Nova cultura remotius

Applicazioni e Servizi Web

Alessandro Pioggia - 0001102149 {alessandro.pioggia2@studio.unibo.it}

31 Agosto 2023

0.1 Introduzione

L'idea alla base di Nova-cultura-remotius è stata concepita durante la pandemia globale sars-cov, al fine di fornire uno strumento didattico che permetta a professori, di erogare attraverso piattaforme online, lezioni accessibili a chiunque. Si tratta di una applicazione web, che consente l'accesso sia a studenti che ad insegnanti e quindi offre funzionalità diversificate in funzione del tipo di accesso. Non si tratta di una struttura admin/user, ogni utente è sullo stesso livello e si tratta di un'interazione bilaterale. L'aspetto interessante riguarda il fatto che di base, una piattaforma di questo genere, da quel che ho potuto appurare, in commercio non è ancora presente e le caratteristiche che a mio avviso la rendono unica sono le seguenti:

- Link di generazione: le lezioni vengono erogate esclusivamente online, creando e condividendo attraverso la chat, riunioni teams. Nell'applicativo presentato purtroppo non è stato possibile gestire la generazione automatica dei link di riunione teams, in quanto la API di riferimento richiede l'utilizzo di Microsoft Graph, che è a pagamento. Se dovesse però essere distribuita a livello commerciale, sarebbe una prerogativa, anche se non altera il funzionamento dell'applicativo, che in sè ha già le funzionalità richieste. Inoltre, non si tratta solo di una questione di costo, i sistemi di meeting online, per essere inclusi all'interno di un applicativo richiedono il consenso dell'utente, dopo accettazione di una normativa della privacy estesa.

- Gestione fondi: i pagamenti vengono gestiti internamente, dunque non è necessario memorizzare i dati della carta, è sufficiente caricare dei fondi "una tantum", che verranno convertiti in token, per l'acquisto delle lezioni.

- Tariffa fissa: una caratteristica molto interessante è rappresentata dal costo, in quanto fisso (1 token, 1 lezione), per evitare sovrapprezzi dovuti all'autorità e prestigio del professore. Un'idea alternativa può essere quella di parametrare il prezzo della lezione alla valutazione complessiva del docente, attraverso una suddivisione in scaglioni. Ad esempio, le lezioni dei professori con media valutazione 5, costano 2 token, mentre le altre 1 token. In questo modo si premia la qualità e si attribuisce valore al sistema di valutazione.

- Scheduling dinamico: l'organizzazione delle lezioni, sia lato studente che insegnante avviene attraverso degli scheduler dinamici e che offrono tante funzionalità, non sono noti servizi di tutoring online che la includano;

- Privacy: di base, al momento del rilascio, il servizio richiede davvero poche informazioni personali, non è nemmeno necessario il numero di telefono e l'indirizzo. Questo punto è centrale nel progetto, in quanto per ovviare all'assenza di queste informazioni, è stata sviluppata una chat interattiva, che permetta di comunicare senza necessariamente avere informazioni precise. Nel caso in cui un insegnante non erogasse la lezione, il proprietario si

assume la resposabilità di rimborso, a proprie spese, se facilmente verificabile. L'applicazione inoltre prevede volutamente l'assenza di foto profilo, per evitare che possano essere fonte di discriminazione di qualsiasi tipo.

Una applicazione web con la quale questo progetto può essere confrontato è www.superprof.it, servizio noto e molto utilizzato a livello italiano con un'utenza molto ampia. Nonostante abbia preso come riferimento il sito indicato, ho deciso di implementare nova-cultura-remotius come un gestionale, creando un margine di apprendimento, che rende l'interazione molto più veloce e flessibile.

0.2 Requisiti

Il progetto mira a sviluppare una piattaforma di teaching online che offrirà a studenti ed insegnanti un ambiente interattivo e sicuro per connettersi e partecipare a lezioni online. Si tratta di un vero e proprio ambiente integrato, arricchito da funzionalità con lo scopo di facilitare l'apprendimento e l'insegnamento.

0.2.1 Funzionalità Obbligatorie

- **Accesso come Insegnante o Studente:** Gli utenti potranno registrarsi e accedere alla piattaforma come insegnanti o studenti.
- **Bacheca delle Lezioni:** Una bacheca mostrerà le lezioni disponibili agli studenti, consentendo loro di filtrare in base a diversi parametri. Si privilegeranno i professori con recensioni più alte.
- **Prenotazione e Disponibilità in Tempo Reale (Calendario):** Gli insegnanti potranno aggiornare la loro disponibilità, mentre gli studenti potranno prenotare lezioni.
- **Portafoglio Virtuale con Token:** Gestione dei token per l'accesso alle lezioni.
- **Chat Integrata:** Confronto diretto tra studenti e insegnanti.
- **Sistema di Autenticazione Sicuro:** Protezione avanzata delle credenziali degli utenti.
- **Sistema di Recensioni:** Feedback sulle lezioni e sugli insegnanti per mantenere elevata la qualità del servizio.

0.2.2 Funzionalità Opzionali

- **Gestione delle Sessioni attraverso JWT:** Implementazione di JSON Web Tokens per sessioni sicure.
- **Autenticazione a più Fattori.**

- **Sistema di Monitoraggio delle Prestazioni:** Valutazione del progresso degli studenti e feedback personalizzato.
- **Gamification:** Badge, obiettivi e token extra per motivare gli studenti.
- **Sistema di Notifiche.**

0.2.3 Principali Tecnologie Utilizzate

- **Stack MEAN:** MongoDB, Express, Angular, NodeJs.
- **socket.io**
- **nestjs:** Framework lato server, basato su express.

0.3 Design

0.3.1 Design dell'architettura di sistema

L'applicativo è stato realizzato sfruttando lo stack MEAN, come già anticipato nell'introduzione ed approfondito nel capitolo delle tecnologie. Lato front-end si è dunque optato per AngularJs, tecnologia la quale architettura si può mappare in un MVVM (Model-View-ModelView). Il mapping è possibile dal momento che:

- Model → servizi e models.ts, ovvero i file che forniscono il dato e lo strutturano;
- View → template;
- ModelView → file del componente, in typescript, che fa da raccordo fra Model e View, legando la logica al template, sfruttando ad esempio il binding.

In aggiunta ad angular, per il templating e la visualizzazione dei dati, si è utilizzato devextreme, un framework con licenza gratuita, che mette a disposizione tool per la cura delle interfacce grafiche. All'aspetto architetturale è stata data molta importanza, a questo proposito, lato back-end, si è optato per NestJs, il quale permette di organizzare i controller ed i servizi in moduli separati, mantenendoli il più possibile indipendenti fra loro (approfondimento nel capitolo delle tecnologie).

0.3.2 Metodologie di sviluppo

Ai fini pratici, è stato preferito optare per un approccio UCD (User Centered Design), trattandosi di un progetto accademico e non avendo a disposizione persone addette al controllo qualità. A questo proposito, è stata sfruttata la virtualizzazione e dunque la creazione di Personas che potessero il più possibile

rispecchiare le necessità dell'utente medio. L'applicazione, con l'ausilio delle tecnologie utilizzate, ha rispettato i seguenti principi: Responsive design, KISS principle(rispettati gli standard di usabilità per evitare complicazioni) e Desktop first (reso possibile da devextreme, dal momento che i componenti sono già ottimizzati e strutturati per essere utilizzati sia lato mobile che desktop). Trattandosi di un progetto individuale non è stato necessario sfruttare la metodologia di sviluppo AGILE, che probabilmente, viste le dimensioni, in ogni caso sarebbe una soluzione probabilmente eccessiva.

Mockup

Inizialmente, sono stati realizzati due mockup, per facilitare la realizzazione dell'applicazione, l'utilizzo di devextreme ha permesso di ridurre al minimo la fase di prototipazione, in quanto i componenti presentavano già interfacce complete, bastava solamente adattare i dati. Essendo desktop first, non è stata inclusa la versione mobile, perché generata automaticamente da devextreme ed i suoi componenti.

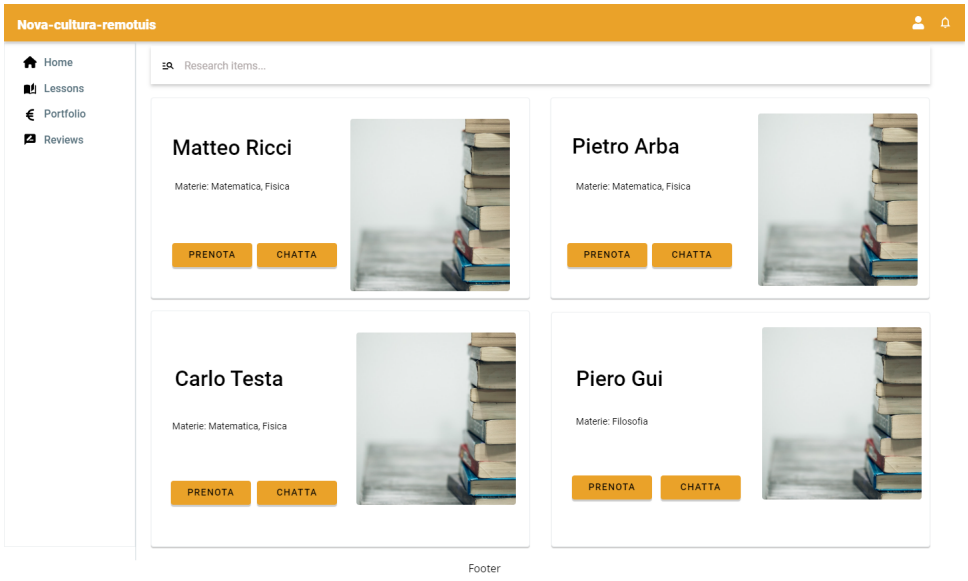

Figure 1: Homepage, come si può osservare è composta dal template base (sidebar menu, header, footer), inoltre la caratteristica dell'interfaccia è la presenza di una lista di card dinamica, con la quale si può interagire.

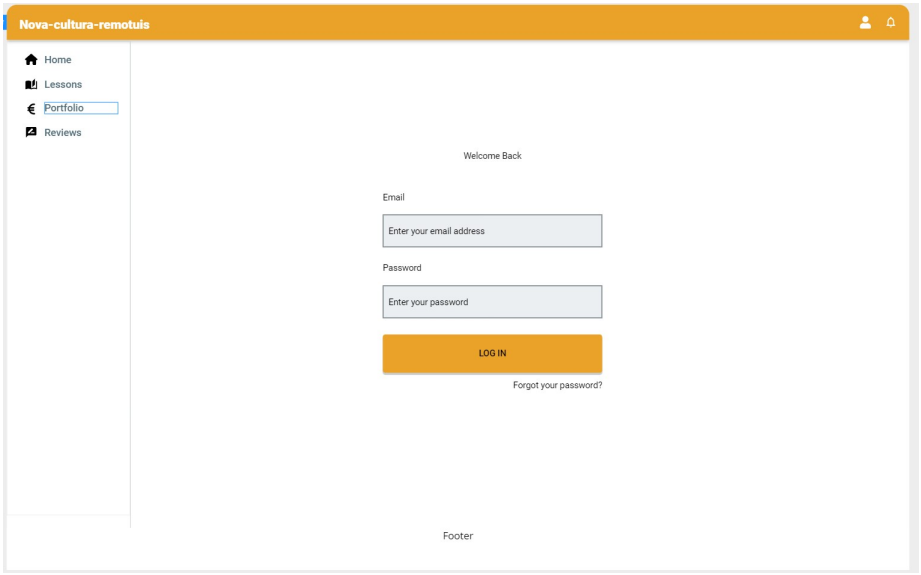

Figure 2: Pagina di login

0.3.3 Target user analysis

Piattaforma di Prenotazione delle Lezioni Online

Descrizione: Un'applicazione web che mette in contatto studenti e professori, permettendo agli studenti di prenotare lezioni private online con vari professori. Ogni lezione ha un costo e gli studenti possono recensire le lezioni e i professori dopo ogni sessione. La piattaforma offre anche una chat per la comunicazione in tempo reale tra studenti e professori.

Personas **1. Persona: Maria**

- *Ruolo*: Studente
- *Età*: 20 anni
- *Obiettivi*: Migliorare le sue competenze in materie specifiche, trovare professori qualificati e avere un'esperienza di apprendimento personalizzata.
- *Caratteristiche*: Studia ingegneria all'università; cerca lezioni soprattutto in matematica e fisica; vuole sapere quanto costerà ogni lezione prima di prenotare; si fida molto delle recensioni quando sceglie un professore; usa la chat per chiedere chiarimenti sui contenuti della lezione prima di prenotare.

2. Persona: Prof. Giuseppe

- *Ruolo*: Professore
- *Età*: 45 anni
- *Obiettivi*: Aumentare il suo reddito offrendo lezioni private, raggiungere un'ampia base di studenti e ottenere feedback costruttivo per migliorare.
- *Caratteristiche*: Insegna storia e filosofia; ha 20 anni di esperienza nell'insegnamento; vuole impostare i suoi orari e tariffe per le lezioni; legge le recensioni per capire le esigenze degli studenti e migliorare la sua offerta; usa la chat per coordinare gli orari e discutere i contenuti delle lezioni con gli studenti.

3. Persona: Luca

- *Ruolo*: Studente
- *Età*: 28 anni
- *Obiettivi*: Imparare nuove competenze per ragioni personali e professionali in un ambiente flessibile.
- *Caratteristiche*: Lavora a tempo pieno in un'azienda IT; interessato a lezioni di lingue straniere e programmazione; preferisce lezioni serali o durante il weekend; vuole prenotare lezioni con professori che hanno una buona reputazione e recensioni positive; usa la chat per risolvere dubbi tecnici o linguistic.

Target User Analysis

- *Età*: Gli utenti hanno un'età compresa tra i 18 e i 50 anni.
- *Educazione*: Gli studenti possono essere sia al liceo/università sia lavoratori che cercano formazione continua.
- *Frequenza d'uso*: Gli studenti potrebbero utilizzare l'applicazione settimanalmente o mensilmente, mentre i professori potrebbero accedere giornalmente.
- *Esigenze e preferenze*: Gli utenti cercano flessibilità negli orari, chiarezza nelle tariffe, affidabilità nella qualità dell'insegnamento e una piattaforma intuitiva.
- *Tecnologia*: Gli utenti sono abbastanza competenti dal punto di vista tecnologico e utilizzano dispositivi mobili e desktop per accedere alla piattaforma.

0.4 Tecnologie

Lo stack utilizzato nel progetto è MEAN, il quale comprende:

- MongoDb;
- ExpressJs;
- Angular [1];
- NodeJs.

I linguaggi utilizzati sono invece:

- Typescript;
- Html;
- css, scss;

Sempre rimanendo coerenti con lo stack tecnologico scelto, sono stati utilizzati due framework, per aumentare la qualità della code base, sia lato client che server, ovvero:

- Nestjs (back-end); [4]
- Devextreme (front-end); [2]

Infine è stato utilizzato mongoose [3], un orm per interagire con il database, oltre a socket.io per la creazione di una chat real time [5].

0.4.1 Nestjs

NestJS è un framework progressivo per la creazione di applicazioni server-side efficienti e scalabili con JavaScript e TypeScript. Basato su express, combina elementi di OOP (Object Oriented Programming) e permette di rendere le applicazioni facilmente manutenibili, grazie alla sua architettura, che comprende:

- moduli;
- decoratori;
- dependency injection, punto cardine di Nestjs, consente una gestione intelligente delle dipendenze;
- Interceptors e guardie;

Questo facilita la creazione di applicazioni modulari, scalabili e facilmente testabili. La struttura dei controller, annotazioni comprese, ricorda quella di .NET core, framework di Microsoft. Infine, è importante considerare che si tratta di una tecnologia molto utilizzata

Figure 3: Descrizione del framework da parte degli autori

0.4.2 Devextreme

L'interfaccia grafica e l'organizzazione del design è dipesa fortemente da Devextreme, un framework che permette di creare interfacce responsive, accessibili e con una buona usabilità. DevExtreme è una suite di componenti UI per la creazione di applicazioni web, mobile e desktop responsive e ricche di funzionalità. È sviluppata da DevExpress e supporta molteplici framework, come Angular, React, Vue e jQuery. Tutto ciò ha permesso di ridurre in maniera importante il tempo necessario per sviluppare l'interfaccia, viste le numerose funzionalità. In particolare, il componente di scheduling, per gestire la prenotazione delle lezioni, è stato molto semplice da realizzare, nonostante la complessità dei task. In particolare, offre:

- componenti UI;
- supporto multimediale;
- strumenti di sviluppo;

0.4.3 Socket.io

Socket.IO è una libreria che permette una comunicazione a bassa latenza, bidirezionale e basata su eventi tra un client e un server. È costruita sopra il protocollo WebSocket e offre garanzie aggiuntive come il fallback a HTTP long-polling o la riconnessione automatica. A partire dalla versione 3, Socket.IO offre ora un supporto di prima classe per TypeScript.

Il canale bidirezionale tra il server Socket.IO (Node.js) e il client Socket.IO (browser, Node.js o un altro linguaggio di programmazione) viene stabilito con

una connessione WebSocket ogni volta che è possibile e utilizzerà l'HTTP long-polling come fallback.

Il Codice di Socket.IO

Il codice di Socket.IO è diviso in due strati distinti:

1. La base a basso livello: chiamata Engine.IO, è il motore interno di Socket.IO.
2. L'API ad alto livello: ovvero Socket.IO stesso.

0.5 Codice

0.5.1 Back-end - web server

Come annunciato, il server web è stato implementato attraverso il framework nodejs, che rende più solida l'architettura del progetto, aumentando la qualità e durabilità del codice. Si sfruttano dunque moduli software, in cui ogni modulo è indipendente e gestisce le richieste in entrata per poi comunicare con il database attraverso un servizio dedicato. Ho deciso di sfruttare il repository pattern per gestire la comunicazione, di conseguenza ogni servizio si appoggia ad un repository. Invece per quanto riguarda il web-socket, la struttura rimane in un modulo isolato ed oltre al controller ed il servizio è presente un chat-gateway, per l'elaborazione di richieste ws in real-time. Quindi nella parte sottostante verranno mostrate entrambe le modalità.

Controller

```
//dependency injection del servizio
constructor(private readonly lessonService: LessonService
    ) {}

@UseGuards(AuthGuard('jwt'))
@Post('create')
async createOne(
@Body() dto: LessonDto,
@Headers('authorization') authHeader: string,
) {
try {
        const token = jwt.decode(authHeader.split(' ')
            [1]);
        return await this.lessonService.createLesson(dto,
             token);
} catch (error) {
        throw new HttpException(error.message, HttpStatus
            .BAD_REQUEST);
}
}
```

Servizio

```
async createLesson(dto: LessonDto, token: any) {
try {
        await this.lessonsRepository.create({
                id: uuidv4(),
                teacherMail: token?.mail ?? '',
                studentMail: '',
                startTime: dto.startTime,
                endTime: dto.endTime,
                subject: dto.subject,
                notes: dto.notes,
        });
} catch (error) {
        throw error;
}
}
```

Repository

```
constructor(
@InjectModel(Lesson.name) private lessonModel: Model<
    LessonDocument>,
) {}

async create(lesson: Lesson): Promise<Lesson> {
const createdUser = new this.lessonModel(lesson);
return createdUser.save();
}
```

0.5.2 Back-end - web socket

Gateway

```
// imports...

@WebSocketGateway({
        namespace: '/chat',
        cors: {
                origin: 'http://localhost:4200',
                methods: ['GET', 'POST'],
                credentials: true,
        },
})
export class ChatGateway implements OnGatewayConnection,
    OnGatewayDisconnect {
        constructor(
        private readonly chatService: ChatService,
        private readonly notificationService:
            NotificationService,
        ) {}
        @WebSocketServer() server;

        private users: { [id: string]: string } = {};
        handleConnection(client: Socket) {
                const userId = this.
                    extractUserIdFromToken(
                client.handshake.query.Authorization as
                    string,
                );
                this.users[userId] = client.id;
                client.emit('connected', { userId, users:
                     this.users });
        }

        handleDisconnect(client: Socket) {
                const userId = this.
                    extractUserIdFromToken(
                client.handshake.query.Authorization as
                    string,
                );
                delete this.users[userId];
        }

        @SubscribeMessage('privateMessage')
        async handlePrivateMessage(
        client: Socket,
        payload: { senderMail: string; receiverMail:
            string; message: string },
        ): Promise<void> {
```

```
                const { senderMail, receiverMail, message
                    } = payload;
                const result = await this.chatService.
                    createMessage({
                        senderMail: senderMail,
                        receiverMail: receiverMail,
                        message: message,
                        timeStamp: new Date(),
                });
                const targetSocketId = this.users[
                    receiverMail];
                if (targetSocketId) {
                        client.to(targetSocketId).emit('
                            receivePrivateMessage',
                            result);
                }
        }
}
```

0.5.3 Front-end (web-server)

Il front-end è stato gestito attraverso Angular, che similmente al back-end è suddiviso in moduli, per favorire questa volta il lazy loading, che consente il risparmio di risorse. In ogni modulo è dunque presente il componente, il servzio (per inviare le richieste http) e la parte di view. Per favorire l'utilizzo di JWT è stato introdotto un interceptor, il quale arricchisce l'header delle request con l'authorization token. QUuindi nella parte sottostante, mostrerò più la parte di logica, piuttosto che quella grafica, in quanto abbastanza comune.

Servizio

```
constructor(
private authenticationService: AuthenticationService,
private http: HttpClient
) {
        this.baseUrl = BASE_URL + 'lessons/';
}

async createLesson(lessonRequest: ILessonRequest) {
        const teacher: ICreateUserRequest =
        await this.authenticationService.
            getAllUserInformations();
        lessonRequest.teacherMail = teacher.mail;
        lessonRequest.studentMail = '';
        this.http.post(`${this.baseUrl}create,
            lessonRequest);
}
```

Interceptor

```
import { Injectable } from '@angular/core';
import {
        HttpRequest,
        HttpHandler,
        HttpEvent,
        HttpInterceptor,
} from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable()
export class AuthInterceptor implements HttpInterceptor {
        constructor() {}

        intercept(
        request: HttpRequest<any>,
        next: HttpHandler
        ): Observable<HttpEvent<any>> {
                const token = sessionStorage.getItem('
                    access_token');

                if (token) {
                        const cloned = request.clone({
                                setHeaders: {
                                        Authorization: `
                                            Bearer ${
                                            token}`,
                                },
                        });

                        return next.handle(cloned);
                } else {
                        return next.handle(request);
                }
        }
}
```

0.5.4 Front-end (web-socket)

Web socket

```
//imports...
@Injectable({
        providedIn: 'root',
})
export class WebsocketService {
        private socket: any;

        constructor() {
                const token = sessionStorage.getItem('
                    access_token');
                this.socket = io('ws://localhost:3000/
                    chat', {
                        query: {
                                Authorization: 'Bearer '
                                    + token,
                        },
                });
        }

        sendMessage(event: string, message: any): void {
                this.socket.emit(event, message);
        }

        onMessage(event: string): Observable<any> {
                const observable = new Subject<any>();
                this.socket.on(event, (data: any) => {
                        observable.next(data);
                });
                return observable;
        }

        onNotificationReceived(): Observable<any> {
                const observable = new Subject<any>();
                this.socket.on('receiveNotification', (
                    notification: any) => {
                        observable.next(notification);
                });
                return observable;
        }

        closeConnection(): void {
                this.socket.disconnect();
        }
}
```

0.6 Test

Il funzionamento dell'applicazione è stato testato su svariati browser, tra cui:

- google chrome;
- opera;
- microsoft edge;
- firefox.

Attraverso l'utilizzo di postman, che consente di effettuare richieste http, è stata testata l'API. Si è inoltre tenuto conto delle Euristiche di Nielsen, per garantire la corretta usabilità del sistema.

0.7 Deployment

Il deployment dell'applicativo è avvenuto attraverso un rilascio su github, che prevede la clonazione in locale.

Passaggi

- Clonazione repository: git clone ...;
- Installazione: npm install
- Esecuzione comandi
  - cd server/; npm start
  - cd ../client; ng serve

L'applicativo è disponibile all'indirizzo: http://localhost:4200/

0.8 Conclusioni

La gestione del progetto è stata secondo il mio punto di vista, adeguata, sia nel rispetto delle tempistiche che nella qualità dell'applicativo. L'utilizzo di framework javascript per la gestione dell'architettura, si è rivelato molto utile, riducendo i tempi di sviluppo. I possibili miglioramenti a mio avviso sono i seguenti:

- integrazione microsoft teams;
- aggiunta di funzionalità alla chat (orario di lettura messaggio, spunta di conferma, ecc...);
- sviluppo di un sistema di gamification per coinvolgere maggiormente gli utenti.

Bibliography

[1] Angular.

[2] Devextreme - html5 javascript ui library for responsive web development.

[3] Mongoose.

[4] Nestjs - a progressive node.js framework.

[5] Socket.io.