Applicazioni Web I – Esame # 2 (scadenza 2022-07-14 alle 23:59)

“Indovinelli”

VERSIONE FINALE – le modifiche sono in rosso

Progettare e implementare un’applicazione web per gestire una piattaforma di gioco che si basa sul pubblicare e risolvere indovinelli.

Nota: nel seguito, “utente” significa un qualsiasi utente che sia autenticato (quindi logged-in), mentre visitatore anonimo è un utente che non è logged-in.

Ogni utente dell’applicazione può pubblicare uno o più indovinelli, che gli altri utenti possono provare a risolvere.

Un indovinello è caratterizzato da una domanda (una stringa di testo), un livello di difficoltà (facile, medio, difficile), una durata (un numero intero di secondi, da 30 a 600), la risposta corretta (una stringa di testo), e 2 suggerimenti (stringhe di testo). La durata è specifica di ogni indovinello, e la durata è misurata a partire dalla prima risposta inviata da un utente (l’istante di creazione dell’indovinello pertanto è irrilevante). Un indovinello può essere pubblicato da un qualsiasi utente, e tutti gli altri utenti possono provare a rispondere, nella durata prevista. ~~La durata è misurata a partire dalla prima risposta inviata (non dal momento di pubblicazione dell’indovinello).~~ Il primo utente che fornisce la risposta corretta ottiene un punteggio, l’indovinello diventa “chiuso”, e non possono essere date ulteriori risposte. Se nessun utente fornisce le risposte corrette all’interno della durata prevista l’indovinello diventa automaticamente “chiuso”.

In particolare, il sistema deve supportare le seguenti funzionalità:

- Un utente può creare un nuovo indovinello, fornendo tutte le necessarie informazioni. Un indovinello nuovo è creato in stato “aperto”, ossia che consente di inviare risposte.
- Un utente può vedere lo stato di ognuno dei suoi indovinelli che ha pubblicato come autore
    - Se l’indovinello è “chiuso”, l’autore vedrà tutte le risposte che sono state date, la risposta corretta decisa dall’autore dell’indovinello, e l’indicazione del vincitore se presente.
    - Se l’indovinello è “aperto”, l’autore vedrà tutte le risposte correnti (aggiornate ogni secondo), e un conto alla rovescia (count-down) che mostra il tempo rimanente.
- Qualsiasi utente può vedere la lista degli indovinelli, divisi tra “aperti” e “chiusi”, vedendo la domanda ed il suo livello di difficoltà
    - Selezionando un indovinello “chiuso” verranno mostrate tutte le risposte che sono state date, la risposta corretta decisa dall’autore dell’indovinello, e l’indicazione del vincitore se presente.
    - Selezionando un indovinello “aperto” si avrà l’opportunità di rispondere (ovviamente senza che siano mostrate le altre risposte). Ogni utente può fornire al massimo una sola risposta per ogni indovinello, e tale risposta non può essere modificata dopo che è stata data. Mentre fornisce la risposta verrà mostrato un conto alla rovescia (count-down) che mostra il tempo rimanente. Se il tempo rimanente è minore del 50%, il primo suggerimento viene mostrato. Se il tempo rimanente è minore del 25%, anche il secondo

suggerimento viene mostrato. Se la risposta è corretta, l'indovinello diventa immediatamente "chiuso" e l'utente otterrà un punteggio pari a 3, 2 o 1 punti, secondo la difficoltà dell'indovinello (rispettivamente difficile, medio, facile).

- I visitatori anonimi vedranno la lista degli indovinelli (domanda e difficoltà) ma non vedranno né le risposte date né la risposta corretta. Devono essere mostrati tutti gli indovinelli, riportando anche il loro stato ("aperto" o "chiuso").
- I visitatori anonimi e gli utenti possono vedere la classifica dei migliori 3 punteggi ottenuti ("top-3"). Nel caso di pari merito, devono essere mostrati tutti gli utenti che hanno i 3 punteggi più alti.

L'organizzazione di queste funzionalità in differenti schermate (e potenzialmente in differenti routes) 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 attivo.
- La valutazione del progetto sarà effettuata muovendosi all'interno dell'applicazione. Non sarà testato il comportamento del bottone di "refresh", né l'immissione diretta di un URL (eccetto /), e il comportamento a fronte dell'immissione diretta di un URL (eccetto /) o del refresh non è definito. Inoltre, l'applicazione non dovrà mai "ricaricarsi" da sola come conseguenza delle normali azioni dell'utente.
- 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 start`". 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`".
- 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 logout) e l'accesso alle API devono essere realizzati tramite `passport.js` e cookie di sessione, utilizzando il meccanismo visto a lezione. Non è richiesto alcun ulteriore meccanismo di protezione a livello delle API. Le credenziali devono essere memorizzate in forma cifrata e con il "sale". La registrazione di un nuovo utente non è richiesta.

Requisiti del database

- Il database del progetto deve essere implementato a cura dello studente, e deve essere precaricato con *almeno 5 utenti*, di cui almeno 3 nel top-3. Tutti gli utenti devono avere, in quanto autori, almeno due indovinelli chiusi ed uno aperto.

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 **pagina di risposta agli indovinelli dove sia mostrato anche il primo suggerimento**. Lo screenshot deve essere inserito nel README linkando un'immagine committata nel repository
    b. Username e password degli utenti creati.~~, e loro caratteristiche (part-time, full-time)~~

Procedura di consegna (importante!)

Per sottomettere correttamente il progetto è necessario:

- Essere **iscritti** all'appello.
- **Avere accettato l'invito** su GitHub Classroom, **associando** correttamente il proprio utente GitHub al proprio nome/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: recentemente 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' – ma da lì creare un tag, non una 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: si consiglia di 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 negli `import` e nei `require()`. Sarebbe bene, al termine, testare il progetto in ambiente Linux (per es. in una VM) per assicurarsi che non ci siano errori di questo tipo.