

Università degli studi di Camerino

Scuola di Scienze e Tecnologie

*Corso di Laurea in Informatica*



**Hack To Win  
Capture The Flag  
by UNICAM**

*Laureando*

**Brahim Jaddi**

*Matricola: 090378*

*Relatore*

**Prof. Fausto Marcantoni**

---

*Anno Accademico 2015/2016*

# Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
1.1	Che cos'è la sicurezza informatica? . . . . .	3
1.2	Che cosa sono i CTF? . . . . .	3
1.3	Obiettivi . . . . .	3
<b>2</b>	<b>Strumenti Utilizzati</b>	<b>4</b>
2.1	HTML[1] . . . . .	4
2.2	JavaScript[2] . . . . .	4
2.3	jQuery[3] . . . . .	4
2.4	CSS[4] . . . . .	4
2.5	Bootstrap[6] . . . . .	5
2.6	Template Ejs[5] . . . . .	5
2.7	Angular.js[7] . . . . .	5
2.8	Node.js[8] . . . . .	6
2.9	Express.js[9] . . . . .	7
2.10	Passport.js[10] . . . . .	7
2.11	Socket.io[11] . . . . .	8
2.12	Redis[13] . . . . .	8
2.13	Tiled map editor[14] . . . . .	8
2.14	Rethinkdb[12] . . . . .	9
	2.14.1 Requisiti del database . . . . .	9
	2.14.2 Diagramma ER . . . . .	10
	2.14.3 Schema logico . . . . .	11
	2.14.4 Query . . . . .	11
<b>3</b>	<b>Sito</b>	<b>18</b>
3.1	Pagina Log in . . . . .	19
3.2	Pagina Registrazione . . . . .	20
3.3	Pagina Home . . . . .	21
3.4	Pagin Profilo . . . . .	22
3.5	Pagina Admin . . . . .	25
	3.5.1 Scheda Utenti . . . . .	25
	3.5.2 Scheda Missioni . . . . .	27
	3.5.3 Scheda Mappe . . . . .	28
<b>4</b>	<b>Realizzazione del Gioco</b>	<b>30</b>
4.1	Parsing della Mappa . . . . .	30
4.2	Algoritmo del Pittore e Collisioni . . . . .	31
4.3	Comandi di gioco . . . . .	32

4.4	Animazione . . . . .	34
4.5	Missioni . . . . .	35
4.5.1	Finestra Missione . . . . .	35
4.5.2	Parsing e Stampa del Testo . . . . .	36
4.5.3	Inizio Missione . . . . .	38
4.5.4	inserimento flag . . . . .	38
<b>5</b>	<b>Realizzazione delle Missioni</b>	<b>39</b>
5.1	Linux Basis . . . . .	40
5.2	Web Security . . . . .	40
5.3	crittografia . . . . .	41
<b>6</b>	<b>Manuale</b>	<b>42</b>
6.1	Installazione . . . . .	42
6.2	Avvio dell'applicazione web . . . . .	42
6.3	Creazione della Mappa . . . . .	43
6.4	Aggiunta di una nuova Missione . . . . .	43
<b>7</b>	<b>Sviluppi Futuri</b>	<b>46</b>
<b>8</b>	<b>Ringraziamenti</b>	<b>46</b>

# 1 Introduzione

## 1.1 Che cos'è la sicurezza informatica?

La **sicurezza informatica** è un argomento di vasta portata che tocca molti aspetti dell'attività individuale nell'ambito delle tecnologie dell'informazione e della comunicazione. Imparare la sicurezza informatica permette di fornire competenze necessarie per l'analisi delle minacce, delle vulnerabilità e del rischio associato all'uso delle tecnologie informatiche al fine di proteggerli da possibili attacchi. Un modo divertente per apprendere le basi della sicurezza informatica, è giocare ai CTF (Capture the flag).

## 1.2 Che cosa sono i CTF?

I **CTF** (Capture the flag) sono dei giochi, in genere fatti a livelli, dove l'utente ha lo scopo di trovare un valore (in genere è una stringa alfanumerica del tipo "as78msa") all'interno di un server apposito creato con delle falle, allo scopo di far mettere in pratica all'utente le proprie conoscenze nell'ambito informatico o insegnare le basi della sicurezza informatica. Ogni valore trovato, di solito, rappresenta la password che serve per accedere al livello successivo. Ad ogni livello, in genere, viene assegnato un punteggio che indica la difficoltà dello stesso.

## 1.3 Obiettivi

Gli obiettivi della tesi sono:

- Creare un gioco CTF che, tramite il controllo di un personaggio, ci permette di muoverci e cercare all'interno di un mondo inventato, delle missioni che hanno lo scopo di insegnare le basi della sicurezza informatica.
- Creare una applicazione web contenente il gioco, che ci permette:
  - La gestione degli utenti;
  - L'aggiunta di mappe per il gioco;
  - L'aggiunta di missione per il gioco;
- Configurare un server linux che conterrà:
  - Livelli per apprendere le basi del sistema operativo linux;
  - Livelli per apprendere le vulnerabilità delle applicazioni web;
  - Livelli per apprendere le basi della crittografia;

## 2 Strumenti Utilizzati

### 2.1 HTML[1]

L'**HTML** (HyperText Markup Language) è il linguaggio solitamente usato per i documenti ipertestuali disponibili sul World Wide Web. Non è un vero linguaggio di programmazione in quanto non prevede alcuna definizione di variabili, funzioni, strutture dati o strutture di controllo, ma serve a descrivere il contenuto logico di una pagina web attraverso appunto i cosiddetti tag. L'uso del linguaggio HTML puro presenta però due grossi problemi: la staticità e l'inefficienza delle sue tecniche per l'impostazione del layout. Fortunatamente HTML supporta anche l'inserimento di script esterni e il ricorso a linguaggi alternativi per supplire a questi due punti deboli.

### 2.2 JavaScript[2]

**JavaScript** è un linguaggio di scripting orientato agli oggetti comunemente usato nei siti web. La caratteristica principale di JavaScript è quella di essere un linguaggio interpretato: quando viene visitata una pagina web, infatti, il codice JavaScript contenuto in essa viene portato in memoria ed eseguito dall'interprete del browser. Il codice viene quindi eseguito direttamente sul client e non sul server.

### 2.3 jQuery[3]

**jQuery** è una libreria di funzioni JavaScript, cross-browser per le applicazioni web che si propone come obiettivo quello di semplificare la programmazione lato client delle pagine HTML. A tale scopo fornisce metodi e funzioni per gestire al meglio gli aspetti grafici e strutturali come l'accesso e la manipolazione degli elementi e dei loro attributi, il riconoscimento e la propagazione degli eventi e gli effetti che questi comportano, e molto altro ancora, il tutto mantenendo la compatibilità tra browser diversi e standardizzando gli oggetti messi a disposizione dall'interprete JavaScript del browser.

### 2.4 CSS[4]

Il **CSS** (Cascading Style Sheets o Fogli di stile) è un linguaggio informatico usato per definire la formattazione di pagine HTML, XHTML e XML. Più precisamente, permette di separare i contenuti di un documento dalla sua formattazione. In questo modo non solo il codice risulta più pulito e leggibile, ma anche la sua manutenzione viene notevolmente semplificata.

## 2.5 Bootstrap[6]

**Bootstrap** è un framework front-end sviluppato per velocizzare la creazione della parte grafica per il front-end delle applicazioni web e siti, fornendo delle strutture basi (Layout). Esso include, oltre al CSS e HTML di base, icone, form, bottoni, tabelle, layout, ecc.

## 2.6 Template Ejs[5]

Per modificare dinamicamente le pagine HTML, o fare dei controlli direttamente sulle pagine a seconda dei dati ricevuti dal server, o modificare il contenuto della pagina a seconda dell'URL in cui ci troviamo ma vogliamo mantenere una formattazione grafica uguale per tutte le pagine dell'applicazione web, utilizziamo i **template**. Il tipo di template utilizzato nella nostra applicazione web è il **template ejs**, che ci permette di utilizzare all'interno delle pagine HTML, grazie ai tag `<% %>`, codice JavaScript permettendo, oltre ad utilizzare un linguaggio uniforme per tutta l'applicazione web, la gestione dei dati inviati dal server senza fare richieste AJAX.

## 2.7 Angular.js[7]

**AngularJS** è un framework, una infrastruttura per la creazione di applicazioni composta da un insieme di funzionalità. Citando la documentazione ufficiale:

*Angular è quello che HTML sarebbe dovuto essere se fosse stato progettato per sviluppare applicazioni.*

Per raggiungere questo obiettivo, AngularJS da un lato esalta e potenzia l'**approccio dichiarativo** dell'HTML nella definizione dell'interfaccia grafica, dall'altro fornisce strumenti per la costruzione di un'architettura modulare e testabile della logica applicativa di un'applicazione. Il framework AngularJS lavora leggendo prima la pagina HTML, che ha incapsulati degli attributi personalizzati addizionali (esempio: `ng-controller`). Angular interpreta questi attributi come delle direttive (comandi) per legare le parti di ingresso e uscita della pagina al modello che è rappresentato da variabili standard JavaScript. Ogni volta che queste variabili vengono modificate dinamicamente, la relativa parte HTML viene cambiata automaticamente senza dover ricaricare la pagina. L'oggetto che permette la creazione di variabili e le lega con il DOM della pagina HTML tramite la direttiva `Angularjs ng-bind` viene spesso denotato con **\$scope**.

Nel progetto angular viene utilizzato solo nella pagina dell'amministratore, permettendo il monitoraggio e la gestione in real-time dell'applicazione web, in quanto volevamo conoscere questo framework e capire le sue piene funzionalità per progetti futuri.

## 2.8 Node.js[8]

Come dice il sito di nodejs:

Node.js® è un runtime Javascript costruito sul motore JavaScript V8 di Chrome. Node.js usa un modello I/O non bloccante e ad eventi, che lo rendono un framework leggero ed efficiente. L'ecosistema dei pacchetti di Node.js, npm, è il più grande ecosistema di librerie open source al mondo.

Fig. 1: Node.js

È importante capire che Node.js non è un webserver. Non può funzionare "da solo" come Apache. Non esistono file di configurazione. Se si vuole che si comporti come un server HTTP, è necessario scriverlo il server HTTP (Con l'aiuto di librerie e altri componenti già pre-esistenti all'interno di Node) dando così un maggiore controllo sul web server. Node.js è un altro modo di eseguire codice sul computer.

La caratteristica principale di Node.js risiede nella possibilità che offre di accedere alle risorse del sistema operativo in modalità **event-driven** e non sfruttando il classico modello basato su processi o thread concorrenti, utilizzato dai classici web server.

Il modello event-driven, o "programmazione ad eventi", si basa su un concetto piuttosto semplice: si lancia una azione quando accade qualcosa. Ogni azione quindi risulta asincrona a differenza dei pattern di programmazione più comune in cui una azione succede ad un'altra solo dopo che essa è stata completata.

Ciò dovrebbe garantire una certa **efficienza delle applicazioni** grazie ad un sistema di callback gestito a basso livello dal runtime.

L'efficienza dipenderebbe dal considerare che le azioni tipicamente effettuate riguardano il **networking**, ambito nel quale capita spesso di lanciare richieste e di rimanere in attesa di risposte che arrivano con tempi che, paragonati ai tempi del sistema operativo, sembrano ere geologiche.

Grazie al **comportamento asincrono**, durante le attese di una certa azione il runtime può gestire qualcos'altro che ha a che fare con la logica applicativa, ad esempio.

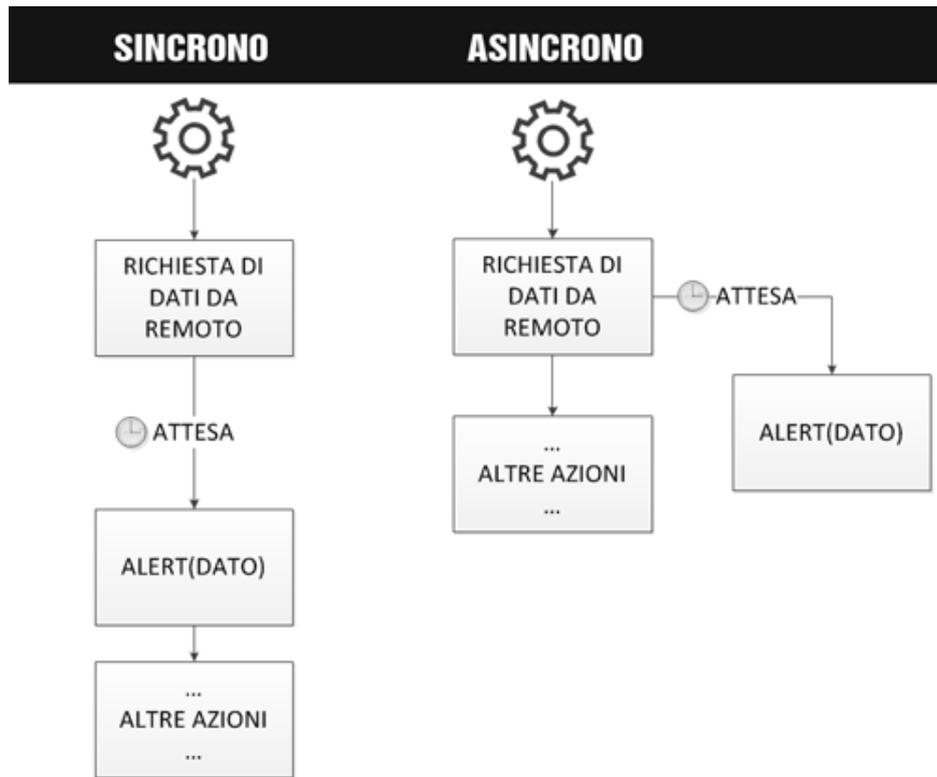


Fig. 2: Differenza tra programmazione asincrona e sincrona

Si è scelto di utilizzare node.js in quanto volevamo introdurci alla programmazione asincrona e soprattutto perché avevamo necessità di utilizzare un web server che si interfacciasse con il gioco nella maniera più veloce.

## 2.9 Express.js[9]

**Express.js** è un framework lato server che permette la creazione e la gestione di web server con la piattaforma Node.js in modo facile e veloce.

## 2.10 Passport.js[10]

**Passport.js** è un modulo di Node.js che permette la gestione dell'autenticazione locale o anche per la gestione dell'autenticazione con i social network come facebook e twitter. **Passport.js** è una funzione di middleware per Node. È stato progettato con un singolo scopo: l'**autenticazione**. Nelle applicazioni moderne, l'autenticazione può essere effettuata in modi diversi. Ad esempio con l'avvento dei social network, l'autenticazione tramite Facebook o Twitter è diventata popolare.

Passport riconosce che ogni applicazione ha requisiti di autenticazione unici. I meccanismi di autenticazione, conosciuti come **strategie**, vengono implementate come moduli indipendenti ma che sono facilmente utilizzabili da passport. Quindi le applicazioni possono scegliere quali strategie implementare, senza creare dipendenze non necessarie. Il tipo di strategia utilizzato nel progetto è di tipo **locale** (in inglese Local strategy) che permette la gestione dell'autenticazione tramite un'username e password che vengono controllati tramite una query al database rethinkdb.

## 2.11 Socket.io[11]

Un altro punto di forza in Node.js è sicuramente l'implementazione della libreria Socket.io che permette di utilizzare la tecnologia Web Socket sul browser. Un esempio tipico di questa tecnologia sono le applicazioni real-time.

Dopo aver installato questa libreria, siamo in grado di far girare la parte server in uno script Node ed intervenire sulla pagina client per mettere in comunicazione full-duplex la parte client e la parte server, passando per il protocollo WS(WSS nel caso di protocollo con connessione sicura).

## 2.12 Redis[13]

**Redis** è uno store di strutture dati chiave-valore in memoria rapido e open source. Redis 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 il caching, la gestione di sessioni, servizi pub/sub e graduatorie. Data la velocità e la semplicità di utilizzo, Redis è una scelta molto comune per applicazioni Web, videogiochi, tecnologie pubblicitarie, IoT e app per dispositivi mobili che necessitano di elevate prestazioni. Questo tipo di database, nel nostro progetto, viene utilizzato per il salvataggio delle sessioni.

## 2.13 Tiled map editor[14]

**Tiled map editor** è un tool per la creazione di mappe per GDR incentrato sull'uso dei **tileset**.

Un tileset è un insieme di immagini (tile), aventi le stesse dimensioni, raggruppate in un'unica immagine permettendo così il caricamento più veloce, dal lato server al lato client, di un'unica immagine invece dell'invio di più immagini alla volta; i tileset sono utilizzati nello sviluppo di videogiochi 2d in quanto sono molto comodi per creare immagini diverse e più grandi utilizzando i vari tile di cui è composta. Nel nostro caso i tileset sono stati utilizzati proprio per la creazione delle mappe del gioco.

Inoltre Tiled permette di inserire in queste immagini degli oggetti personalizzati

di dimensioni arbitrarie ai quali si possono associare degli attributi. La possibilità di aggiungere questi oggetti è stata fondamentale per lo sviluppo di questo gioco.

## 2.14 Rethinkdb[12]

**RethinkDB** è un database open source orientato ai documenti, cioè salva i dati in documenti JSON con schemi dinamici(cioè il modello viene creato all'aggiunta di un documento). Con lo sviluppo di RethinkDB è stato fatto un passo molto importante verso la gestione delle applicazioni real-time. Dal punto di vista dei maggiori marchi mondiali (Google, Amazon, Twitter), è complicato gestire il traffico di informazioni con i database tradizionali, che pur garantendo la consistenza e la sicurezza dei dati, non garantiscono la scalabilità e la disponibilità. La priorità è di avere un sistema che fornisce a tutti gli utenti una risposta entro un certo tempo definito, real-time, e che riesca a gestire un flusso enorme di dati ogni secondo. RethinkDB permette di apportare gli aggiornamenti alle applicazioni nel modo più veloce possibile, e renderli immediatamente visibili agli utenti.

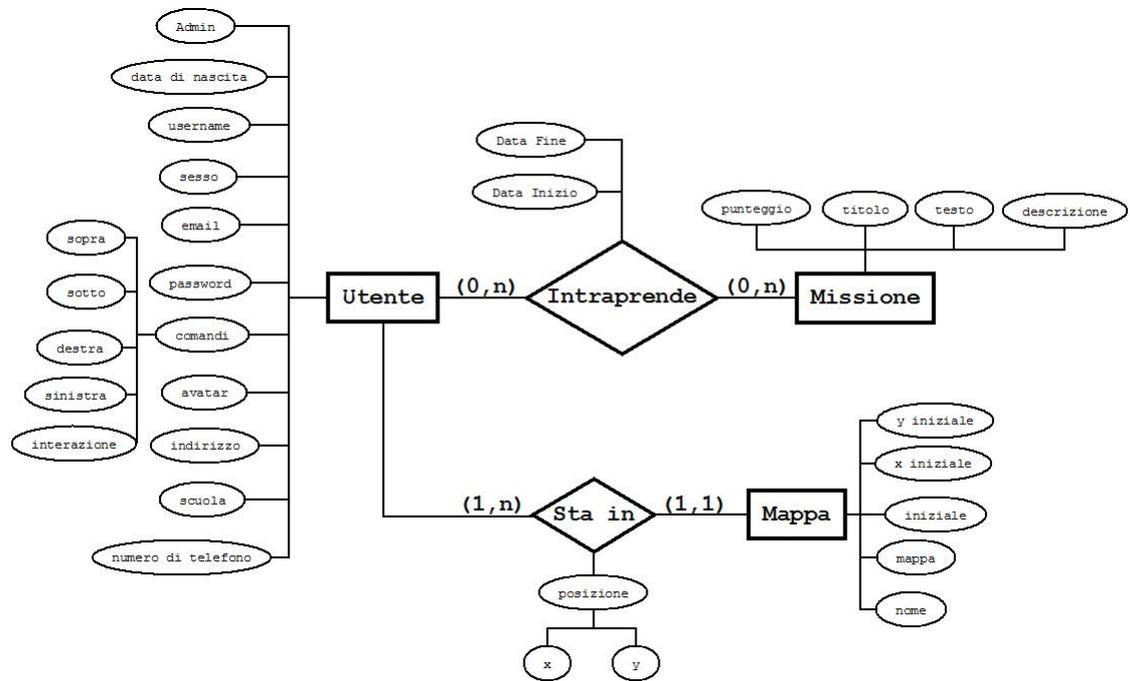
Rethinkdb è stato scelto perchè necessitavamo di un database che si interfacciasse con il lato client nella maniera più veloce possibile permettendo al gioco di interagire con il database in real-time per il salvataggio della posizione del personaggio e per la ricezione e controllo delle missione che l'utente deve affrontare.

### 2.14.1 Requisiti del database

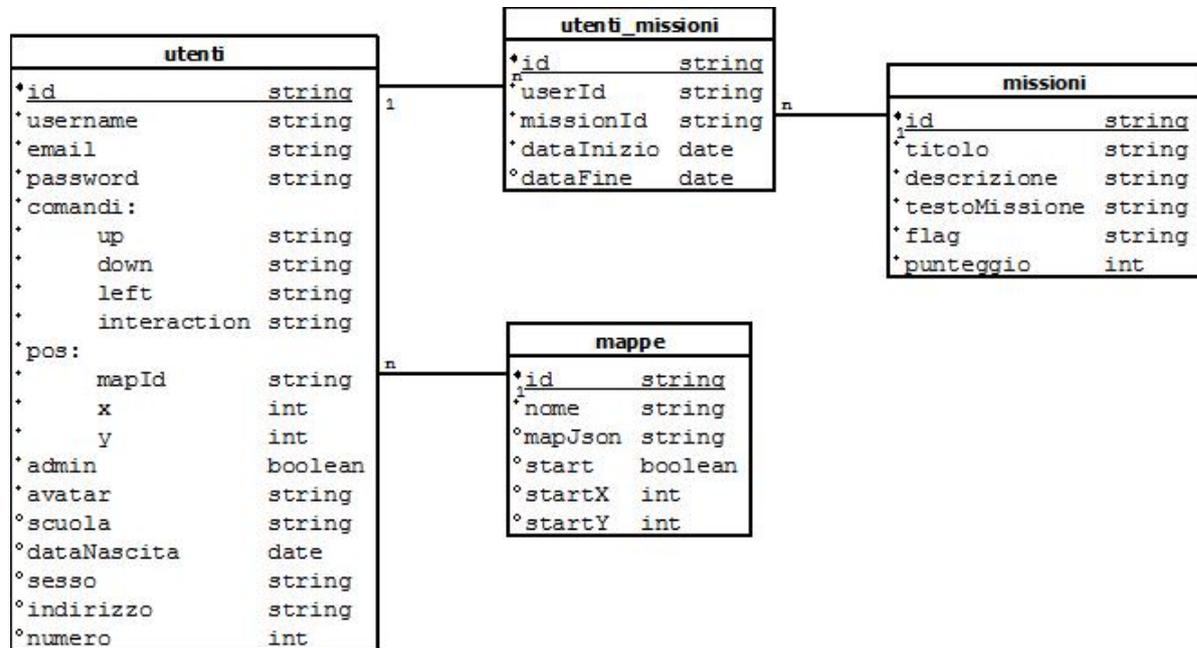
Le informazioni che devono essere salvate permanentemente nel database sono:

- Dati relativi all'utente per l'iscrizione ed il login (email,password e username).
- Dati personali facoltativi relativi ad ogni utente (scuola frequentata, data di Nascita, sesso, indirizzo, numero di telefono.
- Comandi di gioco dell'utente.
- Mappe di gioco.
- Missioni con i dati titolo, descrizione, testo della missione, flag per completare la missione e punteggio relativo.
- Missioni affrontate da ogni utente con la data in cui è stata iniziata e la data in cui è stata finita (facoltativa).

## 2.14.2 Diagramma ER



### 2.14.3 Schema logico



### 2.14.4 Query

Le query che sono state utilizzate per accedere al database sono le seguenti:  
Conta il numero di missioni iniziate:

```
1 exports.missioniIniziate = function(id) {
2   User.get(id).getJoin({ userDate: true }).run().then(function(missioni) {
3     return missioni.length;
4   })
5   .error(function(err) {
6     return err;
7   });
8 }
```

Conta il numero di missioni finite:

```
1 exports.missioniFinite = function(id) {
2   UserMissioni.filter({ userId: id })
3     .hasFields("dataFine").run().then(function(missioni) {
4     return missioni.length;
5   })
6   .error(function(err) {
7     return err;
8   });
9 }
```

Restituisce tutte le missioni inserite nel database:

```
1 exports.getAllMission = function(callback) {
2   Missioni.run().then(function(mappe) {
3
4     return callback(mappe);
5
6   }).error(function(err) {
7
8     throw err;
9   });
}
```

```

9
10     });
11 }

```

Restituisce la missione con uno specifico id:

```

1 exports.getMission = function(id) {
2     //20/12/2016 return non c'era
3     return Missioni.get(id).run().then(function(missione) {
4         return missione;
5     })
6     .error(function(err) {
7         throw err;
8     });
9 }

```

Aggiunge una missione al database:

```

1 exports.getMission = function(id) {
2     //20/12/2016 return non c'era
3     return Missioni.get(id).run().then(function(missione) {
4         return missione;
5     })
6     .error(function(err) {
7         throw err;
8     });
9 }

```

Aggiorna i dati una missione:

```

1 exports.updateMission = function(id, mission) {
2     return Missioni.get(id).update(mission);
3 }

```

Elimina una missione:

```

1 exports.deleteMission = function(id) {
2     Missioni.get(id).delete().run();
3     UserMissioni.filter({ missionId: id }).delete().run();
4 }

```

Calcola il punteggio totale di un utente:

```

1 exports.punteggioTot = function(id) {
2     UserMissioni.hasFields("dataFine").filter({ userId: id })
3     .getJoin({ missione: true }).run()
4     .then(function(usermissioni) {
5         var somma = 0;
6         usermissioni.forEach(function(element, index) {
7             somma += element.missione.punteggio;
8         });
9         return somma;
10    })
11    .error(function(err) {
12        return err
13    });
14 }

```

Restituisce tutte le missioni affrontate dall'utente:

```

1 exports.getUserMissions = function(id, callback) {
2     UserMissioni.filter({ userId: id }).getJoin({ missione: true }).run()
3     .then(function(missioni) {
4         var tempo = new Array();
5         missioni.forEach(function(element, index) {
6             if (element.dataFine) {
7                 var ms = Math.abs(element.dataFine - element.dataInizio);
8                 tempo.push({
9                     nomeMissione: element.missione.titolo,
10                    punteggio: element.missione.punteggio,
11                    dataInizio: element.dataInizio,
12                    dataFine: element.dataFine,
13                    giorni: giorni(ms),
14                    ore: ore(ms),
15                    minuti: minuti(ms),

```

```

16         secondi: secondi(ms)
17     });
18     } else {
19         tempo.push({
20             nomeMissione: element.missione.titolo,
21             punteggio: element.missione.punteggio,
22             dataInizio: element.dataInizio,
23         });
24     }
25 });
26     callback(tempo);
27 })
28     .error(function(err) {
29         throw err;
30     });
31 }

```

Restituisce le informazioni necessarie per compilare la tabella statistiche nella pagina Profilo dell'utente relativa ad un giocatore:

```

1 exports.statistiche = function(id, callback) {
2     UserMissioni.filter({ userId: id }).hasFields("dataFine").getJoin({ missione: true }).run()
3     .then(function(missioni) {
4         var tempo = new Array();
5         missioni.forEach(function(element, index) {
6             var ms = Math.abs(element.dataFine - element.dataInizio);
7             tempo.push({
8                 nomeMissione: element.missione.titolo,
9                 punteggio: element.missione.punteggio,
10                dataInizio: element.dataInizio,
11                dataFine: element.dataFine,
12                giorni: giorni(ms),
13                ore: ore(ms),
14                minuti: minuti(ms),
15                secondi: secondi(ms)
16            });
17        });
18        callback(tempo);
19    })
20    .error(function(err) {
21        throw err;
22    });
23 }

```

le funzioni **giorni()** **ore()** **minuti()** **secondi()** utilizzate nella query **statistiche** sono le seguenti:

```

1 function giorni(m) {
2     return Math.floor(m / (24 * 60 * 60 * 1000));
3 }
4
5 function ore(m) {
6     return Math.floor((m - giorni(m) * 24 * 60 * 60 * 1000) / (60 * 60 * 1000));
7 }
8
9 function minuti(m) {
10    return Math.floor(((m - giorni(m) * 24 * 60 * 60 * 1000) - (ore(m) * 60 * 60 * 1000)) / (60 * 1000));
11 }
12
13 function secondi(m) {
14    return Math.floor(((m - giorni(m) * 24 * 60 * 60 * 1000) - ore(m) * 60 * 60 * 1000 - minuti(m) * 60 * 1000) /
15    1000);
16 }

```

Query che permette di modificare l'avatar dell'utente:

```

1 // query che modifica l'avatar dell'utente
2 exports.changeAvatar = function(data) {
3     User.get(data.id).update({avatar:data.avatar}).run();
4 }

```

Query che salva la posizione del giocatore:

```

1 exports.savePosition = function(id, position) {
2     User.get(id).update({ pos: position }).run();
3 }

```

Query che aggiorna i dati di una mappa:

```
1 exports.updateMap = function(id, map) {
2     return Mappe.get(id).update(map).run();
3 }
```

Query che permette di inserire una nuova mappa nel database:

```
1 exports.addMap = function(map) {
2     var mappa = new Mappe(map);
3     return mappa.save();
4 }
```

Query che rende tutte le mappe che vengono considerate come iniziali non più iniziali. La mappa iniziale è quella mappa che viene associata automaticamente ad un utente non appena questo si iscrive per la prima volta:

```
1 exports.removeStartMap = function() {
2     r.table(Mappe.getTableName()).replace(r.row.without('start')).run();
3     r.table(Mappe.getTableName()).replace(r.row.without('startX')).run();
4     r.table(Mappe.getTableName()).replace(r.row.without('startY')).run();
5 }
```

Query che elimina una mappa dal database:

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

Query che restituisce i dati della mappa iniziale di gioco:

```
1 // query che prende la mappa iniziale
2 var getStartingMap = function() {
3     return Mappe.filter({ start: true }).run();
4 }
```

Query che restituisce tutte le mappe del gioco:

```
1 // query che prende tutte le mappe
2 exports.getAllMaps = function(callback) {
3     Mappe.run().then(function(mappe) {
4         return callback(mappe);
5     }).error(function(err) {
6         console.log(err);
7     });
8 }
```

Query che restituisce la posizione dell'utente:

```
1 exports.getUserPosition = function(id) {
2     return User.get(id).pluck('pos').run();
3 }
```

Query che restituisce tutti gli utenti iscritti al sito:

```
1 exports.getAllUsers = function() {
2     return r.table(User.getTableName()).without("password").run();
3 }
```

Query che permette di registrare un utente al sito:

```
1 // Funzione che permette all'utente di registrarsi con i dovuti controlli
2 exports.addUser = function(req, res) {
3
4     // controlli campi
5     req.checkBody('username', 'E\' richiesto un username').notEmpty();
6     req.checkBody('email', 'E\' richiesta una e-mail').notEmpty();
7     req.checkBody('email', 'L\'email non valida').isEmail();
8     req.checkBody('password', 'E\' richiesta una password').notEmpty();
9     req.checkBody('password2', 'Le Password non corrispondono').equals(req.body.password);
```

```

10
11 var errors = req.validationErrors();
12 if (errors) {
13     res.render('../register', {
14         errors: errors
15     });
16 } else {
17     getStartingMap().then(
18         function(map) {
19             var utente = new User({
20                 username: req.body.username,
21                 email: req.body.email,
22                 password: bcrypt.hashSync(req.body.password, 10),
23                 comandi: {
24                     up: "w",
25                     down: "s",
26                     left: "a",
27                     right: "d",
28                     interaction: "e"
29                 },
30                 pos: {
31                     mapId: map.id,
32                     x: map.startX,
33                     y: map.startY
34                 },
35                 avatar: "player.png",
36                 admin: false
37             });
38             User.filter({ username: utente.username }).run()
39                 .then(function(result) {
40                     if (result.length != 0) {
41                         res.render('../register', { error_msg: "Il nome utente è già stato
42                             utilizzato." });
43                     } else {
44                         User.filter({ email: utente.email }).run()
45                             .then(function(result) {
46                                 if (result.length != 0) {
47                                     res.render('../register', { error_msg: "l'email è già
48                                         stata utilizzata." });
49                                 } else {
50                                     utente.save().then(function(result) {
51                                         req.flash('success_msg', 'Registrazione avvenuta
52                                             con successo.');
```

Query che permette di modificare la password di un utente:

```

1 // query per la modifica delle informazioni relative all'utente
2 exports.updateUserPassword = function(data, socket) {
3     User.get(data.id).run().then(function(user) {
4         if (!bcrypt.compareSync(data.currentPassword, user.password)) {
5             socket.emit("Error_msg", "La password corrente inserita è sbagliata.\n Quindi non è stato
6                 possibile modificare la password.");
7         } else {
8             User.get(data.id).update({ password: bcrypt.hashSync(data.newPassword, 10) }).run();
9         }
10 })

```

Query che permettere di modificare l'username di un utente:

```

1 exports.updateUsername = function(id, user, socket) {
2     User.get(id).run().then(function(utente) {
3         if (utente.username != user.username) {
4             User.filter({ username: user.username }).run().then(function(result) {
5                 if (result.length != 0) {
6                     socket.emit("Error_msg", "L'username inserito è già stato utilizzato.\n Quindi non è
7                         stato possibile modificare l'username.");
8                 } else {
9                     User.get(id).update({ username: user.username }).run();

```

```

10         })
11     }
12 })
13 }

```

Query che permette di modificare l'indirizzo e-mail di un utente:

```

1 exports.updateEmail = function(id, user, socket) {
2     User.get(id).run().then(function(utente) {
3         if (utente.email != user.email) {
4             User.filter({ email: user.email }).run().then(function(result) {
5                 if (result.length != 0) {
6                     socket.emit("Error_msg", "L'e-mail inserita è già stata utilizzata.");
7                 } else {
8                     User.get(id).update({ email: user.email }).run();
9                 }
10            })
11        }
12    })
13 }

```

Query che permette di modificare le informazioni di un utente:

```

1 exports.updateUserInformation = function(id, user) {
2     User.get(id).update(user).run();
3 }

```

Query che aggiunge un utente amministratore:

```

1 exports.addAdmin = function() {
2     // controlli campi
3     getStartingMap().then(function(map) {
4         var admin = new User({
5             username: "Admin",
6             email: "no@reply.it",
7             password: bcrypt.hashSync("admin", 10),
8             comandi: {
9                 up: "w", //w
10                down: "s", //s
11                left: "a", //a
12                right: "d", //d
13                interaction: "e" //e
14            },
15            pos: {
16                mapId: map.id,
17                x: map.startX,
18                y: map.startY
19            },
20            avatar: "player.png",
21            admin: true
22        });
23        User.filter({ username: admin.username }).run()
24            .then(function(result) {
25                if (result.length != 0) {
26                    console.log('utente funder già presente nel database');
27                    // res.render('../register', { error_msg: "Il nome utente è già stato utilizzato" });
28                } else {
29                    User.filter({ email: admin.email }).run()
30                        .then(function(result) {
31                            if (result.length != 0) {
32                                console.log('l'e-mail del funder è già stata utilizzata');
33                                // res.render('../register', { error_msg: "l'email è già stata utilizzata." });
34                            } else {
35                                admin.save().then(function(result) {
36                                    req.flash('success_msg', 'Admin aggiunto con successo. ');
37                                    res.redirect('/login');
38                                }).error(function(err) {
39                                    res.render('../register', { error_msg: err });
40                                });
41                            }
42                        })
43                    }
44                })
45            })
46        .error(function(err) {
47            console.log("c'è stato un errore sull'aggiunta dell'admin:", err);
48        });

```

```
49 }
```

Query che permette di modificare i comandi di un utente:

```
1 exports.updateComandi = function(data) {
2   User.get(data.id).update({
3     comandi: {
4       up: data.up,
5       down: data.down,
6       left: data.left,
7       right: data.right,
8       interaction: data.interaction
9     }
10  }).run()
11  .then(function() {
12    console.log("comandi aggiornati");
13  })
14  .error(function(err) {
15    console.log("error:", err);
16  });
17 };
```

Query che permette di controllare se una missione sia già stata completata da un determinato utente:

```
1 exports.checkMissionAccomplished = function(idMissione, idUtente) {
2   return UserMissioni.filter({ missionId: idMissione, userId: idUtente }).nth(0).run()
3   .then(function(mission) {
4     if (mission.dataFine === undefined) {
5       return false;
6     } else {
7       return true;
8     }
9   });
10 };
```

Query che controlla se il valore datogli corrisponda con il flag della missione:

```
1 exports.checkMissionFlag = function(idMissione, flag) {
2   return Missioni.get(idMissione).run().then(function(mission) {
3     return (mission.flag == flag);
4   });
5 };
```

Query che elimina un utente dal database:

```
1 exports.deleteUser = function(id) {
2   r.table(User.getTableName()).get(id).delete().run();
3   r.table(UserMissioni.getTableName()).filter({ userId: id }).delete().run();
4 }
```

Query che permette di promuovere un utente al ruolo di amministratore:

```
1 exports.makeAdmin = function(id) {
2   r.table(User.getTableName()).get(id).update({ admin: true }).run();
3 }
```

Query che modifica il ruolo di un utente da amministratore a non amministratore:

```
1 exports.removeAdmin = function(id) {
2   r.table(User.getTableName()).get(id).update({ admin: false }).run();
3 }
```

Query che considera una missione come completata dall'utente:

```
1 exports.missionAccomplished = function(idMissione, idUtente) {
2   UserMissioni.filter({ missionId: idMissione, userId: idUtente }).nth(0).update({ dataFine: new Date() }).run()
3   ;
3 };
```

Query che permette di considerare una missione come iniziata da uno specifico utente:

```

1 exports.missionStarted = function(idMissione, idUtente) {
2     var utenteMissione = new UserMissioni({
3         userId: idUtente,
4         missionId: idMissione,
5         dataInizio: new Date()
6     });
7     utenteMissione.save().then(function(result) {
8     }).error(function(err) {
9         console.log("errore:", err);
10    });
11
12 };

```

Query che conta il numero di missioni iniziate dall'utente:

```

1 exports.countUserMissionStarted = function(idMissione, idUtente) {
2     return UserMissioni.filter({ missionId: idMissione, userId: idUtente }).count().execute()
3 };

```

Query che conta il numero di utenti nel database:

```

1 exports.countUsers = function() {
2     return User.count().execute();
3 }

```

### 3 Sito

Il web server che gestisce il tutto è stato scritto con **Nodejs**(paragrafo 2.8), con l'aiuto del framework **Expressjs**(paragrafo 2.9). Il sito gira sul protocollo HTTPS con una chiave privata e un certificato creato da me, finchè non riceverò un certificato valido per inserire il web server su un server di UNICAM.

La struttura file del sito si presenta nel seguente modo:

```

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

```

dove le cartelle **public** e **views** contengono la parte **front-end** cioè la parte visibile all'utente con cui egli può interagire e le altre cartelle e file servono per la parte

**back-end** cioè la parte che permette l'effettivo funzionamento della parte front-end.

### 3.1 Pagina Log in

La **pagina di login** è la prima pagina che viene visualizzata quando si accede per la prima volta al sito o quando non si è loggati:

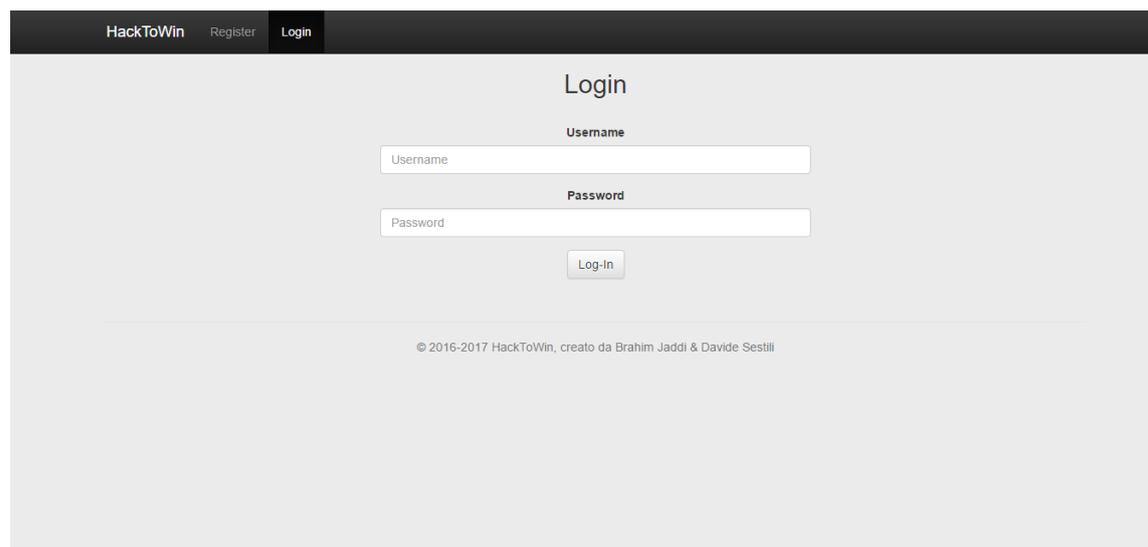


Fig. 3: Log-In Page

La pagina si presenta con due textbox<sup>1</sup> dove la prima viene utilizzata per inserirci l'username e la seconda serve per la password. Per la gestione back-end del routing vengono gestiti i seguenti metodi:

- Il metodo **GET**: quando l'utente fa la richiesta per la pagina login è necessario dire al layout di visualizzare il contenuto della pagina login.html nel proprio body con i dovuti controlli:

```
1 // Route che si attiva quando l'utente si connette alla pagina "urlsito/login"
2 // Login
3 router.get('/login',function (req, res) {
4     // controlla se l'utente è già loggato
5     if(req.user){
6         // lo reindirizza alla pagina home
7         res.redirect('/home');
8     }
9     else{
10        // inserisce nel layout , al posto di <%-body%>,la pagina login.html
11        res.render('../login');
12    }
13 });
```

<sup>1</sup>textbox sono le caselle di testo utilizzare per prendere i valori dell'utente

- Il metodo **POST**: si attiva quando l'utente clicca sul pulsante **login**. Tramite la funzione **authenticate** del modulo **Passportjs**(paragrafo 2.10) controlla se i dati inseriti corrispondono nel database Rethinkdb(paragrafo 2.14): In caso affermativo, crea la sessione relativa e la salva nel database Redis(paragrafo 2.12), reindirizzando poi l'utente alla pagina Home(paragrafo 3.3), in caso di errore l'utente viene reindirizzato alla pagina login con il relativo messaggio di errore.

```

1 // Route che si attiva quando l'utente preme il pulsante di login
2 router.post('/login', passport.authenticate('local', {successRedirect: '/home', failureRedirect: '/login',
  failureFlash: true}));

```

## 3.2 Pagina Registrazione

La pagina di **registrazione** permette ad un nuovo utente di creare un account, necessario per accedere alle funzionalità del sito con un'interfaccia facile e veloce:

Fig. 4: Pagina di Registrazione

Le informazioni richieste all'utente per la creazione dell'account sono le seguenti: Username, E-mail, Password.

I metodi gestiti lato server dal routing sono:

- Il metodo **GET**: invia il contenuto della pagina registrazione che sarà inserito all'interno del body del layout grazie al **Template Ejs**(paragrafo 2.6), se l'utente non è loggato.

```

1 // Route che si attiva quando l'utente si connette alla pagina "urlsito/register"
2 router.get('/register', function (req, res) {

```

```

3 // controlla se l'utente è già loggato
4 if(req.user){
5 // lo reindirizza alla pagina home
6 res.redirect('/home');
7 }else {
8 // inserisce nel layout , al posto di <%-body%>,la pagina register.html
9 res.render('../register');
10 }
11 });

```

- Il metodo **POST**: viene richiamato quando si invia il contenuto della form di registrazione cliccando sul pulsante **Registrati**. Questo route richiama direttamente la query **addUser** dal file `api.js`. La query si occupa di fare i controlli dei campi e in caso di errore notifica l'utente dell'errore oppure in caso dell'avvenuta registrazione invia un messaggio di successo come spiegato nel paragrafo 2.14 relativo al database. C'è solo da puntualizzare che la password non viene salvata in chiaro nel database, ma viene utilizzata la funzione **bcrypt** che genera un **hash** il quale viene salvato nel database. Ogni volta che si tenta di fare il login, prima si genera l'hash della password immessa, poi viene controllato se questo valore corrisponde a quello salvato nel database. C'è da dire che `bcrypt` non è molto veloce nel calcolare l'hash di una password, ma anche se un malintenzionato riuscisse a leggere l'hash di una password impiegherebbe anni per decriptarla, rendendo così questa funzione di hash molto sicura.

```

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

```

### 3.3 Pagina Home

La pagina **home** è una pagina statica che serve per dare informazioni generali ai nuovi utenti sul sito web e sul gioco. Questa pagina può essere visualizzata solamente quando un utente effettua il login grazie alla seguente funzione middleware **verificaAutenticazione**:

```

1 // Questa funzione richiama la funzione isAuthenticated che si trova nell'oggetto richiesta.
2 // La funzione isAuthenticated ritorna true se esiste una sessione false altrimenti
3 function verificaAutenticazione(req,res,next){
4   if(req.isAuthenticated()){
5     return next();
6   }
7   else{
8     req.flash('error_msg','Non sei loggato per visualizzare questa pagina');
9     res.redirect('/login');
10  }
11 }

```

### 3.4 Pagin Profilo

La pagina **profilo** è suddivisa in due: la prima parte relativa alle informazioni dell'utente:

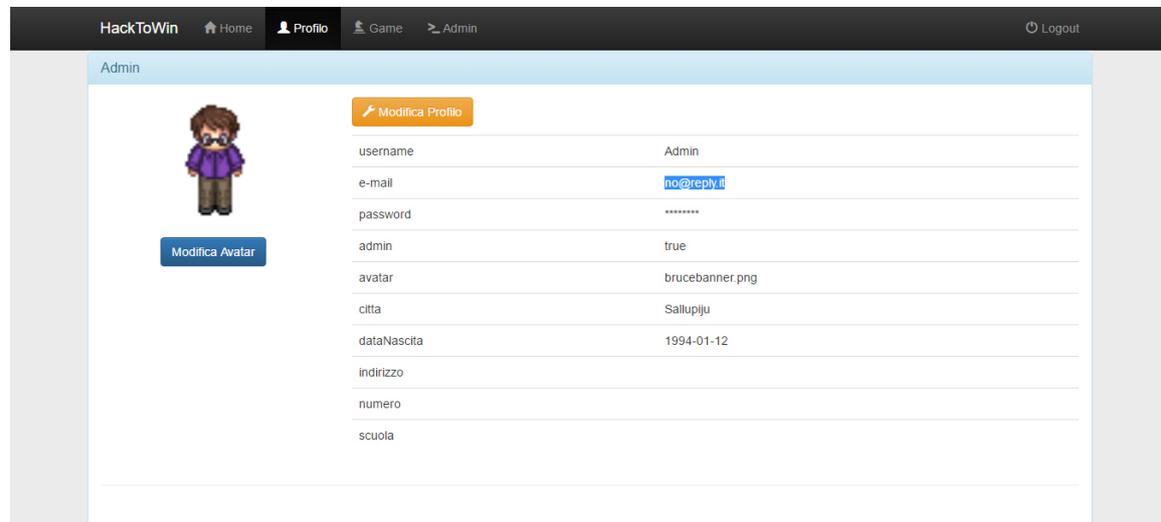


Fig. 5: Pagina Profilo:Informazioni

A sinistra è possibile vedere un'immagine, che rappresenta il corrente avatar dell'utente, con sotto un pulsante che quando viene premuto permette la scelta e la modifica dell'avatar tra un elenco preimpostato di avatar. Per la visualizzazione di questo elenco, il server si occupa della lettura e scrittura su un array del nome dei file immagini presenti nella cartella **/public/img/Avatar**, e li invia tramite l'utilizzo delle socket al client. Quando il lato front-end riceve l'array, attraverso un ciclo for, crea dinamicamente nella pagina profilo un tag img, per ciascun valore, con l'attributo src (indica url dell'immagine sul sito) avente **/img/Avatar**, concatenato al valore dell'array. Ogni tag img in più ha l'attributo onclick (l'evento che si genera quando si clicca sull'immagine) con una funzione che, tramite una socket, invia il nome dell'avatar scelto e l'id dell'utente al server. Il server quando riceve l'evento "modifica avatar" richiama la query che modifica l'avatar nel database passandogli i parametri ricevuti.

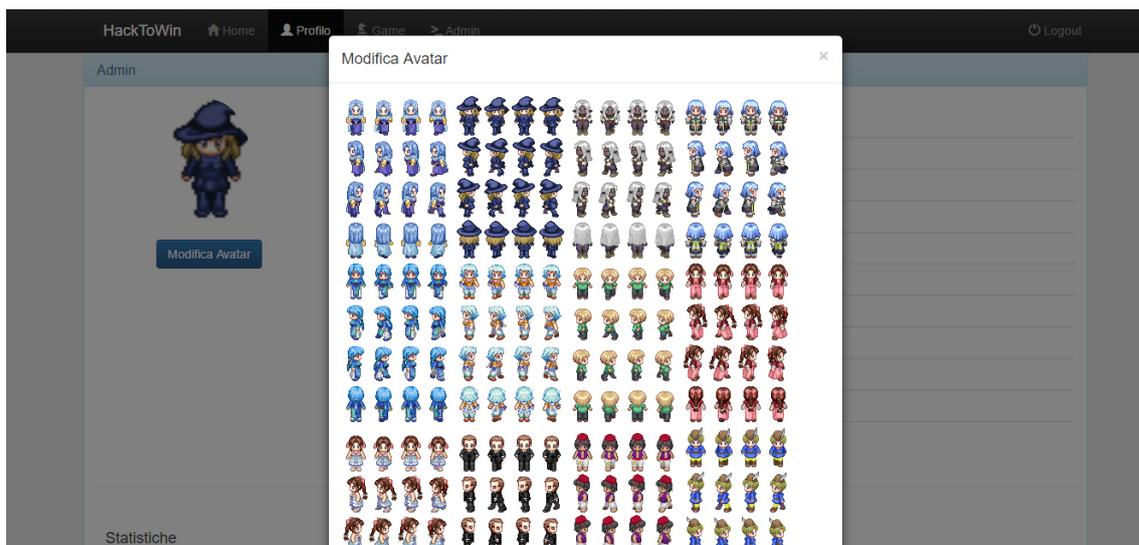


Fig. 6: Pagina Profilo: Modifica Avatar

A destra invece viene mostrata una tabella con le informazioni relative all'utente loggato. Queste informazioni possono essere modificate cliccando sul pulsante **Modifica Profilo**. I campi che si possono modificare sono:

- Username: il nome utilizzato per fare il login. La modifica può avvenire solamente se il nuovo nome è diverso da quello corrente e se non è presente nel database.
- E-mail: questo campo viene modificato nel caso in cui l'email inserita rispetti le seguenti condizioni:
  - L'e-mail nuova sia diversa da quella corrente
  - L'e-mail inserita sia un e-mail valida
  - L'e-mail non sia già presente nel database
- Password Corrente, Nuova Password, Conferma Password: questi campi servono per modificare la password dell'utente. Se il campo Password Corrente è vuoto il programma considera che non si vuole modificare la password, altrimenti fa i dovuti controlli per la modifica della password.
- Scuola: è un campo facoltativo che rappresenta il nome della scuola o università dell'utente
- Data di Nascita: è un campo facoltativo che rappresenta la data di nascita dell'utente. Per questo campo è stato utilizzato il tag input con tipo **da-**

te permettendo così all'utente di scrivere la data in modo corretto oppure utilizzando lo strumento calendario che il tag ci fornisce.

- Maschio o Femmina: rappresenta il sesso dell'utente: per l'implementazione sono stati utilizzate due **radiobox** che permettono un'unica selezione.
- Città: campo facoltativo che rappresenta il nome della città di provenienza dell'utente.
- Numero di telefono: campo facoltativo che permette l'inserimento di soli numeri fino ad un massimo di 10 che sarebbe lo standard in italia.

Infine è presente il bottone **Modifica Profilo** che quando viene premuto, richiama una funzione che, dopo aver fatto i dovuti controlli, tramite una socket, invia un oggetto contenente i valori dei campi e l'id dell'utente al server. Il server analizza l'oggetto ricevuto e se ci sono errori, tramite una socket, invia un messaggio di errore altrimenti richiama la query che si occupa della modifica delle informazioni relative all'utente passandole l'oggetto.

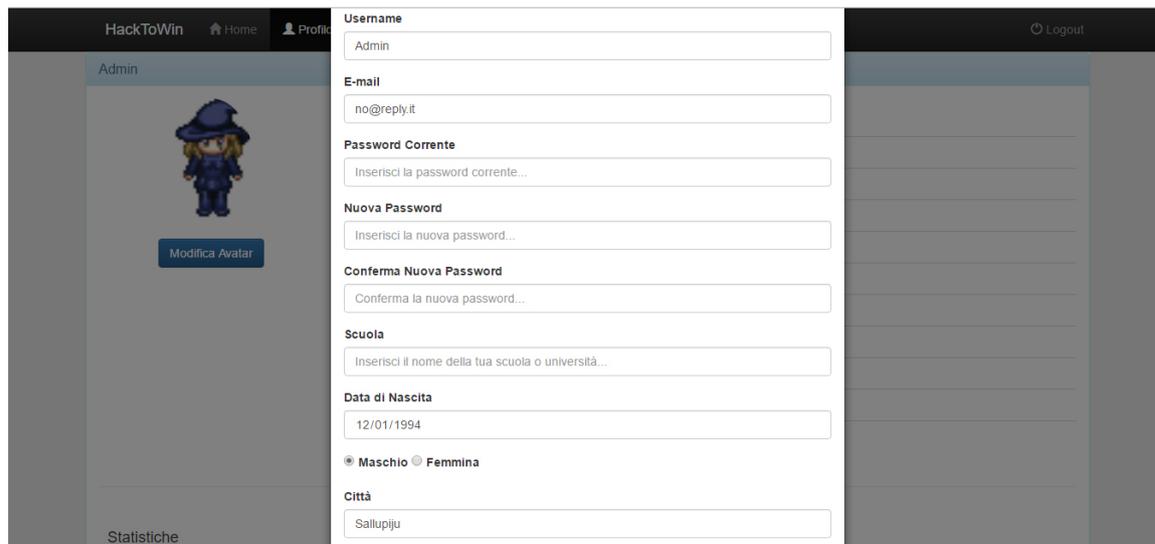


Fig. 7: Pagina Profilo: Modifica Informazioni Profilo

La seconda parte invece contiene una tabella con il nome **statistiche** che rappresenta le missioni completate dell'utente. Questa tabella viene popolata con il template ejs richiamando l'array con il nome **stat**, ricevuto quando l'utente apre la pagina profilo e contenente i dati che ha ricevuto dalla query missioni completate.

## 3.5 Pagina Admin

La pagina **Admin** è visibile solo agli amministratori. In essa sono presenti tre schede, accedibili da una barra di navigazione laterale. Ogni scheda viene gestita da un **controller**<sup>2</sup> di Angularjs(2.7). Le tabelle sono state create con una libreria di Bootstrap(2.5) che ha come nome **bootstrap-table**. Essa fornisce oltre alla creazione dinamica di una tabella, una barra degli strumenti che può essere personalizzata con l'inserimento e la gestione di bottoni, una barra di ricerca e molti altri strumenti che servono per migliorare e rendere più user-friendly una tabella. Il codice angularjs che gestisce la pagina admin si trova all'interno del file `/public/app/app.js`.

### 3.5.1 Scheda Utenti

La **scheda utenti** viene utilizzata per la gestione degli utenti. Il controller che la gestisce ha nome **GetUsersCtrl**.

	Permessi	Username	IP	E-mail	Stato	Item Operate
+	★	Admin	::1	no@reply.it	●	🚩
+		braccone	-	braccone@server.it	●	🔗 🗑️ ✖️

Fig. 8: Pagina Admin: Scheda Utenti

Come possiamo vedere dalla figura, in alto sono presenti due pannelli:

- il pannello **utenti connessi**: permette il controllo in tempo reale di tutti gli utenti connessi. La variabile, che permette ciò, si chiama **\$scope.connectedUsers**

<sup>2</sup>In Angularjs(2.7) un controller viene definito come un semplice oggetto Javascript. Con esso possiamo regolare il modo in cui i dati (grezzi) confluiscono all'interno del livello di presentazione.

e viene aggiornata grazie ad un socket inviata dal server. Per tenere e aggiornare il numero degli utenti connessi, all'inizio veniva tramite una funzione il conteggio del numero di sessioni salvate nel database 2.12, ma ogni volta che un utente si disconnetteva veniva creata una sessione vuota che scade dopo un certo tempo rendendo così il conteggio sbagliato. Poi, è stata creata una variabile che ogni volta che un utente si logga viene incrementata e ogni volta che effettua il logout viene decrementata, rendendo così il conteggio corretto.

- il pannello **utenti iscritti**: permette il controllo del numero di utenti iscritti in tempo reale tramite la variabile **\$scope.iscritti** che viene aggiornata, tramite una socket, inviata dal server ogni dieci secondi.

In basso invece viene mostrata la tabella con le informazioni relative agli utenti. Essa viene aggiornata in tempo reale grazie alla variabile **\$scope.users**. Tramite la libreria bootstrap-table è possibile definire una serie di opzioni come l'aggiunta della barra di ricerca, quale barra degli strumenti devi utilizzare, che tipo di formattazione deve utilizzare e tutta una serie di opzioni necessarie per rendere la tabella più completa possibile e facile da utilizzare. La struttura base per tutte le tabelle che andremo ad utilizzare è la seguente:

```
1 $scope.users = {
2   options: {
3     rowStyle: function(row, index) {
4       return { classes: 'none' };
5     },
6     cache: false, // indica il salvataggio in cache dei dati della tabella
7     height: 526, // altezza della tabella
8     striped: false, // un tipo che rende il colore background delle righe alternato
9     pagination: true,
10    pageSize: 10,
11    pageList: [5, 10, 25, 50, 100, 200], // come devono essere distribuita la visualizzazione dei dati
12
13    //in questo caso in più pagine(5 elementi
14    //nella prima, 10 nella seconda,ecc)
15
16    search: true, // attiva la barra di ricerca
17    showColumns: true, // mostra in alto a destra che permette la scelta dei campi da visualizzare
18    showRefresh: false, // mostra il pulsante per aggiornare la tabella
19    minimumCountColumns: 2, // numero minimo di colonne da mostrare quando si raggiunge la più piccola
20    //dimensione dello schermo
21    clickToSelect: true, // permette la selezione della riga
22    showToggle: true, // mostra il pulsante toggle che permette di vedere la tabella in un altro modo
23    maintainSelected: true, // serve per mantenere il controllo delle righe selezionate
24    detailView: true, // mostra il pulsante + per vedere i dettagli
25    detailFormatter: detailFormatter, // richiama la funzione per la formattazione dei dettagli
26    // contiene i campi che si vogliono visualizzare nella tabella e le relative funzioni di
27    //formattazione
28    columns: [] // array che contiene un 'oggetto per la descrizione dei campi
29  }
30 };
```

I dati necessari per la tabella vengono, prima prelevati con la query **getAllUsers**, e poi inviati dal server con una socket. Angular.js(2.7) quando riceve l'evento socket assegna i dati ricevuti alla variabile **\$scope.users.data** aggiornando così la tabella utenti in tempo reale.

I campi per la tabella sono:

- **Username.**

- **E-mail.**
- **Permessi:** se è presente il simbolo ★ allora l'utente è un amministratore.
- **IP:** indica l'indirizzo ip da dove l'utente si è connesso. Se l'utente è offline allora sarà presente il simbolo -. Il server recupera l'ip grazie all'oggetto **request** della socket e lo inserisce nei dati da inviare alla tabella.
- **Stato:** indica con ● che l'utente è online e con ● che l'utente è offline.
- **Strumenti:** rappresenta i pulsanti necessari per apportare o avere più informazioni sull'utente. I pulsanti sono:
  - : quando viene cliccato, invia, con una socket, l'id dell'utente richiedendo al server di inviare le missioni completate del relativo utente. Una volta che il server esegue la query e reinvia i dati relativi, viene mostrata una finestra con una tabella relativa alle missioni completate dall'utente.
  - : questo pulsante serve per rendere un utente amministratore inviando al server, dopo un messaggio di conferma, l'id del relativo utente. Il server passa il dato ricevuto alla query **makeAdmin**.
  - : questo pulsante serve per togliere i privilegi di amministratore ad un utente amministratore. Il processo avviene grazie all'invio dell'id del l'utente che si vuole degradare tramite una socket, dopo un messaggio di conferma. Il server passa il dato ricevuto alla query **removeAdmin**.
  - : questo pulsante serve per rimuovere il relativo utente inviando, dopo una conferma, l'id relativo al server che richiamando la query **deleteUser**. Per poter eliminare più utenti alla volta, prima è necessario selezionarli in modo da attivare il pulsante  che permette la cancellazione di più utenti, inviando, dopo la conferma, gli id degli utenti selezionati al server, tramite una socket. Quest'ultimo richiama ciclicamente la query **deleteUser** sugli id ricevuti.  
**NOTA BENE:** l'utente Admin non può essere eliminato in quanto è l'utente fondatore.

### 3.5.2 Scheda Missioni

La **scheda missioni** viene utilizzata per la creazione e modifica delle missioni del gioco. Il controller di angular che la gestisce ha nome **GetMissionsCtrl**.

Per creare una missione, basta semplicemente cliccare sul pulsante  che farà comparire una finestra con i campi da compilare. I campi sono:

- **Titolo:** rappresenta il titolo che verrà visualizzato nella parte superiore centrale della finestra missioni.
- **Descrizione Iniziale:** è necessaria per dare all'utente la possibilità di scegliere se iniziare la missione o meno.
- **Descrizione Effettiva:** rappresenta le informazioni necessarie per svolgere la missione.
- **flag:** rappresenta il valore che l'utente deve trovare.
- **Punteggio:** è il valore attribuito alla missione a seconda della difficoltà.

Le due descrizioni sono gestite dal parser descritto nel paragrafo 4.5.2, premettendo così l'implementazione dei link sul canvas che normalmente non sono possibili. Una volta compilati i campi cliccando sul pulsante **crea missione** vengono inviati i dati al server che tramite la query **addNewMission** aggiunge la missione nel database. Per poterla inserire effettivamente nel gioco basta semplicemente creare, con lo strumento **Tiled(2.13)**, un oggetto senza immagine con le proprietà **id** relativo alla missione creata e una proprietà booleana con il nome **missione** e valore **true** nel punto della mappa in cui si desidera inserire la missione.

È possibile modificare una missione cliccando sul pulsante  che mostrerà una finestra con i campi relativi alla missione da modificare. Una volta completato il processo di modifica, viene inviato al server l'id e campi modificati in modo che il lato back-end, richiamando la query **updateMission**, apporta le modifiche nel database. Per eliminare una missione basta cliccare sul relativo pulsante . Per l'eliminazione di più missioni contemporaneamente basta, selezionare le missioni che si desidera rimuovere, e cliccare il pulsante  ricordandosi di eliminare gli oggetti relativi sulla mappa con lo strumento **Tiled(2.13)**.

### 3.5.3 Scheda Mappe

La **scheda mappe** serve per la gestione delle mappe. Essa permette di svolgere le seguenti funzioni:

- La creazione di nuove mappe tramite il click del pulsante . Per il caricamento dell'immagini e del file è stato semplicemente utilizzato il tag `input` di HTML di tipo `file`. Per personalizzarlo graficamente è stata utilizzata la libreria **bootstrap-fileinput**.
- La modifica delle mappe tramite il pulsante .

- La rimozione di una o più mappe avviene con gli stessi tipi di pulsanti spiegati precedentemente nella scheda missioni(3.5.2). L'unica controllo specifico è che una mappa che ha l'attributo start a true non può essere cancellata finché non viene trasformata un'altra mappa iniziale. Per mappa iniziale si intende quella mappa in cui si trova l'utente quando si logga sul gioco per la prima volta. **Nota Bene:** di mappa iniziale ce ne può essere una soltanto quindi ogni volta che si vuole far diventare una mappa iniziale,prima tramite la query **removeStartMap** vengono cancellati i campi start, startx e starty nelle mappa iniziale, poi viene aggiunta o modificata la mappa che si vuole rendere iniziale.
- Il download del file JSON di una mappa tramite il pulsante ↓ che utilizza la funzione **download()** della libreria front-end **downloadjs**.

## 4 Realizzazione del Gioco

Il gioco è stato realizzato completamente da zero, cioè sono state implementate delle classi che vanno dalla gestione del personaggio fino alla stampa per ogni fotogramma degli elementi necessari che ci danno l'effetto di gioco (come l'animazione, il senso della profondità, il senso di collisione ecc.). Il linguaggio utilizzato per l'implementazione di questo motore grafico, è il **JavaScript**(2.2). Si è utilizzato l'oggetto **canvas** fornito dal linguaggio **HTML**(2.1), che, con le relative funzioni, ci permette la stampa del gioco nel browser. Inoltre per una migliore efficienza le funzioni che gestiscono la stampa vengono ciclizzate per ogni fotogramma grazie alla funzione **requestAnimationFrame**. Questa funzione viene fornita da qualsiasi browser permettendo l'esecuzione delle funzioni, con un thread, in parallelo alla logica del gioco.

È stato scelto di gestire la logica del gioco tutta lato client, permettendo così un effetto fluidità per tutti gli utenti senza penalizzare quelli che hanno una connessione internet di basso livello.

### 4.1 Parsing della Mappa

Per poter dire a javascript di stampare e riconoscere, nel modo in cui vengono disegnate nello strumento Tiled(2.13), i diversi elementi della mappa, è stata implementata una classe che prende in input il file JSON generato, lo analizza, e crea, per ogni elemento della mappa un oggetto javascript, che a seconda del tipo gli verrà associata la relativa classe o verrà direttamente stampato nell'oggetto canvas. Il file JSON della mappa in generale ha le seguenti proprietà:

- **Layers:** rappresenta i livelli e l'ordine con cui devono essere stampati a video. Ogni livello contiene l'attributo **data**. Questo attributo è una matrice quadrata che contiene, per ogni elemento, l'id delle immagini da stampare. Ogni elemento rappresenta posizione a video del tile da stampare.
- **Tilesets:** rappresenta le immagini utilizzate all'interno delle mappa con il relativo nome e id.

Per una migliore efficienza invece di caricare l'immagine per ogni fotogramma richiamando le immagini dal lato server, è stata implementato il metodo **preDrawCache()** per caricare all'inizio del gioco le mappe già strutturate con le immagini assegnate, salvandole nella cache del browser web del client.

## 4.2 Algoritmo del Pittore e Collisioni

Gli oggetti presenti nelle nostre mappe si possono dividere in **solidi** ed **oggetti**. I **solidi** sono quegli oggetti che non hanno alcuna immagine associata, essi servono per controllare le collisioni del giocatore e a loro volta si suddividono in:

- **Solidi semplici:** Sono quei solidi che non hanno alcun attributo.
- **Porte:** Servono per spostare il giocatore da una stanza ad un'altra, hanno come proprietà personalizzata un attributo con il nome "porta" booleano, che indica che quella è una porta, ed un attributo di nome "id" che contiene l'id della mappa a cui è collegata quella porta.
- **Missioni:** hanno un attributo "missione" booleano che sta ad indicare che quella è una missione ed un attributo "id" che contiene l'id della missione collegata al solido. Se l'utente preme il tasto di interazione vicino ad uno di questi solidi, come verrà spiegato nel paragrafo Missioni, gli verrebbe mostrata la schermata missione ad esso collegata.
- **Edifici:** ha l'attributo "casa" booleano, un attributo chiamato "titolo" di tipo string ed un altro chiamato "descrizione" di tipo string. Quando l'utente urta uno di questi solidi nella parte bassa dello schermo comparirà una scritta che fornisce informazioni relative al contenuto di quell'edificio.

Questi solidi vengono utilizzati per la gestione delle collisioni, cioè le parti della mappa che indicano i limiti del movimento del personaggio. Per poter gestire le collisioni dell'utente con i solidi del mondo circostante è stato creato un oggetto che tra i suoi attributi ha anche una lista contenente tutti i solidi della mappa ricevuti durante il Parsing della Mappa, inserendo gli oggetti che non hanno immagini associate. Quindi il personaggio ogni volta si muove, tramite la funzione **checkCollision**, controlla che non ci sono solidi nella direzione in cui vuole muoversi. Nel caso in cui l'utente effettivamente collide con un oggetto quest'ultimo gli viene restituito così da poter controllare quali attributi abbia e vedere se sia un solido semplice, un edificio, una missione oppure una porta.

Quando il personaggio collide con un solido deve fermarsi e fare una delle seguenti azioni:

- Se il solido con cui è andato a collidere è una porta allora cambia la mappa di gioco corrente con quella con id uguale a quello della porta
- Se il solido con cui collide è un edificio allora mostra una finestra in cui sono presenti le informazioni relative a quell'edificio
- Se il solido è una missione allora permette di visualizzarla premendo il tasto di interazione

Gli **oggetti**, invece, non hanno attributi aggiuntivi a quelli di base, hanno semplicemente associato a loro un'immagine. Osservando i videogiochi in 2 dimensione con visuale dall'alto più popolari ci siamo accorti che nella maggior parte era stato implementato "l'algoritmo del pittore". Esso consiste nel capire quali elementi devono essere stampati prima di altri dando l'effetto profondità, in modo da rendere l'esperienza di gioco più verosimile e più piacevole. Quindi questi oggetti con un'immagine associata servono per dare un effetto profondità al gioco, ossia se il personaggio sta dietro ad uno di questi oggetti parte del suo corpo sarebbe coperto dallo stesso, mentre se gli sta davanti sarà parte dell'oggetto ad essere coperto dall'utente.

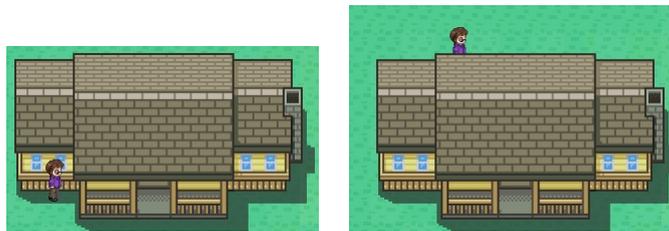


Fig. 9: Differenza tra il personaggio davanti alla casa (sinistra) e dietro (destra), come si può notare se il personaggio sta dietro la casa viene coperto da quest'ultima

Per implementare l'algoritmo del pittore abbiamo creato una classe "Oggetto" associata ad ogni mappa con un attributo "lista" che andrà a contenere tutti i dati di ogni oggetto avente un'immagine associata presente nella mappa. Sempre all'interno della classe oggetto c'è un metodo chiamato "**sortByY**" che ordina tutti gli oggetti della lista da quello la cui base sta più in alto nell'asse delle y a quello che sta più in basso. Ordinando gli oggetti in questo modo basta semplicemente stamparli sequenzialmente per poter ottenere l'effetto da noi voluto.

### 4.3 Comandi di gioco

I comandi permettono all'utente di spostarsi all'interno dell'area di gioco e di interagire con l'ambiente circostante. Questi comandi permettono di effettuare le seguenti attività:

- Muoversi verso l'alto, di default è il tasto "w";
- Muoversi verso il basso, di default "s";
- Muoversi verso destra, di default "d";
- Muoversi verso sinistra, di default "a";

- Interagire con l'ambiente di gioco, ossia iniziare le missioni. Questa attività è associata di default al tasto "e";

L'effetto movimento è dato grazie alle seguenti tipologie:

- Il primo caso avviene quando ci troviamo alle estremità della mappa. Quando viene premuto un pulsante di movimento viene semplicemente modificata l'immagine relativa al personaggio di un pixel per ogni fotogramma.
- Come possiamo notare, nel canvas viene stampata solo una porzione della mappa. Quindi non possiamo semplicemente modificare la posizione dell'immagine relativa al personaggio. Quindi è stato creato un rettangolo, avente le dimensioni del canvas, con al centro l'immagine dell'avatar. Ogni volta che viene premuto un tasto di movimento, modifica la porzione di mappa da stampare di un pixel in più nella direzione desiderata e un pixel in meno nella direzione opposta, mantenendo sempre il personaggio al centro del canvas, dandoci così una sensazione di movimento.

Per poter associare tasti diversi alle varie attività eseguibili nel gioco è stato implementato un menu per la modifica dei tasti accessibile tramite il menù principale



Fig. 10: Schermata di modifica dei comandi

Cliccando sopra al rettangolo blu, in cui c'è il tasto associato all'azione, la casella cambia colore indicando che è possibile modificarlo premendone un'altro.

w	up
s	down
a	left
d	right
Enter	interaction

Fig. 11: Modifica dei comandi

Il salvataggio avviene con il click del pulsante "Salva" che invia, tramite una socket, i nuovi comandi e l'id dell'utente. Quando il server li riceve, tramite una query, modifica i comandi dell'utente nel database.

#### 4.4 Animazione

Quando il personaggio si muove all'interno della mappa di gioco, è necessario animarlo, per dare un migliore effetto di movimento. Per poter animare il nostro personaggio ci siamo avvalsi degli **spritesheet**. Uno **spritesheet** è un file, simili ai tileset, contenente diverse immagini(sprite) che rappresentano i diversi stadi di animazione. Essi vengono utilizzate per la stampa di oggetti nei videogiochi in due dimensioni. Nel nostro caso sono stati utilizzati spritesheet solo per il personaggio da animare. I vari sprite che sono contenuti all'interno sono stati divisi in questo modo:

- nella prima riga sono presenti tutti gli sprite inerenti al movimento verso il basso del personaggio
- nella seconda riga sono presenti gli sprite inerenti al movimento verso sinistra del personaggio
- nella terza riga gli sprite relativi al movimento verso destra
- nell'ultima riga quelli relativi al movimento verso l'alto



Fig. 12: sprite sheet

Per poter avere l'effetto animazione, semplicemente è stata creata una classe che, attraverso un ciclo scorre gli sprite, contenuti nella riga dello spritesheet relativa alla direzione di movimento, stampandoli per ogni fotogramma finché il personaggio si muove. Quando il personaggio si ferma stampa solo il primo sprite della riga corrispondente all'ultima direzione in cui si muoveva il personaggio.

## 4.5 Missioni

Il gioco consiste nel superare delle missioni inserite nei computer che si possono incontrare esplorando l'ambiente di gioco. Lo scopo di queste missioni è sempre quello di trovare un valore nascosto (flag) utilizzando le proprie conoscenze riguardanti la sicurezza informatica, queste missioni sono di vario tipo e possono essere inserite dagli amministratori tramite la pagina Admin.

### 4.5.1 Finestra Missione

Una volta trovato un computer in una stanza è possibile accedere alle informazioni iniziali della missioni premendo il tasto di interazione.

Queste informazioni serviranno all'utente per decidere se accettare la sfida oppure no.



Fig. 13: prima finestra della missione, dove l'utente legge la descrizione iniziale della missione

Dopo aver iniziato la missione verrà stampata una schermata contenente le istruzioni che l'utente dovrà eseguire per trovare il flag.



Fig. 14: Schermata in cui l'utente può inserire il flag

#### 4.5.2 Parsing e Stampa del Testo

Poiché non è possibile utilizzare i classici tag html all'interno del canvas, per poter stampare il testo che viene stampato all'interno della finestra delle missioni, tenendo conto dei fine linea e dei collegamenti ipertestuali ad altre pagine, è stato necessario implementare un parser che fosse in grado di separare questi elementi



### 4.5.3 Inizio Missione

Quando l'utente nella schermata della missione clicca il pulsante "Inizia Missione", il gioco deve capire che l'utente ha iniziato quella missione in quel momento, solo se la missione non sia stata iniziata dall'utente precedentemente. Questa informazione è necessaria per il calcolo delle statistiche, infatti per ogni missione affrontata dall'utente è possibile controllare la data di inizio e quella di fine.

Nell'istante in cui l'utente inizia una missione viene inviata una socket dal client al server con un dato che indica al server quale missione è appena stata iniziata e da quale utente. Dalla parte server, una volta ricevuto il messaggio dal client, controlla se quella missione sia già stata iniziata. Se la missione risulta già iniziata non verrà effettuata alcuna operazione aggiuntiva sul database, viceversa se non risulta iniziata allora creerà nel database una tabella nuova utenti\_missioni il cui il campo userId avrà l'id dell'utente che sta affrontando la missione, il campo missionId l'id della missione che ha deciso di affrontare mentre il campo dataInizio conterrà la data corrente.

### 4.5.4 inserimento flag

Una volta iniziata la missione l'utente potrà inserire un flag all'interno dell'unica textbox presente nel canvas e controllare se sia quello valido cliccando sul tasto Conferma.

La textbox è stata implementata grazie alla libreria **CanvasInput.js**; un modulo che permette di gestire textbox all'interno del canvas.



Fig. 16: finestra dove è possibile inserire il flag

Per poter controllare che il flag sia quello giusto il client invia un messaggio al server tramite una socket con i seguenti dati

- **idMissione:** contiene l'id della missione che l'utente sta cercando di completare
- **idUtente:** contiene l'id dell'utente
- **flag:** contiene il valore che l'utente ha inserito nella textbox

Una volta ricevuti i dati dal client il server controllerà se la missione sia già stata completata oppure no. Se la missione era già stata completata in precedenza allora si avranno due possibilità:

- 1 Il flag è quello giusto, quindi invia al client un messaggio di successo.
- 2 Il flag non è quello giusto quindi invia al client un messaggio di insuccesso.

Se invece la missione non era già stata completata in precedenza si avranno queste due possibilità:

- 1 Il flag è quello giusto e quindi, oltre ad inviare un messaggio di successo al client inserisce la data di fine missione all'interno dell'apposita tabella
- 2 Il flag non è quello giusto e quindi invia solamente un messaggio di insuccesso al client.

## 5 Realizzazione delle Missioni

Le missioni che sono state realizzate per questo progetto hanno la seguente tipologia:

- **Linux Basis:** permettono all'utente di applicare le conoscenze dei comandi base e del file system del sistema operativo linux.
- **Web Security:** permettono all'utente di applicare le conoscenze relative ai siti e applicazioni web.
- **crittografia:** permettono all'utente di applicare o scoprire il funzionamento degli algoritmi di crittografia.

## 5.1 Linux Basis

Per questa tipologia di missioni è stato usato un sistema virtuale con le seguenti caratteristiche:

O.S: Ubuntu 16.04

RAM: 2048 MB

CPU: Intel Core i7-3630QM @ 2.4Ghz, Single Core

Rete: NAT

Inanzi tutto, prima della creazione dei utenti relativi ai livelli, è necessario restringere i permessi di esecuzione, utilizzando il comando **chmod**, a tutti gli utenti (escluso l'utente root) dei comandi relativi alla modifica del file system e quelli relativi alla modifica dei permessi (**sudo**, **su**, **unmount**, **mount**, **chmod**, **chown** ecc.).

I livelli non sono altro che degli utenti linux creati appositamente avente la cartella **/home/levelX**, dove la X indica il numero del livello, avente i permessi di solo lettura e esecuzione da parte degli altri utenti che non sono root. Per ogni livello è presente, in genere, i file necessari al completamento dello stesso. Questi file hanno i seguenti permessi: per esempio nella home del livello1 è presente il file readme che, se andiamo a vedere i permessi, inanzi tutto ci dice che il file appartiene all'utente livello2 e che è di solo lettura da parte del l'utente livello1 e livello2. Quindi se si vuole creare un nuovo livelloX è necessario che ci sia il livello(X+1) in quanto il valore da trovare corrisponde alla password per il livello successivo.

Ora è necessario permettere agli utenti di connettersi tramite **ssh**<sup>3</sup> al server per poter completare i livelli. Per fare ciò, è stato installato il pacchetto **openssh-server**. L'utente quindi ogni volta che dovrà affrontare il livelloX è necessario che si connetta tramite ssh con username levelX e la relativa password. Man mano che il livello cresce, la difficoltà aumenta.

## 5.2 Web Security

Questi tipi di livelli sono incentrati sulla comprensione delle vulnerabilità delle applicazioni web, attraverso l'utilizzo di strumenti che ci permettono di analizzare e trovare la falla direttamente dal client. L'applicazione web è stata creata con Node.js tramite il framework Expressjs. È stato utilizzato il database Rethinkdb per salvare gli utenti, che hanno come username webX (la X indica il numero del livello), che rappresentano il livello che l'utente deve affrontare. Ad ogni livello viene associata la relativa pagina di login: per esempio se vogliamo accedere al

---

<sup>3</sup>**SSH** (Secure SHell, shell sicura) è un protocollo che permette di stabilire una sessione remota cifrata tramite interfaccia a riga di comando con un altro host di una rete informatica.

livello web1 dobbiamo andare sul url **http://ipserver:3000/web1/login**. I livelli vanno dal trovare la password all'interno del sorgente, modificare i cookie, fino all'utilizzo dell'**Sql Injection**<sup>4</sup>.

Questa applicazione può essere inserita nel server ubuntu precedentemente creato per i livelli Linux basis in modo da avere un'unico server dove girano tutti i livelli.

### 5.3 crittografia

Questi tipi di livelli servono per insegnare ad un utente ad analizzare il funzionamento di un algoritmo di criptazione e cercare un modo per aggirarlo. I livelli sono integrati nel server linux, precedentemente descritto nel paragrafo Linux Basis. Ad ogni livello di crittografia viene associato un utente **criptoX**(la X indica il numero del livello), dove nella **home** avrà il file **data.txt** contenente i dati da decriptare per completare la missione. Gli algoritmi di criptazione implementati, per adesso, sono:

- **Cifrario di Cesare:** è uno dei più antichi algoritmi crittografici di cui si abbia traccia storica. È un cifrario a sostituzione mono alfabetica in cui ogni lettera del testo in chiaro è sostituita nel testo cifrato dalla lettera che si trova un certo numero di posizioni dopo nell'alfabeto. Questi tipi di cifrari sono detti anche cifrari a sostituzione o cifrari a scorrimento a causa del loro modo di operare: la sostituzione avviene lettera per lettera, scorrendo il testo dall'inizio alla fine.
- **Cifrario Monoalfabetico:** è un cifrario che consiste nel prendere le lettere dell'alfabeto e cambiarle con altre lettere o simboli creando un nuovo alfabeto che viene utilizzato per cifrare il testo. Un esempio è il seguente:

Plaintext Alphabet	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext Alphabet	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A

Fig. 17: Cifrario Monoalfabetico

- **Cifrario di Hill:** è un cifrario a sostituzione polialfabetica basato sull'algebra lineare. Nella cifratura, ogni lettera è per prima cosa codificata in numero. Lo schema usato più di frequente è semplicemente: A = 0, B = 1, ..., Z = 25, ma questa non è una caratteristica essenziale del cifrario. Un blocco n di lettere è quindi considerato come uno spazio vettoriale di

---

<sup>4</sup>**SQL injection** è una tecnica di code injection, usata per attaccare applicazioni di gestione dati, con la quale vengono inseriti delle stringhe di codice SQL malevole all'interno di campi di input in modo che vengano eseguiti (es. per fare inviare il contenuto del database all'attaccante)

dimensione  $n$ , e moltiplicato per una matrice  $n \times n$ , modulo 26 (se si usa un numero più alto di 26 nel modulo base, allora si può usare uno schema numerale diverso per codificare le lettere, ed è possibile anche utilizzare spazi e punteggiatura). L'intera matrice è considerata la chiave del cifrario e deve essere casuale, a patto che sia invertibile. Per svolgere questa missione viene fornita la chiave e il testo cifrato.

- **One time pad:** è considerato uno degli algoritmi più sicuri in circolazione. Viene generata una chiave della lunghezza del testo che bisogna criptare. Si esegue uno xor tra la chiave e il testo, generando il testo cifrato. Viene considerato molto sicuro in quanto la chiave per criptare è generata casualmente e viene utilizzata una volta solo (One Time). Ovviamente la chiave per questo livello viene fornita all'interno del file data.txt.

## 6 Manuale

### 6.1 Installazione

Per avviare l'applicazione web è necessario aver installato i seguenti strumenti:

- Nodejs(2.8): si consiglia di installare una versione  $\geq 7.0$ ;
- Rethinkdb(2.14): si consiglia di installare la versione 2.3.5;
- Redis(2.12): si consiglia di installare la versione  $\geq 3$

### 6.2 Avvio dell'applicazione web

Per avviare l'applicazione web è necessario, aprire un terminale nella cartella del progetto, e eseguire i seguenti comandi:

- Se si cerca di avviare per la prima volta l'applicazione, siccome il database Rethinkdb(2.14) utilizza un username e una password, che sono salvati nel file config, è necessario usare il seguente comando:

– **rethinkdb –initial-password "qui ci va la password salvata nel file config.js"**

Per le altre volte è sufficiente scrivere il comando **rethinkdb**

- Una volta avviato il database è necessario far partire l'applicazione web con il comando:

– **node server.js**

## 6.3 Creazione della Mappa

Se si vuole creare una nuova mappa prima è necessario crearla con lo strumento Tiled(2.13) e, poi aggiungerla nel gioco come spiegato nella sezione Scheda Mappe. Per la creazione di una mappa con lo strumento Tiled, è possibile seguire la documentazione sul sito <http://doc.mapeditor.org/manual/> o seguire i video tutorial sul sito [www.gamefromscratch.com](http://www.gamefromscratch.com).

## 6.4 Aggiunta di una nuova Missione

Per l'aggiunta di una missione al gioco è necessario prima aggiungerla nel database, come spiegato nella sezione Scheda Missioni, e poi dato l'id, si segue la seguente procedura:

- Aprire lo strumento Tiled(2.13) con la mappa in cui si desidera inserire la missione. Per esempio la mappa relativa all'internet caffè.

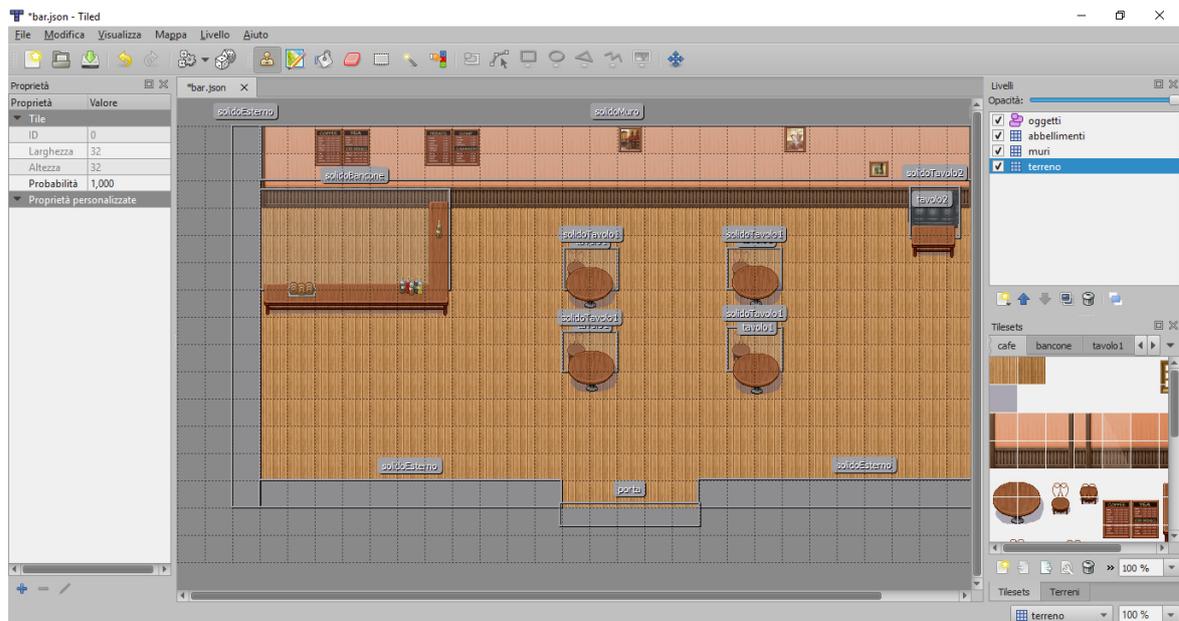
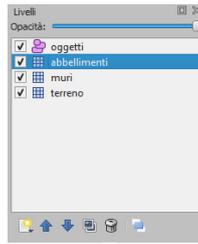
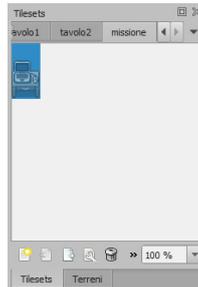


Fig. 18: Mappa Bar

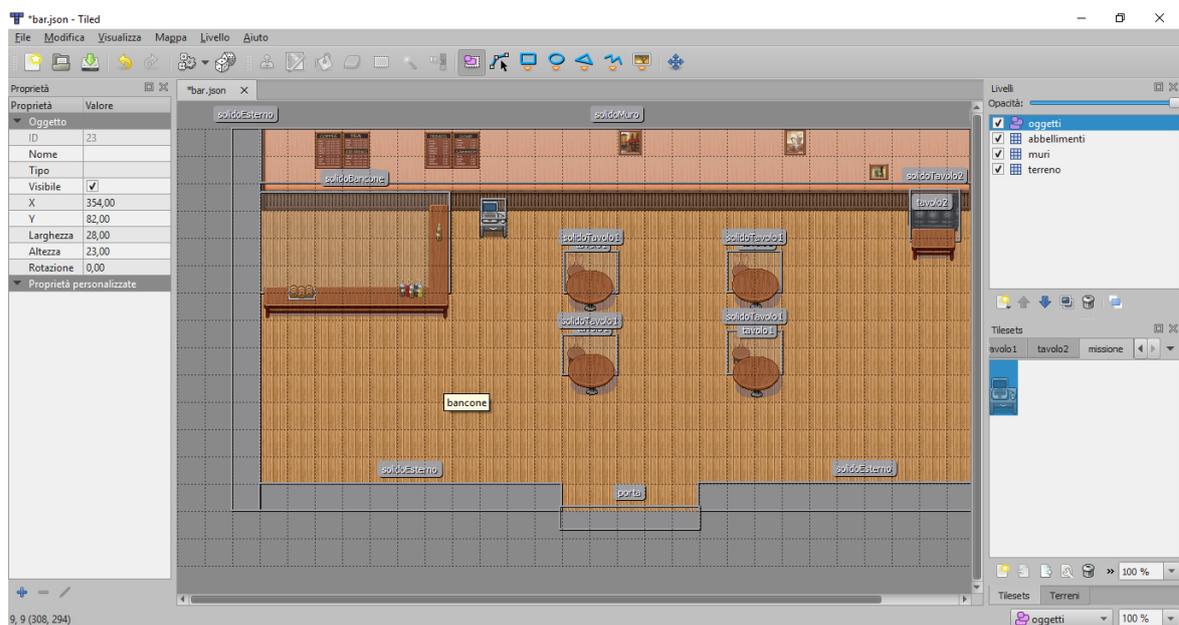
- Selezionare il livello con il nome abbellimenti nella finestra layers.



- Caricare l'immagine  con la finestra tileset:



- Assicurandosi di avere selezionato il seguente pulsante  situato nella barra superiore di Tiled, e selezionando l'immagine caricata dalla finestra tileset, è possibile inserire il computer in un punto desiderato della mappa.
- Per associare la missione a questo computer è necessario selezionare il livello oggetti, cliccare sul pulsante  situato nella barra superiore, e disegnare un rettangolo sopra all'immagine del computer delle dimensioni desiderate, ricordandosi che questo rettangolo corrisponde all'oggetto solido descritto nel paragrafo 4.2, con cui il personaggio collide.



- Ora è necessario aggiungere due proprietà personalizzate, cliccando il tasto + sulla finestra proprietà:
  - **missione**: di tipo booleano. Serve per dire al gioco che l'oggetto con cui collide è una missione;
  - **id**: di tipo stringa. Indica l'id della missione nel database.



Una volta inserita la missione nella mappa con lo strumento Tiled, è necessario esportare la mappa in JSON e ricaricarla tramite la Pagina Admin, cliccando il pulsante modifica (✎) relativo alla mappa come spiegato nel paragrafo Scheda Mappe.

## 7 Sviluppi Futuri

In futuro vorrei inserire la musica nel gioco rendendo l'esperienza di gioco più piacevole, e vorrei implementare la conferma della registrazione con l'e-mail. Infine vorrei inserire nuovi livelli e mappe.

## 8 Ringraziamenti

Ringrazio innanzitutto il professor Fausto Marcantoni per avermi dato la possibilità di svolgere la tesi nell'ambito della sicurezza informatica, un ambito del quale mi sono appassionato. Un grazie va ai compagni di studi di otto anni: Gianmarco Mazzante, Michelangelo Diamanti e Davide Sestili. Un grazie va ai amici nuovi e vecchi che sono stati influenti in parte diretta o indiretta nella mia vita.

Infine un grazie di cuore ai miei genitori, Saadia e Lmiloudi: sono loro ad avermi dato l'opportunità di portare avanti gli studi nonostante le difficoltà, dandomi così la possibilità di intraprendere questo percorso importante nella mia vita.

GRAZIE MAMMA e BABBO.

## Riferimenti bibliografici

- [1] Sito di W3School ([www.w3schools.com/html/](http://www.w3schools.com/html/)) dove è possibile imparare l'HTML
- [2] Sito di W3School ([www.w3schools.com/js/](http://www.w3schools.com/js/)) dove è possibile imparare il linguaggio JavaScript
- [3] Sito della libreria JQuery ([www.jquery.com](http://www.jquery.com))
- [4] Sito di W3School ([www.w3schools.com/css/](http://www.w3schools.com/css/)) dove è possibile imparare CSS
- [5] Sito per i Template Ejs ([www.embeddedjs.com](http://www.embeddedjs.com))
- [6] Sito del framework bootstrap ([www.getbootstrap.com](http://www.getbootstrap.com))
- [7] Sito di Angularjs ([www.angularjs.org](http://www.angularjs.org))
- [8] Sito del framework nodejs ([www.nodejs.org](http://www.nodejs.org))
- [9] Sito del framework expressjs ([www.expressjs.com](http://www.expressjs.com))
- [10] Sito di passportjs ([www.passportjs.org](http://www.passportjs.org))
- [11] Sito per la libreria socket.io ([www.socket.io](http://www.socket.io))
- [12] Sito del database rethinkdb ([www.rethinkdb.com](http://www.rethinkdb.com))
- [13] Sito del database redis([www.redis.io](http://www.redis.io))
- [14] Sito dello strumento Tiled map editor([www.mapeditor.org](http://www.mapeditor.org))
- [15] Manuale per Tiled
- [16] Video tutorial ([www.gamefromscratch.com](http://www.gamefromscratch.com)) per la spiegazione di come creare una mappa con Tiled ([www.doc.mapeditor.org/manual/](http://www.doc.mapeditor.org/manual/))
- [17] Sito da dove ho preso le immagini del personaggio ([www.untamed.wild-refuge.net](http://www.untamed.wild-refuge.net))
- [18] Sito del sistema operativo ubuntu ([www.ubuntu-it.org](http://www.ubuntu-it.org))
- [19] Sito di Stackoverflow per la risoluzione dei problemi ([www.stackoverflow.com](http://www.stackoverflow.com))