della complessità e della grandezza dello script, il tempo dello scaricamento può risultare abbastanza lungo.

2.1.2 Node.js[5]

Node.js è un *framework* che serve a realizzare applicazioni Web in JavaScript, permettendoci di utilizzarlo anche per la programmazione Web lato server, dato che, come detto sopra, JavaScript viene utilizzato per la scrittura "*client-side*". La piattaforma è costruita sul JavaScript Engine V8, che è il *runtime* di Google utilizzato anche da Chrome e disponibile sulle principali piattaforme, anche se risulta maggiormente performante su sistemi operative *UNIX-like*. Nella maggior parte dei *web server* tradizionali, per ogni richiesta o connessione in entrata, si genera un nuovo *thread* di esecuzione per gestire la richiesta. Ora, si immagini di avere milioni di richieste: ciò comporta la creazione di altrettanti *thread* portando, inevitabilmente, ad un sovraccarico del server. Nelle applicazioni moderne, si devono gestire dati in tempo reale e il compito principale dell'applicazione è quello di prendere un dato da un database o da un *filesystem* e inviarlo ai destinatari nel minor tempo possibile. Utilizzando un server tradizionale, avremo che, ad ogni richiesta, verrebbe generato un *thread* il quale

dovrebbe interrogare il database e mettersi in attesa della risposta, per un tempo indefinito, senza fare niente. Node.js, a differenza dei server tradizionali, è *single thread* e gestisce le richieste in maniera asincrona e non bloccante; utilizzando, perciò, una programmazione ad eventi o *event-driven*.

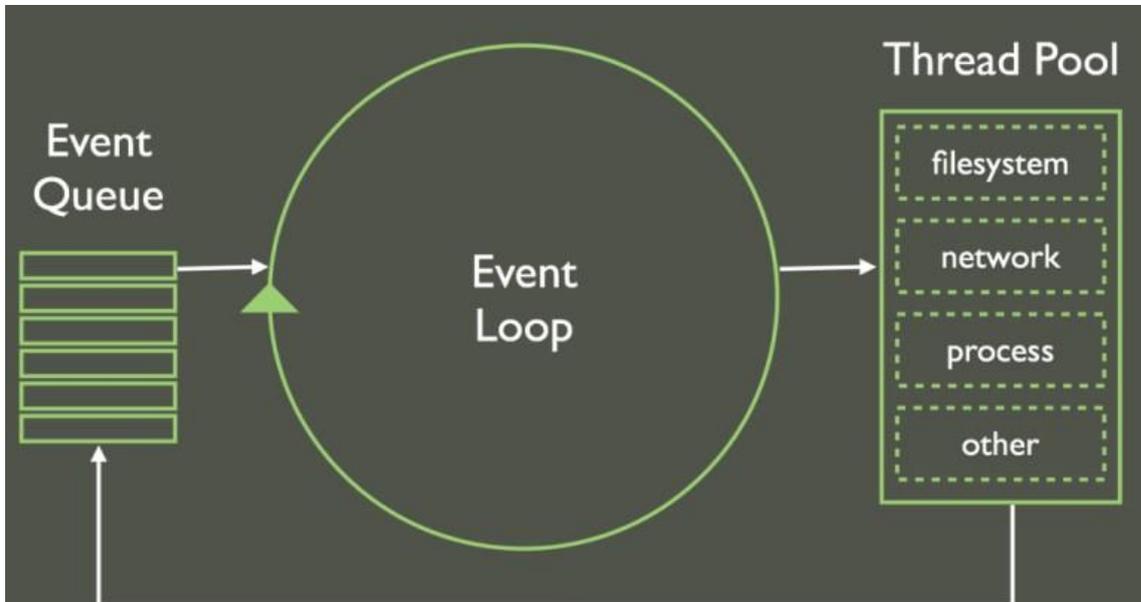


Figura 7 Programmazione event-driven

Il singolo *thread* di Node si mette in attesa di richieste: una volta ricevuta una richiesta, crea una funzione di *callback* associato ad essa ed interroga la risorsa interessata. Subito dopo si rimette in attesa di altre richieste senza aspettare la risposta della risorsa. Una volta che la risorsa ha terminato, notifica il comportamento inserendo un messaggio nella coda degli eventi (“*Event Queue*” nella Figura 7). Il *thread* si accorge dell’arrivo del messaggio e fa scattare la funzione di *callback* che si occuperà di restituire una risposta al client.

Node.js è, quindi, una buona scelta per le applicazioni che devono elaborare un elevato volume di messaggi brevi che richiedono una bassa latenza, chiamate applicazioni real-time (RTA). Alcuni casi di utilizzo possono essere i seguenti:

- Applicazioni collaborative
- Applicazioni e-commerce
- Applicazioni per la comunicazione istantanea: messaggi, videoconferenze e chiamate.

Node.js, in questi casi, è molto efficiente e facilita la gestione di più richieste client. Inoltre, la possibilità di sviluppare tutta l’applicazione in JavaScript, ci permette di lavorare con uno stesso formato di dati (JSON) e di riutilizzare alcuni pezzi di codice sia nel Frontend che nel Backend. Il framework offre, infine, anche un gestore di pacchetti, npm, che, come possiamo apprendere dal sito ufficiale di Node.js, è il più grande ecosistema di librerie open source al mondo.

2.1.3 Nodemailer[7]

Nodemailer è un modulo per le applicazioni Node.js, che consente agli utenti di inviare e-mail. Il progetto è stato avviato nel 2010 quando non esisteva ancora un'opzione valida per inviare messaggi di posta elettronica; ad oggi è la soluzione maggiormente utilizzata dagli utenti di Node.js per ovviare a questa tipologia di problema. Per installarlo, è sufficiente utilizzare il gestore di pacchetti di Node.js, *npm*.

Le sue funzioni sono:

- Un modulo singolo con zero dipendenze: il codice è facilmente controllabile;
- Grande attenzione alla sicurezza;
- Supporto *Unicode* per l'utilizzo di qualsiasi carattere, incluse le *emoji*;
- Supporto per Windows: si può installare con *npm* su Windows, così come su qualsiasi altro modulo, non ci sono dipendenze compilate;
- Possibilità di utilizzare il contenuto HTML in alternativa al testo normale;
- Possibilità di aggiungere allegati ai messaggi;
- Invio sicuro delle e-mail tramite TLS/STARTTLS;
- Diversi metodi di trasporto oltre al supporto SMTP integrato;
- Firma dei messaggi con DKIM;
- Supporto *plugin* personalizzato per la manipolazione dei messaggi;
- Autenticazione OAuth2;
- Proxy per connessioni SMTP;
- Codice ES6: non più perdite di memoria involontarie;
- Account di prova generati automaticamente da *Ethereal.email*.

Requisiti:

- Node.js v6+.

Installazione:

```
1. npm install nodemailer --save
```

Per inviare e-mail, avremo bisogno di un oggetto *transporter*:

```
1. let transporter = nodemailer.createTransport(options[,defaults])
```

dove:

- **transporter**: è l'oggetto per l'invio delle e-mail;
- **options**: definisce i dati della connessione. Come opzioni generali abbiamo:
 - **port**: è la porta dove avviene la connessione (di default è settata a 587 se l'opzione *secure* è impostata a *false*, mentre è settata a 465 se *secure* è impostata a *true*);
 - **host**: è l'hostname o l'indirizzo IP a cui connettersi ('localhost' di default);

- **auth:** definisce i dati di autenticazione;
- **secure:** se impostato a true la connessione userà TLS (Transport Layer Security) quando si conatterà al server. Se impostato a false (di default), allora sarà utilizzato TLS se il server supporta l'estensione STARTTLS.
- **defaults:** definisce i valori predefiniti per le opzioni dell'e-mail.

Una volta configurato il transporter, possiamo inviare e-mail tramite il comando:

```
1. transporter.sendMail(data[, callback])
```

dove:

- **data:** definisce i contenuti della mail;
- **callback:** è una funzione di callback opzionale da eseguire una volta che il messaggio è stato consegnato o fallito
 - **err:** indica l'errore se l'invio della mail è fallito;
 - **info:** include il risultato: il formato esatto dipende dal meccanismo di trasporto utilizzato.

2.1.4 HTML[8]

L'**Hyper Text Markup Language** (letteralmente tradotto come linguaggio a marcatori per ipertesti), è il più importante linguaggio di markup per il web. Nasce alla fine degli anni '80 da Tim Berners-Lee al CERN di Ginevra. HTML è un linguaggio di pubblico dominio la cui sintassi è stabilita dal World Wide Web Consortium (W3C). L'HTML non è un linguaggio di programmazione, in quanto non prevede alcuna definizione di variabili, funzioni, strutture di controllo o strutture dati, ma è un linguaggio di markup, cioè descrive il contenuto, testuale e non, di una pagina web. Normalmente il programma che deve interpretare il codice HTML viene chiamato browser. Il browser effettua la richiesta di una pagina al server http che risponde con la pagina HTML. Il browser ha poi il compito di trasformare le istruzioni in HTML (in puro testo), in elementi grafici e ipertestuali. Una cosa molto importante è che l'HTML non è dinamico, ma bensì statico: ciò significa che se non si agisce direttamente nel codice HTML, esso rimarrà invariato nel tempo. L'unica possibilità che il codice cambi è dovuta a codice script che viene inserito in appositi *tag*.

2.1.5 CSS[9]

L'acronimo CSS sta per **Cascading Style Sheets** (fogli di stile a cascata) e designa un linguaggio di stile per i documenti web. I CSS istruiscono un browser o un altro programma utente su come il documento debba essere presentato all'utente, per esempio definendone i font, i colori, le immagini di sfondo, il layout, il posizionamento delle colonne o di altri elementi sulla pagina, ecc. La storia dei CSS procede su binari paralleli rispetto a quelli di HTML, di cui vuole

essere l'ideale complemento. Da sempre infatti, nelle intenzioni del W3C, HTML dovrebbe essere semplicemente visto come un linguaggio strutturale, alieno da qualunque scopo attinente la presentazione di un documento. Per questo obiettivo, ovvero arricchire l'aspetto visuale e la presentazione di una pagina, lo strumento designato sono appunto i CSS, il quale scopo principale è separare il contenuto dalla presentazione.

2.1.6 Bootstrap[10]

Bootstrap nasce nell'anno 2010 per opera degli sviluppatori Mark Otto e Jacob Thornton. Inizialmente si presentava come un progetto interno a Twitter, ma successivamente è diventato indipendente ed è, perciò, utilizzabile dagli sviluppatori di tutto il mondo come base per la realizzazione di interfacce web. Al momento è considerato il più efficiente e utilizzato framework per rendere i siti web *responsive*. La sua definizione esatta è "*HTML, CSS, and JS toolkit from Twitter*", ovvero una raccolta di strumenti grafici, stilistici ed impaginazione che permettono di avere a disposizione una gran quantità di funzionalità e di siti modificabili e adattabili a seconda delle nostre esigenze. Bootstrap è compatibile con tutte le ultime versioni dei principali browser e la sua funzione principale è il *responsive web design*, ovvero che il design delle pagine web è in grado di attenersi dinamicamente a seconda della grandezza e delle caratteristiche del dispositivo utilizzato. Esso si pone, perciò, come una libreria multi dispositivo e multiplatforma.

2.1.7 JQuery[11]

jQuery è una libreria JavaScript per applicazioni web. Nasce con l'obiettivo di semplificare la manipolazione, la selezione, la gestione degli eventi e l'animazione di elementi DOM in pagine HTML, nonché implementare funzionalità AJAX (*Asynchronous JavaScript and XML*). Le sue caratteristiche permettono agli sviluppatori JavaScript di astrarre le interazioni a basso livello tra interazione e animazione dei contenuti delle pagine. L'approccio di tipo modulare di jQuery consente la creazione semplificata di applicazioni web e versatili contenuti dinamici.

2.1.8 Template Ejs[12]

Uno dei più classici problemi della programmazione lato server è la separazione del codice di programmazione (nel nostro caso JavaScript) dall'HTML. Per risolvere questo problema, la comunità JavaScript ha creato i **template engine**. In questa applicazione web è stato utilizzato il template Ejs, che permette di utilizzare all'interno delle pagine HTML codice JavaScript facendo in modo di gestire i dati inviati dal server senza fare richieste AJAX, oltre ad utilizzare un linguaggio uniforme per tutta l'applicazione.

2.1.9 Express.js[13]

Express.js è il framework più utilizzato per lo sviluppo di applicazioni Node.js. È costruito attorno alla configurazione e alla semplicità granulare del middleware Connect. Permette la gestione e la creazione di web server in modo semplice e rapido.

2.1.10 Angular.js[14]

AngularJS è un framework web open source principalmente sviluppato da Google e dalla comunità di sviluppatori individuali che ruotano intorno al framework nato per affrontare le molte difficoltà incontrate nello sviluppo di applicazioni su singola pagina. Ha l'obiettivo di semplificare il test e lo sviluppo di questa tipologia di applicazioni fornendo un framework lato client con architettura MVC (*Model View Controller*) e MVVM (*Model View View Model*) insieme a componenti comunemente usate nelle applicazioni Web. AngularJS lavora leggendo prima la parte HTML, che ha incapsulati degli attributi personalizzati aggiuntivi (esempio: *ng-controller*). Il framework interpreta questi attributi come delle direttive per legare le parti di ingresso e uscita della pagina al modello che è rappresentato da variabili standard JavaScript. Il valore di queste variabili può essere impostato manualmente nel codice o recuperato da risorse JSON statiche o dinamiche.

2.1.11 Passport.js[15]

Passport.js è il middleware di autenticazione per Node.js. Nelle moderne applicazioni Web, l'autenticazione può assumere una varietà di forme. Tradizionalmente, gli utenti effettuano l'accesso fornendo un nome utente ed una password. Con l'avvento dei social network, il *single sign-on* che utilizza un provider OAuth come Twitter o Facebook è diventato un metodo di autenticazione popolare. Passport riconosce che ogni applicazione ha requisiti di autenticazione univoci. I meccanismi di autenticazione, noti come strategie, sono impacchettati come moduli individuali. Le applicazioni possono scegliere quale strategia utilizzare, senza creare dipendenze non necessarie.

2.1.12 Socket.IO[16]

Socket.IO è una libreria JavaScript per realizzare applicazioni web in real time. Consente la comunicazione in tempo reale e bidirezionale tra Web Client e Server. È composta da due parti: una libreria lato client che viene eseguita nel browser e una libreria lato server per Node.js. Entrambi i componenti hanno le API (Application Programming Interface) quasi identiche. Come Node.js, utilizza una programmazione ad eventi, o event-driven. Socket.IO utilizza principalmente il protocollo WebSocket con il *polling* (ossia la verifica ciclica di tutte le unità o

periferiche di input/output da parte del sistema operativo tramite test dei bit di bus associati ad ogni periferica, seguita da un eventuale operazione di scrittura o lettura) come opzione di fallback, pur fornendo la stessa interfaccia. Sebbene possa essere utilizzato come semplice wrapper per WebSocket, offre molte più funzionalità, tra cui la trasmissione a più socket, la memorizzazione dei dati associati a ciascun client e l'input/output asincrono.

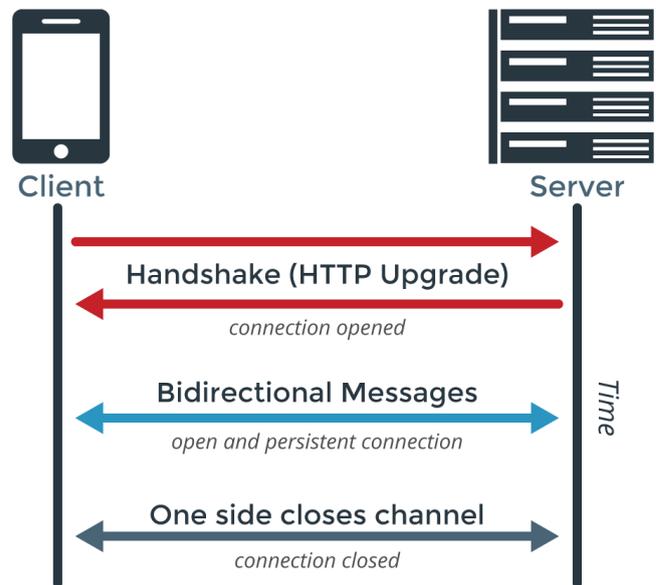


Figura 8 Diagramma Web Socket

Il protocollo WebSocket è una tecnologia web che fornisce canali di comunicazione full-duplex attraverso una singola connessione TCP. È stata creata per essere implementata sia lato client (browser) che lato server, ma può essere utilizzato anche da qualsiasi applicazione client-server. Il protocollo è un'implementazione basata sul protocollo TCP. La sua unica correlazione con l'HTTP è nel modo in cui fa l'handshake durante un upgrade *request* tra server. Il protocollo WebSocket permette maggiore interazione tra un browser e un server, facilitando la realizzazione di applicazioni che forniscono contenuti in tempo reale. Questo è reso possibile fornendo un modo standard per il server di mandare i contenuti ai browser senza dover essere sollecitato dal client e permettendo ai messaggi di andare e venire tenendo la connessione aperta. In aggiunta, le comunicazioni sono fatte attraverso la porta TCP 80, che è un vantaggio per gli ambienti che bloccano porte non standard utilizzando dei firewall.

2.1.13 Tiled Map Editor[17]

Tiled è un editor generico di mappe con tileset. Un tileset (letteralmente insieme di tessere) è un insieme di immagini che hanno le stesse dimensioni, dette *tile*, utilizzate accostate tra loro per comporre immagini di maggiori dimensioni. Questa tecnica caratterizza tradizionalmente la programmazione di videogiochi

con grafica bidimensionale, dove un piccolo tileset è meno oneroso di una grande immagine sotto vari aspetti. Tiled Map Editor è uno strumento gratuito per consentire la facile creazione di layout di mappa. È abbastanza versatile da permettere la specifica di cose più astratte come le aree di collisione. Tutti questi dati possono essere salvati in un comodo formato tmx standard o in formato JSON.

2.1.14 Redis[18]

Redis (*Remote Dictionary Server*) è uno store di strutture dati chiave-valore in memoria rapido e open source. Offre una serie di strutture dati in memoria molto versatili, che permettono di creare un'ampia gamma di applicazioni personalizzate. I principali casi d'uso per Redis sono la gestione di sessioni, il *caching*, servizi pub/sub e graduatorie. Si tratta dello store chiave-valore più utilizzato. Dispone di licenza BSD, è scritto in C, il suo codice è ottimizzato e supporta diverse sintassi di sviluppo. Data la velocità e la semplicità di utilizzo, Redis è una scelta molto comune per applicazioni Web, videogiochi, IoT e app per dispositivi mobili che necessitano di elevate prestazioni. Tutti i dati gestiti da Redis si trovano nella memoria principale del server, mentre la maggior parte di sistemi di gestione di database memorizzano i dati su disco o su SSD. Poiché non devono raccogliere i dati su un disco, i database in memoria come Redis non accumulano alcun ritardo e possono usare algoritmi più semplici per accedere ai dati, sfruttando di meno la CPU. In genere, l'esecuzione di un'operazione richiede meno di un millisecondo. Redis consente agli utenti di memorizzare le chiavi che consentono la mappatura ai differenti tipi di dati. L'elemento fondamentale dei tipi di dati è *String*, che può essere composta da dati di testo o binari con una dimensione massima di 512 MB. Redis supporta, inoltre, altri tipi di dati: list (elenchi di stringhe nell'ordine di aggiunta), set (stringhe prive di ordine), sorted set (insieme di stringhe ordinate in base ad un determinato valore), hash (memorizzano un elenco di campi e valori), hyperloglog (conta gli elementi univoci in un dataset). Redis offre una serie di strumenti che semplificano sviluppo e funzionamento rendendoli più rapidi; tra questi Pub/Sub, per pubblicare messaggi su diversi canali per l'invio a utenti iscritti al servizio, ideale per sistemi di chat e di messaggistica; chiavi TTL con una scadenza predefinita dopo la quale si cancellano automaticamente, per non intasare il database con dati non necessari; contatori atomici che assicura che la race condition non crei risultati discordanti; Lua, una sintassi di scripting potente ma leggera. Redis impiega un'architettura master-slave e supporta la replica asincrona, in cui i dati vengono replicati su diversi server slave. In questo modo è possibile ottenere migliori prestazioni in lettura (poiché le richieste vengono suddivise tra i diversi server) e ripristino rapido in caso di interruzione del server principale. Per fornire la massima durabilità, Redis supporta sia **snapshot** point-in-time (copiando i dataset di Redis su disco) sia la creazione di file **AOF** (Append Only File) in cui vengono memorizzate tutte le modifiche ai dati sul disco. Entrambi i metodi

permettono il ripristino rapido dei dati di Redis in caso di interruzione.

2.2 Database

2.2.1 RethinkDB[19]

RethinkDB è un database orientato ai documenti in tempo reale e open source. Memorizza i documenti in formato JSON (*JavaScript Object Notation*), dati fondamentalmente non strutturati, in un formato eseguibile con distribuzione (*sharding* e *replica*). Fornisce anche il *push* in tempo reale dei dati JSON sul server, che ridefinisce l'intero sviluppo di applicazioni web in real-time. RethinkDB utilizza **ReQL** come linguaggio di interrogazione, completamente diverso dagli altri linguaggi *NoSQL*. È basato su tre principi chiave:

- È integrato nel linguaggio di programmazione: le *query* vengono create effettuando chiamate di funzione nel linguaggio di programmazione che stiamo utilizzando. Non è necessario concatenare stringhe o costruire oggetti JSON specializzati per interrogare il database;
- Tutte le *query* ReQL sono concatenabili: si inizia con una tabella e si possono concatenare gli oggetti in maniera incrementale utilizzando l'operatore “.”;
- Tutte le *query* vengono eseguite sul server: mentre le *query* sono costruite sul client in un linguaggio di programmazione familiare, vengono eseguite interamente sul server del database una volta richiamato il comando *run* e trasmesso ad una connessione attiva del database.

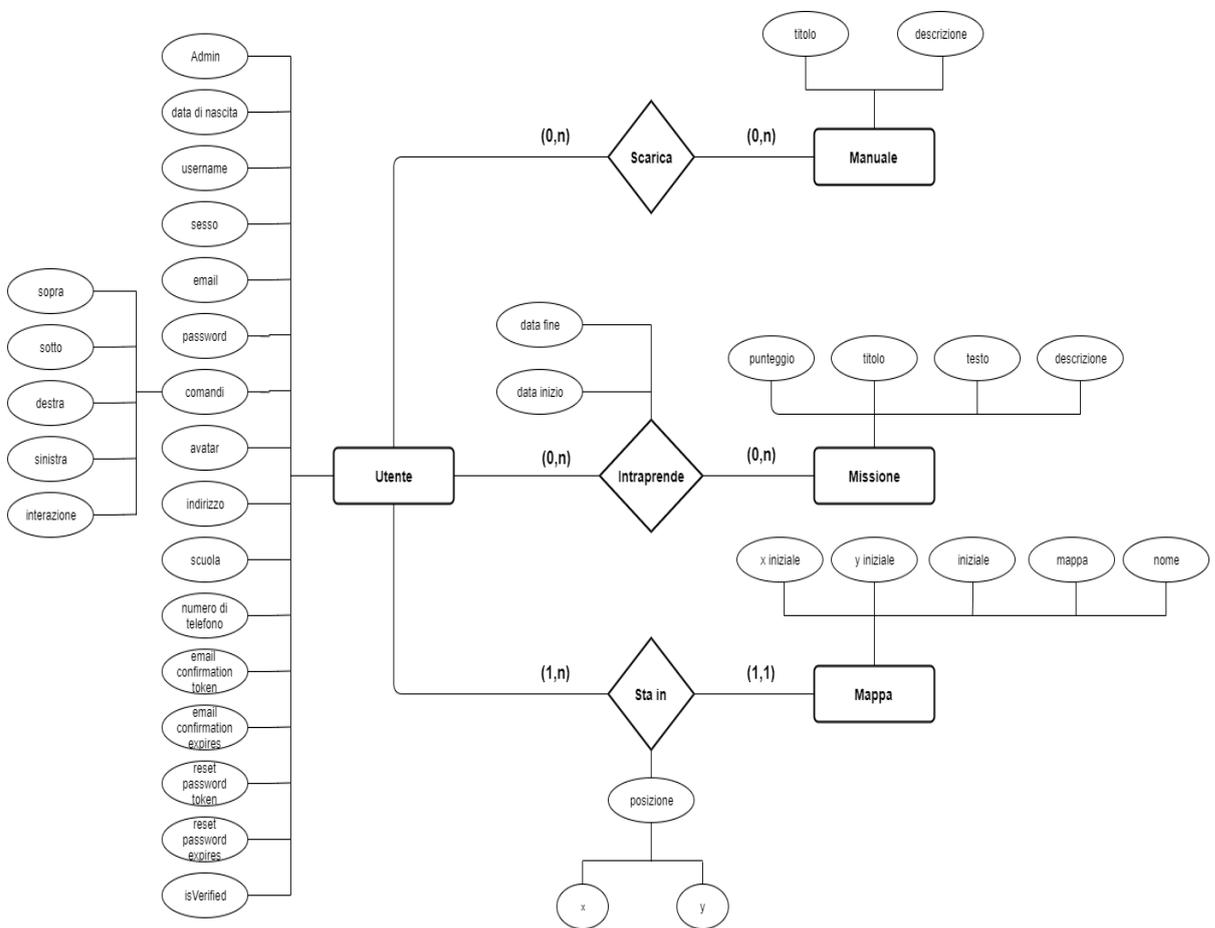
2.2.2 Requisiti del Database

Nel database, dovranno essere salvate permanentemente le seguenti informazioni:

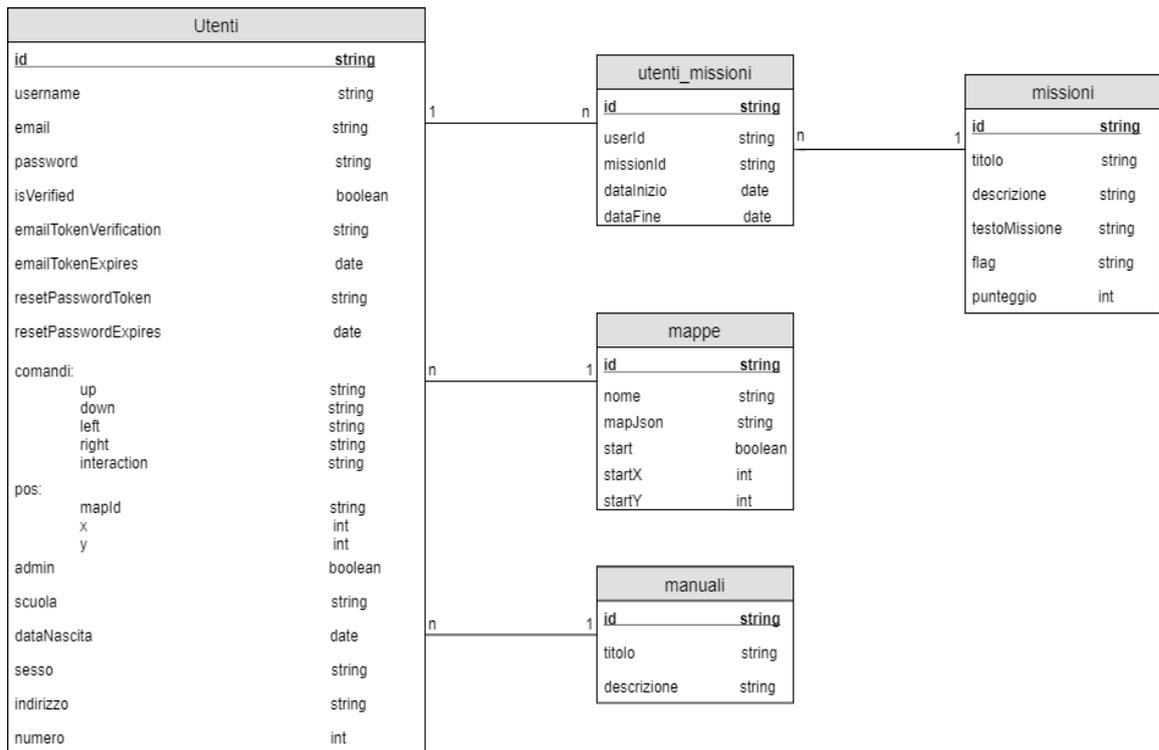
- Dati riguardanti l'utente per effettuare la registrazione e, quindi, il login:
 - Username;
 - Email;
 - Password;
 - Token per la verifica dell'account e la sua scadenza;
 - Token per la richiesta di reset password e la sua scadenza;
 - Un booleano per controllare se l'account è stato verificato o meno.
- Dati personali relativi ad ogni utente, inseriti facoltativamente dalla loro pagina profilo:
 - Scuola frequentata;
 - Data di nascita;
 - Sesso;
 - Indirizzo;
 - Numero di telefono.

- Le mappe di gioco;
- Le missioni insieme ai dati:
 - Titolo;
 - Descrizione;
 - Testo della missione;
 - Flag;
 - Punteggio.
- Manuali in formato .pdf, .zip o .rar con i dati titolo e descrizione;
- Account email, con i dati user, pass e service, utilizzato per inviare la mail agli utenti in fase di conferma account, password dimenticata e per l'invio di email ad uno o più utenti dal pannello di amministratore.

2.2.3 Diagramma ER



2.2.4 Schema Logico



2.2.5 Query

Di seguito, elencherò le *query* che ho utilizzato per accedere al database:

Restituisce l'account email che l'Admin utilizza per effettuare l'invio delle mail agli utenti, per la conferma dell'account e per il recupero della password:

```

1. exports.getAdminEmailPwd = function(callback) {
2.     return adminEmail.run().then(function(email) {
3.         return callback(email);
4.     })
5.     .error(function(err) {
6.         throw err;
7.     });
8. }
    
```

Modifica l'account email che l'Admin utilizza per effettuare l'invio delle mail agli utenti, per la conferma dell'account e per il recupero password:

```

1. exports.updateAdminEmail = function(data, callback) {
2.     adminEmail.delete().then(function(result) {
3.         adminEmail.insert(data).then(function(result) {
4.             return callback(data);
5.         });
6.     });
7. }
    
```

Restituisce il manuale con un id specifico:

```
1. exports.getManuals = function(id) {
2.     return Manuali.get(id).run().then(function(manuale) {
3.         return manuale;
4.     })
5.     .error(function(err) {
6.         throw err;
7.     });
8. }
```

Aggiunge un nuovo manuale:

```
1. exports.addNewManual = function(newManual) {
2.     var manuali = new Manuali(newManual);
3.     return manuali.save();
4. }
```

Aggiorna i dati di un manuale:

```
1. exports.updateManual = function(data) {
2.     return Manuali.get(data.id).update(data);
3. }
```

Elimina un manuale:

```
1. exports.deleteManual = function(id) {
2.     Manuali.get(id).delete().run();
3. }
```

Restituisce tutti i manuali presenti all'interno del database:

```
1. exports.getAllManuals = function(callback) {
2.     Manuali.run().then(function(manuale) {
3.
4.         return callback(manuale);
5.
6.     }).error(function(err) {
7.
8.         throw err;
9.
10.    });
11.
12. }
```

Query per la registrazione dell'utente al sito:

```
1. exports.addUser = function(req, res) {
2.
3.     // controlli campi
4.     req.checkBody('username', 'E\' richiesto un username').notEmpty();
5.     req.checkBody('email', 'E\' richiesta una e-mail').notEmpty();
6.     req.checkBody('email', 'L\'email non è valida').isEmail();
7.     req.checkBody('password', 'E\' richiesta una password').notEmpty();
```

```

8.     req.checkBody('password2', 'Le Password non corrispondo').equals(req.body.password);
9.
10.
11.
12.     var errors = req.validationErrors();
13.     if (errors) {
14.         var captchaId = uniqid();
15.         // creazione captcha
16.         var captcha = svgCaptcha.create();
17.         var textCaptcha = captcha.text;
18.         client.hset("captcha", captchaId, textCaptcha, function(err){
19.             return res.render('./register', {captchaId : captchaId, captchaImg : captcha.data, errors: errors });
20.         });
21.     } else {
22.         // Check recaptcha, verifica se la checkbox del recaptcha sia stata cliccata
23.
24.         if (req.body['g-recaptcha-response'] === undefined || req.body['g-recaptcha-response'] === '' || req.body['g-
recaptcha-response'] === null) {
25.             return captchaFunction(res, "Il captcha Google non è stato verificato");
26.         }
27.
28.         var secretKey = "6Ler61sUAAAAGFFbFAQp9iQsMaUfp0hg-EUWN-M";
29.         var verificationUrl = "https://www.google.com/recaptcha/api/siteverify?secret=" + secretKey + "&response=" + re
q.body['g-recaptcha-response'] + "&remoteip=" + req.connection.remoteAddress;
30.         request(verificationUrl, function(err, response, body) {
31.             var body = JSON.parse(body);
32.             if(body.success !== undefined && !body.success){
33.
34.                 return captchaFunction(res, "Verifica captcha Google fallita");
35.             }
36.         });
37.         captchaId = req.body.idCaptcha;
38.         text = req.body.captcha;
39.
40.         client.hget("captcha", captchaId, function(err, cText){
41.             if (cText != text){
42.                 return captchaFunction(res, "Il captcha è errato");
43.             }else{
44.                 verifier.verify(req.body.email, function(err, info) {
45.                     if (err) {
46.
47.                         console.log(err);
48.                         console.log(req.body.email);
49.
50.                         var error = [{ msg: "L'email inserita non è valida." }];
51.                         return captchaFunction(res, "L'email inserita non è valida");
52.                     }
53.                 } else {
54.                     getStartingMap().then(
55.                         function(map) {
56.                             mappa = map[0];
57.                             var utente = new User({
58.                                 username: req.body.username,
59.                                 email: req.body.email,
60.                                 password: bcrypt.hashSync(req.body.password, 10),
61.                                 comandi: {
62.                                     up: "w",
63.                                     down: "s",
64.                                     left: "a",
65.                                     right: "d",
66.                                     interaction: "e"
67.                                 },
68.                                 pos: {
69.                                     mapId: mappa.id,
70.                                     x: mappa.startX,
71.                                     y: mappa.startY
72.                                 },
73.                                 avatar: "player.png",
74.                                 admin: false,
75.                                 isVerified: false,
76.                             });
77.                             User.filter({ username: utente.username }).run()
78.                                 .then(function(result1) {
79.                                     if (result1.length != 0) {
80.                                         return captchaFunction(res, "Il nome dell'utente è già stato utilizzato.");
81.                                     } else {
82.                                         User.filter({ email: utente.email }).run()
83.                                             .then(function(result2) {
84.                                                 if (result2.length != 0) {
85.                                                     return captchaFunction(res, "L'email è già stata utilizzata.");
86.                                                 } else {
87.                                                     //creazione token di conferma utente
88.                                                     let tokenConfirmation = Math.floor((Math.random()* 90) + 74);
89.                                                     let tokenConfirmationExpires = Date.now() + 3600000;
90.

```

```

91.         tokenConfirmation = encrypt(""+tokenConfirmation,utente.password);
92.
93.         utente.emailConfirmationToken = tokenConfirmation;
94.         utente.emailConfirmationExpires = tokenConfirmationExpires;
95.
96.         utente.save().then(function(result) {
97.             async.waterfall([
98.                 function(callback){
99.                     let encodedMail = new Buffer(req.body.email).toString('base64');
100.
101.                     let link = "https://" + req.get('host') + "/verify?mail="+encodedMail
102.                        + "&id="+tokenConfirmation;
103.
104.                     let mailOptions={
105.                         from : authCred.auth.user,
106.                         to : req.body.email,
107.                         subject : "Conferma Account HackToWin Unicam",
108.                         html : "Ciao,<br> Per favore clicca sul link per verificare
109.                         il tuo account.<br><a href="+link+">Clicca qui per verificare il tuo account</a>"
110.                     };
111.                     callback(null,mailOptions,tokenConfirmation);
112.                 },
113.                 function(mailData,secretKey,callback) {
114.                     console.log(mailData);
115.                     smtpTransport.sendMail(mailData, function(error, response){
116.                         if(error){
117.                             console.log(error);
118.                             return callback(true,"Error in sending email");
119.                         }
120.                         console.log("Message sent: " + JSON.stringify(response));
121.                     });
122.                     req.flash('success_msg', 'Registrazione avvenuta con successo. Controlla la tua casella di posta elettronica per verificare l account');
123.                     res.redirect('/login');
124.                 }
125.             ]),function(err,data) {
126.                 console.log(err,data);
127.                 res.json({error : err === null ? false : true, data : data});
128.             });
129.         }).error(function(err) {
130.             return captchaFunction(res,err);
131.         });
132.     }
133. });
134. console.log("Success (T/F): " + info.success);
135. console.log("Info: " + info.info);
136. }
137. });
138. }
139. });
140. }
141. }

```

Query per permettere all'utente di inviare la richiesta per modificare la password:

```

1. exports.passwordDimenticata = function(req, res) {
2.     async.waterfall([
3.         function(callback) {
4.             User.filter({email: req.body.email}).run()
5.                 .then(function(user) {
6.                     if(!user || user.length == 0) {
7.                         res.render('../forgot', {error_msg: "Non esiste un account registrato con l'email inserita"});
8.                     }else{
9.                         // Persona esiste --> invia email.
10.                        callback(null,user[0]);
11.                    }
12.                });
13.            },

```

```

14.     function(user, callback) {
15.         let token = Math.floor((Math.random() * 100) + 54);
16.         let tokenExpire = Date.now() + 3600000; // + 1 ora.
17.
18.         token = encrypt(""+token,user.password);
19.         User.get(user.id).update({resetPasswordToken:token,resetPasswordExpire:token
    Expire}).run().then(function() {
20.             let encodedMail2 = new Buffer(req.body.email).toString('base64');
21.             let link2 = "https://"+req.get('host')+"/reset?mail="+encodedMail2+"&id=
    "+token;
22.             let mailOptions={
23.                 from : authCred.auth.user,
24.                 to : req.body.email,
25.                 subject : "Reset Password",
26.                 html : "Ciao,<br> Hai ricevuto questa mail perchè tu o qualcun altro
    ha effettuato la richiesta di resettare la password del tuo account.<br> Per favore cli
    cca sul seguente link, o incollalo nella barra del tuo browser per completare la richies
    ta.<br><a href="+link2+">Clicca qui per resettare la password</a>"
27.             };
28.             callback(null,mailOptions,token);
29.         });
30.
31.     },
32.     function(mailData,secretKey,callback) {
33.         console.log(mailData);
34.         smtpTransport.sendMail(mailData, function(error, response){
35.             if(error){
36.                 console.log(error);
37.                 return callback(true,"Error in sending email");
38.             }
39.             console.log("Message sent: " + JSON.stringify(response));
40.
41.             req.flash('success_msg', 'Email inviata all indirizzo '+req.body.email+'
    Per favore controlla la tua casella di posta elettronica');
42.             res.redirect('/login');
43.         });
44.     }
45. ],function(err,data) {
46.     console.log(err,data);
47.     res.json({error : err === null ? false : true, data : data});
48. });
49. }

```

Query per verificare il token ricevuto dalla mail inviata dal sistema dopo aver effettuato la registrazione:

```

1. exports.checkAccountVerification = function(res,req,uemail,token){
2.     User.filter({email:uemail}).run().then(function(users) {
3.         if (users.length!=1){
4.             req.flash('error_msg', 'Qualcosa è andato storto. ');
5.             res.redirect('/login');
6.             return;
7.         }
8.         if (token!=users[0].emailConfirmationToken){
9.             req.flash('error_msg', 'Token errato: Rieffettua la registrazione o contatta
    re admin. ');
10.            res.redirect('/login');
11.            return;
12.        }
13.        if (Date.now() <= users[0].emailConfirmationExpires){
14.            User.get(users[0].id).update({emailConfirmationToken:"0",emailConfirmationEx
    pires:Date.now()-36000, isVerified:true}).run().then(function(){
15.                req.flash('success_msg', 'Account Verificato');

```

```

16.         res.redirect('/login');
17.     });
18. }else{
19.     req.flash('error_msg', 'Token scaduto: Rieffettua la registrazione. ');
20.     res.redirect('/login');
21.     return;
22. }
23. });
24. }

```

Query per verificare il token ricevuto dalla mail inviata dal sistema dopo aver inviato la richiesta per il reset della password:

```

1. exports.checkToken = function(res,req,uemail,token){
2.     User.filter({email:uemail}).run().then(function(users) {
3.         if (users.length!=1){
4.             req.flash('error_msg', 'Qualcosa è andato storto. ');
5.             res.redirect('/login');
6.             return;
7.         }
8.         if (token!=users[0].resetPasswordToken){
9.             req.flash('error_msg', 'Token errato: Rieffettua la modifica della password. ');
10.            res.redirect('/login');
11.            return;
12.        }
13.        if (Date.now() <= users[0].resetPasswordExpire){
14.            res.render('./reset', {
15.                user_email:uemail,
16.                user_token:token
17.            });
18.        }else{
19.            req.flash('error_msg', 'Token scaduto: Rieffettua la modifica della password ');
20.            res.redirect('/login');
21.            return;
22.        }
23.    });
24. }

```

Modifica della password nella pagina “Password Dimenticata”:

```

1. exports.updatePassword = function(req, res){
2.     if(req.body.password != req.body.password2) {
3.         return res.render('./reset', {error_msg: "Le password non corrispondono",user_email:req.body.email,user_token:req.body.token});
4.     }
5.     User.filter({email: req.body.email}).run().then(function(users) {
6.
7.         if(req.body.password != req.body.password2) {
8.             return res.render('./reset', {error_msg: "Le password non corrispondono",user_email:req.body.email,user_token:req.body.token});
9.         } else {
10.            if (users.length !=1){
11.                res.render('./reset', {error_msg: "Email errata.",user_email:req.body.email,user_token:req.body.token});
12.            }
13.            return;
14.            // token check.
15.            if (req.body.token!=users[0].resetPasswordToken){

```

```

16.         req.flash('error_msg', 'Token errato: Rieffettua la modifica della passw
ord.');
```

```

17.         res.redirect('/login');
```

```

18.         return;
```

```

19.     }
20.     // token date Expire.
21.     if (Date.now() > users[0].resetPasswordExpire){
22.         req.flash('error_msg', 'Token scaduto: Rieffettua la modifica della pass
word.');
```

```

23.         res.redirect('/login');
```

```

24.         return;
```

```

25.     }
26.
27.     async.waterfall([
28.         function(callback) {
29.             // update token and token expire to previous date. n
30.             User.get(users[0].id).update({password: bcrypt.hashSync(req.body.pas
sword2, 10),resetPasswordToken:"0",resetPasswordExpire:Date.now()- 36000}).run().then(fu
nction() {
31.
32.                 let mailOptions={
33.                     from : authCred.auth.user,
34.                     to : req.body.email,
35.                     subject : "Password modificata",
36.                     html : "Ciao,<br> La tua password è stata modificata con suc
cesso</a>"
37.                 };
38.                 callback(null,mailOptions);
39.             });
40.         },
41.         function(mailData,secretKey,callback) {
42.             console.log(mailData);
43.             smtpTransport.sendMail(mailData, function(error, response){
44.                 if(error){
45.                     console.log(error);
46.                     return callback(true,"Error in sending email");
47.                 }
48.                 console.log("Message sent: " + JSON.stringify(response));
49.
50.                 req.flash('success_msg', 'Password modificata');
```

```

51.                 res.redirect('/login');
```

```

52.             });
53.         }
54.     ],function(err,data) {
55.         console.log(err,data);
56.         res.json({error : err === null ? false : true, data : data});
57.     });
58. }
59. });
60. }
```

2.3 CAPTCHA

In questo paragrafo tratterò uno dei più comuni ed utilizzati test *anti-bot* per le applicazioni Web riguardanti autenticazione, forum, blog e tutto ciò che può essere bersaglio di spam: il *CAPTCHA*. Spiegherò, inizialmente, il test di Turing, il quale, nella sua versione “inversa”, ha permesso lo sviluppo e la creazione del *CAPTCHA*. Successivamente, verranno discusse le varie versioni del test: dalle box contenenti stringhe alfanumeriche spesso distorte e difficilmente leggibili,

agli ultimissimi *reCAPTCHA* implementati da Google, fino ad arrivare all'*Invisible reCAPTCHA*, che è, attualmente, la versione più recente ed innovativa del test *CAPTCHA*, la quale risulta meno invasiva e ulteriormente sicura rispetto ai suoi predecessori.

2.3.1 Test di Turing

Il Test di Turing è un criterio che serve a determinare se una macchina sia in grado di pensare, suggerito da Alan Turing nell'articolo *Computing machinery and intelligence* redatto sulla rivista *Mind* nel 1950. Turing prende spunto dal "gioco dell'imitazione" a tre partecipanti: un uomo A, una donna B ed una terza persona C.

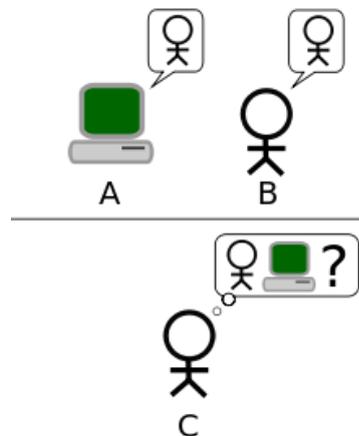


Figura 9 Gioco dell'imitazione

La persona C è tenuta separata dagli altri due e, tramite una serie di domande, deve stabilire qual è l'uomo e qual è la donna. Contemporaneamente A dovrà ingannare C e indurlo a commettere un'identificazione sbagliata, mentre B dovrà aiutarlo. Affinché C non possa disporre di alcun indizio (ad esempio l'analisi della voce o della grafia), le risposte alle domande di C devono essere dattiloscritte o similarmemente trasmesse. Il test di Turing si basa sul presupposto che una macchina si sostituisca ad A. Se la percentuale di volte in cui C indovina chi sia la donna e chi sia l'uomo è simile prima e dopo la sostituzione di A con la macchina, allora la macchina stessa dovrebbe essere considerata intelligente, dal momento che sarebbe indistinguibile da un essere umano, in questa determinata situazione. Per macchina intelligente, Turing ne intende una in grado di pensare, cioè capace di concatenare le idee e di esprimerle: si limita, quindi, tutto alla produzione di espressioni non prive di significato. Le macchine di Turing sono macchine a stati finiti in grado di simulare altre macchine a stati discreti. Per sostenere il test, una macchina deve essere programmata tenendo conto della descrizione di un uomo in termini discreti: ossia come stati interni, segnali, simboli. Dalla complessità del software, emergeranno le funzioni intellettuali; sulla quale aspettativa si fonda l'intelligenza artificiale, il cui scopo è la costruzione di una macchina in grado di riprodurre le funzioni cognitive umane. Oltre alla precedentemente descritta, ci sono altre versioni del test di Turing;

quella che andrò ad esaminare dettagliatamente è il test di Turing inverso.

2.3.2 Test di Turing Inverso e CAPTCHA

Il test di Turing **inverso** è una modifica del test di Turing dove è stato invertito l'obiettivo di uno o più ruoli tra le macchine e l'uomo. La sfida sarebbe che il computer fosse in grado di determinare sé stesso interagendo con un essere umano o con un altro computer. Questa è un'estensione della domanda originale a cui Turing ha cercato di rispondere, ma potrebbe offrire un protocollo sufficientemente accurato per definire una macchina in grado di "pensare" in maniera tale da essere definita tipicamente umana.

Il **CAPTCHA** è, effettivamente, una forma di test di Turing inverso. Con l'acronimo *CAPTCHA*, derivato dall'inglese "*Completely Automated Public Turing-test-to-tell Computers and Humans Apart*" ("Test di Turing pubblico e completamente automatico per distinguere computer e umani"), si denota un servizio per impedire che i bot utilizzino determinati servizi, come ad esempio i forum, la registrazione presso siti web, la scrittura di commenti e, generalmente, tutto ciò che può essere usato per creare spam o per violare la sicurezza con operazioni di *hacking* come il *brute force*. Il termine è stato coniato nel 2000 da Luis von Ahn, Manuel Blum e Nicholas J. Hopper dell'Università Carnegie Mellon e da John Langford della IBM. I *CAPTCHA* sono stati sviluppati per la prima volta nel 1997 dal settore ricerca e sviluppo di AltaVista, capitanato da Andrei Broder, per impedire ai bot di aggiungere URL al loro motore di ricerca. Essi cercavano di creare immagini resistenti agli attacchi degli OCR (sistema di riconoscimento ottico dei caratteri) e così consultarono il manuale degli scanner della Brother, su cui erano indicate tutte le caratteristiche che un testo deve avere per poter essere riconosciuto dallo scanner: cioè caratteri ben definiti e lineari, sfondo omogeneo e ben distinguibile dal testo, mancanza di differenze tra i font utilizzati nel testo, ecc. Applicando al contrario queste indicazioni, si riuscì ad ottenere una situazione dove il testo avesse caratteri storti, font diversi, colori del testo simili a quello dello sfondo; cioè un testo la cui scansione sarebbe stata molto difficile. Broder e il suo gruppo di sviluppo sostennero che, grazie all'introduzione di questa tecnologia, avrebbero ridotto lo spam di oltre il 95%. Indipendentemente dal team di AltaVista, Luis von Ahn e Manuel Blum, nel 2000, realizzarono e diffusero l'idea del test *CAPTCHA*, intendendo con ciò qualunque tipo di programma che fosse in grado di distinguere tra persone e computer. Loro inventarono vari tipi di test, compresi i primi a ricevere un'ampia diffusione grazie all'uso da parte di Yahoo!.

Ad oggi, i CAPTCHA sono utilizzati per impedire che i bot utilizzino determinati servizi, come i forum, la registrazione presso siti web, la scrittura di commenti e, in generale, tutto ciò che potrebbe essere usato per creare spam o per violare la sicurezza con operazioni di *hacking* come il *brute force*.

2.3.3 Evoluzione dei CAPTCHA



Figura 10 Immagini distorte per test CAPTCHA

La prima tipologia dei test CAPTCHA è quella di una *box* contenente una sequenza alfanumerica, spesso di difficile lettura e interpretazione (come da figura) perché distorta o offuscata, che vi viene chiesto di decifrare prima di concludere la vostra operazione. Possono essere anche indovinelli semplici o operazioni matematiche elementari. Questa tipologia di test CAPTCHA presenta, però, numerose problematiche: infatti, l'uso di test basati sulla lettura di testi o altre attività legate alla percezione visiva impedisce o limita fortemente l'accesso alle risorse protette agli utenti con problemi di vista e, poiché tali test sono progettati specificatamente per non essere leggibili da strumenti automatici, i normali ausili tecnologici usati dagli utenti ciechi, ipovedenti o daltonici non sono in grado di interpretarli. Nelle nuove generazioni di CAPTCHA, create per resistere ai più sofisticati programmi di riconoscimento testi, può diventare abbastanza complicato riuscire a riconoscere il testo da parte di molti utenti, anche nel pieno possesso della propria capacità visiva. Sono state, inoltre, scoperte alcune contromisure che permettono agli *spammer* di riuscire a superare i test. Infatti sono stati realizzati software per aggirare questa tipologia di test CAPTCHA che si sono rivelati efficaci nel 92% dei casi. Inoltre sono stati creati anche alcuni programmi per riconoscere i caratteri scritti, utilizzando tecniche apposite e non quelle standard degli OCR. Il metodo più efficace per il riconoscimento di tali test, però, risulta quello di utilizzare un essere umano per risolverlo: è infatti possibile affidare a persone pagate il compito di risolvere i CAPTCHA, ed è stato affermato dal W3C che un operatore può facilmente risolvere centinaia di test in un'ora.

A fine 2014, Google ha annunciato "**No CAPTCHA reCAPTCHA**", un nuovo tipo di test che riconosce i bot in modo più semplice per gli utenti. Per gli utenti si tratta di una semplificazione: basta infatti, nella maggior parte dei casi, mettere una spunta su una casella con scritto "**Non sono un robot**" (Figura 11)

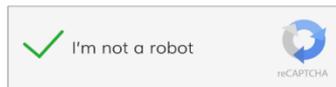


Figura 11 reCAPTCHA di Google

Con questo semplice gesto, il programma è in grado di acquisire una serie di informazioni (indirizzo IP, cookie, movimento del mouse e tempo impiegato per rispondere) sufficienti per riconoscere la maggior parte degli utenti. In casi di difficile disambiguazione persona-bot, il CAPTCHA ripropone, però, un secondo test dove, invece che il solito codice alfanumerico da decifrare, si dovrà scegliere quali tra alcune fotografie proposte assomigliano di più a un'immagine data (**Figura 12**).

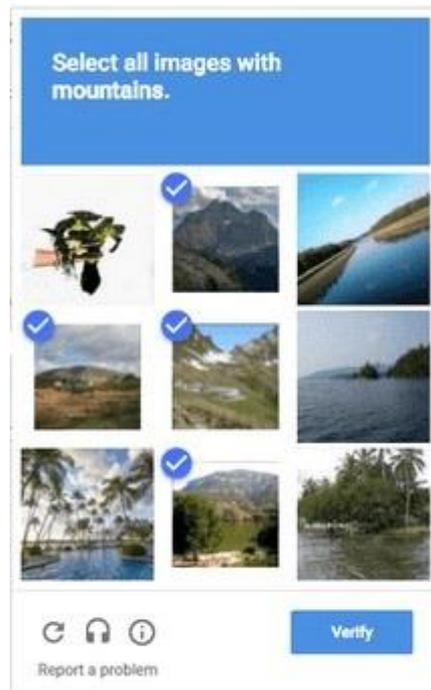


Figura 12 Test di Immagini di reCAPTCHA

Infine, è stato recentemente implementato l'**Invisible reCAPTCHA**, che di fatto stabilirà automaticamente, senza apparenti interventi visibili, se l'utente è una persona o un bot. Integrando una serie di nuovi algoritmi, un'intelligenza artificiale sarà in grado di rilevare parametri come il movimento del mouse e l'indirizzo IP con una precisione inedita, il tutto senza fare domande o richiedere *step* aggiuntivi agli utenti. Se il comportamento dell'utente sarà assimilabile a quello di un essere umano, l'autenticazione avverrà in modo completamente invisibile; altrimenti, se *Invisible reCAPTCHA* avesse qualche dubbio, allora ritorneranno i classici test.

3 Struttura del Sito e Gestione Utenti

In questo capitolo verranno illustrate, all'atto pratico, gli obiettivi e le modifiche apportate alla piattaforma. La struttura del sito è così composta:

```
1. ----/config
2. -----config.js
3. ----/models
4. -----api.js
5. ----/public
6. -----/app
7. -----app.js
8. -----/css
9. -----/js
10. -----/img
11. -----bower.json
12. ----/routes
13. -----index.js
14. -----users.js
15. ----/ssl files
16. ----/views
17. -----/includes
18. -----nav.html
19. -----/layouts
20. -----layout.html
21. -----404.html
22. -----admin.html
23. -----forgot.html
24. -----game.html
25. -----home.html
26. -----login.html
27. -----profilo.html
28. -----register.html
29. -----reset.html
30. ----package.json // contiene le informazioni delle librerie necessarie per il lato server
31. ----server.js // contiene la struttura principale del sito con l'aggiunta delle socket lato server
```

Come precedentemente indicato nel paragrafo 1.3, il web server che gestisce Hack To Win è stato implementato in Node.js, con la partecipazione del framework Express.js. La parte *front end* del sito, cioè la parte responsabile dell'acquisizione dei dati di ingresso e della loro elaborazione e, quindi, la parte visibile all'utente e con la quale può interagire è contenuta nelle cartelle **public** e **views**; il resto fa tutto parte del *back end*, ossia quella parte che permette l'effettivo funzionamento delle interazioni tra l'utente e l'interfaccia. Nei successivi paragrafi vedremo, quindi, degli *screenshot* dell'applicazione e come sono state implementate le modifiche richieste.

3.1 Pagina di Login

Quella di login è la prima pagina che viene visualizzata dall'utente quando vuole accedere al sito o quando non si è loggati:

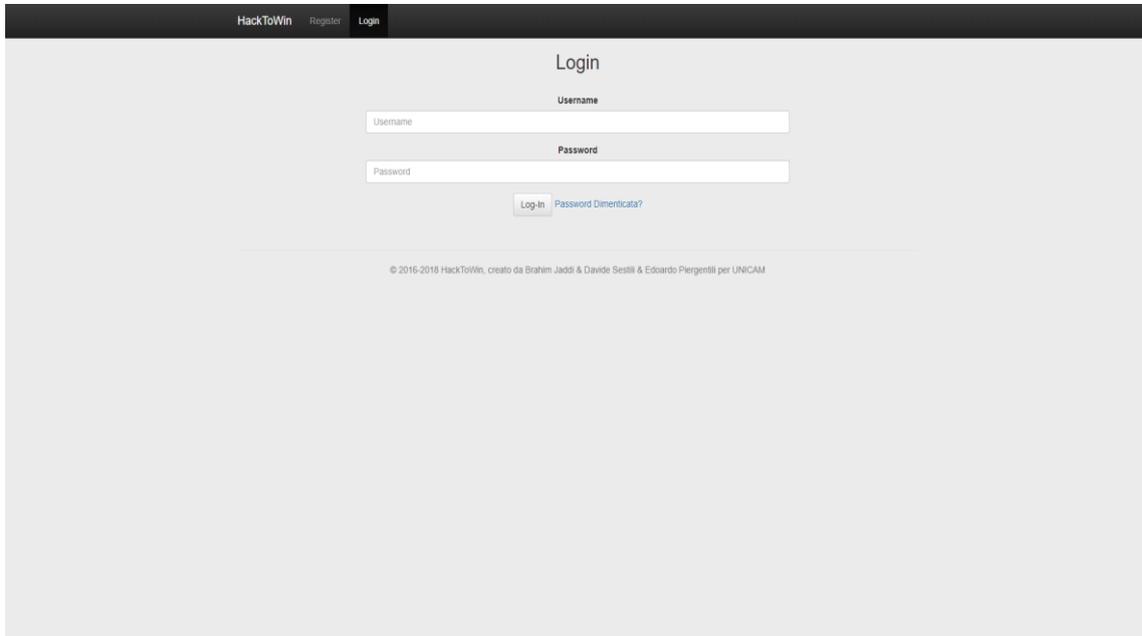


Figura 13 Screenshot della Pagina di Login

La richiesta era di aggiungere un link per permettere all'utente (qualora fosse già registrato) di recuperare la password. È stato, perciò, aggiunto un link che rimanderà alla pagina di “Password Dimenticata”. La gestione del back end del *routing*, ossia la gestione delle **URL** (*Uniform Resource Locator*) è stata così effettuata:

- Il metodo **GET**: quando l'utente clicca sul link, è necessario reindirizzarlo in una nuova pagina, dove gli sarà possibile inoltrare la richiesta per effettuare il cambio della password, grazie ai dovuti controlli:

```
1. // Route che si attiva quando l'utente clicca sul link password dimenticata
2.
3. router.get('/forgot',function (req, res) {
4.     res.render('../forgot', {
5.         user: req.user
6.     });
7. });
```

3.2 Pagina di Password Dimenticata

La pagina di Password Dimenticata permette all'utente registrato di inviare al sistema la richiesta per recuperare la password:

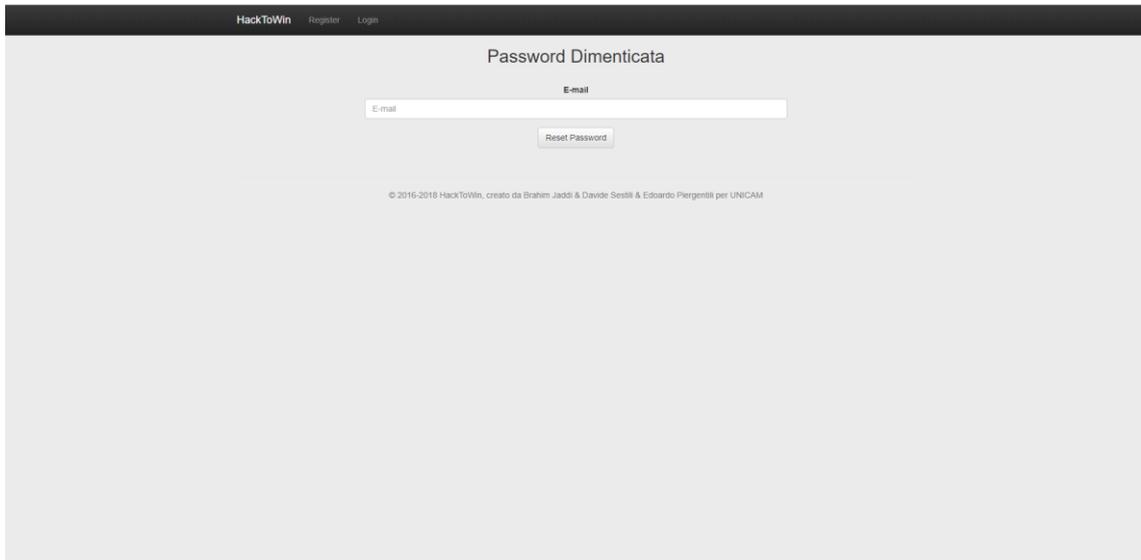


Figura 14 Screenshot della Pagina Password Dimenticata

La pagina presenta una textbox dove l'utente è chiamato ad inserire l'email con cui si è registrato. Quando l'utente clicca sul pulsante **Reset Password** invocherà il metodo **POST**:

```
1. // Route che si attiva quando l'utente fa richiesta di post sulla pagina /forgot
2. router.post('/forgot',api.passwordDimenticata);
```

Questa *route* richiama la query **passwordDimenticata** dal file **api.js**. La query, gestita da una funzione asincrona, controllerà se esiste un account associato all'email inserita nella textbox: se la ricerca ha esito negativo, verrà stampato a video un errore e l'utente sarà reindirizzato alla pagina "Password Dimenticata"; altrimenti entrerà in gioco una funzione con le seguenti funzionalità:

- Creo una variabile *token* a cui sarà assegnata una stringa univoca generata *randomicamente*, in modo da non avere contemporaneamente più token uguali;
- Creo una variabile *tokenExpire* che corrisponderà alla durata della validità del token, impostata a un'ora;
- Assegno alla variabile *token* il risultato ottenuto cifrando, grazie alla funzione **encrypt**, il token precedentemente ottenuto alla password dell'utente che ha inviato la richiesta. Successivamente invio le variabili *token* e *tokenExpire* al database;
- Creo il link da inviare all'utente grazie al quale potrà raggiungere la pagina per modificare la password. La URL del link è così strutturata:

```
1. let link2 = "https://" + req.get('host') + "/reset?mail="+encodedMail2+"&id="+token;
```

dove **encodedMail2** è la codifica, in *Base64* (sistema di codifica che consente la traduzione di dati binari in stringhe di testo ASCII, rappresentando i dati sulla base di 64 caratteri ASCII diversi), dell'indirizzo email dell'utente;

- Successivamente, grazie alle funzionalità di Nodemailer, gestisco la struttura della mail da inviare impostando i parametri **from:** , **to:** , **subject:** e **html**. L'utente riceverà, quindi, una mail contenente il link precedentemente creato.

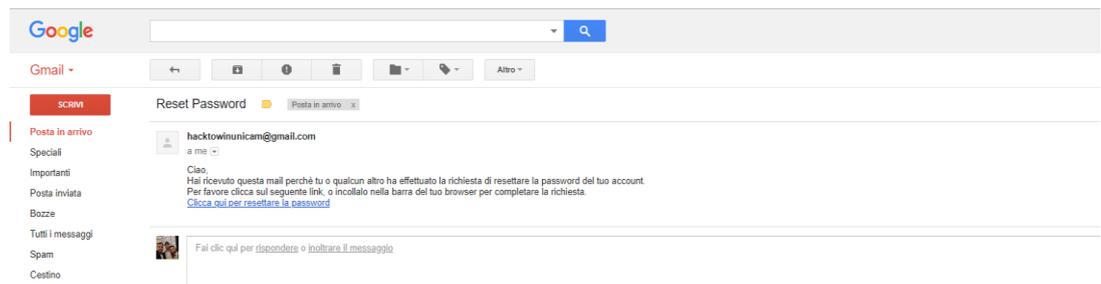


Figura 15 Screenshot dell'email ricevuta dall'utente che richiede il reset della password

3.3 Pagina di Modifica Password

La pagina di Modifica Password permette all'utente di inserire la nuova password per accedere al sito:

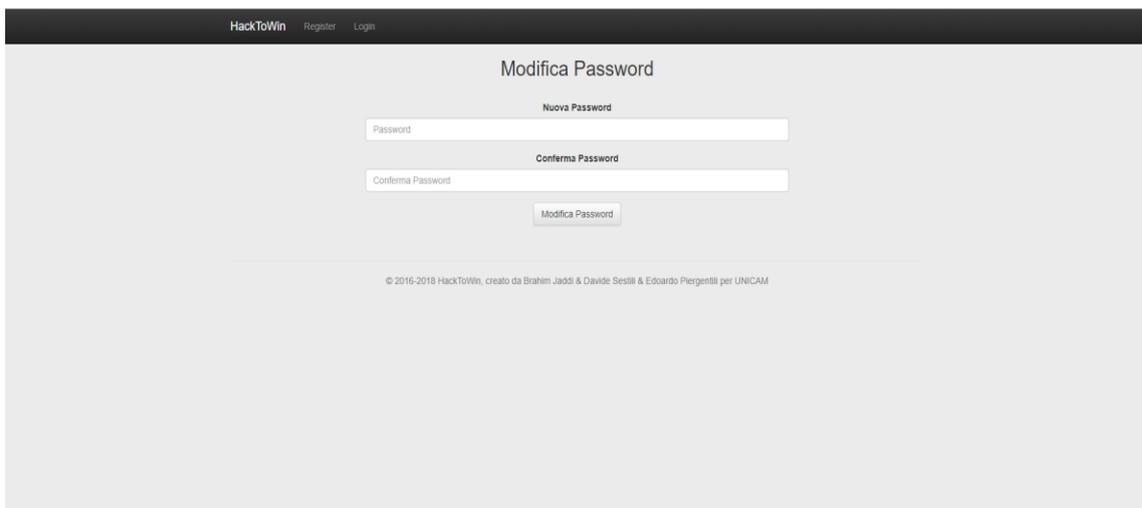


Figura 16 Screenshot della Pagina Modifica Password

- Il metodo **GET**: Quando l'utente clicca sul link ricevuto per email, sarà reindirizzato alla pagina di Modifica Password, grazie al layout che visualizzerà il contenuto del file *reset.html* nel proprio body grazie al **Template Ejs** e ai seguenti controlli:

```

1. // Route che si attiva quando l'utente clicca sul link che viene inviato tramite mail quando richiede il reset della password
2. router.get('/reset',function (req, res) {
3.     let token = req.query.id;
4.     let uemail = req.query.mail;
5.     if (uemail === undefined){
6.         req.flash('error_msg', 'Qualcosa è andato storto. ');
7.         res.redirect('/login');
8.         return;
9.     }
10.    uemail = (new Buffer(uemail, 'base64')).toString('utf8');
11.    api.checkToken(res,req,uemail,token);
12. });

```

Questa *route*, una volta recuperato il token e decodificata la mail dal link inviato in precedenza dal sistema (come descritto nel paragrafo 3.2), richiamerà direttamente la *query* **checkToken** dal file **api.js**. La *query* si occuperà di controllare la presenza del token sul database e se è scaduto o errato.

- Il metodo **POST**: viene attivato quando si vuole inviare il contenuto della *form* cliccando sul pulsante **Modifica Password**:

```

1. // Route che si attiva quando l'utente fa richiesta di post sulla pagina /reset
2. router.post('/reset',api.updatePassword);

```

La route richiama la *query* **updatePassword** da **api.js** che si occuperà di controllare, inizialmente, se le password inserite nelle due *textbox* della pagina corrispondono; successivamente controllerà l'esistenza e la validità del token e, una volta controllato che tutto sia corretto, inserirà la nuova password nel database e resetterà i valori **resetPasswordToken** e **resetPasswordExpire** e invierà una mail all'utente (analogamente a come descritto nel paragrafo 3.2) per notificargli che la modifica della password ha avuto successo.

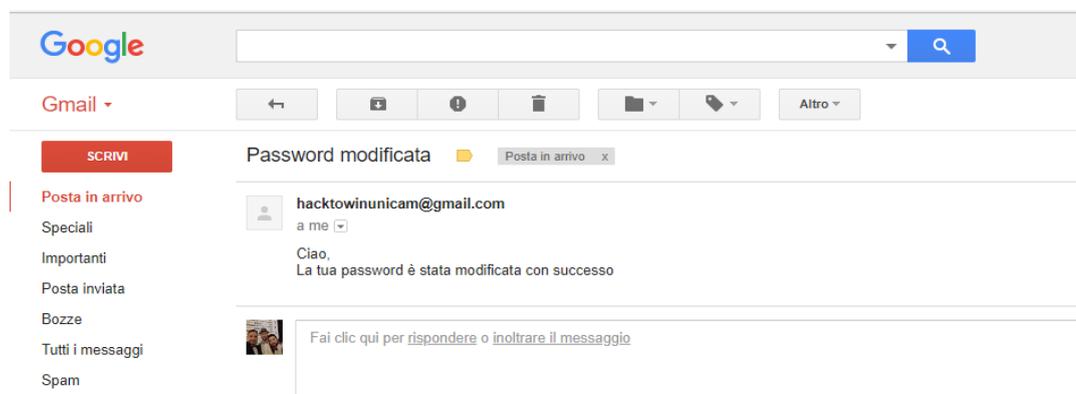


Figura 17 Screenshot email password modificata

3.4 Pagina di Registrazione

La pagina di Registrazione permette ad un nuovo utente di iscriversi al sito:

Figura 18 Screenshot della Pagina Registrazione

La richiesta era di implementare un controllo *CAPTCHA*, oltre ai consueti controlli sul database e la validità dell'email, per evitare attacchi *spam* al server. Come attacchi *spam* si intende l'invio anche verso indirizzi generici, non verificati o sconosciuti, di messaggi ripetuti ad alta frequenza o a carattere di monotematicità tale da renderli indesiderati. Come precedentemente spiegato nel paragrafo 1.3, sono state aggiunte due tipologie di test *CAPTCHA* per poter testare le loro differenze, la loro efficacia e per vedere se fossero compatibili. È stato inoltre richiesto di aggiungere un sistema per verificare l'account appena registrato. Tutto questo è stato gestito grazie ai seguenti metodi:

- Il metodo **GET**: entra in funzione quando l'utente clicca sul pulsante **Register** sulla barra di navigazione. Il contenuto della pagina **register.html** verrà visualizzato all'interno del body del layout tramite il **Template Ejs**. Verrà inoltre restituito il valore associato al campo *CAPTCHA* in versione **svg[20]** memorizzato in una chiave del database Redis, in modo da aggiornare l'immagine del test ad ogni richiesta di registrazione effettuata al server. Nella pagina **register.html**, verrà anche inserito il **reCAPTCHA di Google[21]** all'interno di un div.

```
1. router.get('/register',function (req, res) {
2.     // controlla se l'utente è già loggato
3.     if(req.user){
4.         // lo reindirizza alla pagina home
5.         res.redirect('/home');
6.     }else {
7.         // genera id univoco
8.         var captchaid = unqiud();
9.         // creazione svgCaptcha
10.        var captcha = svgCaptcha.create();
11.        var textCaptcha = captcha.text;
12.        client.hset("captcha", captchaid, textCaptcha, function(err){
```

```

13.         res.render('../register', {captchaId : captchaId, captchaImg : captcha.data
    });
14.     });
15. }
16. });

```

- Il metodo **POST**: entra in gioco quando l'utente clicca sul pulsante **Registrati**:

```

1. // Route che si attiva quando l'utente fa la richiesta di post sulla pagina /register
2. router.post('/register',api.addUser );

```

Questa route richiama la query **addUser** dal file **api.js**. La query si occuperà, inizialmente, di fare i controlli dei campi **Username**, **Email**, **Password** e **Conferma Password** verificando la loro validità e la loro eventuale esistenza all'interno del database. Successivamente si andrà a creare il **CAPTCHA** versione **svg**, si caricherà il valore su una chiave del database Redis e si andranno a gestire eventuali errori per quanto riguarda la risoluzione del test. Una volta terminato questo procedimento, si procederà all'implementazione del test *reCAPTCHA* di Google; per il quale sono necessari determinati passaggi:

- Innanzitutto bisogna registrare il sito web sulla piattaforma Google *reCAPTCHA*, raggiungibile al link indicato nei riferimenti sitografici [21], per ottenere le chiavi necessarie a codificare il modulo: **site key** e **secret key**;
- Lato **front end**: per visualizzare il *widget* è necessario includere, all'interno della pagina **register.html**, la risorsa JavaScript necessaria e un tag **g-recaptcha**; il quale è un elemento *DIV* con la chiave del nostro sito inserita all'interno dell'attributo **data-sitekey**. Lo script deve essere caricato utilizzando il protocollo *HTTPS* e può essere incluso in qualsiasi punto della pagina senza restrizioni;
- Lato **back end**: Verranno gestiti i controlli effettivi sul *CAPTCHA* direttamente sulla query **addUser**:
 - **g-recaptcha-response**: è la chiave che il browser genererà al momento dell'invio del modulo. Se è vuoto o nullo significa che l'utente non ha selezionato il *CAPTCHA* e, quindi, verrà restituito un errore;
 - **verificationUrl**:

```

1. var verificationUrl = "https://www.google.com/recaptcha/api/siteverify?secret=" +
    secretKey + "&response=" + req.body['g-recaptcha-
    response'] + "&remoteip=" + req.connection.remoteAddress;

```

dove **secretKey** è la chiave segreta rilasciata da Google e **req.connection.remoteAddress** fornirà l'indirizzo IP dell'utente connesso (parametro opzionale).

Quando l'utente cliccherà sulla *checkbox* per verificare il *CAPTCHA*, se il sistema non riuscirà a riconoscere direttamente che l'utente sia umano, apparirà una finestra con un quesito e delle immagini da selezionare (come in Figura 19).

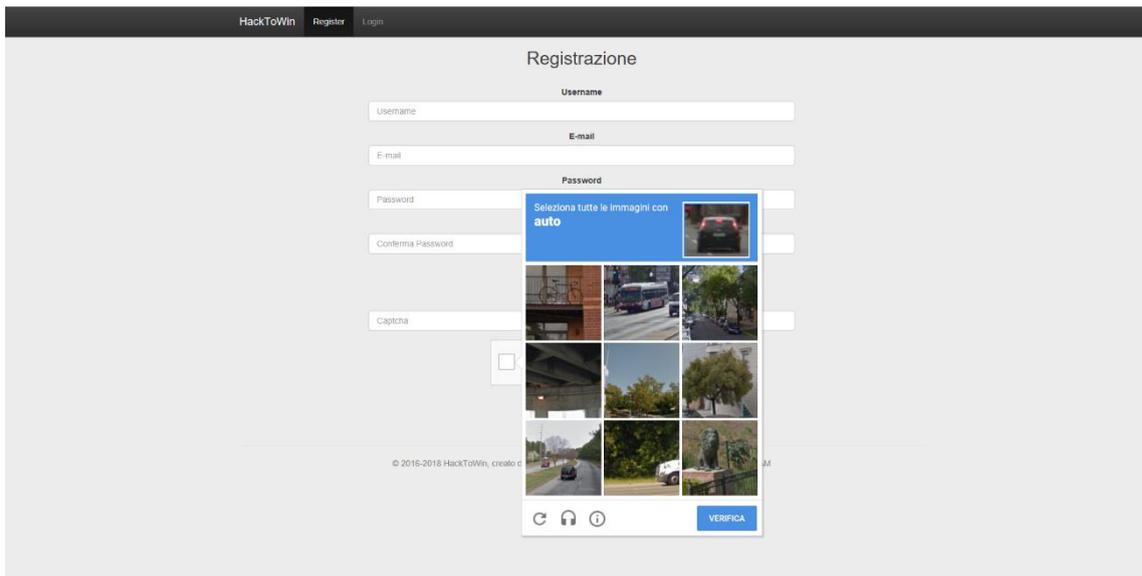


Figura 19 Screenshot test reCAPTCHA

Una volta che l'utente ha portato a termine tutte le richieste della *form* di registrazione e cliccato sul bottone **Registrati**, analogamente a come avvenuto nel paragrafo 3.2, entrerà in gioco una funzione per la creazione di un token univoco da inviare, tramite mail, all'utente per verificare il suo account:

- Creo una variabile *tokenConfirmation* a cui sarà assegnata una stringa univoca generata *randomicamente*, in modo da non avere contemporaneamente più token uguali;
- Creo una variabile *tokenConfirmationExpires* che corrisponderà alla durata della validità del token di conferma account, impostata a un'ora;
- Assegno alla variabile *tokenConfirmation* il risultato ottenuto cifrando, grazie alla funzione **encrypt**, il token precedentemente ottenuto alla password dell'utente che ha inviato la richiesta. Successivamente invio le variabili *tokenConfirmation* e *tokenConfirmationExpires* al database;
- Creo il link da inviare all'utente grazie al quale potrà raggiungere la pagina per modificare la password. La URL del link è così strutturata:

```
1. let link = "https://" + req.get('host') + "/verify?mail="+encodedMail+"&id="+tokenConfirmation;
```

dove **encodedMail** è la codifica, in *Base64*, dell'indirizzo email dell'utente;

- Successivamente, grazie alle funzionalità di Nodemailer, gestisco la struttura della mail da inviare impostando i parametri **from:** , **to:** , **subject:** e **html**. L'utente riceverà, quindi, una mail contenente il link precedentemente creato.

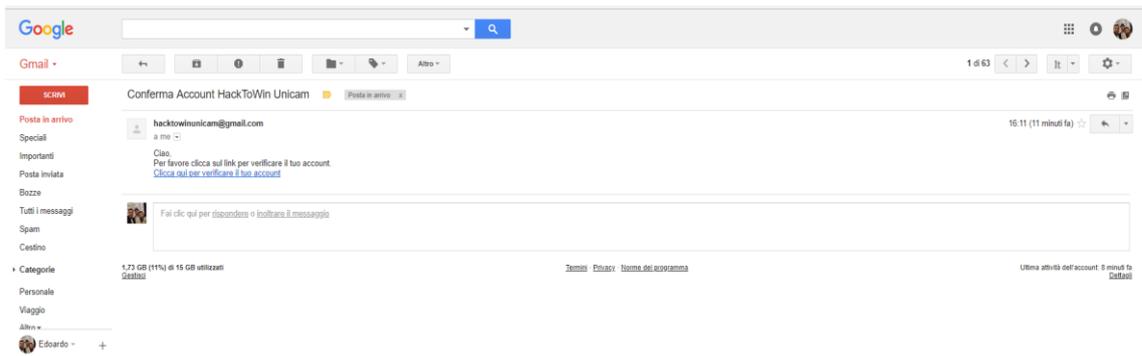


Figura 20 Screenshot Email Conferma Account

Quando l'utente clicca sul link ricevuto per email, inizierà la procedura per verificare l'account:

- Il metodo **GET**:

```

1. // Route che si attiva quando l'utente clicca sul link per verificare l'account
2. router.get('/verify', function (req, res) {
3.   let token = req.query.id;
4.   let uemail = req.query.mail;
5.   if (uemail === undefined){
6.     req.flash('error_msg', 'Qualcosa è andato storto.');
```

La route, una volta recuperato il token e decodificata la mail dell'utente dal link inviato dal sistema, richiamerà la query **checkAccountVerification**; la quale, una volta controllata l'esistenza e la validità del token, imposterà il valore del parametro booleano **isVerified** (settato di default a *false*) a *true* e lo invierà al database. Verranno anche azzerati i valori *tokenConfirmation* e *tokenConfirmationExpires*. Successivamente l'utente verrà reindirizzato alla pagina di **login** e verrà avvisato, tramite un messaggio di successo stampato direttamente sulla pagina, l'avvenuta verifica dell'account. Ora è finalmente in grado di effettuare il login per accedere al sito.

3.5 Pagina Admin

La pagina **Admin** è accessibile solamente agli utenti che hanno ottenuto i privilegi di amministratore ed il codice che la gestisce è all'interno del file **app.js**. Rispetto alla precedente versione dell'applicazione, è stato richiesto di implementare un sistema per poter inviare, direttamente dalla pagina, delle email ad uno o più utenti registrati al sito, un pannello dove è sia possibile impostare l'account email con cui vengono inviate le mail dal sistema e un pannello dove poter gestire i manuali da inserire all'interno del sito.

3.5.1 Scheda Utenti

La **scheda Utenti** permette all'amministratore l'effettiva gestione degli utenti registrati grazie al controller **GetUsersCtrl**.

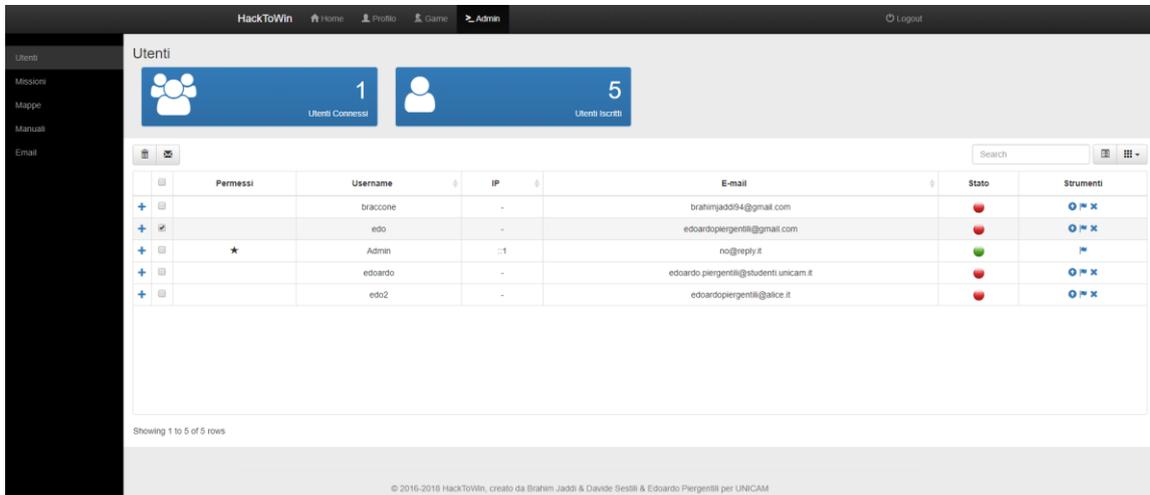


Figura 21 Screenshot Pannello Admin - Scheda Utenti

La modifica apportata a questo pannello è stata l'aggiunta del bottone  (implementato grazie a **Bootstrap**), disabilitato se non è stato selezionato almeno un utente tramite la checkbox presente all'interno della tabella contenente gli utenti registrati. Una volta cliccato il bottone, apparirà la finestra **Invia Email** (Figura 22) dove potremo inserire **Oggetto Email** e **Testo Email**.

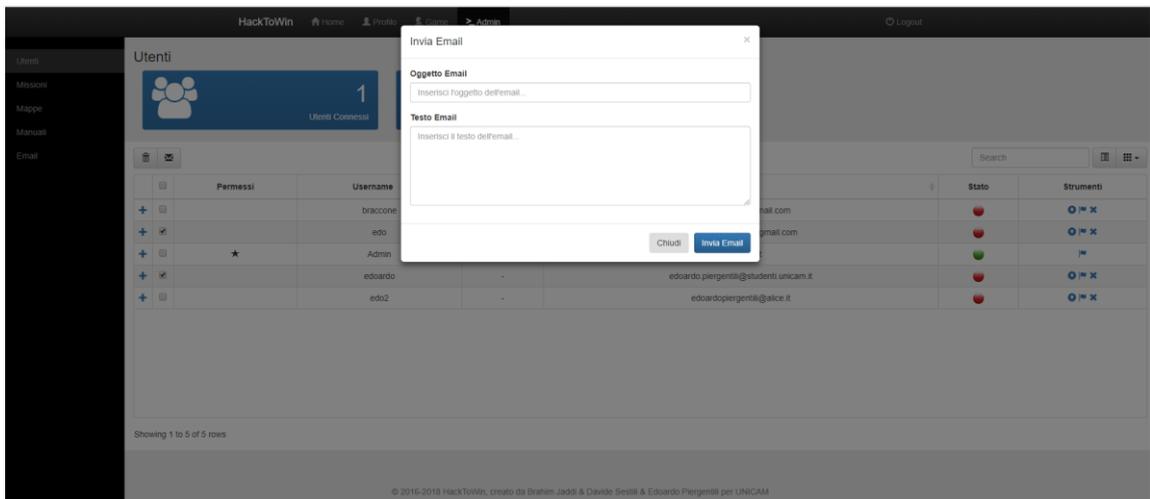


Figura 22 Screenshot Scheda Utenti - Invia Email

L'invio dell'email è gestito tramite le socket **sendMail**: la *socket.emit* è all'interno del file **app.js** mentre la *socket.on* è all'interno del file **server.js**. Nella *socket.on* vengono richiamate due query direttamente dal file **api.js**: **getCred** e **getSmtptTransport**. La prima query serve per restituire l'account email, utilizzato dal sistema per l'invio, impostato nel pannello **Email**; mentre la seconda query serve a gestire l'invio vero e proprio della mail grazie al modulo di

Node.js **Nodemailer**. Una volta cliccato su **Invia Email**, verrà notificato, tramite *alert*, che la mail è stata inviata con successo.

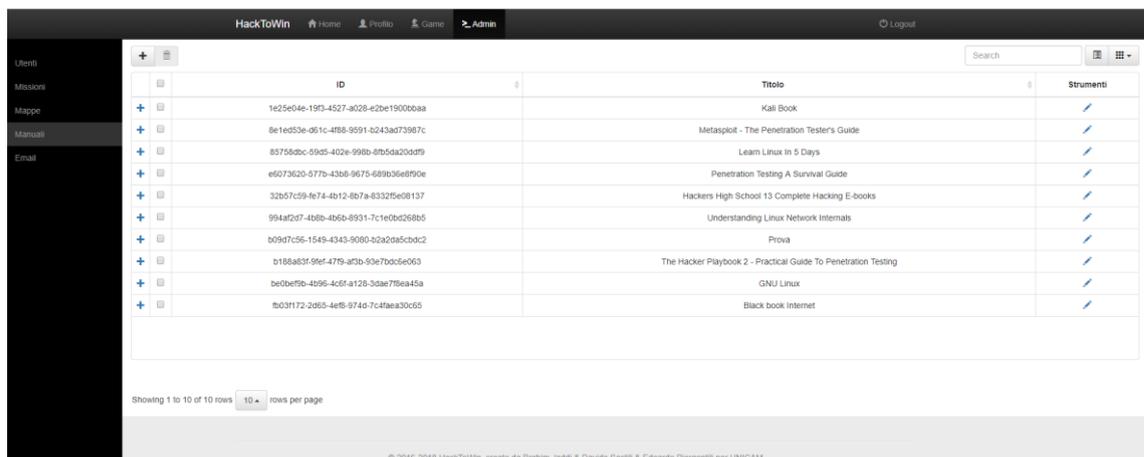
3.5.2 Scheda Manuali

La **scheda Manuali** serve per la gestione dei manuali. Tramite questa scheda possiamo svolgere le seguenti funzioni:

- L'aggiunta di nuovi manuali tramite il pulsante  il quale, una volta cliccato, aprirà una finestra che ci permetterà di inserire alcuni campi da compilare:
 - **Titolo:** il titolo da dare al manuale che verrà visualizzato all'interno del gioco nella **finestra Manuali**;
 - **Descrizione:** è possibile aggiungere una breve descrizione in modo da dare un'ulteriore facilitazione all'utente.
 - Il caricamento dei file è gestito utilizzando il tag **input** di HTML e i file permessi sono **pdf**, **zip** e **rar**.

Una volta compilati i campi, invieremo i dati al server cliccando sul pulsante **Aggiungi Manuale** e aggiungeremo il manuale caricato al database tramite la query **addNewManual**. Per aggiungere effettivamente un manuale al gioco, bisognerà creare, tramite **Tiled Map Editor**, un oggetto senza immagine con le proprietà **id** che indicherà l'id del manuale da inserire e una proprietà booleana **manuali** impostata a *true* nel punto della mappa in cui si vuole inserire il manuale.

- La modifica dei manuali presenti tramite il pulsante , il quale mostrerà una finestra con i campi, precedentemente descritti, da modificare. Modifiche che saranno apportate al database tramite la query **updateManual**.
- La rimozione di uno o più manuali tramite il pulsante , che richiamerà la query **deleteManual**.

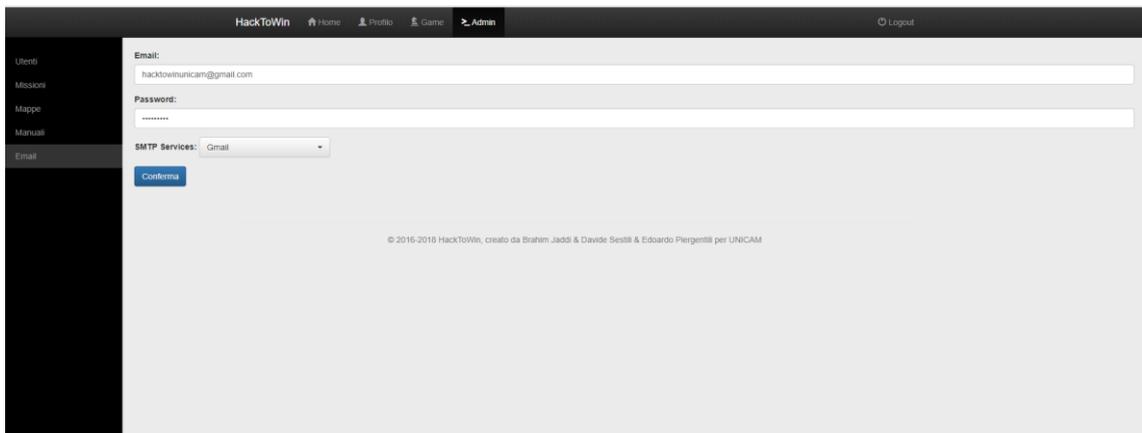


ID	Titolo	Strumenti
1e25e04e-19f3-4527-a028-e2be1500b0aa	Kali Book	
8e1e053e-051c-4f88-9591-0243ad73907c	Metasploit - The Penetration Tester's Guide	
857588bc-59d5-402e-996b-8fb56a20d0f9	Learn Linux in 5 Days	
e6073620-577b-43b6-9675-689b36e8f90e	Penetration Testing A Survival Guide	
32b57c59-4e74-4b12-8b7a-83325e08137	Hackers High School 13 Complete Hacking E-books	
994af2d7-4b6b-4b6b-8931-7c1e6bd26b05	Understanding Linux Network Internals	
b09d7c56-1549-4343-9080-02a29a5c9d32	Priva	
0188a83f-96ef-47f9-af5b-93e7005e063	The Hacker Playbook 2 - Practical Guide To Penetration Testing	
be0be95b-4096-405f-a128-3dae7f8ea45a	GNU Linux	
8b031f72-2965-4ef8-9740-7c4faea30c65	Black book Internet	

Figura 23 Screenshot Pannello Admin - Scheda Manuali

3.5.3 Scheda Email

La **scheda Email** serve per impostare l'account con il quale il sistema invia le email. La scheda presenta tre campi: **Email**, **Password** e **SMTP Services**. In questi campi sono già presenti dei valori di default. Nel campo SMTP Services è presente un elenco dei servizi supportati da Nodemailer: una volta scelto il servizio appropriato, il sistema imposterà automaticamente i valori **host** e **port** relativi al servizio scelto, in modo da configurare il server *SMTP*, ossia il protocollo standard per la trasmissione di email (*Simple Mail Transfer Protocol*) specifico per ogni account. La modifica dell'account avverrà cliccando sul pulsante **Conferma**, il quale invierà i dati al server e salverà i dati sul database tramite la query **setCred**.



The screenshot shows the 'Email' configuration page in the HackToWin Admin Panel. The page has a dark sidebar on the left with navigation links: 'Utenti', 'Missioni', 'Mappe', 'Manuali', and 'Email'. The main content area is light gray and contains the following fields:

- Email:** A text input field containing 'hacktowinunicam@gmail.com'.
- Password:** A text input field with masked characters (dots).
- SMTP Services:** A dropdown menu with 'Gmail' selected.
- Conferma:** A blue button to save the configuration.

At the bottom of the page, there is a small copyright notice: '© 2016-2019 HackToWin, creato da Ibrahim Jaddi & Davide Sestili & Edoardo Pergentili per UNICAM'.

Figura 24 Screenshot Pannello Admin - Scheda Email

4 Gestione Biblioteca

Passando al gioco vero e proprio, ho aggiornato la mappa iniziale del gioco inserendo l'edificio della biblioteca e ho inserito la mappa dell'interno della biblioteca. La mappa si presenta come una classica ed elegante biblioteca con all'interno delle librerie, sulle quali è stato inserito l'argomento su cui il manuale caricato si basa.

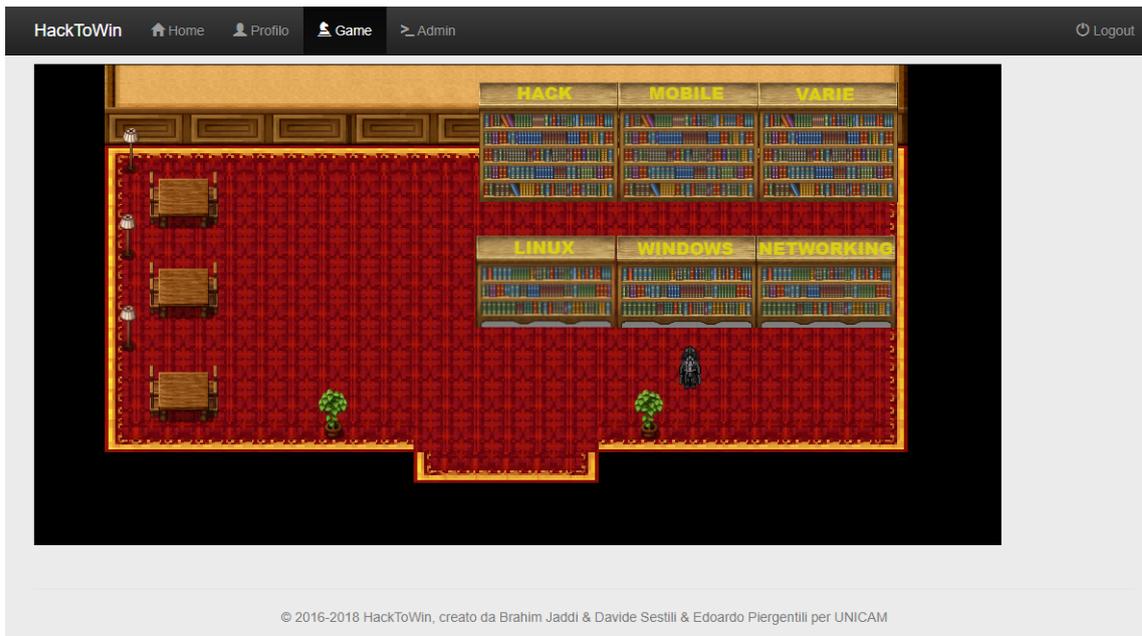


Figura 25 Screenshot Biblioteca

4.1 Descrizione

La mappa è stata disegnata tramite lo strumento **Tiled Map Editor** e caricata sul database, tramite la pagina **Admin**, in un file JSON. L'applicazione prende in input il file JSON, lo analizza e crea, per ogni elemento della mappa, un oggetto JavaScript al quale, a seconda della tipologia, verrà assegnata la relativa classe o verrà stampata nel *canvas*. Il **canvas** è un oggetto appartenente al linguaggio HTML che ci permette di stampare il gioco sul browser. Il file JSON della Biblioteca ha le seguenti classi:

- **Livelli:** In questa classe troviamo i vari livelli della mappa e l'ordine con cui vengono stampati. Ogni livello ha un attributo **data** che è una matrice quadrata contenente, per ogni elemento, l'**id** delle immagini da stampare;
- **Tileset:** In questa classe inseriamo le immagini da utilizzare all'interno della mappa. Ogni *tileset* ha un attributo **nome** e **id**.

Gli oggetti presenti nella mappa sono:

- **Solidi semplici:** non hanno attributi e servono sostanzialmente a gestire le collisioni;
- **Porte:** Servono per muovere il giocatore da una mappa all'altra. Hanno

due proprietà personalizzate: un attributo booleano **porta**, il quale indica al sistema che il giocatore si trova a collidere con una porta e un attributo **id**, che contiene l'id della mappa a cui la porta è collegata.

- **Manuali:** Hanno due proprietà personalizzate: **id** di tipo *string* che contiene l'id del manuale che si vuole scaricare e **manuali** di tipo booleano che indica al sistema che il giocatore sta per interagire con un manuale.

Per la gestione delle collisioni, ossia delle parti della mappa in cui l'avatar del giocatore non può accedere, viene utilizzato un oggetto che, tra i suoi attributi, ha anche una lista contenente tutti i solidi ricevuti durante il *parsing*, ossia quel processo che analizza un flusso continuo di dati in input, in modo da determinare la sua struttura grazie ad una data grammatica formale, della mappa. Qualora il personaggio collida con una porta, allora il sistema cambierà la mappa corrente con quella il cui id è stato assegnato al solido porta; se collide con un edificio allora il sistema stamperà una finestra contenente le informazioni riguardo l'edificio e se collide con un manuale, tramite il pulsante di interazione, verrà visualizzata la finestra contenente tutte le informazioni sul manuale il cui id è stato assegnato al solido.

4.2 Finestra Manuali

Gli id dei manuali sono stati inseriti negli scaffali all'interno della mappa Biblioteca. Quando l'utente interagirà con lo scaffale, apparirà una finestra come in Figura 26.

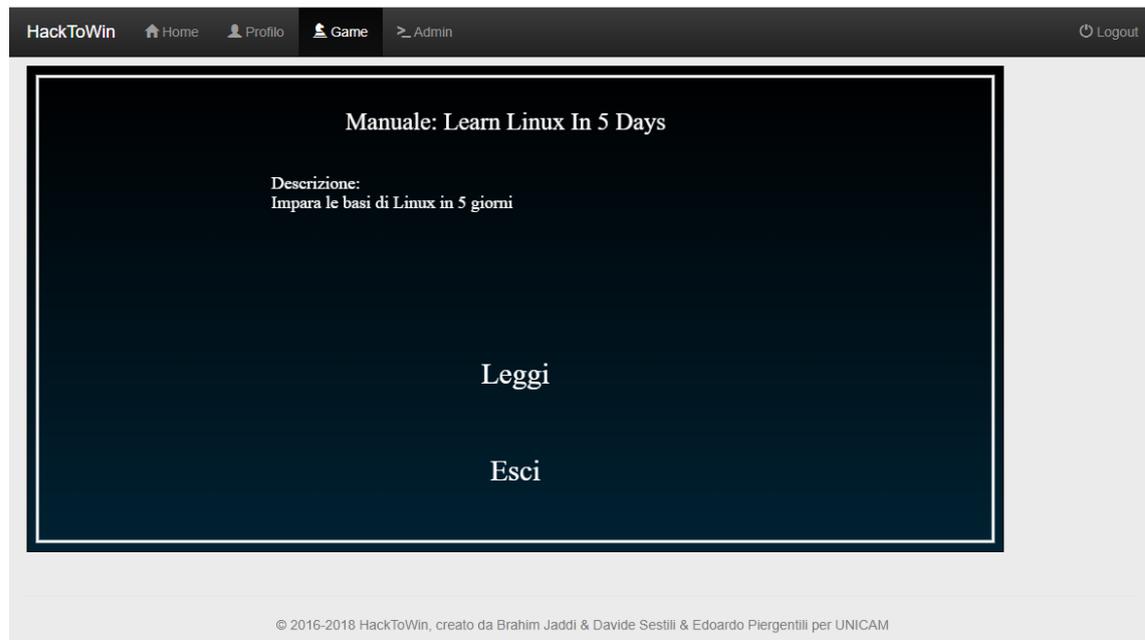


Figura 26 Screenshot Finestra Manuali

Nella finestra verranno stampati il **titolo** e la **descrizione** del manuale che sono stati inseriti dalla **scheda Manuali** del pannello **Admin** (paragrafo 3.5.2) e un link **Leggi** che permetterà all'utente di scaricare il manuale. Per stampare il testo all'interno della finestra è stato utilizzato il *parser* implementato precedentemente dai miei colleghi.

5 Manuale

5.1 Installazione

Gli strumenti che dovremo installare per avviare l'applicazione sono i seguenti:

- **Node.js [6]** : la versione consigliata è **8.11.3**;
- **RethinkDB [19]** : la versione consigliata è **2.3**;
- **Redis [18]** : la versione consigliata è **4.0.10**.

5.2 Avvio Applicazione

Per avviare l'applicazione è necessario azionare inizialmente il database. Dato che per accedere al database con RethinkDB necessitiamo di un username e di una password, che sono salvati nel file di configurazione **config.js**, dobbiamo digitare il seguente comando in un terminale aperto nella cartella contenente l'applicazione:

```
1. rethinkdb --initial-password""
```

dove tra virgolette andrà scritta la nostra password. Per le volte successive sarà sufficiente inserire solamente il comando `rethinkdb`.

Una volta partito il database, avvieremo l'applicazione web tramite il comando:

```
1. node server.js
```

Potremo, quindi, accedere finalmente all'applicazione tramite il browser che più ci aggrada.

5.3 Creazione Mappe

Per creare una nuova mappa è necessario scaricare il programma **Tiled Map Editor [17]**, dove è inoltre possibile trovare la manualistica per poter imparare i comandi e le funzionalità del programma.

5.4 Aggiunta Manuali

Per aggiungere un manuale al gioco, dobbiamo dapprima inserirlo nel database tramite la **scheda Manuali** descritta nel paragrafo 3.5.2. Una volta caricato, nella tabella verrà stampato, oltre al titolo, l'**id** univoco del manuale; il quale dovrà essere inserito nella mappa per identificare tale manuale. Apriamo, quindi, Tiled Map Editor con la mappa in cui desideriamo inserire il manuale (o creiamone una nuova). In questo esempio farò riferimento alla mappa Biblioteca, ma come linea di principio il funzionamento è valido per qualsiasi altra mappa si voglia utilizzare.

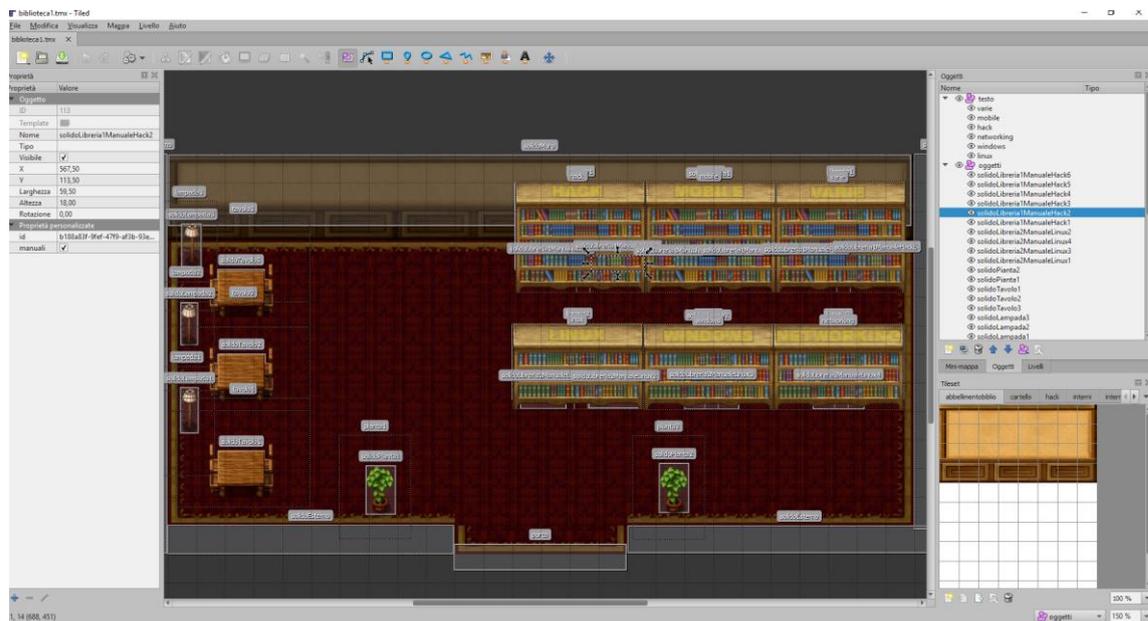


Figura 27 Screenshot Tiled Map Editor - Mappa Biblioteca

Per associare un manuale ad un oggetto, ad esempio una libreria, basterà cliccare sul pulsante  (**Inserisci Rettangolo**), che troviamo sulla barra degli strumenti, e disegnare un rettangolo delle dimensioni desiderate. Il rettangolo ottenuto corrisponderà ad un oggetto solido con il quale il personaggio collide. Quindi selezioniamo e rinominiamo il solido appena creato nella finestra **Oggetti** e aggiungiamo due **Proprietà Personalizzate**, cliccando il tasto aggiungi dalla finestra **Proprietà**. Le proprietà da inserire sono:

- **id** di tipo *String*, che indica l'id del manuale salvato sul database;
- **manuali** di tipo *Boolean* che indica al sistema che il giocatore sta interagendo con un manuale.

Proprietà	Valore
▼ Oggetto	
ID	75
Template	
Nome	solidoLibreria2ManualeLinux4
Tipo	
Visibile	<input checked="" type="checkbox"/>
X	815,33
Y	257,67
Larghezza	56,67
Altezza	28,00
Rotazione	0,00
▼ Proprietà personalizzate	
id	994af2d7-4b8b-4b6b-8931-7...
manuali	<input checked="" type="checkbox"/>

Figura 28-Screenshot Proprietà Personalizzate

Terminato il processo, salvare la mappa nel formato JSON (di default il programma salva le mappe in formato TMX) e inserirla nel database tramite la **Scheda Mappe** della **Pagina Admin** (paragrafo 3.5), cliccando il pulsante modifica relativo alla mappa che si vuole modificare oppure, se vogliamo aggiungere una nuova mappa, basterà cliccare sul pulsante aggiungi della Scheda Mappe.

6 Conclusioni e Sviluppi Futuri

Grazie a questo studio, si è cercato di ampliare e migliorare una piattaforma precedentemente creata da due miei colleghi del Corso di Laurea in Informatica di UNICAM nel loro lavoro di tesi. Si è cercato di renderla il più possibile sicura aggiungendo sistemi di sicurezza in fase di registrazione e di login e si è cercato di agevolare l'utente implementando anche un sistema per aiutarlo a recuperare la password dell'account. Inoltre è stato dato maggiore potere amministrativo agli utenti con tale privilegio, dandogli la possibilità di gestire una vera e propria *mailing list* in grado di contattare uno o più utenti registrati alla piattaforma. È stata creata anche una nuova stanza all'interno del gioco dove gli utenti, scaricando i manuali inseriti e aggiornati dagli admin, possono integrare i loro studi o soddisfare le loro curiosità su argomenti che variano dai sistemi operativi alla sicurezza informatica e al mondo hacking. Essendo questo un mondo che mi appassiona, ho trovato soddisfacente e gratificante contribuire allo sviluppo di questo software, ritenendolo uno strumento di apprendimento impegnativo e, allo stesso tempo, divertente per qualsiasi persona voglia iscriversi alla piattaforma.

Una delle più grandi difficoltà che ho riscontrato durante il mio lavoro di tesi è stato quello di non riuscire ad implementare un sistema in grado di inserire i manuali direttamente dalla pagina Admin all'interno della mappa in cui si vuole visualizzarli; senza passare per il *tool* Tiled Map Editor. Quindi, in futuro, mi piacerebbe trovare e sviluppare una soluzione per ovviare al problema.

7 Ringraziamenti

Sono finalmente giunto al termine di questo percorso di studi; inizio quindi con il ringraziare il prof. **Fausto Marcantoni** per avermi dato la possibilità di lavorare su questa tesi e per tutti i consigli, aiuti e suggerimenti che mi ha dato.

Percorso che, purtroppo, non è mai stato lineare e breve; anzi, di curve e di chilometri ne sono stati macinati parecchi, l'asfalto era sconnesso e ho anche sbagliato strada. Nonostante tutto sono riuscito a rimettermi sulla retta via e a concludere questo viaggio. Non ho nulla da recriminare o da rimpiangere, né tantomeno posso prendermela con qualcuno: dopotutto sono stato io il navigatore! Ho dovuto comprare uno zaino gigante per portare con me tutte le esperienze, belle e brutte, che questa "avventura" mi ha donato; e vi posso assicurare che ne sono state parecchie, sia dall'uno che all'altro lato.

Allora come non iniziare i ringraziamenti se non da chi questo viaggio l'ha permesso e che come me e, probabilmente, peggio, l'ha vissuto: la mia **famiglia**. Un grazie veramente di CUORE a mio babbo **Carlo**, a mia mamma **Lidia**, a mia nonna **Maria** e a mio fratello **Alessandro**; anche se, oltre ai ringraziamenti, sarebbero state più giuste le mie scuse.

Anche se, come detto precedentemente, sono stato io il navigatore, questo non è stato un viaggio in solitaria: non mi è mai piaciuto andare da solo, infatti il mio primo anno universitario è stato un fallimento sociale, ma un successo per quanto riguarda gli studi (sì, due domande me le sono fatte!); anzi, la mia forza è sempre stata quella di circondarmi di un gruppo di persone fantastiche. Questo l'ho capito sin dall'infanzia; infatti il mio prossimo ringraziamento va a chi mi sopporta praticamente dai banchi dell'asilo, che ora sono diventati banchi decisamente più grandi... banconi! Grazie ai **Compari del Campari: Andrea, Edoardo, Francesco, Jonathan, Luca, Marco, Matteo, Michele, Riccardo**, (dato che ne sono tre metto anche i cognomi: **Angelini, Molinas, Sforza**), **Simone** e al sempre presente amico di Roma **Mattia**.

Quindi, dopo un breve tratto di strada lineare e solitaria, sulla strada ho notato un gruppo di ragazzi che avevano bisogno di un passaggio. Da lì è nata qualcosa di più di un'amicizia e siamo tutti diventati navigatori dello stesso viaggio: ogni curva, ogni buca, ogni sterrato e ogni rettilineo lo abbiamo fatto insieme; ognuno con il suo punto di vista, ma sempre e comunque uniti. Se il percorso che ho fatto è stato incredibilmente bello, è perché ho guidato insieme a voi. Grazie veramente tanto al famigerato gruppo **Giggett lu Porc: Artur, Daniele, Danilo, Daris, Marco, Michele, Riccardo**. Per le strade di Camerino, abbiamo trovato anche altre splendide persone, alcune sono diventate splendide compagne per i miei amici, altre hanno sposato a pieno la nostra causa e sono diventati splendidi compagni di serata: grazie a voi **Erika, Marika, Giulia, Sara, Fabio, Elena, Laura, Omaina, Claudio, Francesco, Elia, Stefano, Luca**, e tanti altri che, un po' per dimenticanza e un po' per non rendere la tesi un elenco di nomi, non

ho citato. Un ringraziamento speciale alla mia coinquilina **Rebecca**, perché tenere a bada me, Daniele e Michele è un'impresa a dir poco titanica.

Finito questo viaggio, ho fermato la macchina e mi sono fermato a guardare le stelle; il mio pensiero si è subito rivolto a voi. Mi piace pensare che tutti insieme, tra una mano sugli occhi per una cavolata che ho fatto e una grassa risata, siate orgogliosi di me per questo piccolo traguardo. Un bacio a nonno **Gildo**, nonna **Ottavia** e nonno **Rinaldo**.

È già tempo di risalire in macchina, un nuovo viaggio mi attende.

il vostro
Edoardo Piergentili

8 Riferimenti Bibliografici e Sitografici

8.1 Bibliografia

- [1] Jaddi Brahim, *Hack To Win Capture The Flag by UNICAM*, Università degli Studi di Camerino, Scuola di Scienze e Tecnologie, Corso di laurea in Informatica, tesi di laurea triennale (anno accademico 2015/2016);
- [2] Shahid Shaikh *Mastering RethinkDB*, Packt Publishing - ebooks Account (16 dicembre 2016);
- [3] Marc Wandschneider, F. Conigliaro (traduttore) *Node.js. Creare applicazioni web in JavaScript*, Apogeo (20 novembre 2013);

8.2 Sitografia

- [4] Sito di HTML.it dove imparare le basi per **JavaScript** (www.html.it/guide/guida-javascript-di-base/);
- [5] Sito di HTML.it dove imparare le basi per **Node.js** (www.html.it/guide/guida-nodejs/);
- [6] Sito dove scaricare **Node.js** (nodejs.org/it/);
- [7] Sito del pacchetto di Node.js **Nodemailer** (www.nodemailer.com);
- [8] Sito di HTML.it dove imparare le basi per l'**HTML** (www.html.it/guide/guida-html/);
- [9] Sito di HTML.it dove imparare le basi per **CSS** (www.html.it/guide/guida-css-di-base/);
- [10] Sito del framework **Bootstrap** (www.getbootstrap.com);
- [11] Sito di HTML.it dove imparare le basi per **jQuery** (www.html.it/guide/guida-jQuery/);
- [12] Sito per i template **Template Ejs** (www.embeddedjs.com);
- [13] Sito del framework **Express.js** (www.expressjs.com);
- [14] Sito di **Angular.js** (www.angularjs.org);
- [15] Sito di **Passport.js** (www.passportjs.org);
- [16] Sito della libreria **Socket.IO** (socket.io);
- [17] Sito del tool **Tiled Map Editor** (www.mapeditor.org);
- [18] Sito del database **Redis** (redis.io);
- [19] Sito del database **RethinkDB** (www.rethinkdb.com);
- [20] Sito del pacchetto di Node.js **svg-captcha** (www.npmjs.com/package/svg-captcha);
- [21] Sito del **reCAPTCHA** di Google (developers.google.com/recaptcha/);