Script

Progetto Tecnologie Internet

1. **Introduzione**
2. **Requisiti funzionali**
3. **Struttura del progetto**
4. **Database**
5. **Tecnologie**
6. **Test Automatici**
7. **Guida Utente**

Introduzione

Questo progetto presenta un'applicazione web full-stack che integra funzionalità di messaggistica istantanea, gestione utenti e condivisione file.

L'architettura si divide in un backend basato su Node.js con Express.js e un frontend sviluppato in React. L'applicazione offre un sistema di autenticazione, chat in tempo reale tramite Socket.io, e la possibilità di caricare e scaricare file.

Requisiti funzionali

Di seguito, si elencano le funzionalità globali dell'applicazione, suddivise tra backend e frontend, con gli strumenti.

Backend:

Servizio	*Funzionalità*	*Strumenti*
Autenticazione e Gestione Utenti	- Registrazione e login utenti - Gestione profili e contatti	Express.js, bcryptjs, jsonwebtoken
Chat in Tempo Reale	- Creazione e gestione di chat individuali e di gruppo - Invio e ricezione di messaggi in tempo reale	socket.io
Gestione File	- Upload e download di file	Multer
Persistenza Dati	- Memorizzazione di utenti, chat e messaggi	MongoDB, Mongoose
Sicurezza e Protezione delle Route	- Middleware per l'autenticazione delle richieste	Express.js middleware

Frontend:

Servizio	*Funzionalità*	*Strumenti*
Interfaccia Utente Reattiva	- Visualizzazione di chat, contatti e messaggi	React, Tailwind CSS
Gestione dello Stato dell'Applicazione	- Mantenimento dello stato di autenticazione e delle connessioni socket	React Context API, Zustand
Routing e Navigazione	- Gestione delle diverse viste dell'applicazione	React Router
Notifiche Utente	- Sistema di notifiche per nuovi messaggi e eventi	React Hot Toast

Struttura del progetto

L'architettura scelta per questo progetto adotta il pattern MVC (Model-View-Controller) sul lato server, con una chiara separazione tra la logica di business (controllers), i modelli dei dati (models) e le route API (views in questo contesto).

Il frontend, sviluppato con React, implementa un'architettura basata su componenti, utilizzando hooks e context per la gestione dello stato. L'applicazione segue il principio del Single Page Application (SPA), con il routing gestito lato client. La comunicazione tra frontend e backend avviene principalmente attraverso API RESTful, con l'aggiunta di WebSocket per le funzionalità in tempo reale.

Sul lato server, si mantiene un certo grado di separazione delle responsabilità dei diversi moduli (autenticazione, chat, gestione file). Questo approccio favorisce la scalabilità e la manutenibilità del codice. L'utilizzo di Socket.io introduce un elemento di architettura event-driven per le funzionalità di chat in tempo reale.

Database

Il database utilizzato in questo progetto è MongoDB. Le interazioni con esso vengono facilitate dal modulo *mongoose* un popolare ODM (Object Data Modeling) per MongoDB.

Di seguito, una breve descrizione dei modelli presenti nella cartella *models*:

- **User**: gli utenti della piattaforma. Gli attributi includono *fullName*, *username*, *password* (salvata dopo essere stata hashata per maggiore sicurezza), *gender*, *profilePic*, e *contacts* (un array di riferimenti ad altri utenti, indicando una relazione tra utenti).
- **Chat**: una chat privata o un gruppo. Include *participants* (i partecipanti alla chat), *messages* (i messaggi inviati nella chat), *isGroup* (un booleano che indica se la chat è un gruppo), *groupName*, *groupImage*, e *admin* (l'amministratore del gruppo, se è un gruppo).
- **Message**: i messaggi inviati dagli utenti. Include *senderId* (utente che ha inviato il messaggio), *chatId* (chat a cui il messaggio appartiene), *isFile* (booleano che indica se il messaggio è un file), *fileName*, *message* e *opened* (un booleano che indica se il messaggio è stato aperto).

Tecnologie Usate

- **Node.js:** Ambiente di runtime JavaScript lato server. Scelto per la sua efficienza nell'handling di operazioni I/O asincrone e per la vasta ecosistema di pacchetti npm.
- **Express.js:** Framework web per Node.js. Utilizzato per la sua semplicità e flessibilità nella creazione di API RESTful e per la gestione efficiente delle route.
- **MongoDB:** Database NoSQL orientato ai documenti. Selezionato per la sua scalabilità e flessibilità nello schema, ideale per dati non strutturati come chat e messaggi.
- **Mongoose:** ODM (Object Data Modeling) per MongoDB e Node.js: Scelto per semplificare le interazioni con MongoDB, fornendo una struttura per la modellazione dei dati e validazione.
- **Socket.io:** Libreria per la comunicazione real-time bidirezionale. Utilizzata per implementare funzionalità di chat in tempo reale, consentendo una comunicazione istantanea tra client e server.
- **React:** Libreria JavaScript per la costruzione di interfacce utente. Efficiente nel rendering e crea facilmente componenti riutilizzabili.
- **Vite:** Build tool e dev server per applicazioni web moderne. Selezionato per la sua velocità nel bundling e hot module replacement, migliorando l'esperienza di sviluppo.
- **JSON Web Tokens (JWT):** Standard per la creazione di token di accesso. Impiegato per l'autenticazione sicura e la gestione delle sessioni utente.
- **Bcrypt.js:** Libreria per l'hashing delle password. Scelta per la sua robustezza nella protezione delle credenziali utente, implementando salt e hashing sicuri.
- **Multer:** Middleware per Node.js per la gestione di upload di file multipart/form-data. Utilizzato per facilitare l'upload e la gestione dei file nel backend.

Test Automatici

L'approccio utilizzato per testare gli endpoint del backend nel file si basa sull'uso dei moduli *supertest* e *jest*. Questo metodo prevede la scrittura di test automatizzati che simulano richieste HTTP ai vari endpoint del server per verificare le risposte attese.

Il procedimento generale è il seguente:

1. **Configurazione dell'ambiente**: Importazione delle Dipendenze e Simulazione del server.
2. **Definizione dei Test**: I test sono organizzati in gruppi logici: test per gli endpoint di autenticazione, per gli endpoint di upload, per gli endpoint di messaggistica, etc…
3. **Simulazione delle Richieste HTTP**: Per ogni test, vengono inviate richieste HTTP al server.
4. **Verifica delle Risposte**: Si verificano le risposte ricevute utilizzando asserzioni (i valori attesi). Questo include la verifica dello status code della risposta e la presenza di

determinate proprietà nel corpo della risposta (ad esempio, la presenza di un token di autenticazione o di un messaggio di successo).

Guida Utente

Per installare e inizializzare localmente il progetto assicurati di avere installato Node.js e npm sul tuo sistema prima di procedere.

1. Clona il Repository:

   git clone https://github.com/giacomo-folli/react-chat.git

2. Installa le dipendenze del backend (dalla cartella *backend/*):

   npm install

3. Installa le dipendenze del frontend (dalla cartella *frontend/*):

   npm install

4. Crea un file *.env* nella directory del backend per configurare le variabili d'ambiente necessarie, come il collegamento al database MongoDB, le chiavi segrete per JWT, ecc. Puoi trovare un esempio delle variabili richieste nel file *.env.example*.
5. Avvia il backend (dalla cartella *backend/*):

   npm run server

6. Avvia il frontend (dalla cartella *frontend/*):

   npm run dev

Quest'ultimo comando avvierà il server di sviluppo. Per accedervi basterà recarsi all'indirizzo indicato nel terminale (tipicamente *http://localhost:3000*).