Versione 1.0

06/10/2019



Presentato da: Marco dal zovo - 931121

online chess

corso di programmazione web e mobile – a.a. 2018/2019

introduzione

*Online Chess* è un applicazione web che permette di affrontare altri giocatori al gioco degli scacchi, tenendo traccia delle loro vittorie e sconfitte per classificarli. Lo scopo dell’applicazione è quello di rendere possibile giocare ad una partita a scacchi ovunque attraverso dei semplici passaggi preliminari.

Le tecnologie che utilizza lato client sono le seguenti:

* **HTML5**
* **CSS3**
* **Javascript**
* **AJAX**

Invece, lato server utilizza **NodeJS**, coadiuvato da **EJS** come motore di rendering delle pagine HTML e **SocketIO** nella gestione della comunicazione socket.

Come tecnologia di storicizzazione dei dati utilizza **MongoDB**.

# breve analisi dei requisiti

## destinatari

L’applicazione è stata sviluppata per rispondere alle esigenze di tutte le tipologie di giocatori – dal più esperto al novizio – offrendo un ambiente di gioco pulito ed equo su qualsiasi dispositivo essi si trovino ad utilizzarla. Soprattutto nel riguardo degli utenti che si affacciano per la prima volta al gioco degli scacchi, *Online Chess* offre alcune funzionalità utili ad apprendere le meccaniche di gioco in meno tempo possibile. Infine, l’applicazione è organizzata su più schede specifiche per tema, tutte quante facilmente raggiungibili attraverso un menù laterale sempre visibile.

## modello di valore

*Online Chess* cerca di soddisfare gli interessi di tutti giocatori, sia che cerchino partite complesse per competere con altri giocatori con gli stessi interessi, sia che intendano giocare delle partite meno impegnative contro loro conoscenti o altri giocatori.

Sicuramente la fascia di giocatori in cerca di una tipologia di gioco competitiva sarà più interessata ad utilizzare l’applicazione per garantirsi una buona posizione all’interno della classifica punti.

## flusso dei dati

Tutti i contenuti presenti su *Online Chess* sono prodotti internamente senza grossi costi in tempistiche ed archiviati su un database - all’interno della stessa rete – in formato **JSON**.

Trasversalmente, i contenuti vengono recuperati – quando richiesti, analizzati e spediti al client sempre in formato **JSON**; dopo essere stati recepiti dal client, vengono trasformati in formato **HTML**, per essere interpretati dal browser.

L’applicazione provvede autonomamente ad aggiornare i contenuti quando lo ritiene necessario, per esempio in caso di una improvvisa disconnessione dalla comunicazione socket oppure dopo aver selezionato una scheda differente dal menù laterale.

## aspetti tecnologici

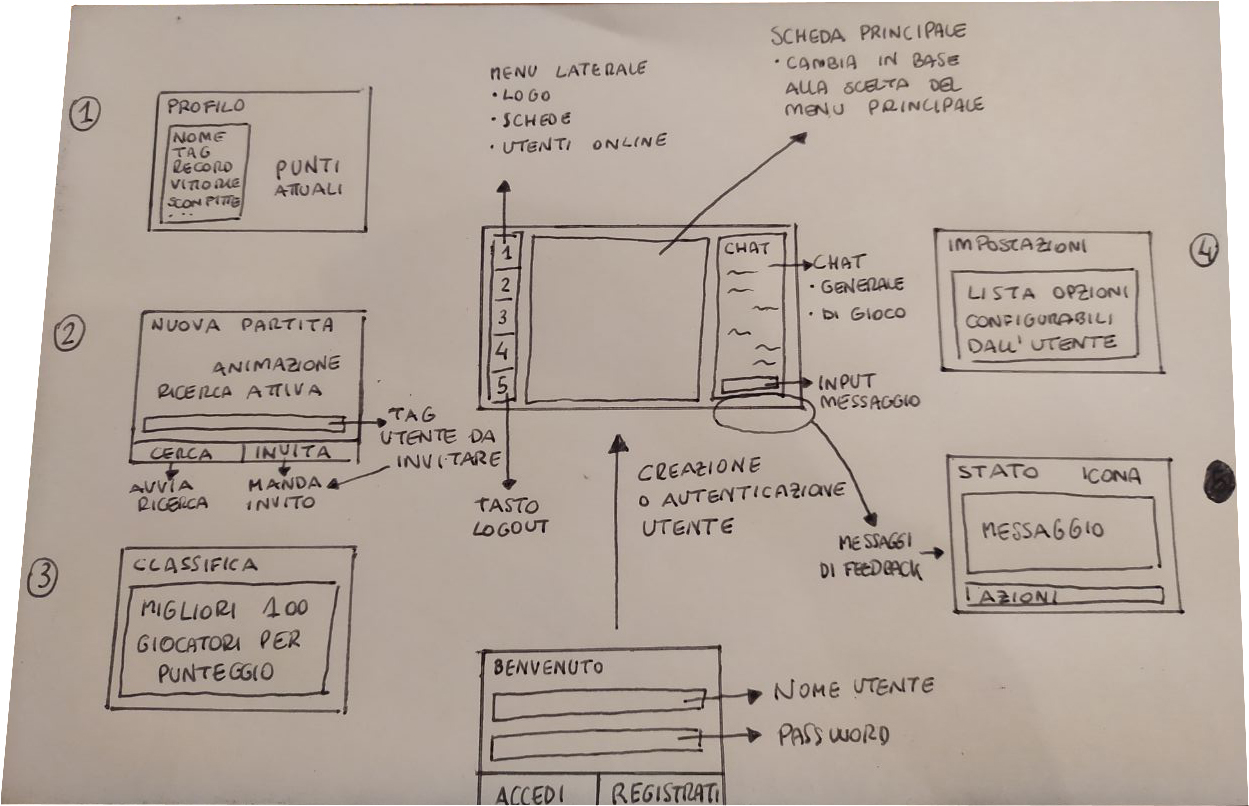
Come detto prima, lato server i contenuti sono archiviati, analizzati e spediti al client utilizzando il formato standard **JSON**; invece, lato client i contenuti vengono acquisiti in formato **JSON**, analizzati ed utilizzati per popolare elementi **HTML** usati come **prefabs** – ovviamente nascosti al giocatore – ed in seguito visualizzati.

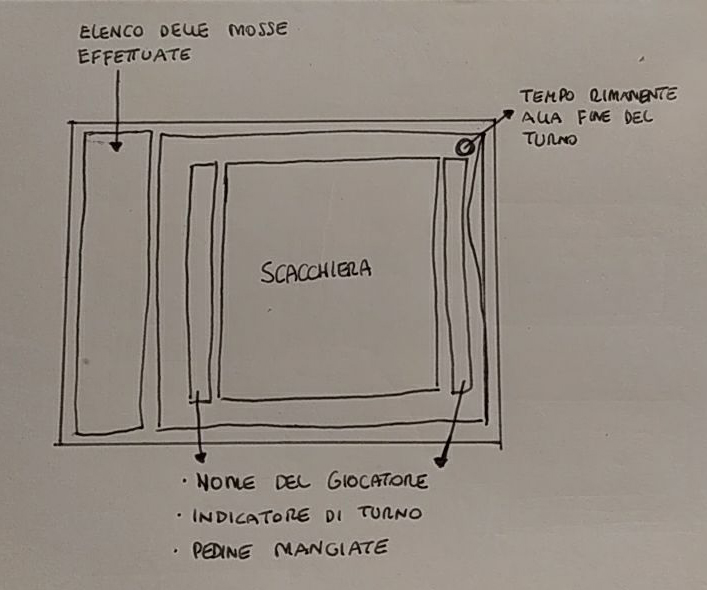
Inoltre, lato server viene utilizzata una gestione del codice tramite modello **MVC**, avendo – per l’appunto – un modello per ogni **collection** su **MongoDB**, vari gestori per le **routes** raggiungibili ed un motore di rendering delle pagine.

Nel dettaglio, *Online Chess* utilizza le seguenti tecnologie:

* **HTML5**, per la costruzione delle interfacce
* **CSS3**, necessario ad applicare lo stile delle interfacce
* **Javascript**, che gestisce gli eventi socket e **DOM**, aggiorna il rendering delle interfacce quando necessario ed effettua chiamate asincrone verso il server
* **AJAX**, utilizzata per comunicare in maniera asincrona con il server
* **MeteorEmoji**, una libreria Javascript che abilita l’utilizzo di emoji anche da client di tipologia *desktop*
* **NodeJS**, attraverso i seguenti moduli:
  + **EJS**, motore di rendering delle interfacce
  + **Express**, framework per applicazioni web
  + **Express-session**, fornisce un middleware per la gestione delle sessioni con Express
  + **I18n-express**, fornisce un middleware per l’internazionalizzazione dei contenuti
  + **SocketIO**, permette una comunicazione real-time e bidirezionale tra client e server
* **MongoDB**, usato come database per storicizzare i contenuti

# interfacce





# architettura

## diagramma dell’ordine gerarchico delle risorse

## descrizione delle risorse

L’applicazione fornisce solamente su due **routes** specifiche pagine HTML:

* **“/”**
* **“/logout”**

Nel primo caso viene stabilito lato server – in base alla presenza o meno della sessione utente – la pagina **HTML** da restituire in risposta:

* **Auth.ejs**, nel caso in cui non sia presente una sessione utente. Questa pagina può compiere le seguenti chiamate **AJAX**:
  + **Signup**, quando l’utente clicca sul bottone per registrarsi
  + **Login**, quando l’utente clicca sul bottone per effettuare l’accesso
  + **Validate**, se in fase di caricamento della pagina trova delle credenziali salvate nel *sessionStorage* (*localStorage* in caso di client mobile), per effettuare un **fast-login**
* **Index.ejs**, nel caso contrario. Questa pagina una volta caricate le risorse Javascript necessarie provvede ad avviare la comunicazione socket con il server che gestirà completamente il comportamento della pagina in seguito alle interazioni del giocatore

Invece, nel secondo caso viene fornita la pagina **logout.ejs** che esegue una chiamata **AJAX** al server per svuotare la sessione utente ed infine reindirizzare alla **route** “/”.

## altri diagrammi

Il database MongoDB ha la seguente struttura:

* **Users**
  + \_id (ObjectId)
  + username (String)
  + hash (String)
  + tag (String)
  + points (Int32)
  + record (Int32)
  + createdAt (Date)
  + lastTimeSeen (Date)
  + socketId (String)
  + gameRoom (Double)
* **Matchmaking**
  + \_id (ObjectId)
  + userTag (String)
  + socketId (String)
  + createdAt (Date)
* **Games**
  + \_id (ObjectId)
  + Players (Array)
    - userTag (String)
    - socketId (String)
    - username (String)
    - units (Array)
      * s (String)
      * u (Int32)
      * x (Int32)
      * y (Int32)
    - graveyard (Array)
      * stessa struttura di *units*
    - class (String)
    - authorizedMoves (Array)
      * u (Int32)
      * x (Int32)
      * y (Int32)
      * moves (Array)
        + x (Int32)
        + y (Int32)
        + empty (Boolean)
        + first (Boolean)
        + promote (Boolean)
        + side (Object)

stessa struttura di *move*

* + rounds (Array)
    - lista di *round* passati
  + roomName (String)
  + createdAt (Date)
  + askedAt (Date)
  + round (Object)
    - count (Int32)
    - by (String)
    - startedAt (Date)
  + startedAt (Date)
* **History**
  + Struttura identica ai documenti di **Games**, con un’unica aggiunta:

end (Object)

* at (Date)
* by (String)
* status (Int32)

# codice

La logica più significativa all’interno dell’applicazione è situata nel calcolo delle mosse possibili per ogni tipologia di pedina. La risorsa commentata è disponibile seguendo il seguente link:

<https://github.com/Dazorn96/node-chess/blob/master/helpers/chess.js>

Nello specifico è interessante la funzione **onKingUnderChess** (riga 135), che per ogni mossa eseguibile dal giocatore verifica – simulando il movimento della pedina – se la pedina *re* del giocatore si trovi sotto stato **scacco**, rimuovendola quando l’esito è positivo.