Applicazioni Web I – Esame #3 (scadenza 2023-09-07 alle 23:59)

“Indovina Chi”

VERSIONE FINALE – le modifiche sono in rosso.

Progettare e implementare un’applicazione web per giocare ad una versione a singolo giocatore del famoso gioco “Indovina Chi”.

Nel gioco, il giocatore deve indovinare l’identità di uno specifico ‘oggetto’ tra un insieme di altri oggetti tramite un insieme di tentativi. Il gioco definisce un catalogo di oggetti, e gli oggetti sono caratterizzati da un insieme di proprietà che possono assumere differenti valori. Ogni oggetto nel catalogo ha una combinazione *unica* di valori per le sue proprietà. Tutte le proprietà devono essere definite (cioè, avere un valore) per tutti gli oggetti.

Un catalogo di oggetti può rappresentare un dominio qualsiasi, come per esempio persone famose, personaggi di cartoni animati, dinosauri, costellazioni, pokemon, sportivi, professori, animali, fiori, ecc. Usare la propria fantasia a piacere. Si richiede che l’applicazione implementi solamente uno dei domini, a propria scelta. Il numero di proprietà e i loro valori devono essere definiti considerando le caratteristiche del catalogo scelto.

Per esempio, se gli oggetti selezionati fossero animali selvaggi, le proprietà potrebbero essere “nutrizione” (con valori: carnivoro, erbivoro, …), “gambe” (con i valori 0, 2 o 4), “vola” (valori si o no), “colore” (con un insieme di possibili valori predeterminati), ecc.

Il gioco consiste in una serie di giocate di varia difficoltà (facile = mostra 12 oggetti, media = mostra 24 oggetti, difficile = mostra 36 oggetti). All’inizio di ogni giocata, l’intera lista di oggetti viene visualizzata (mostrando un’immagine per ogni oggetto, posizionata in una griglia bidimensionale), e un “oggetto segreto” viene scelto casualmente dal server. Si applicano le seguenti due condizioni:

1. Per evitare comportamenti scorretti, l’identità dell’oggetto segreto non deve **mai** essere trasferita al client.
2. L'insieme di valori associati alle proprietà di ogni oggetto non devono essere mostrati, perché il giocatore deve essere in grado di capirli dall’immagine corrispondente all’oggetto stesso.

Ogni giocata consiste in un insieme di tentativi. In ogni tentativo, il giocatore seleziona una proprietà e un possibile valore di tale proprietà tra i possibili valori della proprietà stessa. Dopo aver confermato la selezione, l’applicazione comunica al giocatore se il tentativo è corretto o no (cioè, se il valore della proprietà dell’oggetto segreto corrisponde a quello selezionato dal giocatore). L’applicazione poi ‘disabilita’ tutti gli oggetti sullo schermo che non sono compatibili con il tentativo effettuato. *Per esempio, se il giocatore seleziona, per il tentativo, “vola=si”, e l’oggetto segreto è un animale che vola, l’applicazione conferma il tentativo, e disabiliterà tutti gli oggetti che non volano. Se l’oggetto segreto non è un animale che vola, l’applicazione comunica che il tentativo non è corretto, e disabilita tutti gli animali che volano.*

Nota: per ogni possibile sequenza di tentativi, l’oggetto segreto sarà sempre nell’insieme degli oggetti ancora non disabilitati.

Commented [1]: Le immagini dei vari oggetti le dobbiamo prendere da internet? Queste immagini devono essere salvate sul client o sul server?

Commented [2]: Si', da internet o da dove desiderate, a vostra scelta. Le immagini devono essere immagazzinate dove ritiene più opportuno (tipicamente potrebbero essere risorse statiche in un'apposita cartella sul server)

Commented [3]: L'utente può scegliere la difficoltà con cui giocare oppure il gioco consiste nel fare i tre livelli e si conclude solo al termine delle 3 giocate? In tal caso, in totale, gli oggetti diversi dovrebbero essere 36+24+12.

Commented [4]: Può scegliere. Massimo oggetti 36. Se la difficoltà è minore, se ne sceglie un sottoinsieme dei 36.

Commented [5]: è permesso passare al client soltanto l'ID dell'oggetto segreto, senza ulteriori dettagli e senza API che permettano di ottenere informazioni su di esso? Se non fosse possibile, la soluzione di mandare l'id non in chiaro al client si può considerare?

Commented [6]: Non è consentito di mandare nessuna informazione (né in chiaro né cifrata) riguardo al segreto sul client. Bisogna pensare le API in modo che questo trasferimento di informazioni non sia necessario.

Commented [7]: Si può suppore che l'utente possa fare più volte la medesima domanda oppure dobbiamo tenere traccia dei particolari tentativi effettuati?

Commented [8]: Sì, ma il valore della proprietà dovrà ovviamente essere diverso

Commented [9]: Se per via di tentativi precedenti differenti una proprietà non comparisse più nel sottoinsieme di tessere ancora in gioco dobbiamo fare in modo che quella domanda non si possa più fare?

Commented [10]: L'implementazione è a sua scelta, certamente ha senso fare per esempio come dice

Per terminare la giocata, il giocatore seleziona uno degli oggetti ancora non disabilitati (questo può essere effettuato in ogni momento, e sarà l'unica opzione se c'è solo un oggetto rimanente). Se corrisponde all'oggetto segreto, il giocatore vince, altrimenti perde.

Al termine della giocata viene assegnato un punteggio:

- 0 punti se il giocatore ha perso;
- un numero intero non negativo se il giocatore ha vinto, calcolato come (n_oggetti-n_tentativi), dove n_oggetti è il numero iniziale degli oggetti nel gioco. Se la differenza è negativa, viene assegnato un punteggio pari a 0.

Dalla pagina principale dell'applicazione, qualsiasi utilizzatore anonimo del sito può selezionare un livello di difficoltà, iniziare una giocata, e se terminata riceverà un punteggio (che non verrà memorizzato).

Dalla pagina principale dell'applicazione, un utente autenticato può selezionare un livello di difficoltà, iniziare una giocata, e se terminata il risultato della giocata sarà memorizzato e aggiunto alla storia personale dell'utente.

Un utente autenticato può accedere ad una pagina con la storia delle sue giocate che contenga quelle completate. Per ognuna deve essere mostrato: data, difficoltà, oggetto segreto, punteggio. Anche il punteggio totale dell'utente (cioè la somma di tutti i punteggi ottenuti dall'utente autenticato) deve essere mostrato sulla stessa pagina.

L'organizzazione delle funzionalità di questo testo in differenti schermate (e potenzialmente in differenti route) è lasciata allo studente ed è oggetto di valutazione.

Requisiti del progetto

- L'architettura dell'applicazione e il codice sorgente devono essere sviluppati adottando le migliori pratiche (best practice) di sviluppo del software, in particolare per le single-page application (SPA) che usano React e HTTP API.
- Il progetto deve essere realizzato come applicazione React, che interagisce con un'API HTTP implementata in Node+Express. Il database deve essere memorizzato in un file SQLite.
- La comunicazione tra il client ed il server deve seguire il pattern "two servers", configurando correttamente CORS, e React deve girare in modalità "development" con lo Strict Mode attivato.
- La valutazione del progetto sarà effettuata navigando all'interno dell'applicazione. Non saranno testati né usati il bottone di "refresh" né l'impostazione manuale di un URL (tranne /), e il loro comportamento non è specificato. Inoltre, l'applicazione non dovrà mai "autoricaricarsi" come conseguenza dell'uso normale dell'applicazione stessa.
- La directory radice del progetto deve contenere un file `README.md` e contenere due subdirectories (`client` e `server`). Il progetto deve poter essere lanciato con i comandi: "`cd server; nodemon index.js`" e "`cd client; npm run dev`". Viene fornito uno scheletro delle directory del progetto. Si può assumere che `nodemon` sia già installato a livello di sistema.
- L'intero progetto deve essere consegnato tramite GitHub, nel repository creato da GitHub Classroom.
- Il progetto **non deve includere** le directory `node_modules`. Esse devono essere ricreabili tramite il comando "`npm install`", subito dopo "`git clone`".

Commented [11]: Per terminare la giocata l'utente ha come unica opzione quella di tentare di indovinare la soluzione o può farlo anche in altro modo?

Commented [12]: E' l'unica opzione.

Commented [13]: L'utente può avere più di una partita attiva in contemporanea (ad esempio una per livello) oppure per iniziare un altra partita deve obbligatoriamente terminare quella corrente?

Commented [14]: Non è richiesto

Commented [15]: Possiamo salvare il resto delle informazioni della partita anonima senza salvare il punteggio?

Commented [16]: Il punto è che non è necessario salvare alcuna informazione perché non si deve associare a nessuno. Non vedo quindi perché sarebbe utile salvare informazioni. Se viene comodo salvarle ma poi non sono usate va bene.

Commented [17]: Salvando le informazioni di tutte le partite e associandole al sessionid di ogni giocatore faccio in modo di poter salvare l'oggetto segreto nel db e poterlo interrogare a necessità

Commented [18]: Non è qualcosa che abbiamo visto. Comunque può fare come ritiene, posto che siano soddisfatte le specifiche.

- Il progetto può usare librerie popolari e comunemente adottate (come per esempio `day.js`, `react-bootstrap`, ecc.), se applicabili e utili. Tali librerie devono essere correttamente dichiarate nei file `package.json` e `package-lock.json` cosicché il comando `npm install` le possa scaricare ed installare tutte.
- L'autenticazione dell'utente (login) e l'accesso alle API devono essere realizzati tramite `passport.js` e cookie di sessione, utilizzando il meccanismo visto a lezione. E' richiesto immagazzinare le credenziali in formato hashed e con sale. La registrazione di un nuovo utente non è richiesta.

Requisiti del database

- Il database del progetto deve essere definito dallo studente e deve essere precaricato con 36 oggetti, almeno 4 proprietà, e almeno 3 utenti, di cui due hanno già effettuato alcune giocate, e uno non ha mai giocato.

Nota importante

- Come già descritto nel testo, l'identità dell'oggetto segreto non deve **mai** essere inviata al browser (eccetto al termine del gioco), per cui le API dovranno essere progettate considerando questo aspetto.

Contenuto del file README.md

Il file README.md deve contenere le seguenti informazioni (un template è disponibile nello scheletro del progetto creato con il repository). In genere, ogni spiegazione non dovrebbe essere più lunga di 1-2 righe.

1. Server-side:
    a. Una lista delle API HTTP offerte dal server, con una breve descrizione dei parametri e degli oggetti scambiati
    b. Una lista delle tabelle del database, con il loro scopo
2. Client-side:
    a. Una lista delle route dell'applicazione React, con una breve descrizione dello scopo di ogni route
    b. Una lista dei principali componenti React implementati nel progetto
3. In generale:
    a. Uno screenshot della **schermata dove è possibile giocare**. L'immagine va embeddata nel README inserendo lì il link ad una immagine committata nel repository.
    b. Username e password degli utenti.

Procedura di consegna (importante!)

Per sottomettere correttamente il progetto è necessario:

- Essere **iscritti** all'appello.
- **Avere accettato l'invito** su GitHub Classroom usando il link fornito per **questo** specifico appello, **associando** correttamente il proprio utente GitHub al proprio nome e matricola.
- Fare il **push del progetto** nel **branch "`main`"** del repository che GitHub Classroom ha generato per lo studente. L'ultimo commit (quello che si vuole venga valutato) deve essere **taggato** con il tag `final`. NB: `final` deve essere scritto tutto minuscolo e senza spazi

Commented [19]: Il database deve contenere un "massimo" di 36 oggetti o ce ne possono essere anche di più? Perché per le proprietà che ho scelto, su soli 36 elementi faccio fatica a popolare la base di dati con una buona omogeneità. Tanto l'estrazione di 12/24/36 oggetti "casuali" (in base alla difficoltà scelta) immagino debba essere comunque implementata per poter giocare

Commented [20]: Non è necessario che li scelga casualmente, quello scelto casualmente è il segreto. Sono richiesti 36, non massimo 36.

NB: qualche anno fa GitHub *ha cambiato il nome del branch di default da master a main*, porre attenzione al nome del branch utilizzato, specialmente se si parte/riutilizza/modifica una soluzione precedentemente caricata su un sistema git.

Nota: per taggare un commit, si possono usare (dal terminale) i seguenti comandi:

```
# ensure the latest version is committed
git commit -m "...comment..."
git push
```

```
# add the 'final' tag and push it
git tag final
git push origin --tags
```

NB: **il nome del tag è "final", tutto minuscolo, senza virgolette, senza spazi, senza altri caratteri**, e deve essere associato al commit da valutare.

E' anche possibile inserire un tag dall'interfaccia web di GitHub (seguire il link 'Create a new release').

Per testare la propria sottomissione, questi sono i comandi che useremo per scaricare il progetto. Potreste volerli provare in una directory vuota:

```
git clone ...yourCloneURL...
cd ...yourProjectDir...
git pull origin main  # just in case the default branch is not main
git checkout -b evaluation final # check out the version tagged with
'final' and create a new branch 'evaluation'
(cd client ; npm install)
(cd server ; npm install)
```

Assicurarsi che tutti i pacchetti (package) necessari siano scaricati tramite i comandi `npm install`. Fare attenzione se alcuni pacchetti sono stati installati a livello globale perché potrebbero non apparire come dipendenze necessarie: potreste voler testare la procedura su un'installazione completamente nuova (per es. in una VM).

Il progetto sarà testato sotto Linux: si faccia attenzione al fatto che Linux è case-sensitive nei nomi dei file, mentre macOS e Windows non lo sono. Pertanto, si controllino con particolare cura le maiuscole/minuscole usate nei nomi dei files, negli `import` e nei `require()`.