Applicazioni Web I – Esame #1 (scadenza 2023-06-25 alle 23:59)

“CMSmall”

⚠️ VERSIONE PRELIMINARE. La versione finale verrà pubblicata il 14 giugno. ⚠️

Progettare e implementare un’applicazione web per gestire un piccolo content management system (CMS) con un minimo insieme di funzionalità. L’applicazione deve soddisfare i seguenti requisiti.

Il CMS ha un “back-office” (cioè, l’area amministrativa e di gestione) e un “front-office” (cioè, la parte dell’applicazione web visibile pubblicamente a tutti, senza autenticazione).

Nel back-office, il CMS consente agli utenti autenticati di creare e gestire contenuti (cioè pagine). Ogni **pagina** ha le seguenti proprietà:

- Un *titolo*;
- Un *autore* (di default, l’utente loggato);
- Una *data di creazione*, cioè quando è stato creato;
- Una *data di pubblicazione*. Secondo il valore di questa data, la pagina può essere “draft” (data di pubblicazione assente), “programmata” (la data di pubblicazione è nel futuro), “pubblicata” (la data di pubblicazione è oggi o nel passato);
- Blocchi di *contenuto*.

Un blocco di **contenuto** può essere di tre tipi: header, paragrafo, o immagine. Una pagina deve avere almeno un header e uno degli altri due tipi di blocchi. Le immagini devono essere selezionabili da una lista di immagini precaricate (devono essere disponibili almeno 4 immagini). I blocchi possono essere riordinati durante la creazione e l’editing della pagina (come specificato nel seguito). Gli header e i paragrafi devono contenere del testo.

Ogni utente autenticato, dopo il login, può vedere tutte le pagine create (da qualunque autore) in una schermata dedicata, con la possibilità di:

- **Creare una nuova pagina**, inserendo tutte le proprietà necessarie e aggiungendo all’interno della pagina almeno un blocco header e almeno uno degli altri blocchi. La data di creazione e l’autore non possono essere cambiati. I blocchi possono essere *riordinati in qualsiasi momento* all’interno della pagina durante il processo di creazione. Come deve funzionare il meccanismo di riordinamento è lasciato allo studente (per es., delle frecce per muovere in su e in giù i blocchi). I blocchi possono anche essere rimossi durante il processo di creazione della pagina.
- **Editare una pagina esistente della quale è autore.** Editando la pagina l’utente autenticato può cambiare tutte le proprietà (tranne la data di creazione e l’autore all’interno della pagina) e il contenuto (incluso aggiungere, rimuovere e cambiare l’ordine dei blocchi), ma ci deve sempre essere almeno un header e un altro tipo di blocco).
- **Cancellare una pagina della quale** all’interno della pagina **è autore.**

Il CMS supporta un tipo speciale di utente autenticato, che ha il ruolo di **amministratore** dell'intera applicazione. Non c'è limite al numero di amministratori che possono essere presenti nel CMS. Qualunque amministratore può effettuare le operazioni di un utente autenticato e in aggiunta può:

- **Editare o cancellare qualsiasi pagina**, anche se l'utente non è l'autore
- **Cambiare l'utente autore di una pagina impostandolo ad un utente differente.**
- Impostare il nome del **sito web**, che deve essere mostrato in cima ad ogni schermata (sia nel back-office che nel front-office).

Nel front-office, invece, gli utenti sia autenticati che non autenticati (anonimi) vedranno l'intero sito web, con il nome definito, la lista delle pagine *pubblicate* ordinate in ordine cronologico per data di pubblicazione e possono vedere tutto il contenuto di ogni pagina, incluse le sue proprietà.

L'organizzazione delle funzionalità di questo testo 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 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`".
- 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 *almeno* 4 utenti, con almeno un utente che sia autore di due pagine, uno che non abbia creato pagine, una che sia un amministratore, e due pagine per stato (draft, pubblicata, programmata).

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 per creare una nuova** pagina e una della **lista di tutte le pagine**, entrambe come non amministratori. Le immagini vanno embeddate nel REDAME inerendo 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, **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

  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()`.