RELAZIONE DI VENERBA MIRCO 872653

COMPONENTI GRUPPO

- Penzo Andrea 873888
- Venerba Mirco 872653

NB. STRUTTURA DEL PROGETTO

- Ogni progetto (client, server, Cordova ed Electron), ha una cartella chiamata fileBat che contiene tutti i file eseguibili (in ambiente windows) di quel progetto
- Lato client
    - Cartella src
        - Cartella app
            - Cartella componenti
                - Cartella che comprende tutti i componenti generati con la cli di Angular con il comando ng generate component.
                - Inoltre, dentro ad alcuni di questi componenti è presente una cartella functions che serve per suddividere le funzioni del componente per tipologia. Come spiegheremo nei paragrafi successivi, non abbiamo creato tanti altri componenti e per questo, abbiamo sfruttato la tecnica dell'export di funzioni per far sì che dentro al file Typescript del componente siano presenti le firme delle funzioni ma l'implementazione è presente dentro alla cartella functions divise per tipologia.
            - Cartella classi
                - Dentro a questa cartella memorizziamo qualsiasi classe Typescript che abbiamo creato per il nostro progetto. Alcuni esempi possono essere l'oggetto inserzione, l'oggetto libro ecc.
            - Cartella Img
                - È l'unica immagine all'interno del nostro progetto e serve per la gestione dei messaggi di riferimento sfruttando la stessa tattica di WhatsApp
            - Cartella Servizi
                - Questa cartella contiene tutti i servizi che abbiamo creato sfruttando la cli di Angular con il comando ng generate service come, per esempio, il servizio per la gestione degli utenti, per la gestione dei libri ecc. e quindi ciascun servizio rappresenta l'interfacciamento dei componenti verso il server.
- Lato server
    - Il file main.ts contiene tutta la configurazione dell'ambiente e delle utility e l'import di tutti i moduli separati che sono i seguenti:
        - Cartella classi che rappresenta l'insieme dei modelli di dati sfruttando la libreria Mongoose.

- La cartella data che contiene tutti i dati di esecuzione di MongoDb
- La cartella datiDiProva che contiene tutti gli inserimenti dei dati di prova divisi per tipologia
- La cartella keys che contiene la gestione delle chiavi per la gestione del https anche se abbiamo implementato questo progetto con http per semplicità. Però sappiamo che l'https è più sicuro ma ci sono le chiavi e certificati da gestire e per questo, come suggerito a lezione, lo abbiamo sviluppato solo con richieste http
- La cartella routes che contiene le varie routes di express divise per tipologia
- Infine la cartella server che contiene la configurazione del server http, del link di mongodb, del https anche se commentato perché abbiamo implementato il progetto con l'http anche se meno sicuro.

COME ESEGUIRE IL PROGETTO

- NB. Questa spiegazione è in ambiente windows
- Lato client
    - Progetto web
        - Nella cartella fileBat ci sono tre file eseguibili:
            - AvviaClient.bat
                - Richiama lo script start del file package json che contiene "ng serve –open" che serve per eseguire il progetto Angular e aprirlo in automatico nel browser tramite l'aggiunta del flag --open
            - AvviaClientConInstallazione.bat
                - Prima di richiamare lo script start del file package json che contiene "ng serve –open", installo tutti i pacchetti e relative dipendenze con il comando npm install
            - ProntoPerCopiare.bat
                - Elimina il contenuto della cartella node_modules per poterlo inviare via mail più leggero
    - Progetto Electron
        - Nella cartella fileBat ci sono 4 file eseguibili:
            - AvviaElectron.bat
                - Richiama lo script start del file package json che contiene "electron --no-sandbox ." che serve per eseguire il progetto electron
            - AvviaElectronConInstallazione.bat
                - Prima di richiamare lo script start del file package json che contiene "electron --no-sandbox .", installo tutti i pacchetti e relative dipendenze con il comando npm install
            - BuildaElectronWindows.bat
                - Questo comando serve per creare il file eseguibile in ambito windows con il comando che è nello script run del package.json e contiene "npx electron-packager ./

ProgettoTaw2020 –platform=win32". Però prima di creare il file eseguibile, installo tutti i pacchetti e relative dipendenze con il comando npm install

- ProntoPerCopiare.bat
  - Elimina il contenuto della cartella node_modules e della cartella che contiene tutti i file necessari per l'eseguibile di Windows (nel nostro caso) per poterlo inviare via mail più leggero

- Progetto Cordova
  - Dentro alla cartella fileBat ci sono tre file eseguibili:
    - Questi file sottostanti usano la piattaforma browser in quanto non avevamo a disposizione un emulatore Android… Comunque nella parte finale del documento abbiamo scritto tutti i passaggi e modifiche da fare per poterla sviluppare anche per l'ambiente Android
    - AvviaCordovaBrowser.bat
      - Per prima cosa richiamiamo lo script builBrowser nel file package.json che è il comando "cordova build browser" che builda l'applicazione nella piattaforma del browser
      - Successivamente richiamiamo lo script runBrowser nel file package.json che è il comando "cordova run browser" che serve per eseguire il progetto cordova nel browser
    - AvviaCordovaBrowserConInstallazione.bat
      - Stessa cosa del file eseguibile AvviaCordovaBrowser.bat, ma prima di fare questo fa le seguenti cose:
        - Aggiunge la piattaforma browser al progetto attraverso il comando "cordova platform add browser"
        - Installa tutti i pacchetti e relative dipendenze con il comando npm install
    - ProntoPerCopiare.bat
      - Elimina le cartelle node_ modules, platforms, plugins per poter inviare successivamente via mail la versione più leggera del progetto

- Lato server
  - Dentro la cartella fileBat ci sono tre file eseguibili:
    - AvviaServer.bat
      - Per prima cosa richiama lo script compile del file package.json con il comando "tsc" che serve per compilare i file Typescript e produrre in output file javascript inserendoli nella cartella src
      - Successivamente richiama lo script start del file package.json che sfrutta il pacchetto "concurrently" per avviare contemporaneamente due servizi:

        - Con il seguente comando “mongod --dbpath ./data”, avvio il servizio per la gestione e connessione al MongoDb server mettendo i file prodotti nella cartella data
        - Con il comando “node ./src/main.js” avvio il web server in ambiente NodeJs usando i file prodotti in fase di compilazione con il comando “tsc”
    - AvviaServerConInstallazione.bat
        - Prima di richiamare gli script compile e start del file package.json, installo tutti i pacchetti e relative dipendenze con il comando “npm install”
    - ProntoPerCopiare.bat
        - Elimina le directory node_modules e data per poter successivamente inviare per mail la versione più leggera del progetto
- Alla fine dell’installazione dei pacchetti contenuti nei relativi package.json, potrebbero venire fuori alcuni warning:
    - alcuni dei quali riguardanti fsevents quando si installano i pacchetti in ambiente windows in quanto non è compatibile ma richiesto da Angular per il corretto funzionamento
    - altri possono riguardare eventuali nuove versioni e quindi delle vulnerabilità nelle versioni più vecchie
    - oppure eventuali versioni di altri pacchetti che vengono richiesti per avere la massima compatibilità però nel nostro caso, avendo funzionalità ridotte e funzionalità basilari, non succede niente anche se ci sono dei warning. Infatti, anche i progetti visti a lezione, alcune volte, presentavano dei warning in fase di installazione dei pacchetti.
        - Un esempio all’interno del nostro progetto server potrebbe essere il seguente: la libreria per la gestione dell’indice autoincrement, richiede una versione di mongoose minore però funzionano lo stesso in quanto usiamo solo funzionalità basilari compatibili lo stesso

COMPONENTI DEL SISTEMA

- Il progetto si può descrivere come un’applicazione MEAN
    - M: database Mongodb per memorizzare i dati come documenti in formato JSON e classificandoli in collezioni distinte
    - E: framework Express per la gestione delle routes e gestione funzioni middleware
    - A: Angular per la realizzazione dell’architettura software lato client
    - N: Node js per l’ambiente si sviluppo di un web server
- Libreria mongoose per l’interfacciamento alla base di dati sfruttando schemi e modelli
- Libreria Socket.io per la comunicazione real-time senza usare richieste http permettendo al server di inviare direttamente dati in broadcast all’utente lato client

UNA DESCRIZIONE DELL’ARCHITETTURA DEL SISTEMA, QUALI SONO I COMPONENTI E IN CHE MODO QUESTI CONCORRONO A SODDISFARE LE FEATURES RICHIESTE

- L’architettura è molto semplice e minimale e questa la si può vedere nei pdf d’uso.

- Lato client è basato principalmente sul framework Bootstrap per la gestione delle view e del layout oltre che ad Angular per l'architettura software. Il layout è basato su card (componente di Bootstrap) per dividere le varie funzionalità all'interno della pagina web e popup per le varie opzioni di modifica dati, filtraggio dati e inserimento dati. Avevamo cercato di dividere la pagina web con tutte le funzionalità in più componenti. Il problema principale che abbiamo avuto è che rispetto al nostro layout che abbiamo pensato, c'erano tutti componenti con almeno un bottone o un campo diverso e questo non ci ha permesso di strutturarla in più componenti uguali e richiamabili passando dei parametri per diversificare. Gli unici che ce lo permettevano potevano essere il componente per la card del logout e la card per l'implementazione futura, ma abbiamo preferito usare lo stesso standard di progettazione per tutta l'interfaccia e quindi abbiamo sfruttato l'export delle funzioni Javascript anche per il logout. Alcuni esempi sono i seguenti:
    - Nelle inserzioni create dall'utente loggato, sono visibili alcuni parametri come il prezzo di riserva e sono invisibili altre funzionalità come la chat privata in quanto non ha senso chattare con sé stesso.
    - Nelle inserzioni in vendita, l'utente loggato non può vedere il prezzo di riserva ma può usare la chat privata con l'inserzionista.
    - Nelle inserzioni partecipate, l'utente loggato può vedere il prezzo di riserva solo se sono già scadute e può anche chattare in privato con l'inserzionista.
    - Inoltre, scegliendo altre card rispetto alle inserzioni, svolgono funzionalità totalmente diverse, una per la gestione degli errori, una per le chat, una per il logout, una per l'aggiornamento dei dati ecc. e quindi non raggruppabili in una stessa card e implementabile tramite passaggio di parametri per diversificare.
- Per questo motivo, non potevamo suddividere il contenuto delle card in un componente denominato "riga inserzione" in quanto ciascun bottone "dettagli", mostrava contenuti diversi rispetto alla tipologia di inserzione selezionata.
- In caso si poteva strutturare il tutto creando vari componenti "riga inserzione" per ciascuna tipologia. Però secondo noi, veniva troppo codice identico per i vari componenti "riga inserzione" e quindi abbiamo preferito suddividere la pagina sfruttando altre tecniche quali:
    - Le card di Bootstrap
    - I modal di Bootstrap
    - L'export di funzioni in Javascript
- Di conseguenza i componenti che abbiamo sviluppato sono i seguenti:
        - Componente per l'home page del moderatore per implementare le seguenti funzionalità:
            - Modifica dei dati di un'inserzione
            - Eliminare un'inserzione
            - Vedere le inserzioni in vendita ed eventuale filtraggio
            - Aggiungere un moderatore, inserendo i dati di anagrafica e in automatico viene impostata la password temporanea "Temporanea2020" che successivamente al primo login, il moderatore nuovo la cambia
            - Eliminare uno studente

        - Vedere le statistiche
        - Aggiornamento dei dati per recuperare i dati aggiornati dopo che l'utente è da un po' connesso in quanto l'unica cosa real-time sono i messaggi e il loro invio e ricezione.
        - Effettuare il logout
        - Gestione degli errori avvenuti in fase di esecuzione e la loro stampa tramite alert bootstrap
        - Due card vuote che servono solo per riempire il layout della pagina, o volendo per la gestione di funzionalità future
- Componente per l'home page dello studente per implementare le seguenti funzionalità:
        - Inserimento nuova inserzione
        - Visualizzazione lista e dettagli inserzione create + eventuale chat pubblica + possibilità di eliminarle se sono scadute
        - Visualizzazione lista inserzioni attualmente in vendita + possibilità di fare una nuova offerta + possibilità di chattare in privato con l'inserzionista + possibilità di chattare in pubblico rispetto ad una inserzione specifica + possibilità di filtraggio
        - Visualizzazione delle proposte attuali che ha fatto l'utente loggato
        - Visualizzazione lista inserzioni vinte + possibilità di chat privata con l'inserzionista + possibilità di chat pubblica rispetto a quell'inserzione + visualizzazione di tutti i dati dell'inserzione selezionata
        - Visualizzazione della lista delle aste a cui l'utente loggato ha partecipato + visualizzazione dati dell'offerta selezionata + possibilità di chat privata e pubblica
        - Lista chat per vedere con che utenti l'utente loggato ha chattato in ordine di tempo + un eventuale notifica tramite alert bootstrap alla ricezione di nuovi messaggi (funzionalità solo se l'utente è loggato e se non sta chattando altrimenti la chat si aggiorna in real-time senza notifica). Non abbiamo gestito la notifica di nuovi messaggi quando l'utente non è connesso, ma si poteva gestire nel seguente modo: inserire un ulteriore campo nella tabella messaggi che rappresenta un flag di visualizzazione del messaggio.
        - Aggiornamento dei dati per recuperare i dati aggiornati dopo che l'utente è da un po' connesso in quanto l'unica cosa real-time sono i messaggi e il loro invio e ricezione.
        - Effettuare il logout
        - 1 card vuota che serve solo per riempire il layout della pagina, o volendo per la gestione di funzionalità future
        - Gestione degli errori avvenuti in fase di esecuzione e la loro stampa tramite alert bootstrap

- Componente per l'home page dell'utente non ancora loggato per implementare le seguenti funzionalità:
    - Visualizzazione lista inserzioni in vendita + possibilità di filtraggio + possibilità di sola visualizzazione della chat pubblica in quanto non autenticato
    - Aggiornamento dei dati per recuperare i dati aggiornati dopo che l'utente è da un po' connesso in quanto l'unica cosa real-time sono i messaggi e il loro invio e ricezione.
    - Effettuare il login
    - Effettuare la registrazione
    - Gestione degli errori avvenuti in fase di esecuzione e la loro stampa tramite alert bootstrap
    - 1 card vuota che serve solo per riempire il layout della pagina, o volendo per la gestione di funzionalità future
- Componente per effettuare il login:
    - Effettuare il login
    - Reindirizzamento per la registrazione
    - Reindirizzamento per resettare la password
    - Accedere senza autenticarsi per vedere solo la lista di inserzioni attualmente in vendita
- Componente per effettuare la registrazione
    - Effettuare la registrazione
    - Reindirizzamento per effettuare il login
    - Reindirizzamento per resettare la password
    - Reindirizzamento per accedere senza autenticarsi per vedere le sole inserzioni in vendita
- Componente per effettuare il recupero della password
    - Resettare la password inserendo l'e-mail dell'utente e viene impostata a quella di default "Temporanea2020". Quando viene fuori l'alert verde di bootstrap con scritto password resettata, si preme la textview con scritto effettua il login
    - Volendo si poteva gestire tramite l'aggiunta di un flag nella tabella utenti che rappresenta se la password è temporanea oppure se è definitiva, inviare per e-mail una password random e così si gestiva nel miglior dei modi la sicurezza.
- Componente per effettuare il cambiamento della password
    - Questo componente viene richiamato quando l'utente fa il login con username e password temporanea. Viene richiesta la password nuova, viene fatto un controllo e successivamente se è andato a buon fine, si viene reindirizzati alla schermata di login per fare l'accesso con le credenziali nuove
- Componente per effettuare la registrazione di nuovi utenti.
    - Effettuare la registrazione
    - Reindirizzamento per effettuare il login

- Reindirizzamento per effettuare il reset della password
- Possibilità di accesso senza autenticazione per vedere le sole inserzioni in vendita

UNA DESCRIZIONE DEL MODELLO DEI DATI UTILIZZATO. QUALI SONO LE COLLEZIONI E QUAL'È LA STRUTTURA DEI DOCUMENTI DI CIASCUNA COLLEZIONE CHE VENGONO MEMORIZZATI NEL DATABASE

- Noi abbiamo gestito il database ragionando come fosse relazionale. Di seguito la lista di collezioni e documenti.
- La nostra logica è la seguente:
    - Per ogni collezione, ciascun documento è caratterizzato da un index auto increment che viene calcolato in automatico sfruttando la libreria mongoose-auto-increment
    - Ogni utente contiene il ruolo che gli viene assegnato e contiene anche l'indice delle aste a cui ha partecipato
    - Ogni inserzione è collegata alla classe user tramite tre index che rappresentano l'utente che ha creato l'inserzione, l'utente che ha fatto l'ultima offerta, l'utente vincitore.
    - Inoltre, ogni inserzione è collegata alla classe libro tramite index per sapere che libro contiene questa determinata inserzione.
    - Ogni messaggio è riferito alla classe inserzione tramite un index per sapere a che inserzioni appartiene un certo messaggio, inoltre ciascun messaggio è collegato eventualmente ad un altro messaggio da cui eredita le caratteristiche privato o pubblico, mittente e destinatario ecc. sfruttando la stessa logica di WhatsApp
    - Inoltre, ciascun messaggio è collegato alla classe user tramite due index per sapere chi sono mittente e destinatario.
- Le collezioni sono:
    - Inserzione
        - idInserzione: number auto increment
        - utente: number index per accedere ai dati dell'utente corretto
        - libro: number index per accedere ai dati del libro corretto
        - dataInizio: Date
        - dataFine: Date con vincolo che sia maggiore della data attuale e della data di inizio
        - prezzoIniziale: number
        - prezzoAttuale: number con settaggio identico al prezzo iniziale in fase di inserimento di una nuova inserzione
        - utentePrezzoAttuale: number index per accedere ai dati dell'utente corretto
        - prezzoRiserva: number con vincolo che sia maggiore del prezzo iniziale
        - vincitore: number index per accedere ai dati dell'utente corretto
    - Libro
        - idLibro: number
        - nome: string
        - corsoDiStudi: string
        - universita: string
        - autore: string
        - annoPubblicazione: number con vincolo che sia compreso tra il 1970 e il 2030
        - edizione: number con vincolo che sia maggiore di 0

        - isbn: string
    - Messaggio
        - idMessaggio: number
        - idInserzione: number
        - messaggioRiferimento: number index per accedere ai dati del messaggio corretto, settato a 0 se senza messaggio di riferimento
        - oggetto: string
        - contenuto: string
        - data: Date
        - mittente: number index per accedere ai dati dell'utente corretto
        - destinatario: number index per accedere ai dati dell'utente corretto, settato a 0 se è un messaggio pubblico
    - User
        - idUser: number
        - ruolo: string[]
        - nome: string
        - cognome: string
        - username: string
        - email: string
        - areaGeografica: string
        - asteParteciapate: number[] index per accedere ai dati delle inserzioni a cui l'utente ha partecipato facendo un'offerta
        - salt: string usata per fare l'hash della password fornita al momento della registrazione, viene memorizzato nel database per i successivi confronti (login)
        - digest: string memorizza la password hashata

VINCOLI CHE ABBIAMO SCELTO

- L'utente quando vince un'inserzione, viene aggiunta tale inserzione nella card della lista delle inserzioni vinte, apre i dettagli e preme il bottone chat privata per procedere all'acquisto contattando lo studente venditore
- Un'inserzione contiene solo un libro
- La registrazione di nuovi moderatori avviene tramite la home page di un moderatore che deve inserire nome, cognome, e-mail, area geografica e poi viene inserita in automatico la password temporanea "Temporanea2020" per effettuare il primo accesso. Successivamente quando il nuovo moderatore fa il primo accesso, viene reindirizzato alla schermata per cambiare la password
- Alla fine di un'asta, se risulta vinta, viene notificata al venditore tramite lo spostamento nella lista delle inserzioni vendute mentre nell'utente che ha fatto l'offerta tramite lo spostamento nella lista delle inserzioni vinte
- Viene usata la stringa "Temporanea2020" come password temporanea e quindi si assume che questa non venga usata come password definitiva per un utente
- Le date di pubblicazione dei libri devono essere comprese tra il 1970 e il 2030 per favorire anche i libri di prossima pubblicazione
- Prezzi delle inserzioni devono essere valori interi

- Il destinatario di un messaggio pubblico viene rappresentato dal valore 0 in quanto non esiste alcun utente con id pari a 0 perché partono da 1.
- Il messaggio di riferimento è pari a 0 se un messaggio non è riferito a nessun'altro in quanto l'id dei messaggi parte da 1
- Qualsiasi errore che viene generato lato server, non viene stampato visibile in prima pagina, ma viene stampato dentro alla card di gestione errore come riportato nei pdf di spiegazione d'uso sfruttando un alert bootsstrap
- La notifica dei nuovi messaggi avviene solo se l'utente è loggato dentro all'applicativo in quanto non usiamo un flag nella tabella messaggi per verificare se è stato letto oppure no.
    - Se l'utente ha la chat aperta e quindi sta chattando in tempo reale, la chat si aggiorna
    - Se invece l'utente non sta chattando in tempo reale e quindi sta guardando le inserzioni o qualcos'altro, viene notificato un nuovo messaggio sfruttando un alert bootstrap dentro alla card delle chat. Questo lo si può vedere nel pdf d'uso lato studente

UNA DESCRIZIONE DELLE API FORNITE DALLA COMPONENTE SERVER. LA DESCRIZIONE DEVE CONTENERE IN MODO CHIARO LA LISTA DEGLI ENDPOINTS, DEGLI EVENTUALI PARAMETRI E IL FORMATO DEI DATI (JSON) CHE VENGONO SCAMBIATI NELLE RICHIESTE HTTP

- Parametri httpHeaders comuni a tutti gli endpoint TRANNE quello del login e quelli che non richiedono l'autenticazione: "authorization": "Bearer " + this.us.getToken(), "cache-control": "no-cache", "Content-Type": "application/json"
- I controlli per sapere se lo studente loggato è un moderatore o uno studente lo faccio in ogni endpoint e sfrutto req.user di express
- Ciascun json restituito tranne quello di root, hanno formati standard ossia:
    - Status code
        - 200 indica che la richiesta è stata eseguita correttamente
        - 501 indica che il server non supporta la funzionalità necessarie a soddisfare la richiesta
    - Endpoint
        - Rappresenta l'interfaccia alla quale si sta richiamando una funzionalità, può essere uno tra i seguenti: "/", "/studenti", "/moderatori", "/inserzioni", "/libri", "/persone", "/statistiche", "/messaggi", "/login"
    - Method
        - Rappresenta il metodo http usato nella richiesta e può essere uno tra i seguenti: "delete", "get", "post", "patch" e "put"
    - Error
        - Rappresenta un flag di errore e può essere settato solo a true o false
    - Message
        - Rappresenta un eventuale messaggio sia di successo che errore per capir cosa è successo
    - Reasons
        - Eventuali errori generati dai catch o altre istruzioni che vengono allegati per analizzarli lato client
    - Eventuali dati da allegare nel caso di estrazioni ecc.
        - Rappresentano liste di oggetti o singoli oggetti che vengono estratti o modificati e per questo inviati al client

- Di seguito nella tabella ci sono gli endpoint, il metodo, i parametri e la loro obbligatorietà, esempi di json restituiti sia in caso di successo e sia in caso di errore. Inoltre, si possono osservare se i parametri devono essere contenuti nel body oppure nell'url tramite parametri
- Infatti, noi abbiamo scelto due tipologie di parametri nelle richieste http:
    - La prima consiste nel body e quindi sono caratterizzati da questo format: req.body.nome
    - La seconda consiste nel passaggio dei dati sfruttando l'url tramite parametri anticipati da un ? e sono formati nel seguente modo: req.query.nome

Endpoint	**Metodo e spiegazione**	**Parametri**	**Json restituiti**
/	Get per ottenere la lista di endpoint disponibili	Nessuno	{ api_version: "1.0", endpoints: ["/", "/studenti", "/moderatori", "/inserzioni", "/libri", "/persone", "/statistiche", "/messaggi", "/login"] }
/statistiche	Get per ottenere le statistiche	Nessuno in quanto tramite req.user accedo ai dati dell'utente loggato e di conseguenza capisco se è un utente o un moderatore	In caso positivo: { statusCode: 200, endpoint: "/statistiche", method: "get", error: false, message: "Estrazione effettuata", reasons: null, asteParteciate: asteParteciate, asteVinte: inserzioniVinte, inserzioniCreate: inserzioniCreate } In caso negativo di errore: { statusCode: 501, endpoint: "/statistiche", method: "get", error: true, message: "Errore estrazione inserzioni create", reasons: errore } e message rappresenta il messaggio che imposto nei vari casi di errore e rappresenta una spiegazione
/libri	Get, per ottenere la lista di libri con la possibilità di filtrarli sfruttando alcuni di questi parametri	req.query.idLibro, req.query.libro, req.query.corsoDiStudi, req.query.universita, req.query.autore, req.query.annoPubblicazione, req.query.edizione, req.query.isbn sono opzionali	In caso positivo: { statusCode: 200, endpoint: "/libri", method: "get", error: false, message: "Libri estratti", reasons: null, libri: libri } In caso di errore: { statusCode: 501, endpoint: "/libri", method: "get", error: true, message: "Errore estrazioni libri", reasons: errore }
/libri	Put per inserire un nuovo libro	req.body.nome req.body.corsoDiStudi req.body.universita req.body.autore req.body.annoPubblicazione	In caso positivo: { statusCode: 200, endpoint: "/libri", method: "put", error: false, message: "Libro inserito", reasons: null, libro: libro }

		req.body.edizione req.body.isbn sono obbligatori	In caso negativo: { statusCode: 501, endpoint: "/libri", method: "put", error: true, message: "Non inserito", reasons: errore }
/libri	Patch per modificare un libro	req.body.idLibro req.body.nome req.body.corsoDiStudi req.body.universita req.body.autore req.body.annoPubblicazione req.body.edizione req.body.isbn sono obbligatori	In caso positivo: { statusCode: 200, endpoint: "/libri", method: "patch", error: false, message: "Libro modificata", reasons: null } In caso negativo: { statusCode: 501, endpoint: "/libri", method: "patch", error: true, message: "Errore modificare libro", reasons: errore }
/libri	Delete per eliminare libri	req.query.idLibro è obbligatorio	In caso positivo: { statusCode: 200, endpoint: "/libri", method: "delete", error: false, message: "Libri eliminati", reasons: null } In caso negativo: { statusCode: 501, endpoint: "/libri", method: "delete", error: true, message: "Errore eliminazione libri", reasons: errore }
/inserzioni	Put per inserire inserzione	req.body.utente req.body.libro req.body.dataInizio req.body.dataFine req.body.prezzoIniziale req.body.prezzoRiserva req.body.prezzoAttuale req.body.utentePrezzoAttuale req.body.vincitore sono obbligatori	In caso positivo: { statusCode: 200, endpoint: "/inserzioni", method: "put", error: false, message: "Inserzione inserita", reasons: null, inserzione: inserzione } In caso negativo: { statusCode: 501, endpoint: "/inserzioni", method: "put", error: true, message: "Non inserita", reasons: errore }
/inserzioni	Patch per modificare un'inserzione	req.body.idInserzione req.body.utente req.body.libro req.body.dataInizio req.body.dataFine req.body.prezzoIniziale req.body.prezzoRiserva req.body.prezzoAttuale req.body.utentePrezzoAttuale sono obbligatori	In caso positivo: { statusCode: 200, endpoint: "/inserzioni", method: "patch", error: false, message: "Inserzione modificata", reasons: null } In caso negativo: { statusCode: 501, endpoint: "/inserzioni", method: "patch", error: true, message: "Errore modificare inserzione", reasons: errore }

/inserzioni	Post per fare una nuova offerta	req.body.idInserzione req.body.idUser req.body.nuovaOfferta sono obbligatori	In caso positivo: { statusCode: 200, endpoint: "/inserzioni", method: "post", error: false, message: "Offerta inviata", reasons: null } In caso negativo: { statusCode: 501, endpoint: "/inserzioni", method: "post", error: true, message: "Errore salvataggio offerta partecipata", reasons: errore }
/inserzioni	Get per estrarre le inserzioni	req.query.idInserzione req.query.nomeLibro req.query.corsoDiStudi req.query.universita req.query.areaGeografica req.query.username req.query.prezzoAttualeMinimo req.query.prezzoAttualeMassimo sono opzionali	In caso positivo: { statusCode: 200, endpoint: "/inserzioni", method: "get", error: false, message: "Estrazione inserzioni effettuata", reasons: null, inserzioni: inserzioni } In caso negativo: { statusCode: 501, endpoint: "/inserzioni", method: "get", error: true, message: "Errore estrazioni inserzioni", reasons: errore }
/inserzioni	Delete per eliminare inserzioni	req.query.idInserzione è opzionale	In caso positivo: { statusCode: 200, endpoint: "/inserzioni", method: "delete", error: false, message: "Inserzioni eliminata", reasons: null } In caso negativo: { statusCode: 501, endpoint: "/inserzioni", method: "delete", error: true, message: "Errore eliminazione inserzione", reasons: errore }
/login	Get per fare il login	Header personalizzati: authorization: "Basic " + btoa(u.username + ":" + u.password), "cache-control": "no-cache","Content-Type": "application/json"	In caso positivo: { statusCode: 200, endpoint: "/login", method: "get", error: false, message: "Login effettuato", reasons: null, token: token_signed, utente: tokendata, temp: true } con temp il flag per la gestione della password temporanea In caso negativo: { statusCode: 501, endpoint: "/passport", method: "use", error: true, message: "Utente inesistente", reasons: null }
/messaggi	Get estrarre i messaggi	req.query.idMessaggio req.query.idInserzione req.query.messaggioRiferimento	In caso positivo: { statusCode: 200, endpoint: "/messaggi", method: "get", error: false, message: "Messaggi

		req.query.oggetto req.query.contenuto req.query.data req.query.mittente req.query.destinatario sono opzionali	estratti", reasons: null, messaggi: messaggi } In caso negativo: { statusCode: 501, endpoint: "/messaggi", method: "get", error: true, message: "Errore estrazioni messaggi", reasons: errore }
/messaggi	Put per inserire un messaggio	req.body.messaggioRiferimento req.body.idInserzione req.body.oggetto req.body.contenuto req.body.destinatario req.body.mittente sono obbligatori	In caso positivo: { statusCode: 200, endpoint: "/messaggi", method: "put", error: false, message: "Messaggio inserito", reasons: null, messaggio: messaggio } In caso negativo: { statusCode: 501, endpoint: "/messaggi", method: "put", error: true, message: "Inserzione inesistente", reasons: null }
/messaggi	Delete per eliminare messaggi	req.query.idMessaggio req.query.idUtente req.query.idInserzione Obbligatorio solo uno tra questi	In caso positivo: { statusCode: 200, endpoint: "/messaggi", method: "delete", error: false, message: "Messaggi eliminati", reasons: null } In caso negativo: { statusCode: 501, endpoint: "/messaggi", method: "delete", error: true, message: "Errore eliminazione messaggi", reasons: errore }
/moderatori	Patch per aggiornare la password temporanea	req.body.password req.body.confermaPassword è obbligatorio	In caso positivo: { statusCode: 200, endpoint: "/moderatori", method: "patch", error: false, message: "Password cambiata", reasons: null } In caso negativo: { statusCode: 501, endpoint: "/moderatori", method: "patch", error: true, message: "Errore salvataggio cambiamenti", reasons: errore }
/moderatori	Put per inserire un moderatore	req.body.nome req.body.cognome req.body.username req.body.email req.body.areaGeografica req.body.asteParteciapate sono obbligatori	In caso positivo: { statusCode: 200, endpoint: "/moderatori", method: "put", error: false, message: "Utente registrato", reasons: null } In caso negativo: { statusCode: 501, endpoint: "/moderatori", method: "put", error: true, message: "Errore registrazione", reasons: errore }

/persone	Get per estrarre le persone	nessuno	In caso positivo: { statusCode: 200, endpoint: "/persone", method: "get", error: false, message: "Persone estratte", reasons: null, utenti: users } In caso negativo: { statusCode: 501, endpoint: "/persone", method: "get", error: true, message: "Errore estrazioni persone", reasons: errore }
/studenti	Post per resettare la password di uno studente	req.body.email è obbligatorio	In caso positivo: { statusCode: 200, endpoint: "/studenti", method: "post", error: false, message: "Password resettata", reasons: null } In caso negativo: { statusCode: 501, endpoint: "/studenti", method: "post", error: true, message: "Errore salvataggio cambiamenti", reasons: errore }
/studenti	Patch per cambiare la password temporanea	req.body.password req.body.confermaPassword sono obbligatori	In caso positivo: { statusCode: 200, endpoint: "/studenti", method: "patch", error: false, message: "Password cambiata", reasons: null } In caso negativo: { statusCode: 501, endpoint: "/studenti", method: "patch", error: true, message: "Errore salvataggio cambiamenti", reasons: errore }
/studenti	Put per inserire uno studente	req.body.password req.body.nome req.body.cognome req.body.username req.body.email req.body.areaGeografica req.body.asteParticipate req.body.confermaPassword sono obbligatori	In caso positivo: { statusCode: 200, endpoint: "/studenti", method: "put", error: false, message: "Utente registrato", reasons: null } In caso negativo: { statusCode: 501, endpoint: "/studenti", method: "put", error: true, message: "Password non coincidono", reasons: null }
/studenti	Delete per eliminare studenti	req.query.idUtente è obbligatorio	In caso positivo: { statusCode: 200, endpoint: "/studenti", method: "delete", error: false, message: "Studente eliminato", reasons: null } In caso negativo: { statusCode: 501, endpoint: "/studenti", method: "delete", error: true, message: "Errore

			eliminazione studente", reasons: errore }
/studenti	Get per estrarre studenti	req.query.idUser req.query.nome req.query.cognome req.query.username req.query.email req.query.areaGeografica req.query.asteParteciate sono opzionali	In caso positivo: { statusCode: 200, endpoint: "/studenti", method: "get", error: false, message: "Studenti estratti", reasons: null, utenti: studenti } In caso negativo: { statusCode: 501, endpoint: "/studenti", method: "get", error: true, message: "Errore estrazione studenti", reasons: errore }

- Inoltre, ce ne sono altri due che consistono nel gestire eventuali endpoint inesistenti o di eventuali errori che vengono generati nella chiamata di alcuni endpoint.
- Inoltre, abbiamo creato un server molto robusto, prima di apportare eventuali modifiche fa tutti i controlli di esistenza, controlla gli indici di riferimento, controlla le relazioni tra i vari campi dell'oggetto, verifica che tutti i riferimenti puntino ad oggetti esistenti ecc. Un esempio potrebbe essere il seguente:
    - Nell'endpoint /inserzioni metodo patch, prima di poter modificare l'inserzione facciamo i seguenti controlli:
        - Il prezzo iniziale deve essere minore del prezzo di riserva
        - La data di fine deve essere maggiore della data di inizio
        - La data di fine deve essere maggiore della data attuale
        - Verifico che esista l'utente che ha creato l'inserzione con l'id ricevuto nel body della richiesta
        - Verifico che l'utente che ha creato l'inserzione sia uno studente
        - Verifico che esista un libro con l'id ricevuto nel body della richiesta
        - Verifico che esista un'inserzione con l'id dell'inserzione da modificare ricevuto nel body della richiesta
        - Verifico che esista un utente con l 'id dell'utente che ha fatto l'ultima proposta
        - Verifico che l'utente che ha fatto l'ultima proposta sia uno studente
        - Finalmente, se tutti i controlli sono andati a buon fine, posso modificare l'inserzione. Altrimenti ciascuna condizione ha un messaggio diverso di errore per riuscire a capire esattamente quale condizione non era stata soddisfatta

AUTENTICAZIONE

- Nel nostro progetto abbiamo usato la basic authentication, che consiste nell'invitare il client ad inviare la coppia username:password. Successivamente viene generato un token chiamato json web token che serve per verificare i permessi nelle richieste http successive
- La logica è la seguente:
    - L'utente usa l'applicativo, entra nella pagina del login ed inserisce le credenziali.

- Preme il bottone login e questo richiama la funzione typescript che a sua volta richiama il servizio della gestione dell'utente per effettuare il login
- Questo servizio effettua una richiesta http al server con i seguenti headers:
    - authorization: "Basic " + btoa(u.username + ":" + u.password)
    - "cache-control": "no-cache"
    - "Content-Type": "application/json"
- Dal header precedente si può notare la configurazione della basic authentication, ossia la coppia username:password. Prima di spedirla al server, si richiama la funzione btoa che serve per la codifica in base 64.
- Però sappiamo che la codifica in base 64 è una funzione invertibile e quindi anche se inviata oscurata, un altro utente può pescarla dal flusso e riconvertirla. Per questo la basic authentication dovrebbe essere sempre utilizzata in abbinata all'https
- Successivamente si richiama, tramite il passport che è il componente per la configurazione della strategia di autenticazione, la logica di verifica se la password coincide a qualche utente, oppure se l'utente è inesistente ecc. tramite user.validaPassword.
- Successivamente, se tutto è andato a buon fine, restituisce lato client il token e l'utente lo salva sfruttando il local storage del browser per riusarlo in futuro dopo la chiusura del browser. Mentre in fase di logout, si setta il token a stringa vuota.
- Invece nelle richieste successive, si allega negli header la seguente stringa: "authorization": "Bearer " + this.token che serve per passare il token che mi sono creato in fase di login al server per verificare se sono autorizzato nel fare certe operazioni
- Il ruolo dell'utente loggato lato server lo si verifica sfruttando la proprietà req.user

COMPONENTI SERVIZI E ROUTES

- Componenti
    - Home page del moderatore
        - Gestione inserzioni
        - Aggiunta moderatore
        - Cancellazione studenti
        - Gestione statistiche
        - Aggiornamento dati
        - Effettuare logout
        - Gestione errori
    - Home page dello studente
        - Inserimento inserzione
        - Lista inserzioni per tipologia + gestione
        - Gestione chat
        - Aggiornamento dati
        - Logout
        - Gestione errori
    - Home page dell'utente non loggato

        - Visualizzazione inserzioni in vendita
        - Effettuare il login
        - Effettuare la registrazione
        - Aggiornamento dei dati
        - Gestione degli errori
    - Pagina per effettuare il login
        - Effettuare il login
        - Reindirizzamento alla pagina per la registrazione
        - Reindirizzamento alla pagina di resettare la password
        - Reindirizzamento alla pagina di non autenticarsi
    - Pagina per effettuare la registrazione
        - Effettuare la registrazione
        - Reindirizzamento alla pagina per il login
        - Reindirizzamento alla pagina di resettare la password
        - Reindirizzamento alla pagina di non autenticarsi
    - Pagina per effettuare il reset della password
        - Effettuare reset della password
        - Reindirizzamento alla pagina per il login
        - Reindirizzamento alla pagina la registrazione
    - NB. La costruzione di altri componenti nel nostro caso non era possibile e questo è spiegato nella prima parte del documento e quindi per diversificare le aree abbiamo sfruttato le card, i modal di bootstrap e le funzioni export di javascript.
    - Volendo si poteva creare per ciascuna card un nuovo componente di Angular, però nel nostro caso erano tutti componenti diversi per almeno una caratteristica e/o bottone e per questo lo abbiamo suddiviso sfruttando la tecnica dell'export di funzioni in più file suddivisi per tipologia. Nel caso avessimo creato più componenti, dovevamo portare i dati anche a livello superiore sfruttando @Output() e passare i dati in input ad un altro componente inserendolo nel tag html come parametro.
- Servizi
    - Gestione inserzioni
        - Inserimento inserzione
        - Estrarre lista inserzioni partecipate
        - Estrarre lista inserzioni create proposte e vinte
        - Estrarre lista inserzioni in vendita
        - Possibilità di fare una nuova offerta
        - Estrarre lista generica di inserzioni
        - Eliminare inserzione
        - Modificare un'inserzione
        - Filtra inserzioni
        - NB. Le funzioni per estrarre le varie tipologie di inserzioni, abbiamo deciso di farlo in questo modo in quanto ciascuna prende parametri diversi ma si poteva farlo in una funzione unica che estrae tramite indice e poi analizzare

le varie inserzioni lato client e quelle che soddisfano certi vincoli si inserivano in una lista temporanea per la loro gestione futura

    - Gestione libri
        - Estrarre i libri
        - Modificare libri
        - Estrarre libro dall'id
        - Eliminare libri
        - Inserimento libri
    - Gestione messaggi
        - Estrarre la lista di messaggi
        - Eliminare messaggi
        - Inviare messaggi
    - Gestione utenti
        - Gestione token di accesso
        - Elimina studente
        - Estrarre lista degli studenti
        - Inserire un moderatore
        - Estrarre utente dall'id
        - Estrarre utente da username
        - Effettuare il login
        - Resettare password
        - Cambia password temporanea studente
        - Cambia password temporanea moderatore
        - Effettuare la registrazione
        - Rinnovare il token però non incluso nel progetto a livello implementativo
        - Effettuare il logout
    - Gestione socket io
        - Connessione
    - Gestione statistiche
        - Funzione per ottenere le statistiche
- Routes
    - Route: "", redirectTo: "/login"
    - Route: "login", Componente: LoginComponent
    - Route: "registrazione", Componente: RegistrazioneComponent
    - Route: "passworddimenticata", Componente: PassworddimenticataComponent
    - Route: "modificapassword", Componente: ModificapasswordComponent
    - Route: "homepagemoderatore", Componente: HomepagemoderatoreComponent
    - Route: "homepagestudente", Componente: HomepagestudenteComponent
    - Route: "homepagenonloggato", Componente: HomepagenonloggatoComponent

CREAZIONE VERSIONE MOBILE

- Per prima cosa, abbiamo creato il progetto Cordova con il seguente comando: "cordova create ProgettoTaw2020"

- Poi abbiamo aperto la directory www del progetto appena creato e abbiamo eliminato tutto il contenuto
- Poi abbiamo effettuato delle modifiche sul progetto web da convertire in Cordova:
    - Abbiamo modificato il tag base del file index.html e lo abbiamo aggiornato in “<base href="./">”
    - Poi abbiamo aggiunto l’import del file javascript di Cordova nel file index.html con il seguente codice: “<script type="text/javascript" src="cordova.js"></script>”
    - Poi abbiamo modificato il file main.ts aggiungendo l’evento device ready con il seguente codice: “document.addEventListener("deviceready", () => {contenuto}, false)”
    - Poi come ultima modifica (se si vuole creare un progetto Android), bisogna modificare l’indirizzo del web server mettendo l’indirizzo http://10.0.2.2:8080 per collegarsi alla sottorete dell’emulatore Android
    - Adesso che tutte le modifiche sono state fatte, bisogna aprire il file angular.json del progetto da convertire e nel campo outToDir mettere il percorso della cartella www del progetto di Cordova
    - Adesso bisogna aprire il terminale nella cartella del progetto da convertire e usare il seguente comando “ng build” che serve per buildare l’applicazione e i file che vengono generati, vengono reindirizzati in automatico nella cartella www del progetto di Cordova sfruttando la proprietà outToDir del file angular.json
    - A questo punto che abbiamo creato e ultimato il nostro progetto di Cordova, bisogna annullare tutte le modifiche del progetto web per farlo tornare al punto di partenza
- Adesso che abbiamo apportato tutte le modifiche e abbiamo un progetto di Cordova quasi funzionante, bisogna aggiungere le piattaforme, nel nostro caso abbiamo usato il browser perché non avevamo un emulatore Android per testarlo. Ma di seguito scriviamo tutti i passaggi per ciascuna piattaforma
- Per la piattaforma browser:
    - Bisogna aggiungere la piattaforma del browser nel progetto con il seguente comando “cordova platform add browser”
    - Poi bisogna buildare il progetto con il seguente comando “cordova build browser”
    - Ed infine bisogna avviare il progetto con il comando “cordova run browser”
- Per la piattaforma Android:
    - Per prima cosa bisogna aggiungere la piattaforma Android al progetto con il comando “cordova platform add android”
    - Poi bisogna buildare l’applicazione con il comando “cordova build android”
    - Poi bisogna installare l’apk generata sull’emulatore Android con il comando “adb install platforms/android/app/build/outputs/apk/debug/app-debug.apk”
    - Però c’è un problema che consiste nelle richieste http nelle versioni nuove di Android in quanto non sono possibili. Sono ammesse solo richieste https. Per risolvere questo problema bisogna modificare il file config.xml:
        - aggiungere: “<application android:usesCleartextTraffic=”true”/>” dentro al tag <edit-config>, e successivamente aggiungere anche il namespace di Android

- Quando sono state apportate anche queste modifiche, ribuildarla, reinstallarla e tutto funziona

CREAZIONE VERSIONE DESKTOP

- Per prima cosa abbiamo creato una cartella ProgettoTaw2020Electron
- Successivamente abbiamo creato il file package.json per tutte le dipendenze e creato il file main.js che genera la finestra del progetto nel desktop
- Adesso bisogna aprire la cartella del progetto da convertire e fare una modifica
- Modificare il tag base nel index.html, scrivendo “<base href="./">”
- Bisogna aprire il file angular.json del progetto da convertire e nel campo outToDir mettere il percorso della cartella dist del progetto di Electron
- Adesso bisogna aprire il terminale nella cartella del progetto da convertire e usare il seguente comando “ng build” che serve per buildare l’applicazione e i file che vengono generati, vengono reindirizzati in automatico nella cartella dist del progetto di Electron sfruttando la proprietà outToDir del file angular.json
- A questo punto che abbiamo creato e ultimato il nostro progetto di Electron, bisogna annullare tutte le modifiche del progetto web per farlo tornare al punto di partenza
- Adesso che abbiamo il nostro progetto Electron finito, bisogna costruire il file eseguibile per le varie piattaforme e nel nostro caso in ambiente Windows.
- Per questo motivo apriamo il terminale nella cartella del progetto Electron e diamo il comando “npx electron-packager ./ ProgettoTaw2020 –platform=win32”
- Adesso avremo una nuova cartella all’interno del progetto Electron che contiene sia il file eseguibile e anche tutti i file necessari per la sua costruzione.
- Doppio click nel file eseguibile per aprirlo