



UNIVERSITÀ DEGLI STUDI DI GENOVA

SCUOLA DI SCIENZE MATEMATICHE, FISICHE E NATURALI

LAUREA IN INFORMATICA

ANNO ACCADEMICO 2022/2023

Sviluppo full-stack per un'applicazione web per la turnazione del personale

Febbraio 2024

Candidato
Pezzano Enrico

Relatrice
Prof.ssa Ribaudo Marina

Abstract

TODO at the end

Indice

Acronimi	ii
1 Introduzione	1
1.1 Scopo del progetto	1
2 Contesto di riferimento	3
2.1 Modello three-tier	3
2.2 Framework utilizzati	10
2.2.1 Angular e PrimeNG	11
2.2.2 Typescript	19
2.2.3 .NET Framework, .NET Core e .NET 8	22
2.2.4 ASP.NET	25
3 Prototipo	28
3.1 Database	29
3.2 Backend	31
3.2.1 Swagger	39
3.3 Frontend	42
3.3.1 PrimeNG	57
3.3.2 Servizi Angular	58
4 Conclusioni e ringraziamenti	60
4.1 Conclusioni	60
4.2 Ringraziamenti	60
Elenco delle figure	62

Capitolo 1

Introduzione

1.1 Scopo del progetto

Tutte le aziende, in particolare quelle di una certa complessità, hanno l'impellente necessità di gestire e razionalizzare le risorse e i servizi che erogano ai propri dipendenti, con particolare attenzione a quei servizi che permettono di utilizzare determinati strumenti aziendali e di implementare le comunicazioni ufficiali tra dipendente e azienda. Parte del processo di digitalizzazione delle aziende prevede, tra le altre iniziative, la creazione e l'utilizzo di sistemi automatici, intelligenti e fruibili sia da computer che da periferica mobile per l'accesso ai servizi aziendali.

L'azienda presso la quale ho svolto il mio tirocinio è Gruppo SIGLA (GS), un'azienda di più di 100 persone, di cui molte impiegate in via temporanea presso clienti in Italia e all'estero, ha interesse nel digitalizzare questi processi rapidamente e nel realizzare sistemi che permettano di aiutare altre aziende a fare altrettanto.

In particolare, Gruppo SIGLA vuole implementare una soluzione che permetta di gestire i propri strumenti e servizi aziendali in modo tale da garantirne un accesso rapido e efficiente per i propri dipendenti. Il servizio sul quale ho lavorato durante il tirocinio è stato quello di tracciamento e feedback delle attività formative erogate dall'azienda. Questo sistema si presenta come una applicazione web, composta da elementi client e server, e è sviluppata tramite l'utilizzo di tecnologie avanzate come .NET per il back-end e Angular per il front-end. Si prevede, inoltre, la possibilità di integrare servizi di autenticazione e invio di email ai dipendenti.

TODO at the end

Questo documento è organizzato nei seguenti capitoli.

Il capitolo 2 descrive, il contesto di riferimento, in particolare si parla del progetto scelto (del perchè e dello stato dell'arte), delle tecnologie utilizzate (per il back-end e per il front-end), delle metodologie di sviluppo e dei requisiti per portare a termine il progetto.

Il capitolo 3 è dedicato alla descrizione del prototipo realizzato, si parla dell'architettura del sistema nel dettaglio, delle tecnologie utilizzate e delle scelte progettuali fatte. Inoltre, si parla della fase di sviluppo e di test del prototipo.

Infine, il capitolo 4, presenta conclusioni e sviluppi futuri.

Capitolo 2

Contesto di riferimento

La stesura del codice di questo progetto di prova finale è stata effettuata grazie a strumenti digitali basati sul web e ai relativi linguaggi di programmazione.

L'adozione di una solida architettura informatica ha giocato un ruolo cruciale. Prima di descrivere nel dettaglio il contesto client-server utilizzato, è necessario introdurre alcuni concetti fondamentali per la comprensione del progetto. Questo capitolo esplorerà le piattaforme, i framework¹ e gli strumenti già esistenti, delineando le ragioni dietro la loro scelta e illustrando come si integrano per supportare l'intera struttura dell'applicazione.

2.1 Modello three-tier

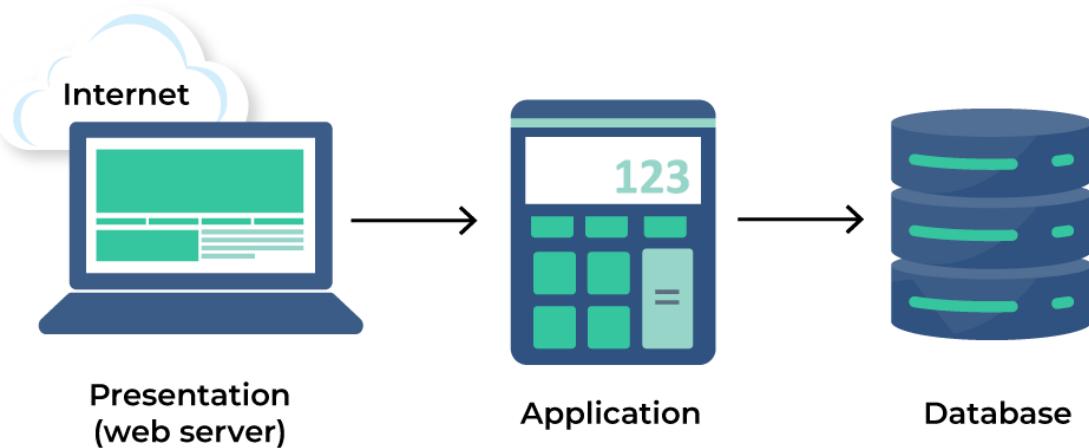


Figura 2.1: Schema di un'applicazione a tre livelli.

¹In informatica, un framework, in italiano struttura o quadro, è un'architettura logica di supporto su cui un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore.

Il modello three-tier è un'architettura software² consolidata, di tipo client-server, in cui l'interfaccia utente, l'elaborazione logica e la gestione dei dati sono sviluppati e mantenuti come moduli indipendenti su server distinti. L'organizzazione in livelli separati aiuta a migliorare lo scopo dell'architettura stessa, aumentando quindi la manutenibilità, la scalabilità, l'affidabilità, la flessibilità e ovviamente la modularità del sistema.

Il maggior beneficio del modello three-tier è che i livelli sono sviluppati e mantenuti indipendentemente e perciò, in caso di qualsiasi modifica, solo un livello viene coinvolto. Inoltre, un ulteriore vantaggio è dato dal fatto che i livelli possono essere scalati separatamente, all'aumentare del carico di lavoro di uno qualsiasi.

Come si può intuire dal nome, il modello three-tier è composto da tre livelli:

1. livello di presentazione
2. livello di applicazione
3. livello di database

Nello sviluppo web, i livelli hanno sempre i nomi precedenti, diversi gli uni dagli altri, ma si occupano di funzionalità analoghe.

Nello specifico, il primo livello è quello che viene presentato all'utente finale, ossia l'interfaccia grafica grazie alla quale l'utente ha la possibilità di interagire con il sistema. Lo scopo principale dello strato superiore del modello three-tier è di mostrare le informazioni provenienti dagli altri livelli in modo chiaro e comprensibile all'utente e di raccogliere eventuali dati inseriti dall'utente stesso. Il livello di presentazione può essere eseguito su un qualsiasi browser web, che farà anche da web server, su un'applicazione desktop o su un'interfaccia grafica GUI, a seconda delle esigenze del progetto. Le applicazioni del livello di presentazione sono solitamente sviluppate utilizzando HTML, CSS e JavaScript, nel caso dei siti web, e possono essere scritte in un'ampia varietà di linguaggi a seconda della piattaforma, nel caso delle applicazioni desktop. Più di recente i framework per lo sviluppo front-end, come Angular, React e Vue.js, sono diventati sempre più popolari.

Nel caso di questo progetto di tirocinio, è stato scelto il framework Angular, come verrà spiegato più approfonditamente nel seguito (2.2.1).

Il secondo livello, quello di applicazione (o di logica) raccoglie informazioni e richieste dal livello superiore e le elabora utilizzando le regole specifiche della logica

²L'insieme delle procedure e delle istruzioni in un sistema di elaborazione dati; si identifica con un insieme di programmi (in contrapposizione a hardware).

di business. Inoltre, il livello di applicazione può aggiungere, eliminare o modificare i dati memorizzati nel livello di database; per esempio, nel caso di un e-commerce, il livello di applicazione fa query³ al database dell'inventario per sapere la disponibilità di un prodotto, oppure aggiunge dettagli al profilo di un cliente, etc. In breve, è il livello responsabile di processare le richieste del client e, una volta ricevute le informazioni necessarie, di inviarle di nuovo al client. Nei sistemi più moderni o complessi, questo livello è solitamente sviluppato utilizzando linguaggi come Java, Python, PHP, Ruby, Perl, ASP.NET, JavaScript o TypeScript, etc. In generale, qualsiasi linguaggio di programmazione che supporti le chiamate API⁴ (in modo da poter comunicare con il livello del database, o con altri moduli) può essere utilizzato per sviluppare il livello di applicazione.

Infine, nel terzo livello, quello di database, detto anche livello di accesso ai dati o back-end, le informazioni elaborate dall'applicazione vengono archiviate e gestite, per poter essere recuperate in seguito, quando necessario. Il back-end si sostanzia in un Database Management System (DBMS) e tra i più diffusi ricordiamo PostgreSQL, MySQL, MariaDB, Oracle, DB2, Informix, Microsoft SQL Server.

— fermarsi qui e mettere solo una figura —

I tre livelli sono collegati tra loro attraverso interfacce ben definite, che consentono a ciascun livello di funzionare in autonomia e una facile variazione e manutenzione dell'applicazione. L'architettura a tre livelli è un design pattern ampiamente utilizzato per la costruzione di applicazioni software scalabili e mantenibili e è comunemente utilizzato nelle applicazioni web e nei sistemi software aziendali.

Si noti che il termine ‘3 Tier’ non può essere riferito ad un'applicazione semplicemente dividendo il suo codice in tre parti. È fondamentale seguire determinate regole per ciascun livello in modo da garantire la corretta implementazione dell'architettura. Le suddette regole non sono complesse e di facile comprensione.

- Innanzitutto, il codice di ogni livello deve essere memorizzato in file separati, i quali possono essere gestiti indipendentemente, meglio se da team diversi. Inoltre, ogni livello deve contenere solo il codice che appartiene a se stesso. Ciò significa che la logica di business deve risiedere soltanto nel livello di applicazione (logica di business), la logica di presentazione nel livello di front-end e

³ Una richiesta di informazioni o di azioni da parte di un database; solitamente scritta in un linguaggio di interrogazione, come ad esempio SQL.

⁴ Un'interfaccia di programmazione delle applicazioni, in acronimo API (dall'inglese application programming interface), è un insieme di definizioni e protocolli con i quali vengono realizzati e integrati software applicativi.

la logica di accesso ai dati nel livello di back-end, o più precisamente nel livello di Data Access Layer (DAL);

- Il livello di presentazione può esclusivamente ricevere richieste da un'entità esterna e rispondere alla stessa, generalmente un utente, ma può anche essere un'altra applicazione software. In aggiunta, non può avere accesso diretto al database o al livello di Data Access Layer, ma può unicamente inviare richieste e ricevere risposte dal livello contenente la logica di business (livello di applicazione);
- Similmente, il livello di applicazione può solo ricevere richieste e rispondere al livello di presentazione, infatti, può solo inviare richieste e ricevere risposte dal Data Access Layer e gli è vietato di accedere direttamente al database;
- Il Data Access Layer (DAL) può ricevere richieste e rispondere solamente al livello della logica di business (applicazione) e non può inviare richieste a nessun altro componente, se non al Database Management System che supporta;
- È oltretutto importante assicurarsi che ogni livello sia totalmente ignaro del funzionamento interno degli altri. Ad esempio, il livello di applicazione deve essere agnostico rispetto al database e non interessato al funzionamento interno dell'oggetto di accesso ai dati. Allo stesso modo, non deve elaborare i dati in modo diverso in base al componente che riceve una richiesta. Il livello di presentazione può prendere i dati e elaborarli in vari modi, come ad esempio costruire un documento HTML, un documento PDF, un file CSV, ma sono dettagli non dovrebbero influenzare il livello di applicazione.

Si noti che, in un'applicazione a tre livelli, tutte le comunicazioni passano attraverso il livello applicazione. Il livello di presentazione e il livello di database non possono comunicare direttamente tra loro.

Di nuovo, il principale beneficio dell'architettura a tre livelli è la separazione logica delle funzionalità. Ogni livello può essere eseguito su un sistema operativo e una piattaforma server diversi, ad esempio un livello di presentazione può essere eseguito su un server web, mentre il livello di applicazione può essere eseguito su un server di applicazioni e il livello di database può essere eseguito su un server database; per far sì che ognuno combaci coi propri requisiti funzionali e che ogni livello sia eseguito su un server dedicato (fisico o virtuale), cosicché i servizi di ciascun livello possano essere personalizzati e ottimizzati senza influenzare gli altri livelli.

Altri vantaggi includono:

- Sviluppo più rapido: poiché ogni livello può essere sviluppato simultaneamente da team diversi, un'organizzazione può portare l'applicazione sul mercato più rapidamente e i programmati possano utilizzare i linguaggi e gli strumenti più recenti e migliori per ogni singolo livello;
- Miglior scalabilità: ogni livello può essere scalato indipendentemente dagli altri, se necessario;
- Miglior affidabilità: un'interruzione (di corrente o per un problema software) in un livello è meno probabile che influenzi la disponibilità o le prestazioni degli altri livelli;
- Miglior sicurezza: poiché il livello di presentazione e il livello di database non possono comunicare direttamente tra loro, un livello di applicazione ben progettato può funzionare come una sorta di firewall interno, prevenendo attacchi come le SQL injection o altri exploit dannosi.

Si noti che esistono altre tipologie di architetture software multi-tier, come ad esempio quella a due livelli, la N-tier, il MVC, etc, ma non sono così diffuse come quella a tre livelli e non saranno approfondite in questo progetto di laurea.

Un paragone degno di nota è quello con il Model-View-Controller (MVC). Le principali **differenze** tra le architetture di tipo three-tier e di tipo MVC sono:

oppure

L'architettura a tre livelli sembra molto simile a un'altra architettura, altrettanto conosciuta, denominata Model-View-Controller (MVC), ma non sono la stessa cosa.

Entrambe le architetture mirano a separare i compiti dell'applicazione e a migliorare lo sviluppo del software suddividendola in diversi componenti. Mentre l'architettura a tre livelli -come visto in precedenza (2.1)- si concentra sul dividere un'applicazione in tre livelli distinti (ogni livello ha il suo ruolo specifico) e sulla comunicazione ben definita tra di loro (come descritto nelle regole), l'architettura MVC, tuttavia, si concentra sulla separazione dell'applicazione in tre componenti distinti: Model, View e Controller (modello, vista e controllore).

Il modello rappresenta i dati e la logica di business dell'applicazione, la vista rappresenta l'interfaccia utente e il controllore agisce come mediatore tra i primi due. Infatti, è il controllore a ricevere l'input⁵ dall'utente, a interagire con il modello per elaborare i dati e ad aggiornare la vista per visualizzarne i risultati.

⁵Un input è un'informazione che viene fornita a un sistema informatico, in genere da un utente.

In breve, entrambe cercano di separare le responsabilità e migliorare lo sviluppo del software, ma lo portano a termine diversamente:

- il modello three-tier si concentra di più sulla separazione dei diversi livelli;
- l'architettura MVC si concentra di più sulla separazione dei componenti all'interno di un livello.

MVC Architecture

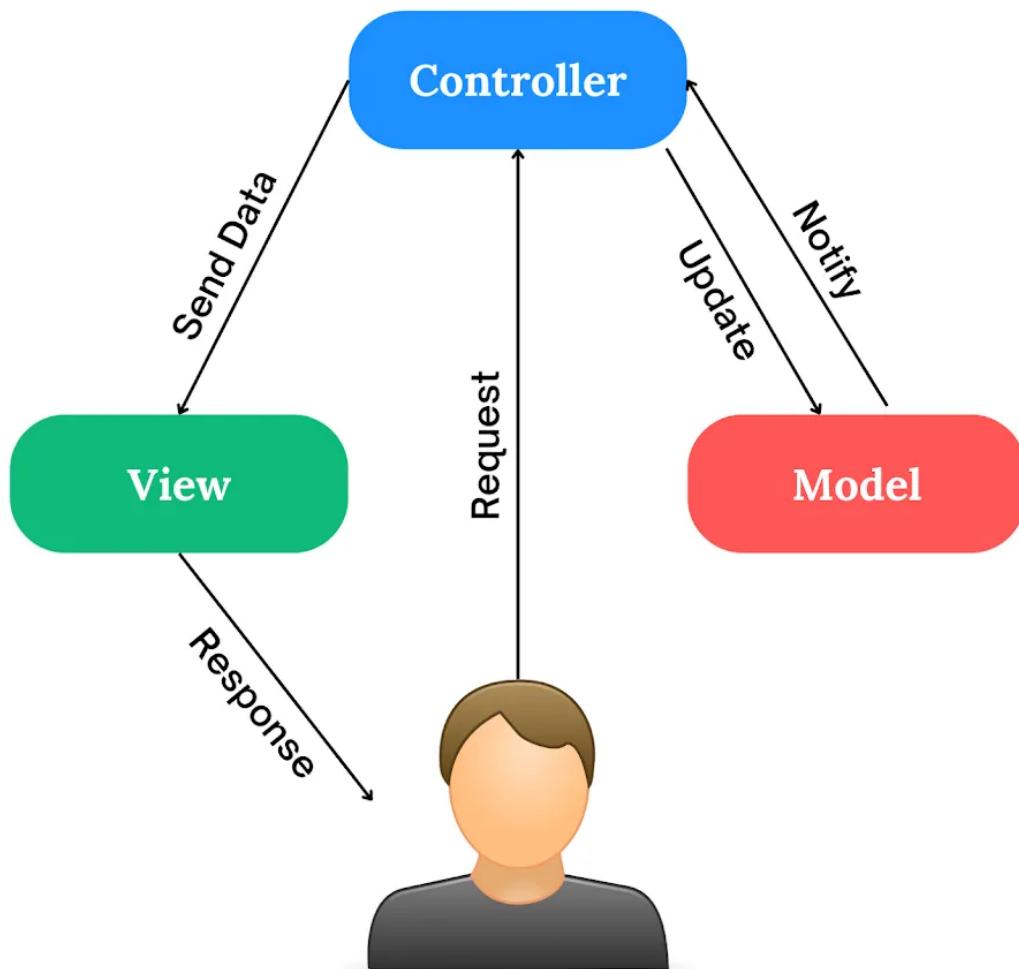


Figura 2.2: Schema teorico dell'architettura MVC.

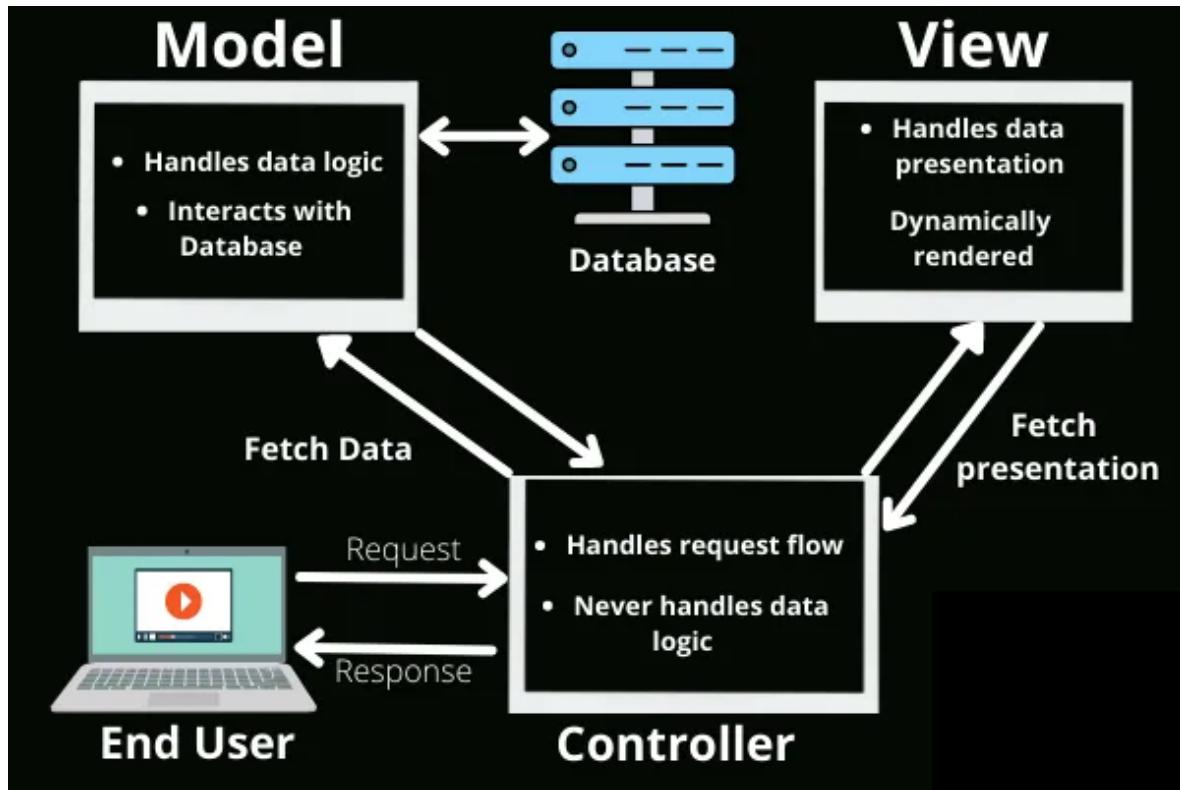


Figura 2.3: Schema pratico dell'architettura MVC.

2.2 Framework utilizzati

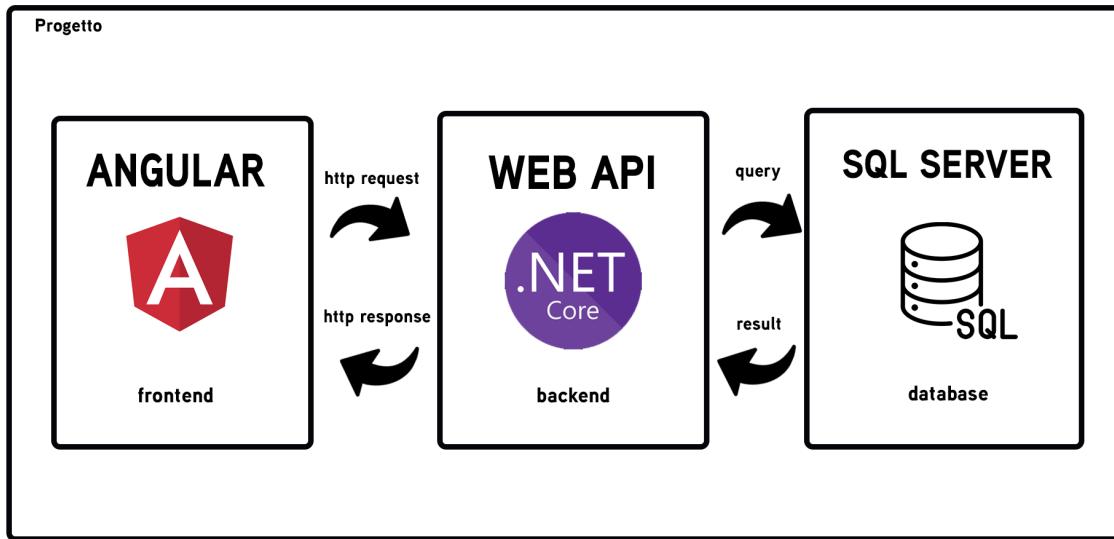


Figura 2.4: Schema dei framework utilizzati.

Dopo aver esaminato l'architettura del modello three-tier, in questa sezione verranno descritti i framework utilizzati per lo sviluppo del progetto di laurea.

Un framework è un'astrazione che unisce codice comune tra progetti di software con scopi simili, fornendo una funzionalità generale, che può essere modificata o estesa per risolvere un problema specifico. Più in generale, incrementano l'efficienza e le prestazioni dello sviluppo web, fornendo una struttura coerente, affinché gli sviluppatori non debbano continuamente ricostruire il codice da zero; i framework sono ottimi quando si tratta di risparmiare tempo.

Aggiuntivamente, offrono ai programmatore una serie di funzionalità extra che possono essere aggiunte al software senza richiedere ulteriori sforzi.

Nel caso di questo progetto di laurea, lo scopo principale è stato quello di implementare nuovi componenti e funzionalità per un'applicazione web già esistente, quindi è stato necessario utilizzare i framework già adottati nel progetto stesso, in modo da poter integrare le nuove funzionalità in uno dei modi più semplici e veloci, senza dover riscrivere totalmente il codice da zero. Uno dei difetti di questo approccio è stato aver ricevuto del software scritto da più sviluppatori, con stili di programmazione diversi e componenti sviluppati da una o più persone. Infatti la re-

pository del progetto (dopo aver implementato le funzionalità richieste dal progetto di laurea) è composta da più di 55 mila file e nel complesso era molto facile perdersi nella moltitudine di file. Ciononostante, è stato possibile integrare le nuove funzionalità in modo semplice e veloce, grazie ai framework di sviluppo utilizzati, sia per il back-end che per il front-end.

2.2.1 Angular e PrimeNG



Figura 2.5: Angular e PrimeNG, due framework.

Questa sotto-sezione si propone di descrivere e analizzare in breve uno dei framework per il front-end più utilizzati al mondo, Angular, insieme al relativo framework -open source⁶- PrimeNG, che fornisce uno dei set di componenti UI più completi per Angular.

Angular, denominato anche Angular 2+ (ossia una versione più moderna di AngularJS), è un framework JavaScript scritto e basato su TypeScript, open source per lo sviluppo di UI di applicazioni web, focalizzandosi sulle singole pagine; è stato sviluppato principalmente dal team di Angular di Google e da una comunità di individui e aziende esterne. È, inoltre, una riscrittura completa del framework originale (AngularJS); entrambi i framework sono stati sviluppati dallo stesso team. Attualmente, Angular è mantenuto da Google.

Le differenze principali tra Angular e AngularJS sono che il secondo usa JavaScript, ha un supporto mobile limitato e una dependency injection meno robusta. Il primo invece (Angular) invece adotta TypeScript, offre un supporto mobile migliore, una dependency injection più potente e un efficiente rilevamento dei cambiamenti. Inoltre, ha template⁷ dinamici, una CLI potente e una struttura modulare per file più piccoli.

⁶Un software open source è un software di cui gli autori (più precisamente i detentori dei diritti) rendono pubblico il codice sorgente, favorendone il libero studio e permettendone il miglioramento da parte di altri programmati indipendenti.

⁷In informatica, un template è un modello predefinito che può essere utilizzato per creare documenti o pagine web, uno scheletro/schema che può essere riempito con contenuti specifici.

In quanto framework, fornendo una struttura standard per gli sviluppatori, Angular ha dei chiari vantaggi. Consente agli utenti di creare applicazioni di grandi dimensioni, manutenibilmente.

La principale funzionalità di Angular è quella di fornire blocchi per costruire le applicazioni web, denominati componenti, grazie ai quali aiuta gli sviluppatori ad impostare rapidamente una web-app scalabile e facilmente manutenibile. Angular conferisce ai programmati la libertà di sviluppare applicazioni che possano essere eseguite agilmente su più piattaforme; inclusi dispositivi mobili, desktop e web. Oltretutto, è stato progettato per essere modulare, cosicché si possano integrare o rimuovere facilmente le funzionalità di cui si ha bisogno o meno.

Nonostante l'ampio utilizzo e la facilità di apprendimento di JavaScript, nei casi analoghi a questo progetto di laurea, Angular si rivela comunque essere una scelta migliore, presentando una serie di benefici -sopra citati i principali- che lo rendono ideale per affrontare la maggior parte, se non tutti, i problemi che gli sviluppatori incontrano con JavaScript.

Alcune delle caratteristiche di Angular sono:

1. Document Object Model (DOM): gestisce un documento XML o HTML come una struttura ad albero in memoria; Angular usa il DOM classico. Ipotizzando che vengano effettuati dieci aggiornamenti sulla stessa pagina HTML, invece di aggiornare i cambiamenti già elaborati, Angular aggiorna l'intera struttura ad albero dei tag HTML;
2. TypeScript (TS): il linguaggio -sviluppato da Microsoft- in cui è scritto Angular. Definisce un insieme di tipi che rende il codice più facile da leggere e da scrivere. Inoltre, tutto il codice TypeScript viene compilato in JavaScript, cossicché ogni piattaforma possa eseguirlo. È altamente consigliato, ma non vincolante per sviluppare con Angular;
3. Data Binding: è un meccanismo che abilita gli utenti a manipolare gli elementi di una pagina web attraverso un browser web. Usato principalmente nelle pagine web che incorporano componenti interattivi, come calcolatori, tutorial, forum e giochi. Inoltre, consente una migliore visualizzazione progressiva di una pagina web quando le pagine contengono una grande quantità di dati. Nel caso di Angular viene utilizzato data binding bidirezionale, infatti lo stato del modello riflette eventuali modifiche apportate agli elementi UI corrispondenti. Al contrario, lo stato UI riflette eventuali modifiche apportate allo stato del modello, questa funzionalità consente al framework di collegare il DOM ai dati del modello attraverso il controller;

4. Tesing: Angular usa un test framework denominato Jasmine, ma non sarà oggetto di discussione in questo progetto di laurea;

Passando all'architettura, è importante notare che Angular è un framework Model-View-Controller, a tutti gli effetti, dando una guida chiara su la web app dovrebbe essere strutturata; permette inoltre un flusso bidirezionale di data-binding mettendo contemporaneamente a disposizione un vero e proprio DOM.

I seguenti sei sono i blocchi fondamentali di un'applicazione Angular:

1. Moduli: una web app Angular possiede un modulo root (denominato AppModule) il quale fornisce il meccanismo di bootstrap per avviare l'applicazione. Ogni modulo può avere uno o più moduli figlio;
2. Componenti: ogni componente dell'applicazione definisce una classe che contiene i dati e la logica dell'applicazione. Generalmente, un componente definisce una parte dell'interfaccia grafica (o UI, in inglese);
3. Modelli (o Template⁸): i template Angular combaciano con HTML per modificare i suoi stessi elementi (HTML) prima che sia visualizzati. Esistono due tipi di data binding: di evento (bisogna aggiornare la web app) e di proprietà (abilita l'utente ad interpolare i valori calcolati dall'applicazione nel codice HTML);
4. Metadati: comunicano ad Angular come processare una classe. Sono usati per decorare la classe stessa, cosicché si possa configurare il comportamento aspettato della classe;
5. Servizi: una classe di servizio è creata quando si hanno dati o logica che non sono associati con la visualizzazione, ma devono comunque essere condivisi tra i componenti. La classe di servizio è sempre associata con il decoratore '@Injectable';
6. Dependency Injection: un pattern di progettazione che permette di mantenere le classi dei componenti precise e efficienti. È una funzionalità che non raccoglie dati da un server, valida un input dell'utente o stampa direttamente a console. Invece, delega i suddetti compiti ai servizi;

⁸ In informatica, un template è un modello predefinito che può essere utilizzato per creare documenti o pagine web, uno scheletro/schema che può essere riempito con contenuti specifici.

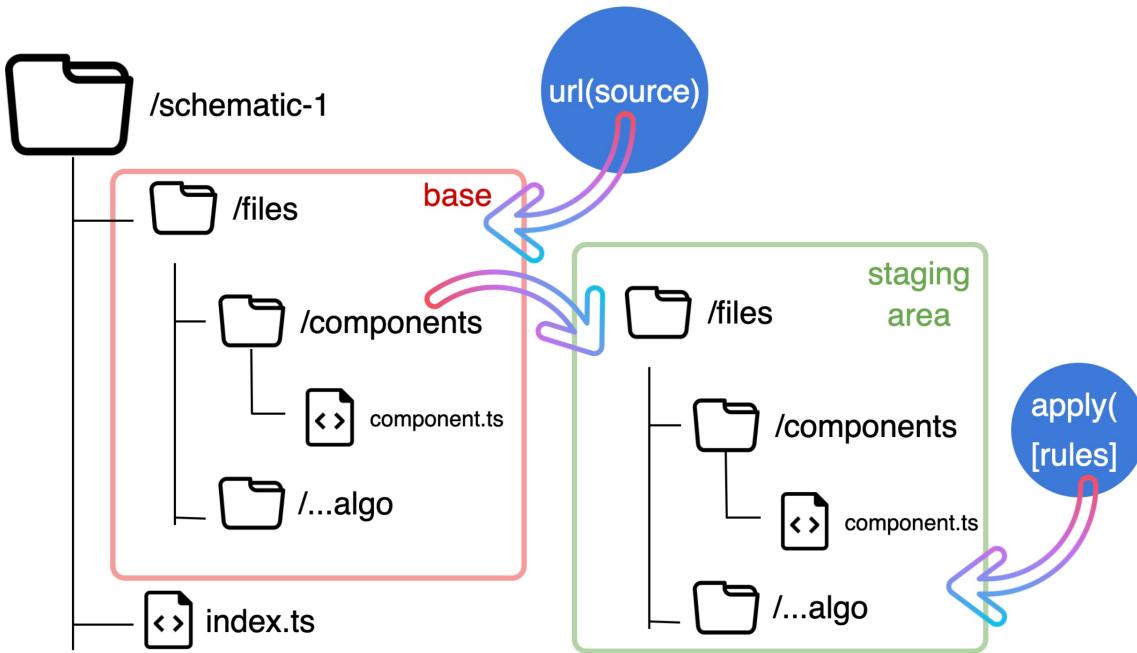


Figura 2.6: Schematica di Angular.

Un'ulteriore funzionalità degna di nota di Angular è la sua Command Line Interface (CLI), la quale permette di creare nuovi progetti, applicazioni, componenti, etc. Inoltre, Angular CLI offre anche la possibilità di eseguire un web server, tramite l'apposito comando da terminale, in modo tale da poter testare l'applicazione in locale, prima di effettuare il deploy su un server remoto.

Molte versioni di Angular sono state rilasciate dalla sua nascita. Ognuna di esse ha contribuito al miglioramento dell'efficienza del framework. I principali **vantaggi** di Angular sono:

1. Componenti su misura: funzionalità che permette agli utenti di creare i propri componenti, i quali possono contenere funzionalità e logiche di visualizzazione in parti riutilizzabili. Inoltre si integrano facilmente con i componenti HTML standard;
2. Data Binding: funzionalità che permette agli utenti di trasformare i dati da codice JS a UI agevolmente, senza dover scrivere codice aggiuntivo;
3. Dependency Injection⁹: funzionalità che permette agli utenti di creare servizi modulari e ‘iniettarli’ dove necessario. Migliora la modularità e la manutenibilità degli stessi servizi;

⁹Un pattern architettonico per la gestione delle dipendenze tra gli oggetti, in cui un oggetto riceve le dipendenze da un'entità esterna anziché crearle direttamente.

4. Testing: i test sono utensili di prima classe, Angular è stato sviluppato da zero tenendolo a mente, infatti permette di testare ogni singola parte dell'applicazione;
5. Esaustivo: fornisce soluzioni per la comunicazione coi server, l'intradamento nell'applicazione e altro tra le funzionalità standard;
6. Compatibilità: Angular è utilizzabile su più piattaforme e compatibile con vari browser. Un'applicazione Angular può essere eseguita su qualsiasi browser (Chrome, Firefox, Safari, etc) o sistema operativo (Windows, Linux, MacOS, etc).

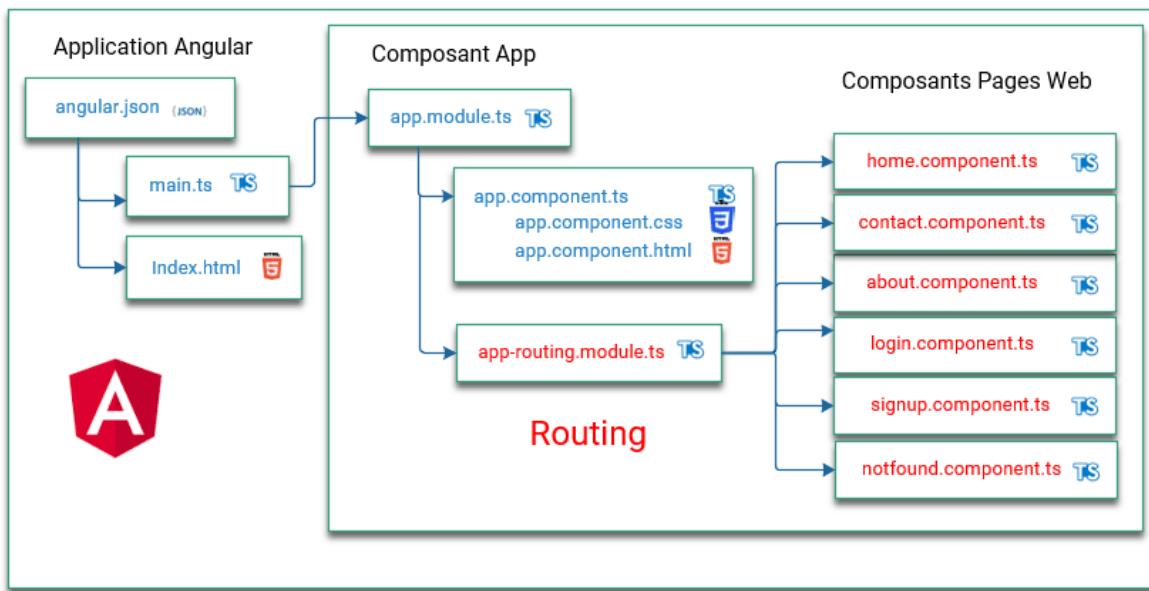


Figura 2.7: Schema dell'intradamento di Angular.

I principali **svantaggi** e limitazioni di Angular sono:

1. Curva di apprendimento ripida: i componenti elementari di Angular che tutti i sviluppatori dovrebbero conoscere comprendono direttive, moduli, decoratori, componenti, servizi, dependency injection, pipe e template. Alcuni argomenti più avanzati includono rilevamento di cambiamenti, zone, compilazione AoT¹⁰ e RxJS. Per i neofiti, Angular potrebbe essere impegnativo da apprendere per via della sua completezza;

¹⁰La compilazione di un linguaggio di programmazione ad alto livello in uno di livello inferiore prima dell'esecuzione di un programma, di solito al momento della composizione dello stesso, per ridurre la quantità di lavoro da eseguire al momento dell'esecuzione.

2. SEO limitate: SEO limitate e scarsa accessibilità all'indicizzazione¹¹ dei motori di ricerca;
3. Migrazione: una delle ragioni per cui le aziende non adottano frequentemente Angular per le migrazioni è la difficoltà intrinseca nella portabilità di codice JS (o basato su jQuery¹²) preesistente in un'applicazione Angular. Inoltre, ogni nuova versione rischia di essere problematica da aggiornare e molte di loro non sono retro-compatibili.
4. Verboso: un problema comune negli utilizzatori Angular è la verbosità del framework.

Tra le aziende più famose che utilizzano Angular ci sono Google, Microsoft, Nike, Forbes, Upwork, HBO, Sony, General Motors, etc.

¹¹Un processo che consiste nel creare una struttura dati, generalmente una tabella hash o un albero, che permetta di accedere rapidamente ai dati in un database.

¹²jQuery è una libreria JavaScript per applicazioni web. Nasce con l'obiettivo di semplificare la selezione, la manipolazione, la gestione degli eventi e l'animazione di elementi DOM in pagine HTML, nonché implementare funzionalità AJAX.

Infine, **PrimeFaces Next Generation (PrimeNG)** è una collezione ricca di componenti UI per Angular, i cui widget sono open source e gratuiti. Sviluppato da PrimeTek Informatics nel 2016, un'azienda con anni di esperienza nello sviluppo di soluzioni UI open source.

Tra le varie funzionalità che rendono PrimeNG un framework popolare tra gli sviluppatori, ci sono:

- Componenti UI completi: PrimeNG offre oltre 80 componenti UI, come ad esempio tabelle, dialoghi, grafici, etc;
- Template-Driven¹³: i componenti di PrimeNG sono template-driven, rendendoli facili da usare e personalizzare;
- Supporto di temi: PrimeNG supporta vari temi, permettendo agli sviluppatori di scegliere tra una vasta gamma.

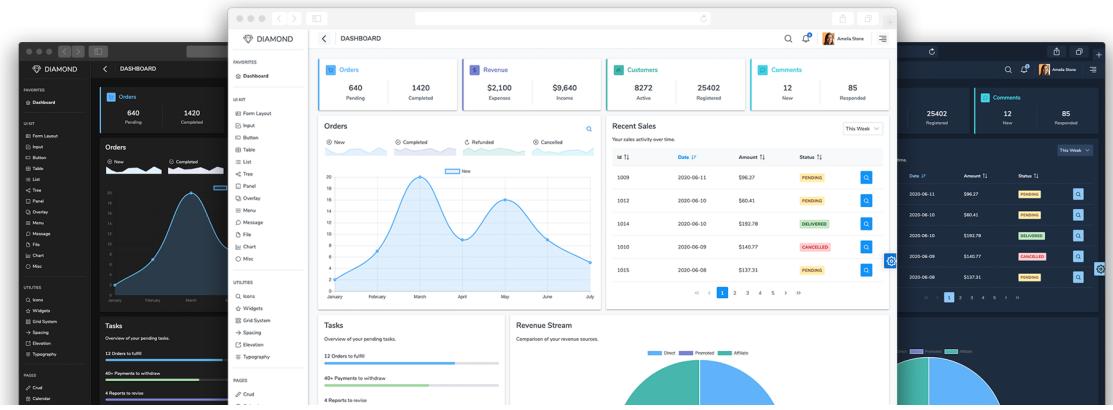


Figura 2.8: Diamond, uno dei template di PrimeNG.

Parecchie aziende usano PrimeNG sia per applicazioni interne che per applicazioni per i clienti. Grazie alla sua numerosa serie di componenti e facilità d'uso, PrimeNG è diventato una delle scelte più popolari e valide tra gli sviluppatori, anche per le applicazioni più complesse.

Tra di loro troviamo Ericsson, Siemens, Lufthansa, UniCredit, Ford, Volvo, Nvidia, eBay, Mercedes-Benz, Audi, HP, Scania, Cisco, Nike, Volkswagen.

¹³Un approccio per la creazione di applicazioni web che si basa su template HTML, che vengono compilati lato client.

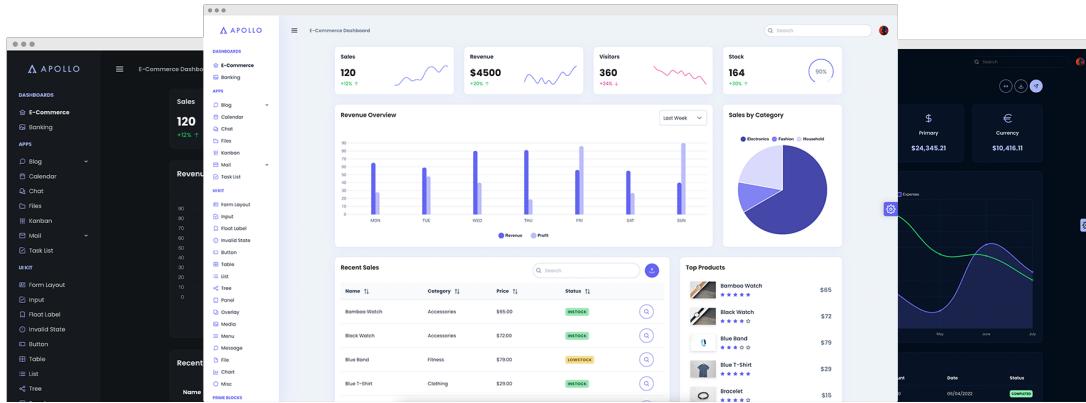


Figura 2.9: Apollo, uno dei template di PrimeNG.

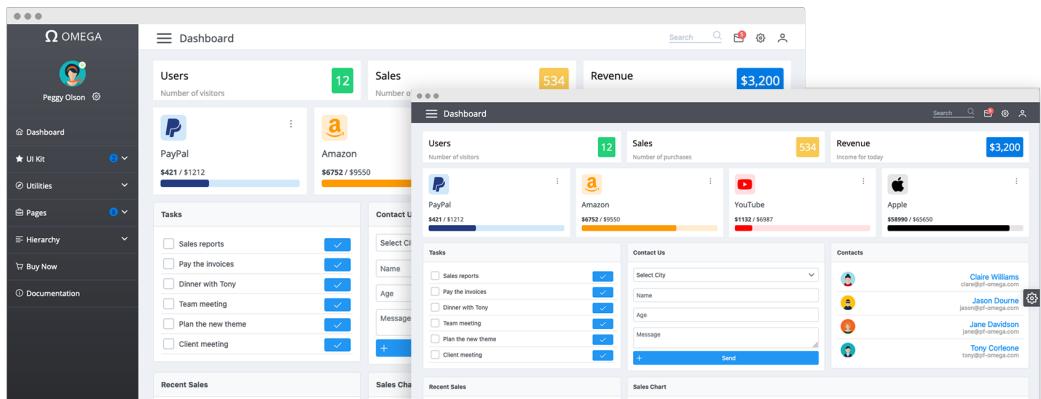


Figura 2.10: Omega, uno dei template di PrimeNG.

2.2.2 Typescript

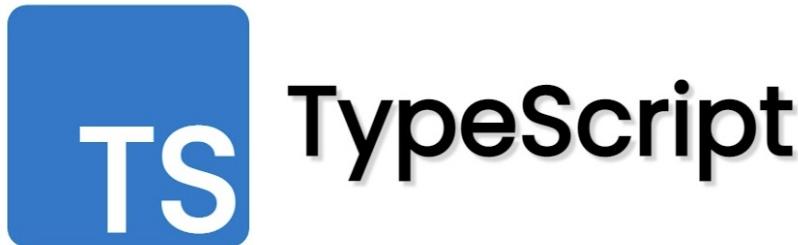


Figura 2.11: Il logo di TypeScript.

TypeScript (TS) è un linguaggio di programmazione open source sviluppato e mantenuto da Microsoft. Fa parte di un approccio moderno alla programmazione e è un superset sintattico e restrittivo di JavaScript, infatti offre tutte le funzionalità di JavaScript e ne aggiunge di nuove e più avanzate, come la possibilità di tipare staticamente il linguaggio. In particolare, siccome TypeScript ‘conosce’ il linguaggio JavaScript, in molti casi genererà i tipi inferenziandoli¹⁴ dal codice. Per esempio, inizializzando una variabile con un valore arbitrario, TypeScript userà il valore come tipo.

```
let helloWorld: string  
let helloWorld = "Hello World!"
```

Figura 2.12: Inizializzazione del classico ‘Ciao Mondo’.

¹⁴L’inferenza è un processo di deduzione logica che permette di ottenere nuove informazioni a partire da quelle già note.

Comprendendo come funziona JavaScript, TypeScript può costruire un sistema di tipi che accetta il codice JavaScript, ma coi tipi; ciò offre un sistema di tipi completo, senza dover aggiungere caratteri extra per renderli espliciti. Infatti, è così che TypeScript è in grado di capire che ‘helloWorld’ è una stringa nell’esempio sopra, nella figura (2.12). Visual Studio Code, sviluppato da Microsoft e uno dei text editor più popolari tra gli sviluppatori, usa TypeScript per facilitare l’esperienza della scrittura del codice.

Inoltre, TypeScript è stato progettato per lo sviluppo di applicazioni di grandi dimensioni e viene transcompilato¹⁵ in JavaScript.

In origine fu pubblicato nel 2012, dopo due anni in cui Microsoft lo sviluppò internamente. L’obiettivo principale era quello di migliorare JavaScript (originariamente introdotto come linguaggio per il livello di presentazione, ma negli anni è cresciuto parecchio e è diventato complicato per mantenere e riusare il codice) in modo che gli sviluppatori potessero lavorare su applicazioni di grandi dimensioni, difficile con JavaScript. Infatti TypeScript è fortemente tipato¹⁶ e orientato agli oggetti¹⁷.

Importanti e numerose funzionalità, le quali aiutano lo sviluppo di applicazioni su larga scala, vengono introdotte da TypeScript:

- Controllo di tipo statico: TypeScript introduce il tipaggio statico, il quale può rilevare errori a tempo di compilazione, piuttosto che durante l’esecuzione, comuni in JavaScript;
- Oggetti e classi: le classi sono una delle caratteristiche principali della programmazione orientata agli oggetti;
- Moduli: possono aiutare ad organizzare e encapsulare il codice;
- Strumenti avanzati: autocompletamento, navigazione e refactoring¹⁸;
- Scalabilità: grazie a funzionalità quali tipi, classi e moduli, TypeScript è particolarmente adatto per applicazioni di grandi dimensioni;

Progetti su larga scala e molte grandi aziende hanno adottato TypeScript per lo sviluppo, per esempio Angular è scritto in TypeScript, come spiegato più approfonditamente nella sezione di Angular (2.2.1).

Di seguito, alcune delle aziende più famose che utilizzano TypeScript:

¹⁵Compilare codice sorgente, traducendolo dal suo linguaggio di programmazione in un qualsiasi altro, o in una versione più vecchia dello stesso linguaggio, producendo codice sorgente tradotto nel linguaggio di destinazione.

¹⁶Un linguaggio di programmazione è fortemente tipato se non permette operazioni tra tipi diversi.

¹⁷Un paradigma di programmazione che si basa sulla rappresentazione di oggetti e sulle relazioni tra di essi.

¹⁸Nell’ingegneria del software, il refactoring è una tecnica strutturata per modificare la struttura interna di porzioni di codice senza modificarne il comportamento esterno.

- Google
- Slack
- Medium
- Doordash
- bitpanda
- Techstack

TypeScript (TS) ha dimostrato di essere un'aggiunta preziosa e affidabile all'ecosistema JavaScript. Le sue funzionalità lo rendono uno strumento potente per lo sviluppo di applicazioni su larga scala, per uno sviluppo web efficiente e, come abbiamo visto, è stato ampiamente adottato nell'industria.

2.2.3 .NET Framework, .NET Core e .NET 8



Figura 2.13: Il logo di .NET Framework.

.NET è un framework software sviluppato da Microsoft, il quale include una vasta libreria di classi denominata Framework Class Library (FCL) e fornisce interoperabilità tra diversi linguaggi di programmazione. Inizialmente rilasciato nel 2002, .NET nacque come software proprietario, ma negli anni successivi Microsoft rilasciò le versioni open source. La FCL fornisce la UI, l'accesso ai dati, connettività ai DB, crittografia, sviluppo di applicazioni web, algoritmi numerici e comunicazioni di rete. Gli sviluppatori scrivono software unendo il loro codice sorgente con il framework di .NET e altre librerie. Tra le varie funzionalità di .NET troviamo una vasta libreria di soluzioni in codice per problemi comuni di programmazione e una macchina virtuale che gestisce l'esecuzione di programmi scritti specificatamente per .NET. Quest'ultimo, inoltre, supporta diversi linguaggi di programmazione in modo da poter permettere interoperabilità tra linguaggi. Ciò significa che ogni linguaggio può usare codice scritto in altri linguaggi. Il 'motore' dell'esecuzione di .NET è chiamato Common Language Runtime (CLR), il quale esegue il codice e gestisce le risorse del sistema operativo, come ad esempio la memoria, i file e i dispositivi di I/O. Ogni programma .NET è sotto la sua supervisione. .NET è usato da milioni di sviluppatori in tutto il mondo per creare applicazioni per una moltitudine di piattaforme e dispositivi, inclusi Windows, web, mobile, etc.

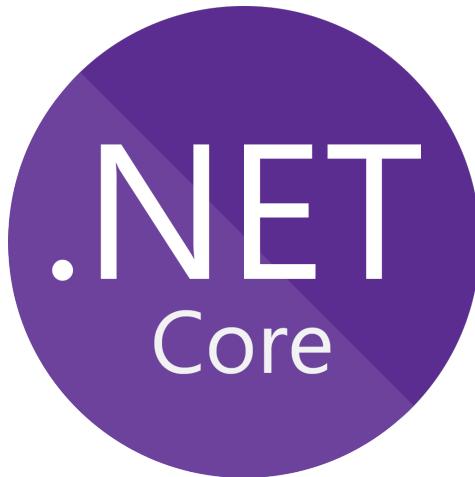


Figura 2.14: Il logo di .NET Core.

Successivamente a .NET Framework, Microsoft rilasciò .NET Core, una versione più leggera e veloce della precedente. Il nuovo framework fu fin da subito open source e multi-piattaforma -può essere eseguito su Windows, Linux e macOS, grazie a un'architettura modulare e flessibile- e fu annunciato come un modello di sviluppo in stile ‘buffet’, nel senso che gli sviluppatori possono scegliere i componenti di cui necessitano, senza dover installare il framework completo. .NET Core supporta la maggior parte delle vecchie librerie .NET, oltre che Visual Basic¹⁹ e F#²⁰. Dal 2016, .NET Core è stato rilasciato in varie versioni, l’ultima delle quali è la 3.1, rilasciata nel dicembre 2019; nel 2020 .NET Core è stato rinominato in .NET 5. L’ultima versione attualmente disponibile è la 8.0, rilasciata nel novembre 2023.



Figura 2.15: Il logo di .NET 8.

In breve, i due elementi principali dell’architettura di .NET sono:

1. Common Language Runtime (CLR): come già detto sopra, è il motore di esecuzione di .NET, il quale esegue il codice scritto in uno dei qualsiasi linguaggi

¹⁹Un linguaggio di programmazione orientato agli oggetti e agli eventi, multi-paradigma, sviluppato da Microsoft.

²⁰F# è un linguaggio di programmazione funzionale, fortemente tipato, multi-paradigma, sviluppato da Microsoft.

supportati da .NET. Carica soltanto le librerie necessarie per eseguire il codice, in modo da ridurre i tempi di avvio e l'uso della memoria. Inoltre, tra le sue funzionalità, sono degne di nota la gestione della memoria e della sicurezza automatica;

2. Framework Class Library (FCL): come già detto sopra, è un vasto insieme di classi e funzioni preimpostate che possono essere utilizzate per una vasta gamma di applicazioni

Negli anni, .NET, ha dimostrato di essere una piattaforma affidabile e flessibile per lo sviluppo di un'ampia tipologia di applicazioni; infatti, il suo supporto per diversi linguaggi di programmazione e piattaforme lo rende un'opzione versatile per gli sviluppatori.

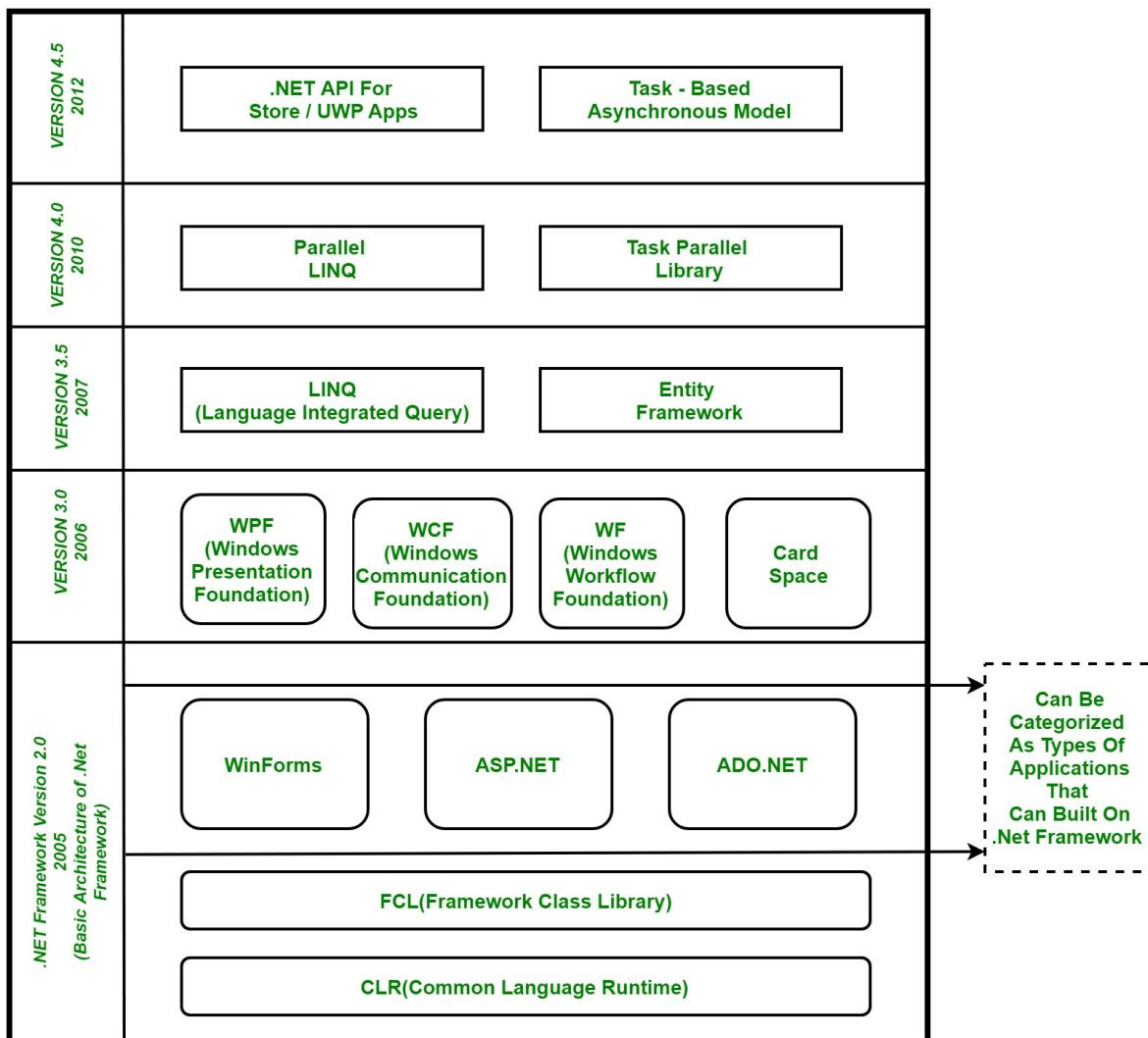


Figura 2.16: La pila tecnologica dei componenti di .NET (2005/2012).

2.2.4 ASP.NET

Una delle principali tecnologie di .NET è Active Server Pages.NET (ASP.NET), utilizzata ampiamente in questo progetto di prova finale. ASP.NET è un framework di sviluppo web, sviluppato da Microsoft, esattamente come il suo ‘fratello maggiore’ .NET; è stato rilasciato per la prima volta nel gennaio 2002 con la versione 1.0 del framework .NET. Attualmente, la versione più recente è la 4.8.1, rilasciata nel 2022. Al giorno d’oggi, in un mondo in cui le applicazioni web fanno sempre più parte della nostra vita quotidiana, scegliere il framework più adatto è essenziale per sviluppare applicazioni web robuste, scalabili e manutenibili. ASP.NET è un framework prominente in quest’ambito, offrendo agli sviluppatori la possibilità di creare applicazioni web moderne e piene di funzionalità.

Le funzionalità chiave di ASP.NET includono:

- Model-View-Controller (MVC): pattern architettonale che facilita la separazione delle responsabilità tra i 3 componenti (modello, vista e controller). Inoltre, il modello MVC favorisce codice organizzato, modulare e facile da mantenere, contribuendo alla scalabilità dell’applicazione web;
- C#²¹: uno degli aspetti distintivi di ASP.NET è la sua integrazione con il linguaggio di programmazione C# per la programmazione lato server. Sfruttando le capacità di C#, ASP.NET introduce principi fortemente tipati, orientati agli oggetti e un vasto ecosistema di librerie e strumenti per lo sviluppo web, ottenendo un codice più efficiente e mantenibile;
- Multi-piattaforma: ASP.NET offre ampio supporto per Windows, Linux e macOS, espandendo la portata delle applicazioni web e facilitando l’adozione di pratiche di sviluppo moderne, come le architetture a microservizi e la containerizzazione²²;
- Supporto Integrated Development Environment (IDE)²³: lo sviluppo ASP.NET è uniformemente integrato con Visual Studio, un potente IDE, il quale offre una miriade di funzionalità, alcune esclusive per ASP.NET.

²¹Un linguaggio di programmazione multi-uso, orientato agli oggetti, sviluppato da Microsoft. Principalmente, C# è tipato staticamente, fortemente tipato, dichiarativo, imperativo, funzionale, ha contesto lessicale e discipline di programmazione orientata ai componenti.

²²Dall’inglese ‘containerization’, è una tecnica di virtualizzazione a livello di sistema operativo che consente di eseguire più applicazioni in modo isolato su un singolo host. Differisce dalla virtualizzazione tradizionale in quanto non esegue un’intera macchina virtuale, ma esclusivamente l’applicazione e le sue dipendenze.

²³Dall’inglese ‘Integrated Development Environment’, è una tipologia di programma per sviluppatori che offre un insieme di strumenti per la scrittura, la compilazione e il debug del codice sorgente.

```

<table class="table">
  <thead>
    <tr>
      <th>@Html.DisplayNameFor(model => model.Name)</th>
      <th>@Html.DisplayNameFor(model => model.PhoneNumber)</th>
      <th>@Html.DisplayNameFor(model => model.Email)</th>
    </tr>
  </thead>
  <tbody>
    @foreach (var item in Model) {
      <tr>
        <td>@Html.DisplayFor(modelItem => item.Name)</td>
        <td>@Html.DisplayFor(modelItem => item.PhoneNumber)</td>
        <td>@Html.DisplayFor(modelItem => item.Email)</td>
      </tr>
    }
  </tbody>
</table>

```

A tooltip for the `item.Email` property is shown, listing methods: `Email`, `Equals`, `GetHashCode`, and `GetType`.

Figura 2.17: Esempio di una semplice applicazione web ASP.NET.

L'utilizzo di ASP.NET offre una serie di vantaggi empirici, tra cui:

- Scalabilità e alte prestazioni: ASP.NET è noto per la sua capacità di gestire carichi elevati e offrire prestazioni eccezionali. Funzionalità come la compilazione ‘just-in-time’²⁴, la memorizzazione dell’output nella cache e la programmazione asincrona contribuiscono alla scalabilità delle applicazioni web, garantendo un’esperienza d’uso scattante all’utente, anche sotto carichi pesanti;
- Sicurezza: funzionalità di priorità nel web development e ASP.NET offre un set di funzionalità robusto per affrontare varie problematiche di sicurezza, come ad esempio meccanismi di autenticazione integrati, procedure di codifica sicura e protezione contro le vulnerabilità più comuni. Tutto ciò migliora la stabilità complessiva delle applicazioni web di ASP.NET;
- Ecosistema ricco e supporto della comunità: l’ecosistema ASP.NET è ricco di librerie, framework e estensioni che accelerano e facilitano lo sviluppo. Inoltre, la comunità attiva intorno a ASP.NET garantisce un supporto continuo, aggiornamenti frequenti e un’ampia varietà di risorse per gli sviluppatori che cercano di migliorare le proprie competenze i specifici ambiti.

²⁴Un compilatore just-in-time (JIT) è un compilatore che trasforma il codice sorgente in codice macchina (binario) al momento preciso dell’esecuzione.

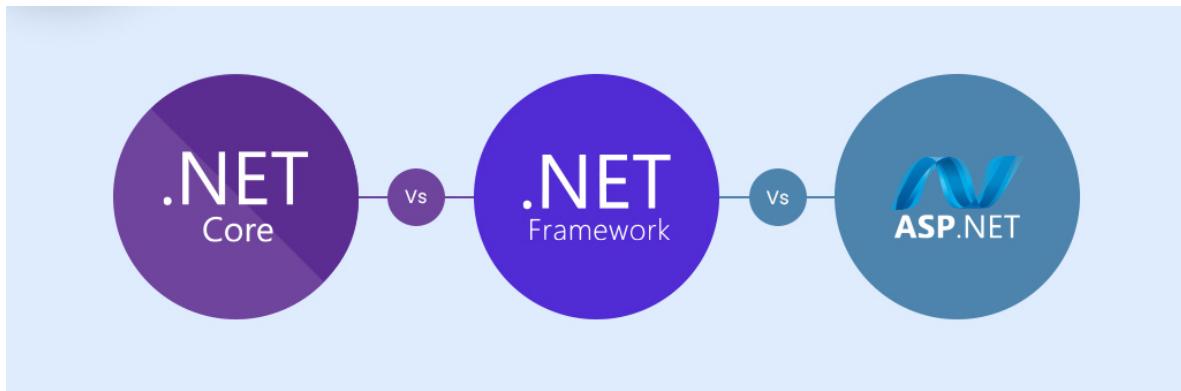


Figura 2.18: Le principali tecnologie Microsoft .NET.

Concludendo, ASP.NET resta una forza trainante nello sviluppo web, offrendo un framework completo e versatile per la loro costruzione. La sua architettura robusta, il supporto per lo sviluppo multi-piattaforma e l'integrazione con potenti strumenti di sviluppo lo rendono una scelta convincente per gli sviluppatori che mirano a fornire soluzioni web di alta qualità, scalabili e sicure.

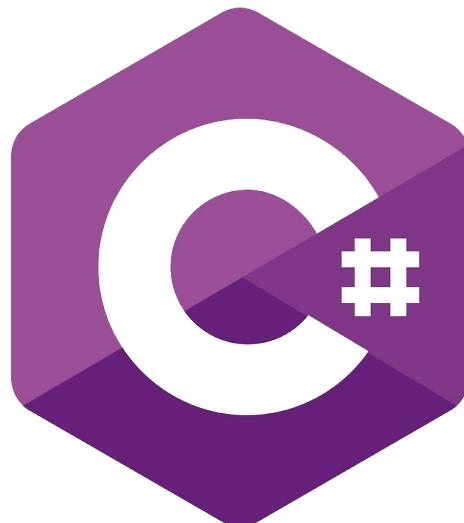


Figura 2.19: Logo di C#.

Capitolo 3

Prototipo

Questo capitolo descriverà il progetto di prova finale realizzato durante il tirocinio. In particolare, si parlerà di ognuno dei tre livelli di sviluppo di un'applicazione web: il database, il back-end e il front-end e delle scelte progettuali fatte per realizzare il prototipo.

La parte di autenticazione all'applicazione web è stata gestita da Gruppo SIGLA (GS) tramite un sistema di autenticazione di terze parti, in modo da poter gestire gli accessi e le autorizzazioni degli utenti in modo centralizzato, dai loro server esterni. Così facendo, il progetto non ha avuto necessità di sviluppare login e registrazione di utenti, ma si è concentrato sulle funzionalità di tracciamento e feedback delle attività formative erogate dall'azienda, in quanto la sicurezza di rete esulava dallo scopo del progetto di prova finale.

TODO mettere qui uno screen del prototipo finale.

3.1 Database

Innanzitutto è stato necessario progettare il database, nello specifico, è stato necessario creare le tabelle necessarie e riempirle con delle tuple¹ di prova per effettuare il giusto tracciamento del funzionamento e dei test del prototipo. Come da richiesta del tirocinio, le tabelle del database di gestione degli utenti non è stata realizzata, in quanto il progetto di tirocinio si è concentrato principalmente sulle funzionalità dell'amministratore e, quindi, il completo controllo -aggiungere, modificare, eliminare corsi formativi- delle attività formative erogate dall'azienda.

Il primo passo è stato quello di concepire le tabelle da integrare al DB già esistente -contenente le tabelle per la gestione di uffici e notizie di lavoro- e quindi fare le migrazioni² necessarie.

Grazie alle API di ASP.NET, è stato possibile creare e riempire le tabelle del database direttamente dal codice del backend, tramite le Object-Relational Mapping (ORM), le quali si comportano come da ponte tra il codice e il DB relazionale. In pratica, è una tecnica per convertire i dati tra un DB relazionale e lo heap³ di un linguaggio di programmazione orientato agli oggetti (può anche essere vista come uno strato che collega la programmazione orientata agli oggetti coi DB relazionali). In questo specifico caso, non c'è stato bisogno di collegare le tabelle. Nella sezione successiva (3.2), vedremo l'implementazione nel dettaglio, senza alcun codice SQL.

Per la tabella dei corsi di formazione, sono stati creati i seguenti campi:

- Id del corso (integer), chiave primaria
- CoursesName (string), nome del corso
- CoursesCapacity (integer), capacità massima di utenti del corso
- CoursesType (string), tipologia del corso
- CoursesDate (string), data di inizio del corso

Una volta fatto ciò, la tabella che tiene traccia delle versioni del database, ‘VersionInfo’ è stata aggiornata con le nuove migrazioni. Le sue colonne sono:

- Version (integer), versione del database

¹ In matematica, una tupla è una sequenza ordinata di elementi. In informatica, una tupla è una struttura dati che rappresenta una collezione di elementi, ognuno dei quali può essere di un tipo diverso.

² Le migrazioni sono un modo per aggiornare il database in modo incrementale, in modo che possa essere mantenuto e aggiornato senza doverlo ricreare ogni volta che cambia il modello di dati.

³ Lo heap è una regione di memoria in cui vengono allocati gli oggetti istanziati durante l'esecuzione di un programma.

- AppliedOn (datetime), data di applicazione della migrazione
- Description (text), descrizione della migrazione

Nella sezione successiva (3.2) verranno analizzate nel dettaglio le porzioni di codice atte a creare la tabella CoursesData e a riempirla con delle tuple di prova.

Infine, per concludere questa sezione, per visualizzare e gestire il database, tramite GUI, è stato utilizzato DBeaver, un software gratuito e open source per la gestione di database relazionali, il quale ne supporta la maggior parte, come MySQL, PostgreSQL, Oracle, SQLite e SQL Server. La scelta è ricaduta su DBeaver per la sua semplicità d'uso e il fatto che sia sviluppato dalla comunità open source. Il nome DBeaver deriva dalla combinazione di Database DB e Beaver, dall'inglese ‘castoro’, una tra le tante legature di parole tipiche dell’informatica.



Figura 3.1: Il logo di DBeaver.

3.2 Backend

In questa sezione descriverò come ho usato ASP.NET per lo sviluppo del back-end del progetto di prova finale.

Lo sviluppo principale è stato fatto appoggiandosi all'applicazione web già esistente, sulla quale Gruppo SIGLA (GS) ha sviluppato il progetto di tirocinio, che poi offre agli studenti. Quindi, basandomi sull'architettura già esistente, ho creato 5 nuove classi e modificato la classe Program.cs, già esistente, per gestire le funzionalità del prototipo. Queste classi sono:

- CoursesData.cs
- 4_CoursesMigration.cs
- CoursesController.cs
- CoursesDataGetListExecutor.cs
- CoursesDataExecuteExecutor.cs

In particolare, la classe CoursesData.cs è stata creata per definire la tabella 'CoursesData' e le sue colonne, tramite la libreria IdentityModel di ASP.NET. La classe viene gestita come un'entità grazie all'attributo [DbModel] prima della sua definizione e le colonne vengono definite come proprietà⁴ della classe stessa -coi rispettivi getter⁵ e setter⁶-, ognuna con un attributo loro attributo precedente che li definisce come proprietà JSON, come si può vedere nel listato successivo (3.1); nella sezione successiva (3.3) verrà descritto nel dettaglio il loro funzionamento.

⁴ In C#, una proprietà è un membro di una classe che fornisce un modo flessibile per leggere, scrivere o calcolare il valore di un campo privato.

⁵ Un getter è un metodo, encapsulato in una proprietà, che **legge** il valore di un campo privato di una classe.

⁶ Un setter è un metodo, encapsulato in una proprietà, che **scrive** il valore di un campo privato di una classe.

```

1  namespace It.gs.backend.Model
2  {
3      using System.Text.Json.Serialization;
4      using It.gs.Repository;
5      using It.gs.Repository.Model;
6
7      public class AddCoursesToDbExecuteInfo : IExecuteInfo {
8          public CoursesData[] Courses {get; set; }
9      }
10     public class DeleteCoursesDataExecuteInfo : IExecuteInfo {
11         public CoursesData[] Courses {get; set; }
12     }
13     public class UpdateCoursesDataExecuteInfo : IExecuteInfo {
14         public CoursesData[] Courses {get; set; }
15     }
16     [DbModel]
17     public class CoursesData
18     {
19         [JsonPropertyName("coursesId")]
20         public int CoursesId { get; set; }
21
22         [JsonPropertyName("coursesName")]
23         public string CoursesName { get; set; }
24
25         [JsonPropertyName("coursesCapacity")]
26         public int CoursesCapacity { get; set; }
27
28         [JsonPropertyName("coursesType")]
29         public string CoursesType { get; set; }
30
31         [JsonPropertyName("coursesDate")]
32         public string CoursesDate { get; set; }
33
34         [JsonPropertyName("count")]
35         public int? Count { get; set; }
36     }
37 }
```

Listing 3.1: Il codice in C# del file CoursesData.cs.

Figura 3.2: Il codice in C# del file CoursesData.cs.

Analizzando nel dettaglio `4_CoursesMigration.cs` invece, si può notare che è il file che contiene la creazione e la migrazione della tabella ‘CoursesData’. Di seguito, il listato (3.3) del codice in C# del file:

```

1  namespace It.gs.backend.Migrations {
2    using FluentMigrator; using It.gs.backend.Model;
3
4    [Migration(4)] // Increment the db version, fourth migration
5    public class CoursesMigration : Migration {
6      public override void Down() {}
7      public override void Up() {
8        IfDatabase("sqlserver").Create.Schema("StarterDb");
9        Create.Table($"{{nameof(CoursesData)}}")
10       .WithColumn($"{{nameof(CoursesData.CoursesId)}}").AsInt32().PrimaryKey().Identity()
11       .WithColumn($"{{nameof(CoursesData.CoursesName)}}").AsString().NotNullable().WithDefaultValue(false)
12       .WithColumn($"{{nameof(CoursesData.CoursesCapacity)}}").AsInt16().NotNullable().WithDefaultValue(10)
13       .WithColumn($"{{nameof(CoursesData.CoursesType)}}").AsString().NotNullable()
14       .WithColumn($"{{nameof(CoursesData.CoursesDate)}}").AsString().NotNullable();
15       var coursesRows = new List<dynamic>{
16         new { CoursesId = 1, CoursesName = "Angular Modulo 1", CoursesCapacity = 10, CoursesType = "Udemy", CoursesDate = "30-01-2024" },
17         new { CoursesId = 2, CoursesName = "Angular Modulo 2", CoursesCapacity = 10, CoursesType = "Udemy", CoursesDate = "30-02-2024" },
18         new { CoursesId = 3, CoursesName = "Angular Modulo 3", CoursesCapacity = 10, CoursesType = "Udemy", CoursesDate = "30-03-2024" },
19         new { CoursesId = 4, CoursesName = "Angular Modulo 4", CoursesCapacity = 10, CoursesType = "Udemy", CoursesDate = "30-04-2024" },
20         new { CoursesId = 5, CoursesName = "Angular Modulo 5", CoursesCapacity = 10, CoursesType = "Udemy", CoursesDate = "30-05-2024" },
21       };
22       foreach(var course in coursesRows)
23         Insert.IntoTable($"{{nameof(CoursesData)}}").Row(course);
24       // Delete.Table($"{{nameof(CoursesData)}}");
25     }
26   }
27 }
```

Figura 3.3: Il codice in C# del file `4_CoursesMigration.cs`.

Grazie alla libreria FluentMigrator di .NET, ho potuto creare la tabella ‘CoursesData’ e riempirla con le tuple di prova precedentemente citate. È ovviamente importante ricordarsi di estendere la classe `Migration` e di implementare i metodi `Down()` e `Up()`, altrimenti il codice non funzionerebbe. Inoltre, in questo caso, è anche necessario tener traccia della versione del DB, incrementando di volta in volta il numero della migrazione, come si può notare dall’attributo `[Migration ()]` a cui viene passato l’intero numero 4. I metodi `Up ()` e `Down ()` si occupano rispettivamente di ‘spostare’ il DB in avanti e indietro nelle migrazioni, a un qualsiasi punto nel tempo. L’unico accorgimento è che la migrazione all’indietro è distruttiva, in quanto le tabelle o le righe vengono eliminate nel processo. È quindi necessario e consigliabile fare preventivamente un backup⁷, se si vuole essere sicuri di mantenere i dati.

Analizzando nel dettaglio la porzione di codice del metodo `Up ()`, si può notare che, riguarda la creazione della tabella `CoursesData`: innanzitutto crea uno schema -`StarterD`- e controlla che sia effettivamente un DB SQLServer. Una volta sicuro, crea la tabella `CoursesData` e le sue colonne, specificandone il tipo e se sono o meno nullabili, in particolare:

- Il metodo `AsInt16` sta per ‘integer a 16 bit’, ovvero un intero a 2 byte;
- Il metodo `AsInt32` sta per ‘integer a 32 bit’, ovvero un intero a 4 byte;

⁷ Una copia di sicurezza dei dati, solitamente memorizzata in un luogo diverso da quello in cui si trovano i dati originali.

- Il metodo `AsString` comunica al DB che il campo su cui è chiamato sarà di tipo stringa;
- Il metodo `NotNullable` stabilisce che il campo non può essere nullo;
- Il metodo `WithDefaultValue` stabilisce il valore di default del campo, in questo caso `false`, ma ovviamente accetta anche `true` come parametro;
- Il metodo `PrimaryKey` stabilisce che il campo è una chiave primaria;
- Il metodo `Identity` stabilisce che il campo su cui è chiamato è un campo identità, ciò significa che il valore per questo campo sarà automaticamente generato dal DB e sarà incrementato di volta in volta.

Successivamente, viene creata una lista dinamica in cui vengono inserite le righe di prova, coi valori dei campi corrispondenti. Infine, viene fatto un ciclo `foreach` in cui, per ogni riga della lista, viene inserita una riga nella tabella `CoursesData`.

Durante lo sviluppo del prototipo, si è presentata la necessità di cancellare la tabella `CoursesData` per debuggare il codice, ecco il motivo della riga 24 commentata.

Passando invece al file `CoursesController.cs`, si può notare che è stato creato per gestire le richieste HTTP relative ai corsi di formazione. Di seguito, il listato (3.4) del codice in C# del file:

```

1 [Route("api/v{version:apiVersion}/[controller]")]
2 [ApiController]
3 [ApiVersion("1.0")]
4 [Authorize(Policy = Policies.Base)]
5 public class CoursesController : ControllerBase{
6     private readonly IRepositoryModule<CoursesData> coursesDataRepo;
7     public CoursesController(IRepositoryModule<CoursesData> CoursesDataRe
8
9     [HttpPost("searchCoursesData")]
10    [ProducesResponseType(typeof(CoursesData[]), 200)] ... (400) ... (500)
11    public async Task<IActionResult> SearchCoursesData([FromBody] DynamicQue
12    await GetList<CoursesData>(<e => StatusCode(500, e), <v => Ok(v), Maybe<C
13
14    [HttpPost("addCoursesData")]
15    [ProducesResponseType(200)] ... (400) ... (500)
16    public async Task<IActionResult> AddCoursesToDb([FromBody] CoursesData
17    var CoursesInfo = new AddCoursesToDbExecuteInfo {Courses = Courses};
18    var result = await this.coursesDataRepo.Execute(CoursesInfo);
19    if(result.IsSuccess) return Ok();
20    else throw result.Error.SourceException;
21 }
22
23 [HttpDelete("deleteCoursesData/{CoursesId}")]
24 [ProducesResponseType(200)] ... (400) ... (500)
25 public async Task<IActionResult> DeleteCoursesData(int CoursesId) {
26    var CoursesInfo = new DeleteCoursesDataExecuteInfo{Courses=new[]{new Cou
27    var result = await this.coursesDataRepo.Execute(CoursesInfo);
28    if (result.IsSuccess) return Ok();
29    else throw result.Error.SourceException;
30 }
31
32 [HttpPut("updateCoursesData")] ... [ProducesResponseType(... ... ...)]
33 public async Task<IActionResult> UpdateCoursesData([FromBody] CoursesDat
34 var CoursesInfo = new UpdateCoursesDataExecuteInfo{Courses = Courses};
35 var result = await this.coursesDataRepo.Execute(CoursesInfo);
36 if (result.IsSuccess) return Ok();
37 else throw result.Error.SourceException;
38 }
39 }
```

Figura 3.4: Il codice in C# del file CoursesController.cs.

Nota che alcuni attributi sono stati sostituiti con dei puntini, per motivi di spazio e per ridondanza (l'unico cambiamento è il parametro, -200, 400 o 500- il resto è identico) e che il namespace e gli using sono stati omessi per gli stessi motivi.

La classe mostrata precedentemente (3.4), necessita di estendere la classe ControllerBase per poter gestire le richieste HTTP citate sopra. Altri dettagli degni di nota sono:

- L'attributo [Route ()] specifica il percorso della richiesta HTTP e il numero di versione dell'API;
- L'attributo [Authorize ()] specifica una politica di autorizzazione, in questo caso Policies.Base;
- Tutti i metodi sono asincroni, in quanto si aspettano una risposta dal DB e, quindi, non possono bloccare il processo principale;
- Il metodo SearchCoursesData () è preceduto dall'attributo [HttpPost ()], il quale specifica che la richiesta HTTP sarà di tipo POST, dal nome 'searchCoursesData' e dall'attributo [ProducesResponseType ()], il quale stabilisce che la risposta del metodo sarà un array di tipo CoursesData, con codice di stato 200, 400 o 500;
- Analogamente per gli altri metodi, AddCoursesToDb (), DeleteCoursesData () e UpdateCoursesData (), con la differenza che 'updateCoursesData' userà il metodo HTTP PUT e 'deleteCoursesData' userà il metodo HTTP DELETE.

A questo punto, CoursesDataExecuteExecutor.cs si occupa di eseguire l'operazione richiesta (richiesta o risposta HTTP), le quali sono state definite dal 'controller' descritto sopra. Qui di seguito, il listato (3.5) del codice in C# del file 'Executor':

```
1  public class CoursesDataExecuteExecutor_ : IExecuteWithTransactionExecutor<CoursesData>
2  {
3      // ...
4  }
```

Figura 3.5: Il codice in C# della classe di CoursesDataExecuteExecutor.cs, successivamente diviso in blocchi di codice per motivi di spazio.

```

1  public async Task<IExecuteResult> Execute(DatabaseSettings settings, IDbConnection connection, IDbTransaction transaction, IExecuteInfo info)
2  {
3      switch(info) {
4          case AddCoursesToDbExecuteInfo i:
5              return await AddCoursesExecute(settings, connection, transaction, i);
6          case DeleteCoursesDataExecuteInfo i:
7              return await DeleteCoursesDataExecute(settings, connection, transaction, i);
8          case UpdateCoursesDataExecuteInfo i:
9              return await UpdateCoursesDataExecute(settings, connection, transaction, i);
10         default: throw new NotSupportedException($"Execute with transaction for type {info.GetType().FullName} not supported");
11     }
12 }
13 private async Task<IExecuteResult> AddCoursesExecute(DatabaseSettings settings, IDbConnection connection, IDbTransaction transaction,
14             ↪ AddCoursesToDbExecuteInfo info) {
15     foreach(var Course in info.Courses)
16     {
17         _ = await Add(settings, connection, transaction, Course);
18     }
19     return await IExecuteResult.From(true);
20 }
21 private async Task<CoursesData> Add(DatabaseSettings settings, IDbConnection connection, IDbTransaction transaction, CoursesData item)
22 {
23     Console.WriteLine("item: ", item);
24     var sql = $"INSERT INTO CoursesData (CoursesName, CoursesCapacity, CoursesType, CoursesDate) VALUES (@{nameof(CoursesData.CoursesName)} ,
25             ↪ @{nameof(CoursesData.CoursesCapacity)} , @{nameof(CoursesData.CoursesType)} , @{nameof(CoursesData.CoursesDate)})";
26     var r = await connection.ExecuteAsync(sql, item, transaction);
27     return item;
28 }
```

Figura 3.6: Il codice di CoursesDataExecuteExecutor.cs, relativo alla scelta del metodo e all'aggiunta delle tuple.

```

1  private async Task<IExecuteResult> DeleteCoursesDataExecute(DatabaseSettings settings, IDbConnection connection, IDbTransaction transaction,
2             ↪ DeleteCoursesDataExecuteInfo info) {
3     foreach(var Course in info.Courses)
4     {
5         _ = Remove(settings, connection, transaction, Course);
6     }
7     return await IExecuteResult.From(true);
8 }
9 private async Task<CoursesData> Remove(DatabaseSettings settings, IDbConnection connection, IDbTransaction transaction, CoursesData item)
10 {
11     var sql = $"DELETE FROM CoursesData WHERE CoursesId = @CoursesId";
12     var affectedRows = await connection.ExecuteAsync(sql, new { CoursesId = item.CoursesId }, transaction);
13     var r = await connection.ExecuteAsync(sql, item, transaction);
14     return item;
15 }
```

Figura 3.7: Il codice di CoursesDataExecuteExecutor.cs, relativo alla cancellazione delle tuple.

```

1  private async Task<IExecuteResult> UpdateCoursesDataExecute(DatabaseSettings settings, IDbConnection connection, IDbTransaction transaction,
2             ↪ UpdateCoursesDataExecuteInfo info) {
3     foreach(var Course in info.Courses)
4     {
5         _ = await Update(settings, connection, transaction, Course);
6     }
7     return await IExecuteResult.From(true);
8 }
9 private async Task<CoursesData> Update(DatabaseSettings settings, IDbConnection connection, IDbTransaction transaction, CoursesData item)
10 {
11     var sql =
12     $"UPDATE {nameof(CoursesData)} " +
13     $"SET " +
14     $" {nameof(CoursesData.CoursesName)} = @{nameof(CoursesData.CoursesName)}, " +
15     $" {nameof(CoursesData.CoursesCapacity)} = @{nameof(CoursesData.CoursesCapacity)}, " +
16     $" {nameof(CoursesData.CoursesType)} = @{nameof(CoursesData.CoursesType)}, " +
17     $" {nameof(CoursesData.CoursesId)} = @{nameof(CoursesData.CoursesId)}, " +
18     $" {nameof(CoursesData.CoursesDate)} = @{nameof(CoursesData.CoursesDate)} " +
19     $" WHERE {nameof(CoursesData.CoursesId)} = @{nameof(CoursesData.CoursesId)}";
20     var r = await connection.ExecuteAsync(sql, item, transaction);
21     return item;
22 }
```

Figura 3.8: Il codice di CoursesDataExecuteExecutor.cs, relativo all'aggiornamento delle tuple.

La classe `CoursesDataExecuteExecutor.cs`, come possiamo vedere dai listati soprastanti, smista la richiesta grazie ai parametri passati. In particolare tramite il parametro `info`, è in grado di eseguire il giusto metodo per ogni tipologia richiesta. Ognuno dei metodi ausiliari `AddCoursesExecute ()`, `DeleteCoursesDataExecute ()` e `UpdateCoursesDataExecute ()` si occupa di chiamare i rispettivi metodi `Add ()`, `Remove ()` e `Update ()`, usando i parametri corretti; questi ultimi invece, imposteranno le query corrette per ogni tipo di operazione richiesta, come si può notare dai listati (3.6), (3.7) e (3.8), per poi chiamare il metodo `ExecuteAsync ()`, il quale esegue effettivamente la query, infatti, `ExecuteAsync ()` è un metodo asincrono, in quanto si aspetta una risposta dal DB e, quindi, non può bloccare il processo principale. Inoltre, è un metodo che restituisce un oggetto di tipo `Task`, il quale rappresenta un'operazione, sempre asincrona, che può restituire un valore, in questo caso un oggetto di tipo `IExecuteResult`, il quale rappresenta il risultato dell'operazione.

È degno di nota anche il file `CoursesDataGetListExecutor.cs`, il quale si occupa di raccogliere la lista dei corsi di formazione, tramite il metodo `GetList ()`, chiamato dalla classe `CoursesController` descritta in precedenza. Di seguito, il listato (3.9) del codice in C# del file:

```

1  public class CoursesDataGetListExecutor_ : IGetListExecutor<CoursesData>, IAddExecutor<CoursesData>
2  {
3      public async Task<CoursesData> Add(DatabaseSettings settings, IDbConnection connection, IDbTransaction transaction, CoursesData item)
4      {
5          var sql = $"INSERT INTO CoursesData (CoursesName, CoursesCapacity, CoursesType, CoursesDate) VALUES (@{nameof(CoursesData.CoursesName)}, "
6          ↪ @{nameof(CoursesData.CoursesCapacity)}), @{nameof(CoursesData.CoursesType)}, @{nameof(CoursesData.CoursesDate)})";
7
8          var r = await connection.ExecuteAsync(sql, item, transaction);
9          if(r > 0) return item;
10         else throw new InvalidOperationException("ERROR!!!");
11     }
12
13     public async Task<IEnumerable<CoursesData>> GetList(DatabaseSettings settings, IDbConnection connection, Maybe<CoreDynamicQueryPart> query)
14     {
15         var (countSql, sql, parameters) = query.EnsureOrderBy(nameof(CoursesData.CoursesId)) .ComposeWithCount<DynamicParameters>($"SELECT * $FROM{nameof(CoursesData)}", nameof(CoursesData.CoursesId), settings);
16         var result = await connection.QueryAsync<CoursesData>(sql: sql, param: parameters);
17         var count = await connection.QuerySingleAsync<int>(sql: countSql, param: parameters);
18         return result.Select(x => { x.Count = count; return x; });
19     }

```

Figura 3.9: Il codice di `CoursesDataGetListExecutor.cs`, classe contenente il metodo `GetList ()`.

Si nota l'implementazione di un altro metodo `Add ()`: le differenze con il metodo `Add ()` della classe `CoursesDataExecuteExecutor.cs` sono minime, in quanto entrambi eseguono una query di inserimento, ma per la classe (3.9) è un metodo pubblico, il quale effettua anche un controllo sull'esito dell'operazione, restituendo un'eccezione in caso negativo. È importante notare che la classe `CoursesDataGetListExecutor` usa la libreria Dapper, una micro ORM per interagire con il DB. Infatti, Dapper mappa i risultati delle query SQL in oggetti C#, il che rende più facile lavorare con i dati

relazionali risultanti.

Infine, è stato necessario configurare il file principale Program.cs, aggiungendogli le righe di codice relative alle nuove classi e ai loro metodi. La prima build⁸ del backend era pronta.

A questo punto, era necessario testare che l'implementazione del backend funzionasse correttamente. Ho utilizzato una API apposita, Swagger.

3.2.1 Swagger



Figura 3.10: Il logo di Swagger.

Swagger è un insieme di strumenti per sviluppatori di API, originariamente specificata e sviluppata da SmartBear Software, nel 2011. In questo caso, il suo utilizzo è stato principalmente quello di testare il corretto funzionamento del backend. Il suo vantaggio principale sta nella possibilità di utilizzarlo direttamente dal browser, senza dover installare alcun software aggiuntivo, è sufficiente digitare un indirizzo URL apposito -per esempio <https://localhost:5001/swagger/index.html>- e si aprirà una pagina web con un'interfaccia grafica dedicata, in cui ci sarà una lista di tutte le API disponibili, sviluppate con ASP.NET.

⁸ Il processo di compilazione di un programma, ovvero la trasformazione del codice sorgente in codice eseguibile, in questo riferita alla compilazione eterogenea di tutti i file del backend.

Figura 3.11: L'interfaccia grafica di Swagger, relativa al progetto di tirocinio.

Una volta autenticatosi con le giuste credenziali -nel mio caso, quelle fornite da Gruppo SIGLA-, si ha l'autorizzazione necessaria per poter eseguire le API sviluppate. Di seguito alcuni esempi di esecuzione e del loro risultato:

Figura 3.12: Esempio di esecuzione di una API per searchCoursesData, il corpo della richiesta è in formato JSON.

<p>200 Success</p> <p>Media type <input checked="" type="button" value="text/plain"/> ▼</p> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>[{ "coursesId": 0, "coursesName": "string", "coursesCapacity": 0, "coursesType": "string", "count": 0, "courseDescription": "string", "courseTags": "string", "courseState": true, "courseLink": "string", "courseDate": "string" }]</pre>	<i>No links</i>
<p>400 Bad Request</p> <p>Media type <input type="button" value="text/plain"/> ▼</p> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>{ "type": "string", "title": "string", "status": 0, "detail": "string", "instance": "string", "additionalProp1": "string", "additionalProp2": "string", "additionalProp3": "string" }</pre>	<i>No links</i>
<p>401 Unauthorized</p>	<i>No links</i>
<p>403 Forbidden</p>	<i>No links</i>
<p>500 Server Error</p>	<i>No links</i>

Figura 3.13: Alcuni esempi di risultati di esecuzione di API tramite Swagger, in ordine 200, ‘Successo’, 400, ‘Cattiva richiesta’, 401, ‘Non autorizzato’, 403, ‘Vietato’ e 500, ‘Errore interno del server’.

3.3 Frontend

Questa sezione descriverà come ho usato Angular per lo sviluppo e la ratio dietro le scelte progettuali riguardanti il livello di presentazione del progetto di prova finale, ossia il frontend.

L'unico requisito di sistema è stato avere Node Package Manager (NPM) installato, in quanto Angular è un framework che si basa su di esso. NPM è un gestore di pacchetti per JS, in particolare è quello di default per Node.js, un ambiente d'esecuzione JS. NPM è composto da un client da linea di comando e un database online di pacchetti pubblici e privati, chiamato Node Package Manager (NPM) registry. Per installare NPM più agevolmente, ho usato Node Version Manager (NVM), un gestore di versioni di Node.js, il quale permette di installare e gestire più versioni di Node.js su un sistema. Di seguito, il codice bash per installare NVM:

```
1 curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash
```

Una volta fatto ciò, è stato necessario installare la versione 16 di Node.js, la quale è stata usata per lo sviluppo del frontend. Di seguito, il codice bash per installare la versione 16 di Node.js:

```
1 nvm install 16
2 nvm use 16.x.x
```

Il passo successivo è stato quello di installare tutte le dipendenze necessarie per far funzionare il frontend del progetto di prova finale. Di seguito, il relativo codice da eseguire su terminale:

```
1 npm install # (npm v 16.18.1)
```

Infine, è stato necessario compilare e avviare il frontend. Di seguito, il relativo codice da eseguire su terminale:

```
1 npm run build # l'equivalente di 'ng build'
2 npm run start # l'equivalente di 'ng serve'
```

Finalmente, è stato possibile accedere all'interfaccia grafica (GUI) del frontend, digitando e navigando all'indirizzo <http://localhost:4200/> su un qualsiasi browser.

Nota che le scritte in verde, precedute da un '#', sono commenti, quindi non vengono eseguite dal terminale.

Passando allo sviluppo del codice vero e proprio, ho utilizzato la CLI di Angular, la quale mi ha permesso di creare nuovi componenti, direttamente da terminale, con un solo comando. Il codice bash per installarla è:

```
1 npm install -g @angular/cli
```

Di seguito, il codice bash per creare un nuovo componente:

```
1 ng generate component education # crea un nuovo componente chiamato 'educationComponent', per gestire i corsi formativi, dal punto di vista di un ut...
```

Se il comando viene eseguito correttamente, verrà creato una nuova cartella, chiamata education, all'interno della cartella src/app/components, con i file necessari per far funzionare il componente, come si può vedere dalla figura (3.14).

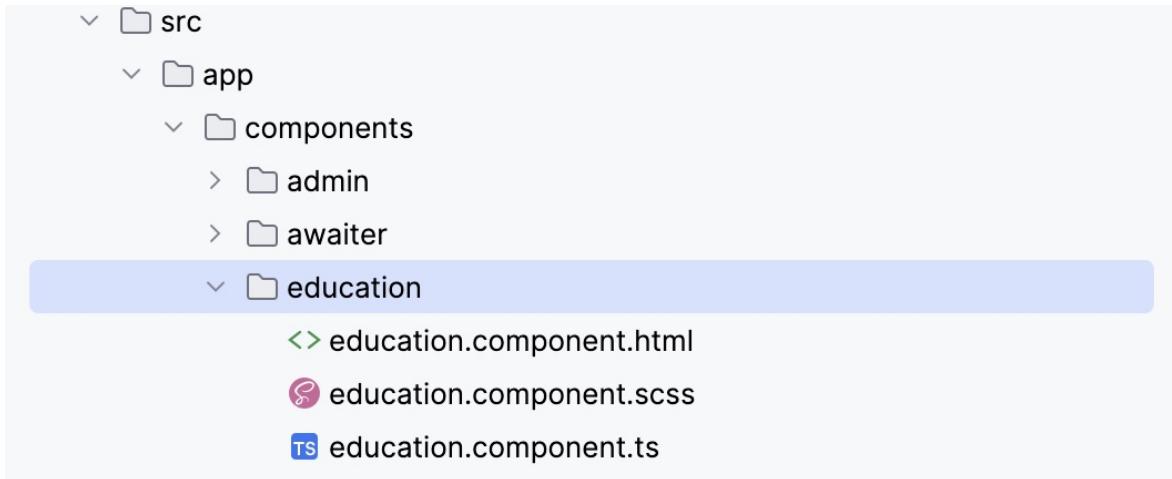


Figura 3.14: La struttura della cartella education, creata tramite la CLI di Angular.

Una volta creato il componente, è stato necessario configurare il file che si occupa di gestire l'instradamento dell'applicazione, app-routing.module.ts, per poter visualizzare correttamente il nuovo componente. Successivamente è stato necessario configurare il file app.module.ts: in particolare, è stato necessario importare il nuovo componente e aggiungerlo all'array declarations del modulo NgModule. Di seguito, il codice in TypeScript del file app.module.ts e del file app-routing.module.ts:

```

1 import {EducationManagementComponentComponent} from './components/admin/educationManagementComponent/ educationManagementComponent.component';
2 import {EducationComponent} from './components/education/ education.component';
3 import {CoursesEffect} from "./redux/courses/courses.effects";
4 import {CoursesResolver} from "./resolvers/courses.resolver";
5 import {EnrollmentComponent} from "./components/enrollment/enrollment.component";

```

Figura 3.15: Tutti i componenti e i servizi importati nel file app.module.ts, anche quelli che verranno descritti successivamente.

```

1  @NgModule({
2    bootstrap: [AppComponent],
3    declarations: [
4      AppComponent,
5      AwaiterComponent,
6      MessageComponent,
7      AppMainComponent,
8      AppConfigComponent,
9      AppLoginComponent,
10     AppNotFoundComponent,
11     AppErrorComponent,
12     AppAccessdeniedComponent,
13     DurationPipe ,
14
15    NoticeComponent,
16    ReservationComponent,
17    OfficesManagementComponent,
18    NoticesManagementComponent,
19    ViewsManagementComponent,
20
21    EducationManagementComponentComponent,
22    EducationComponent,
23    EnrollmentComponent ,
24  ],
25  imports: [
26    ...
27  ],
28  providers: [
29    ...
30  ]
31 })
32 export class AppModule {}

```

Figura 3.16: L'array declarations del modulo NgModule del file app.module.ts, con l'aggiunta dei componenti sviluppati per il progetto di prova finale.

Nota che i puntini rappresentano l'omissione di codice non rilevante ai fini della spiegazione.

```

1 import ...
2 import { EnrollmentComponent } from "./components/enrollment/enrollment.component";
3 import { EducationComponent } from "./components/education/education.component";
4 import { EducationManagementComponentComponent } from "./components/admin/educationManagementComponent/ educationManagementComponent.component";
5
6 @NgModule({
7   imports: [ RouterModule.forRoot([
8     { path: "", redirectTo: "/notice", pathMatch: "full" },
9     {
10       path: "",
11       component: AppLayoutComponent,
12       children: [ { ... } ]
13       {
14         path: "education",
15         component: EducationComponent,
16         canActivate: [GuestUserGuard, BaseUserGuard, VipUserGuard, AdminUserGuard],
17         resolve: { coursesData: "CoursesResolver" },
18       },
19       {
20         path: "educationManagementComponent",
21         component: EducationManagementComponentComponent,
22         canActivate: [AdminUserGuard],
23         resolve: { coursesData: "CoursesResolver" },
24       }
25     ],
26   ],
27   { path: "error", component: AppErrorComponent },
28   { path: "access", component: AppAccessdeniedComponent },
29   { path: "notfound", component: AppNotFoundComponent },
30   { path: "login", component: AppLoginComponent },
31   ],
32   { scrollPositionRestoration: "enabled" }
33 ),],
34 exports: [RouterModule],
35 providers: [GuestUserGuard, BaseUserGuard, VipUserGuard, AdminUserGuard],
36 })
37 export class AppRoutingModule {}

```

Figura 3.17: Il codice del file app-routing.module.ts, il quale si occupa di instradare l'applicazione, a seconda dell'utente e della 'guardia' attivata.

Nota che i puntini rappresentano l'omissione di codice non rilevante ai fini della spiegazione.

Una volta che il componente è stato configurato correttamente all'interno dei file sopra citati, è stato possibile iniziare a sviluppare il codice del componente stesso. Ossia i file:

- `education.component.html`, il quale si occupa di gestire la parte HTML del componente, è quindi il template dello stesso;
- `education.component.ts`, il quale si occupa di gestire la parte TypeScript del componente;

Nota che il file `education.component.css` è stato omesso, in quanto vuoto, perché il CSS di ogni componente è stato gestito tramite il file `styles.css`, come si può notare dalla figura (3.22).

```

1 <div class="card">
2   <div class="flex"> <span style="font-size: 25px;">Gestione corsi (Admin)</span> </div>
3   <div class="container mt-5">
4     <h4 class="mb-4">{{ showAddForm ? 'Aggiungi' : 'Modifica' }} corso</h4>
5     <form (ngSubmit)="showAddForm ? addCourse(courseForm) : updateCoursesData(courseForm)" #courseForm="ngForm">
6       <div class="p-grid">
7         <div class="p-col-12 p-md-6">
8           <div class="p-inputgroup">
9             <span class="p-float-label">
10               <input *ngIf="showAddForm" pInputText name="name" [(ngModel)]="course.coursesName" required>
11               <input *ngIf="!showAddForm" pInputText name="name" [(ngModel)]="editedCourse.coursesName" required> <label>Nome corso</label>
12             </span>
13           </div>
14           <ng-container *ngIf="courseForm.controls['name']">
15             <ng-container *ngIf="courseForm.controls['name'].invalid && (courseForm.controls['name'].dirty || courseForm.controls['name'].touched)">
16               <div class="error-message" *ngIf="courseForm.controls['name'].errors.required">
17                 Nome corso obbligatorio
18               </div>
19             </ng-container>
20           </ng-container>
21         </div>
22         <br>
23         <div class="p-col-12 p-md-6">
24           <div class="p-inputgroup">
25             <span class="p-float-label">
26               <input *ngIf="showAddForm" pInputText name="capacity" [(ngModel)]="course.coursesCapacity" required>
27               <input *ngIf="!showAddForm" pInputText name="capacity" [(ngModel)]="editedCourse.coursesCapacity" required> <label>Numero
28                 postazioni</label>
29             </span>
30           </div>
31           <ng-container *ngIf="courseForm.controls['capacity']">
32             <ng-container *ngIf="courseForm.controls['capacity'].invalid && (courseForm.controls['capacity'].dirty || courseForm.controls['capacity'].touched)">
33               <div class="error-message" *ngIf="courseForm.controls['capacity'].errors.required">
34                 Numero postazioni obbligatorio
35               </div>
36             </ng-container>
37           </ng-container>
38         </div>
39         <br>
40         <div class="p-col-12">
41           <div class="p-inputgroup">
42             <span class="p-float-label">
43               <p-dropdown *ngIf="showAddForm" [options]="types" [(ngModel)]="selectedType" placeholder="Seleziona tipologia corso"
44                 <optionLabel="types" name="types" [style]="{{ 'width': '100%' }}" required></p-dropdown>
45               <p-dropdown *ngIf="!showAddForm" [options]="types" [(ngModel)]="selectedType" placeholder="Seleziona tipologia corso"
46                 <optionLabel="types" name="types" [style]="{{ 'width': '100%' }}" required></p-dropdown> <label>Seleziona tipologia corso
47             </span>
48           </div>
49           <ng-container *ngIf="courseForm.controls['coursesType']">
50             <ng-container *ngIf="courseForm.controls['coursesType'].invalid && (courseForm.controls['coursesType'].dirty || courseForm.controls['coursesType'].touched)">
51               <div class="error-message" *ngIf="courseForm.controls['coursesType'].errors.required">
52                 Tipologia corso obbligatoria
53               </div>
54             </ng-container>
55           </ng-container>
56         </div>
57         <br>
58         <div class="p-col-12 p-md-6">
59           <div class="p-inputgroup">
60             <span class="p-float-label">
61               <p-calendar *ngIf="showAddForm" [(ngModel)]="course.coursesDate" name="date" dataType="string" dateFormat="dd/mm/yy" [showIcon] = "true" required></p-calendar>
62               <p-calendar *ngIf="!showAddForm" [(ngModel)]="editedCourse.coursesDate" name="date" dataType="string" dateFormat="dd/mm/yy" [showIcon] = "true" required></p-calendar> <label>Data inizio corso </label>
63             </span>
64           </div>
65           <ng-container *ngIf="courseForm.controls['coursesDate']">
66             <ng-container *ngIf="courseForm.controls['coursesDate'].invalid && (courseForm.controls['coursesDate'].dirty || courseForm.controls['coursesDate'].touched)">
67               <div class="error-message" *ngIf="courseForm.controls['coursesDate'].errors.required">
68                 Data inizio obbligatoria
69               </div>
70             </ng-container>
71           </ng-container>
72         </div><br><br><br>
73         <div *ngIf="showAddForm">
74           <button pButton type="submit" label="Aggiungi corso" class="p-button-rounded p-button-success" [disabled]="courseForm.invalid || selectedUsers.length > course.coursesCapacity"></button>
75           <div *ngIf="successAdd" class="success-message">
76             Corso aggiunto correttamente!
77           </div>
78           <div class="error-message" *ngIf="selectedUsers.length > course.coursesCapacity">
79             Il numero di utenti inseriti supera la disponibilità del corso
80           </div>
81         </div>
82         ...

```

Figura 3.18: Il codice HTML del file education.component.html, in particolare il form per l'aggiunta e la modifica dei corsi di formazione.

Notiamo che una parte del codice è stata omessa, tramite dei puntini, in quanto identica al ‘div’ precedente, ma invece di gestire l’aggiunta dei corsi, ne gestisce la modifica.

Analizzando il codice soprastante (3.18), la parte del template HTML contenente la totalità dei div che compongono il form⁹ per la gestione dei corsi di formazione -da parte dell’admin- troviamo tre azioni principali disponibili:

- addCourse, il quale raccoglie i dati inseriti dall’utente tramite un form apposito e i container di PrimeNG -i quali verranno trattati più approfonditamente nella sezione (3.3.1)- e li invia al backend, tramite i metodi TS appositi;
- updateCoursesData, il quale condivide il form d’inserimento dati con addCourse, ma solamente dopo che l’utente ha cliccato il pulsante di modifica apposito, nella tabella dei corsi di formazione. Ovviamente, invierà comunque i dati al backend, ma stavolta per aggiornare una tupla già esistente;
- deleteCoursesData, il quale si occupa di cancellare una tupla, e quindi un corso, già esistente, tramite il metodo TS apposito e una semplice query.

Inoltre, ogni campo del form è dotato degli appositi attributi required, per garantire che l’utente non possa inviare un form vuoto, e ngModel, per legare i dati inseriti dall’utente al componente TS corrispondente e ogni azione è accompagnata da un messaggio di successo o di errore, il quale verrà mostrato all’utente, tramite un div apposito, in caso di esito positivo o negativo.

⁹ Un form è un’interfaccia grafica che permette all’utente di inserire dati in un’applicazione web

```

1 <p-table styleClass="p-datable-courses-data" [value]="coursesData$ | async" dataKey="chargePointId"
2 currentPageReportTemplate="Showing {first} to {last} of {totalRecords} entries" responsiveLayout="scroll"
3 [rows]="5" [rowHover]="true" [paginator]="true" [showCurrentPageReport]="true" [rowsPerPageOptions]=[5,10,15]"
4 [loading]="isLoading$ | async" [totalRecords]="totalRecords$ | async" [filterDelay]=300"
5 [globalFilterFields]="globalFiltersFields" [(selection)]="selectedCoursesData" [lazy]="true" [customSort]="true"
6 (onLazyLoad)="sort($event)" [tableStyle]={`${min-width}: '25rem'}`>
7   <ng-template pTemplate="caption">
8     <div class="flex">
9       <span style="font-size: 25px;">Elenco corsi creati</span>
10      <span class="p-input-icon-left ml-auto">
11        <i class="pi pi-search"></i>
12        <input pInputText [value]="lastSearch$ | async" type="text" (input)="filter($event.target.value)" placeholder="Course Search" />
13      </span>
14    </div>
15  </ng-template>
16  <ng-template pTemplate="header">
17    <tr>
18      <th pSortableColumn="coursesName">
19        <div class="p-d-flex p-jc-between p-ai-center">
20          Nome corso
21          <p-sortIcon field="coursesName"></p-sortIcon>
22        </div>
23      </th>
24      <th>
25        <div class="p-d-flex p-jc-between p-ai-center">
26          Partecipanti max
27        </div>
28      </th>
29      <th pSortableColumn="coursesType">
30        <div class="p-d-flex p-jc-between p-ai-center">
31          Tipologia
32          <p-sortIcon field="coursesType"></p-sortIcon>
33        </div>
34      </th>
35      <th pSortableColumn="coursesDate">
36        <div class="p-d-flex p-jc-between p-ai-center">
37          Data inizio
38          <p-sortIcon field="coursesDate"></p-sortIcon>
39        </div>
40      </th>
41      <th>
42        <div class="p-d-flex p-jc-between p-ai-center">
43          Modifica
44        </div>
45      </th>
46      <th>
47        <div class="p-d-flex p-jc-between p-ai-center">
48          Elimina
49        </div>
50      </th>
51    </tr>
52  </ng-template>
53  <ng-template pTemplate="body" let-coursesData>
54    <tr class="p-selectable-row" [pRowToggler]="coursesData.coursesId">
55      <td>
56        <span class="p-column-title">Nome corso</span>
57        {{coursesData.coursesName}}
58      </td>
59      <td>
60        <span class="p-column-title">Num. postazioni max</span>
61        {{coursesData.coursesCapacity}}
62      </td>
63      <td>
64        <span class="p-column-title">Tipologia</span>
65        {{coursesData.coursesType}}
66      </td>
67      <td>
68        <span class="p-column-title">Data inizio</span>
69        {{coursesData.coursesDate}}
70      </td>
71      <td>
72        <span class="p-column-title">Modifica</span>
73        <p-button icon="pi pi-file-edit" (click)="editCoursesData(coursesData); showAddForm = false;"></p-button>
74      </td>
75      <td>
76        <span class="p-column-title">Elimina</span>
77        <p-button icon="pi pi-trash" (click)='deleteCoursesData(coursesData)'></p-button>
78      </td>
79    </tr>
80  </ng-template>
81  <ng-template pTemplate="emptymessage">
82    <tr>
83      <td colspan="9">No Courses Data found.</td>
84    </tr>
85  </ng-template>
86 </p-table>

```

Figura 3.19: Il codice HTML del file education.component.html, in particolare la tabella per la visualizzazione dei corsi di formazione, coi relativi pulsanti per la modifica e la cancellazione.

Notiamo che in entrambe le figure (3.18) e (3.19), il codice HTML è complementato da alcuni tag di PrimeNG -come p-inputText, p-dropdown, p-calendar, p-table, p-sortableColumn, p-sortIcon, p-button-, ma ne parleremo più approfonditamente nella sezione (3.3.1).

Al fine di creare la tabella per la visualizzazione dei corsi di formazione esistenti, ho utilizzato il componente p-table di PrimeNG, in modo da visualizzare correttamente i dati dei corsi provenienti dal backend. In particolare, il template HTML è strettamente legato ai tag ng-template, i quali permettono di definire un template personalizzato per la visualizzazione dei dati, in base alle esigenze dello sviluppatore. Così facendo, risulta il codice di una tabella HTML classica, ma potenziata da PrimeNG e che comunica correttamente con il livello della logica di business. Un'altra funzionalità offerta da PrimeNG e usata nel codice soprastante è la possibilità di ordinare i dati della tabella, tramite il tag p-sortableColumn e p-sortIcon, i quali permettono di ordinare i dati in base a una colonna, in ordine crescente o decrescente, e di visualizzare la relativa icona che indica l'ordine corrente.

```

1 import ...
2 import { CoursesData } from 'src/app/redux/courses/courses.state';
3 import { selectCoursesData, selectCoursesDataFilters } from 'src/app/redux/courses/courses.selectors';
4 import { addCoursesData, changeCoursesDataFilters, deleteCoursesData, updateCoursesData } from 'src/app/redux/courses/courses.actions';
5 @Component({
6   selector: 'app-educationManagementComponent',
7   templateUrl: './educationManagementComponent.component.html',
8   styleUrls: ['./educationManagementComponent.component.scss'],
9 })
10 export class EducationManagementComponentComponent implements OnInit, OnDestroy{
11   coursesData$: Observable<CoursesData[]> = of([]);
12   public editedCourse: Partial<CoursesData> = {};
13   course: Partial<CoursesData> = { coursesName: '', coursesCapacity: null, coursesType: '', coursesDate: '' };
14   types: any = [ { types: 'Udemy' }, { types: 'Coursera' }, { types: 'brilliant' }, { types: 'UniGe' }, { types: 'Angular University' } ];
15   selectedType: any = null;
16   usersId: any[] = [ { userId: 'Utente 1' }, { userId: 'Utente 2' }, { userId: 'Utente 3' }, ];
17   selectedUsers: any[] = [];
18   showAddForm = true;
19   successAdd = false;
20   successUpdate = false;
21   expandedMap: { [key: number]: boolean } = {};
22   lastSearch$: Observable<string> = of(null);
23   coursesDataFilters$: Observable<DynamicQueryPart> = of({});
24   isLoading$: Observable<boolean>;
25   totalRecords$: Observable<number> = of(0);
26   globalFiltersFields = Object.keys(this.course);
27   filtersSubj$ = new Subject<Action>();
28   filtersSub: Subscription;
29   private firstQuery = true;
30   private _selectedCoursesData: CoursesData;
31   get selectedCoursesData() { return this._selectedCoursesData; }
32   set selectedCoursesData(value: CoursesData) { this._selectedCoursesData = value; }
33   constructor(
34     private store: Store<AppState>,
35     private route: ActivatedRoute
36   ){
37     this.coursesData$.subscribe((data) => {
38       if (data) { data.forEach((course) => { this.expandedMap[course.coursesId] = false; }); } });
39     this.isLoading$ = store.select(selectIsLoading).pipe(distinctUntilChanged());
40     this.coursesDataFilters$ = store.select(selectCoursesDataFilters);
41     this.filtersSub = this.filtersSubj$.asObservable().pipe(debounceTime(1000)).subscribe((a) => this.store.dispatch(a));
42     this.lastSearch$ = store.select(selectCoursesDataFilters).pipe(take(1), map((f) => f.filtering && f.filtering.length > 0 ? f.filtering[0]
43       => true : null));
44   }
45   addCourse(form: NgForm) {
46     if (this.selectedType) this.course.coursesType = this.selectedType.types;
47     if (this.selectedUsers.length > 0) this.selectedUsers.map((user) => user.userId);
48     this.store.dispatch( addCoursesData({ item: this.course, _id: getRandomId() }) );
49     this.resetFormFields(form);
50     this.successAdd = true;
51     this.successUpdate = false;
52   }
53   resetFormFields(form: NgForm) {
54     form.resetForm();
55     this.course = { coursesId: null, coursesName: '', coursesCapacity: null, coursesDate: null };
56     this.selectedType = null;
57     this.selectedUsers = [];
58   }
59   deleteCoursesData(course: CoursesData){ if (course) this.store.dispatch( deleteCoursesData({item: course, _id: getRandomId() }) ); }
60   updateCoursesData(form: NgForm){
61     if (this.selectedUsers.length > 0) this.selectedUsers.map((user) => user.userId);
62     if (this.selectedType) this.editedCourse.coursesType = this.selectedType.types;
63     this.store.dispatch( updateCoursesData({ item: this.editedCourse, _id: getRandomId() }) );
64     this.editedCourse = {};
65     this.resetFormFields(form);
66     this.successUpdate = true;
67     this.successAdd = false;
68     this.showAddForm = true;
69   }
70   editCoursesData(course: CoursesData) {
71     this.editedCourse = { ...course };
72     this.showAddForm = false;
73   }
74   async filter(value: string) {
75     ... //vedi figura successiva
76   }
77   async sort($event: LazyLoadEvent) {
78     ... //vedi figura successiva
79   }
80   ngOnInit(): void {
81     this.coursesData$ = this.store.select(selectCoursesData).pipe(startWith(this.route.snapshot.data.CoursesData));
82     this.coursesData$.pipe(
83       filter(data => !data)
84     ).subscribe(() => {
85       this.totalRecords$ = this.coursesData$.pipe(map((x) => (x ? (x[0] ? x[0].count : 0) : 0)));
86     });
87   }
88   ngOnDestroy(): void { this.filtersSub?.unsubscribe(); }
}

```

Figura 3.20: Il codice TypeScript del file education.component.ts.

```

1  async filter(value: string) {
2    const currentQueryParams = await firstValueFrom(this.coursesDataFilters$.pipe(take(1)));
3    const newQueryParams: DynamicQueryPart = {
4      ...currentQueryParams,
5      paging: currentQueryParams.paging
6        ? currentQueryParams.paging
7        : { skip: 0, take: environment.defaultNumberOfRowsPerPage },
8      filtering: value
9        ? [{ column: this.globalFiltersFields.join(','), predicate: 'LK', value, kind: 'STRING' } as Filtering]
10       : []
11    };
12    this.filtersSubj$.next(changeCoursesDataFilters({ queryParams: newQueryParams, _id: getRandomId() }));
13  }
14  async sort($event: LazyLoadEvent) {
15    if (this.firstQuery) {
16      this.firstQuery = false;
17      return;
18    }
19    const currentQueryParams = await firstValueFrom(this.coursesDataFilters$.pipe(take(1)));
20    const newQueryParams = {
21      ...currentQueryParams,
22      ordering: $event.sortField
23        ? [{ column: $event.sortField, columnPrefix: '', descending: $event.sortOrder > 0}]
24        : [],
25      paging: currentQueryParams.paging
26        ? ($event.first !== null || true)
27        : { skip: $event.first, take: $event.rows }
28        : currentQueryParams.paging
29        : { skip: 0, take: environment.defaultNumberOfRowsPerPage }
30    };
31    this.filtersSubj$.next(changeCoursesDataFilters({ queryParams: newQueryParams, _id: getRandomId() }));
32  }

```

Figura 3.21: I metodi filter e sort del file education.component.ts, i quali si occupano di filtrare e ordinare i corsi di formazione, listati qui per motivi di spazio rispetto alla figura precedente.

Come si può vedere dalle figure (3.20) e (3.21), il componente TS si occupa di gestire la totalità della logica di business del componente educationComponent, in particolare, le nozioni principali sono:

- alcuni import sono stati omessi, in quanto non rilevanti ai fini della spiegazione;
- gli import alle righe 2, 3 e 4 sono relativi ai file e ai servizi necessari per far funzionare correttamente la logica di business del componente;
- il componente implementa l’interfaccia OnInit e OnDestroy, in modo da poter eseguire del codice all’inizializzazione e alla distruzione del componente;
- il componente è dotato di variabili, le quali si occupano di gestire la totalità dei dati e delle azioni del componente, come ad esempio coursesData\$ e course, le quali si occupano rispettivamente di tenere in memoria un array dei corsi esistenti e di fissare i dati di un corso, in modo da poterla usare come puntatore per un ciclo di aggiunta o modifica di un corso;
- il metodo addCourse si occupa di raccogliere i dati del backend e iniettarli nel livello di presentazione;
- il metodo resetFormFields si occupa di resettare i campi del form, in modo da poter inserire nuovi dati;

- il metodo `deleteCoursesData` si occupa di cancellare un corso esistente, tramite il metodo TS apposito e una semplice query;
- il metodo `filter` si occupa di filtrare i dati dei corsi di formazione;
- il metodo `sort` si occupa di ordinare i dati dei corsi di formazione ricevuti dal backend, per poterli visualizzare correttamente nella tabella del livello di presentazione.

```

1 $gutter: 1rem; //for primeflex grid system
2
3 @import "assets/layout.scss";
4
5 @import "../node_modules/primeng/resources/primeng.min.css";
6 @import "../node_modules/primeflex/primeflex.scss";
7 @import "../node_modules/primeicons/primeicons.css";
8
9 @import "assets/overrides/styles/theme.scss";

```

Figura 3.22: Il codice CSS del file `styles.css`, il quale si occupa di gestire il CSS di tutti i componenti, importando temi e stili da PrimeNG.

In questo caso, il progetto, non concentrandosi sull'apprendimento del dettaglio di uno stile, ma sull'implementazione di un prototipo, ho deciso di usare un tema predefinito di PrimeNG, il quale è stato importato nel file `styles.css`, come si può vedere dalla figura (3.22). Nonostante la leggera perdita di personalizzazione, la scelta di usare un tema predefinito ha permesso di efficientare i tempi di sviluppo, in quanto non è stato necessario scrivere CSS da zero -spesso complicato-, ma solo importare un tema già pronto.

TODO spiegazione di `course.effects`, `course.actions`, `course.reducers`, `course.state` e `course.selectors` ALtri file degni di nota sono:

- `courses.actions.ts`, il quale contiene tutti gli export delle azioni relative ai corsi di formazione, come si vede dalla figura (3.23);
- `courses.effects.ts`, il quale si occupa principalmente di mappare le informazioni in entrata o in uscita dal componente, come si vede dalla figura (3.24);
- `courses.reducer.ts`, il quale contiene alcuni metodi ausiliari per il corretto funzionamento della logica di business, in particolare per filtrare o manipolare le informazioni dal o verso il backend, come si vede dalla figura (3.25);
- `courses.state.ts`, il quale contiene la struttura di base dello stato dei corsi di formazione, come si vede dalla figura (3.26);
- `courses.selectors.ts`, il quale gli export necessari per il funzionamento di ogni metodo filtro o di stato dei corsi di formazione, come si vede dalla figura (3.27).

```
1 import {createAction, props} from '@ngrx/store';
2 import {DynamicQueryPart, WithId} from '../state';
3 import {SuccessPayload, ErrorPayload} from '../message/message.actions';
4 import {CoursesData} from './courses.state';
5
6 export const searchCoursesData = createAction('[API] Search Courses Data', props<{queryParams: DynamicQueryPart} & WithId>());
7 export const searchCoursesDataSuccess = createAction('[API] Search Courses Data Success', props<SuccessPayload<{result: CoursesData[]} > & WithId>());
8 export const searchCoursesDataError = createAction('[API] Search Courses Data Error', props<ErrorPayload & WithId>());
9 export const changeCoursesDataFilters = createAction('[API] Change Courses Data Filters', props<{queryParams: DynamicQueryPart} & WithId>());
10
11 export const addCoursesData = createAction('[API] Add Courses Data', props<{item: Partial<CoursesData>} & WithId>());
12 export const addCoursesDataSuccess = createAction('[API] Add Courses Data Success', props<SuccessPayload<{result: CoursesData[]} > & WithId>());
13 export const addCoursesDataError = createAction('[API] Add Courses Data Error', props<ErrorPayload & WithId>());
14
15 export const deleteCoursesData = createAction('[API] Delete Courses Data', props<{item: Partial<CoursesData>} & WithId>());
16 export const deleteCoursesDataSuccess = createAction('[API] Delete Courses Data Success', props<SuccessPayload<{result: CoursesData[]} > & WithId>());
17 export const deleteCoursesDataError = createAction('[API] Delete Courses Data Error', props<ErrorPayload & WithId>());
18
19 export const updateCoursesData = createAction('[API] Update Courses Data', props<{ item: Partial<CoursesData>} & WithId>());
20 export const updateCoursesDataSuccess = createAction('[API] Update Courses Data Success', props<SuccessPayload<{result: CoursesData[]} > & WithId>());
21 export const updateCoursesDataError = createAction('[API] Update Courses Data Error', props<ErrorPayload & WithId>());
```

Figura 3.23: Il codice TypeScript del file courses.actions.ts.

```

1 import { Injectable } from '@angular/core';
2 import { ofType, createEffect, Actions } from '@ngrx/effects';
3 import { map, mergeMap, catchError } from 'rxjs/operators';
4 import { CustomHttpClient } from '../../services/custom-http-client.service';
5 import { environment } from '../../../../../environments/environment'; // src/environments/environment';
6 import { addCoursesData, addCoursesDataError, addCoursesDataSuccess, changeCoursesDataFilters, deleteCoursesData, deleteCoursesDataError,
7   deleteCoursesDataSuccess, searchCoursesData, searchCoursesDataFilters, searchCoursesDataError, searchCoursesDataSuccess, updateCoursesData, updateCoursesDataError,
8   updateCoursesDataSuccess,} from './courses.actions';
9 import { CoursesData } from './courses.state';
10 import { HttpErrorResponse } from '@angular/common/http';
11 import { of } from 'rxjs/internal/observable/of';
12
13 @Injectable()
14 export class CoursesEffect{
15
16   constructor(
17     private actions$: Actions,
18     private httpClient: CustomHttpClient
19   ){ }
20
21   _searchCoursesData = createEffect(
22     () => this.actions$.pipe(
23       ofType(searchCoursesData),
24       mergeMap(a => {
25         const response$ = this.httpClient.post<CoursesData[]>(`${environment.apiUrl}/courses/searchCoursesData`, a.queryParams, { responseType: 'json'});
26         return response$.pipe(
27           map(r => r ?? []),
28           map(r => searchCoursesDataSuccess({ result: r, _id: a._id })),
29           catchError((err: HttpErrorResponse, _) => {
30             return [searchCoursesDataError({ error: err.message, _id: a._id })];
31           })
32         );
33       })
34     )
35   _changeCoursesDataFilters = createEffect(
36     () => this.actions$.pipe(
37       ofType(changeCoursesDataFilters),
38       map(a => searchCoursesData({queryParams: a.queryParams, _id: a._id}))
39     )
40   )
41   _addCoursesData = createEffect(
42     () => this.actions$.pipe(
43       ofType(addCoursesData),
44       mergeMap(a => {
45         const response$ = this.httpClient.post<CoursesData[]>(`${environment.apiUrl}/courses/addCoursesData`, [a.item], { responseType: 'json'});
46         return response$.pipe(
47           map(r => r ?? []),
48           map(r => addCoursesDataSuccess({result: r, _id: a._id })),
49           catchError((err: HttpErrorResponse, _) => {
50             return [addCoursesDataError({error: err.message, _id: a._id})];
51           })
52         );
53       })
54     )
55   _deleteCoursesData = createEffect(
56     () => this.actions$.pipe(
57       ofType(deleteCoursesData),
58       mergeMap(a => {
59         const response$ = this.httpClient.delete<CoursesData[]>(`${environment.apiUrl}/courses/deleteCoursesData/${a.item.courseId}`, { responseType: 'json'});
60         return response$.pipe(
61           map(r => r ?? []),
62           map(r => deleteCoursesDataSuccess({result: r, _id: a._id })),
63           catchError((err: HttpErrorResponse, _) => {
64             return [deleteCoursesDataError({error: err.message, _id: a._id})];
65           })
66         );
67       })
68     )
69   _updateCoursesData = createEffect(() =>
70     this.actions$.pipe(
71       ofType(updateCoursesData),
72       mergeMap(a =>
73         this.httpClient.put<CoursesData[]>(`${environment.apiUrl}/courses/updateCoursesData`, [a.item], { responseType: 'json' })
74       ).pipe(
75         map(r => r ?? []),
76         map(r => updateCoursesDataSuccess({ result: r, _id: a._id })),
77         catchError((err: HttpErrorResponse, _) =>
78           of(updateCoursesDataError({ error: err.message, _id: a._id }))
79         )
80       )
81     )
82   );
83 }

```

Figura 3.24: Il codice TypeScript del file courses.effects.ts.

```

1 import { createReducer, on, Action } from "@ngrx/store";
2 import { CoursesData, initialCoursesDataState as initialCoursesDataState, initialCoursesDataFiltersState as initialCoursesDataFiltersState } from
   ↪ "./courses.state";
3 import { DynamicQueryPart, WithId } from "../state";
4 import { addCoursesDataError, addCoursesDataSuccess, changeCoursesDataFilters, deleteCoursesDataError, deleteCoursesDataSuccess, searchCoursesData
   ↪ searchCoursesDataSuccess, updateCoursesDataError, updateCoursesDataSuccess } from "./courses.actions";
5
6 const _coursesDataFiltersReducer = createReducer(
7   initialCoursesDataFiltersState,
8   on(changeCoursesDataFilters, (_, a) => ({...a.queryParams, _id: a._id})))
9 );
10 export function coursesDataFiltersReducer(state: DynamicQueryPart & WithId, action: Action): DynamicQueryPart & WithId{
11   return _coursesDataFiltersReducer(state, action);
12 }
13
14 const _coursesDataReducer = createReducer(
15   initialCoursesDataState,
16   on(searchCoursesDataSuccess, (_, a) =>
17     Object.assign([...a.result ? a.result : []], { _id: a._id})),
18   on(searchCoursesDataError, (_1, a) => Object.assign([], { _id: a._id})),
19 );
20
21 export function coursesDataReducer(state: CoursesData[] & WithId, action: Action): CoursesData[] & WithId{
22   return _coursesDataReducer(state, action);
23 }
24
25 const _addCoursesDataReducer = createReducer(
26   initialCoursesDataState,
27   on(addCoursesDataSuccess, (_, a) =>
28     Object.assign([...a.result ? a.result : []], { _id: a._id})),
29   on(addCoursesDataError, (_1, a) => Object.assign([], { _id: a._id})),
30 );
31
32 export function addCoursesDataReducer(state: CoursesData[] & WithId, action: Action): CoursesData[] & WithId{
33   return _addCoursesDataReducer(state, action);
34 }
35
36 const _deleteCoursesDataReducer = createReducer(
37   initialCoursesDataState,
38   on(deleteCoursesDataSuccess, (_, a) =>
39     Object.assign([...a.result ? a.result : []], { _id: a._id})),
40   on(deleteCoursesDataError, (_1, a) => Object.assign([], { _id: a._id})),
41 );
42
43 export function deleteCoursesDataReducer(state: CoursesData[] & WithId, action: Action): CoursesData[] & WithId{
44   return _deleteCoursesDataReducer(state, action);
45 }
46
47 const _updateCoursesDataReducer = createReducer(
48   initialCoursesDataState,
49   on(updateCoursesDataSuccess, (state, action) => ({ ...state, _id: action._id, coursesData: action.result })),
50   on(updateCoursesDataError, (state, action) => ({ ...state, _id: action._id, error: action.error }))
51 );
52
53 export function updateCoursesDataReducer(state: CoursesData[] & WithId, action: Action): CoursesData[] & WithId{
54   return _updateCoursesDataReducer(state, action);
55 }

```

Figura 3.25: Il codice TypeScript del file courses.reducer.ts.

```

1 import { DynamicQueryPart, WithId } from "../state";
2
3 export interface CoursesData {
4   coursesId: number;
5   coursesName: string;
6   coursesCapacity: number;
7   coursesType: string;
8   coursesDate: string;
9   count: number;
10 }
11
12 export const initialCoursesDataState: CoursesData[] & WithId = Object.assign([], { _id: '' });
13
14 export const initialCoursesDataFiltersState: DynamicQueryPart & WithId = { _id: '' };

```

Figura 3.26: Il codice TypeScript del file courses.state.ts.

```

1 import { createFeatureSelector, createSelector } from "@ngrx/store";
2 import { DynamicQueryPart, WithId } from "../state";
3 import { CoursesData } from "./courses.state";
4
5 export const selectCoursesDataState = createFeatureSelector<CoursesData[]>('coursesData');
6 export const selectCoursesDataFiltersState = createSelector<DynamicQueryPart>('coursesDataFilters');
7
8 export const selectCoursesData = createSelector(selectCoursesDataState, (state: CoursesData[] & WithId) => state);
9 export const selectCoursesDataFilters = createSelector(selectCoursesDataFiltersState, (state: DynamicQueryPart & WithId) => state);

```

Figura 3.27: Il codice TypeScript del file courses.selectors.ts.

Aggiuntivamente, dopo aver complementato il funzionamento dei file sopra citati, ho aggiunto un componente Angular per visualizzare un calendario visibile a larghezza intera, come si può vedere dalla figura (3.28) della pagina dell'applicazione web. Così facendo, l'utente base può visualizzare i corsi di formazione esistenti a colpo d'occhio, in modo da poter scegliere il corso di formazione più adatto alle proprie esigenze.

TODO aggiungere screen fullcalendar

Figura 3.28: Il calendario visibile a larghezza intera, il quale visualizza i corsi di formazione esistenti.

Dopo averlo installato tramite la CLI di Angular, ho dovuto configurare il file app.component.ts e app.module.ts, in modo da poterlo usare correttamente.

```

1 export class AppComponent implements OnInit {
2   calendarOptions: CalendarOptions = {
3     initialView: 'dayGridMonth',
4     plugins: [dayGridPlugin]
5   };
6 ...
7 }

```

Figura 3.29: Il codice TS del file app.module.ts, necessario per il funzionamento del calendario.

Di seguito, il codice HTML e TS per gestire la logica di business del componente fullcalendarComponent:

```
1 <full-calendar [options]="calendarOptions"></full-calendar>
```

Figura 3.30: Il codice HTML del file fullcalendar.component.html.

```

1  ngOnInit(): void {
2    this.coursesData$ = this.store.select(selectCoursesData).pipe(startWith(this.route.snapshot.data.CoursesData));
3    this.coursesData$.pipe(
4      filter(data => !!data)
5    ).subscribe((data) => {
6      this.totalRecords$ = this.coursesData$.pipe(map((x) => (x ? (x[0] ? x[0].count : 0) : 0)));
7
8      let tmp = [];
9      data.forEach((course) => {
10        let day:string = course.coursesDate[0] + course.coursesDate[1];
11        let month:string = course.coursesDate[3] + course.coursesDate[4];
12        let year:string = course.coursesDate[6] + course.coursesDate[7] + course.coursesDate[8] + course.coursesDate[9];
13
14        tmp.push({ title: course.coursesName, date: year + '-' + month + '-' + day }); // yyyy-mm-dd
15        console.log(tmp);
16      });
17
18      this.calendarOptions = {
19        ...this.calendarOptions,
20        events: tmp
21      };
22    })
23  }

```

Figura 3.31: Il codice TypeScript del file education.component.ts, necessario per il funzionamento della logica di business del calendario.

Infine, è stato necessario sviluppare i file mancanti per il completo funzionamento del progetto di prova finale, quindi i file per la gestione di tutti i componenti restanti, in modo da l'applicazione web potesse gestire correttamente la visualizzazione, la modifica, la ricerca, l'aggiunta e la cancellazione dei corsi di formazione da parte dell'admin e la visualizzazione dei corsi di formazione da parte dell'utente base. Inoltre, in fase finale del tirocinio, ho aggiunto un componente per la gestione delle iscrizioni da parte degli utenti base. Le figure dei codici, dei file sopra citati, sono omesse, in quanto sono consone al codice già mostrato e descritto in precedenza.

3.3.1 PrimeNG

Grazie a PrimeNG, ho potuto implementare una GUI funzionale e minimalistica. Innanzitutto, è stato necessario installare i giusti pacchetti e le giuste dipendenze, tramite CLI, successivamente, la scelta dei componenti utilizzati e descritti in precedenza, è stata fatta in base alle esigenze del progetto e alle funzionalità offerte da PrimeNG. In particolare, i componenti usati sono:

- p-table, per la visualizzazione dei corsi di formazione;
- p-sortableColumn e p-sortIcon, per l'ordinamento dei dati della tabella;
- p-inputText, p-dropdown, p-calendar, per la gestione dei campi di input e delle date;
- p-button, per la gestione dei pulsanti di modifica e cancellazione;

- p-fullCalendar, per la visualizzazione dei corsi di formazione esistenti in un calendario.
- ngtemplate, per la gestione dei template personalizzati.

Di seguito, un esempio di come è stato usato il componente ng per la visualizzazione dei corsi di formazione:

```

1  <ng-template pTemplate="body" let-coursesData>
2      <tr class="p-selectable-row" [pRowToggler]="coursesData.coursesId">
3          <td>
4              <button type="button" pButton pRipple
5                  class="p-button-text p-button-rounded p-button-plain"
6                  [icon]="isRowExpanded(coursesData.coursesId) ? 'pi pi-chevron-down' : 'pi pi-chevron-right'"
7                  (click)="toggleRow(coursesData.coursesId)">
8              </button>
9          </td>
10         <td>
11             <span class="p-column-title">Nome corso</span>
12             {{coursesData.coursesName}}
13         </td>
14         <td>
15             <span class="p-column-title">Tipologia</span>
16             {{coursesData.coursesType}}
17         </td>
18         <td>
19             <span class="p-column-title">Data inizio</span>
20             {{coursesData.coursesDate}}
21         </td>
22         <td>
23             <span class="p-column-title">Prenota</span>
24             <p-button icon="pi pi-check" (click)="addEnrollment(coursesData);"></p-button>
25         </td>
26     </tr>
27
28     <tr>
29     </tr>
30
31     <tr *ngIf="isRowExpanded(coursesData.coursesId)">
32         <td [attr.colspan]="globalFiltersFields.length + 1">
33             <div class="expanded-content">
34                 <p>immagine corso scelta con switch</p>
35             </div>
36         </td>
37     </tr>
38
39 </ng-template>
```

Figura 3.32: Il codice HTML del file education.component.html, in particolare la tabella di PrimeNG.

3.3.2 Servizi Angular

I servizi Angular sono tra le funzionalità più potenti e flessibili del medesimo framework, in quanto permettono di creare e gestire la logica di business in modo efficiente e modulare. Durante questo tirocinio di prova finale, sono stati usati al fine di gestire la comunicazione tra il livello di presentazione e il livello della logica di business. Inoltre, sono stati scelti per la loro capacità di essere iniettati in qualsiasi componente. Di seguito, un esempio di come è stato usato un servizio Angular per la gestione dei corsi di formazione:

```
1  @Injectable({ providedIn: "root" })
```

Figura 3.33: Il codice TS del file course.resolver.ts, contenente l'implementazione di un servizio Angular.

```
1  @Injectable({ providedIn: "any" })
```

Figura 3.34: Il codice TS del file admin-user-guard.ts, contenente l'implementazione di un servizio Angular.

Capitolo 4

Conclusioni e ringraziamenti

4.1 Conclusioni

Concludendo, l'utilizzo a 360° degli strumenti e delle tecnologie riportare è stato essenziale per la realizzazione di questo progetto di prova finale.

Dopo aver studiato e approfondito l'architettura del modello 3-tier e la documentazione necessaria per lo sviluppo, posso ritenermi soddisfatto dell'esperienza svolta in questo tirocinio. È stata un'ottima occasione per approfondire le mie capacità in alcuni linguaggi e impararne altri.

L'applicazione web finale è risultata moderna e intuitiva, al possa coi framework odierni.

Il portale di sviluppo, fornito da Gruppo SIGLA, mi ha dato l'opportunità di esplorare meglio le dinamiche lavorative nel contesto della programmazione di applicazione web, seguendo l'architettura del modello 3-tier.

4.2 Ringraziamenti

Ringrazio tutte le persone che mi sono state accanto in questi anni e percorso universitario e non, in particolare: la mia famiglia per avermi supportato in ogni modo possibile;

i miei amici, sia quelli universitari che quelli di sempre, per le risate e i momenti passati insieme;

i miei colleghi di lavoro, per avermi aiutato a crescere professionalmente e essere stati sempre disponibili nei miei confronti;

i miei colleghi universitari, per avermi aiutato a superare gli esami e per avermi supportato in questo percorso, dalle sessioni di studio più intense, alle serate passate insieme;

i miei professori, per aver confermato la mia passione per l'informatica, per avermi trasmesso ulteriore fame di conoscenza in questo ambito e per essere stati disponi-

bili in ogni momento: dai dubbi più banali, alle richieste di chiarimenti più complesse e ai consigli riguardanti la mia carriera universitaria;

Infine, ringrazio la mia relatrice, per avermi accompagnato in questa prova finale di tesi, per avermi aiutato a superare gli ultimi ostacoli e dubbi e per avermi consigliato in ogni momento.

Senza la vostra conoscenza, presenza e supporto, non sarei riuscito a raggiungere questo traguardo.

Grazie di cuore.

Elenco delle figure

2.1	Schema di un'applicazione a tre livelli.	3
2.2	Schema teorico dell'architettura MVC.	8
2.3	Schema pratico dell'architettura MVC.	9
2.4	Schema dei framework utilizzati.	10
2.5	Angular e PrimeNG, due framework.	11
2.6	Schematica di Angular.	14
2.7	Schema dell'intradamento di Angular.	15
2.8	Diamond, uno dei template di PrimeNG.	17
2.9	Apollo, uno dei template di PrimeNG..	18
2.10	Omega, uno dei template di PrimeNG.	18
2.11	Il logo di TypeScript.	19
2.12	Inizializzazione del classico ‘Ciao Mondo’.	19
2.13	Il logo di .NET Framework.	22
2.14	Il logo di .NET Core.	23
2.15	Il logo di .NET 8.	23
2.16	La pila tecnologica dei componenti di .NET (2005/2012).	24
2.17	Esempio di una semplice applicazione web ASP.NET.	26
2.18	Le principali tecnologie Microsoft .NET.	27
2.19	Logo di C#.	27
3.1	Il logo di DBeaver.	30
3.2	Il codice in C# del file CoursesData.cs.	32
3.3	Il codice in C# del file 4_CoursesMigration.cs.	33
3.4	Il codice in C# del file CoursesController.cs.	35
3.5	Il codice in C# della classe di CoursesDataExecuteExecutor.cs, successivamente diviso in blocchi di codice per motivi di spazio.	36
3.6	Il codice di CoursesDataExecuteExecutor.cs, relativo alla scelta del metodo e all'aggiunta delle tuple.	37
3.7	Il codice di CoursesDataExecuteExecutor.cs, relativo alla cancellazione delle tuple.	37

3.8 Il codice di CoursesDataExecuteExecutor.cs, relativo all'aggiornamento delle tuple.	37
3.9 Il codice di CoursesDataGetListExecutor.cs, classe contenente il metodo GetList ().	38
3.10 Il logo di Swagger.	39
3.11 L'interfaccia grafica di Swagger, relativa al progetto di tirocinio.	40
3.12 Esempio di esecuzione di una API per searchCoursesData, il corpo della richiesta è in formato JSON.	40
3.13 Alcuni esempi di risultati di esecuzione di API tramite Swagger, in ordine 200, Successo, 400, ‘Cattiva richiesta’, 401, ‘Non autorizzato’, 403, ‘Vietato’ e 500, ‘Errore interno del server’.	41
3.14 La struttura della cartella education, creata tramite la CLI di Angular. .	43
3.15 Tutti i componenti e i servizi importati nel file app.module.ts, anche quelli che verranno descritti successivamente.	43
3.16 L'array declarations del modulo NgModule del file app.module.ts, con l'aggiunta dei componenti sviluppati per il progetto di prova finale. . .	44
3.17 Il codice del file app-routing.module.ts, il quale si occupa di instradare l'applicazione, a seconda dell'utente e della ‘guardia’ attivata.	44
3.18 Il codice HTML del file education.component.html, in particolare il form per l'aggiunta e la modifica dei corsi di formazione.	46
3.19 Il codice HTML del file education.component.html, in particolare la tabella per la visualizzazione dei corsi di formazione, coi relativi pulsanti per la modifica e la cancellazione.	48
3.20 Il codice TypeScript del file education.component.ts.	50
3.21 I metodi filter e sort del file education.component.ts, i quali si occupano di filtrare e ordinare i corsi di formazione, listati qui per motivi di spazio rispetto alla figura precedente.	51
3.22 Il codice CSS del file styles.css, il quale si occupa di gestire il CSS di tutti i componenti, importando temi e stili da PrimeNG.	52
3.23 Il codice TypeScript del file courses.actions.ts.	53
3.24 Il codice TypeScript del file courses.effects.ts.	54
3.25 Il codice TypeScript del file courses.reducer.ts.	55
3.26 Il codice TypeScript del file courses.state.ts.	55
3.27 Il codice TypeScript del file courses.selectors.ts.	56
3.28 Il calendario visibile a larghezza intera, il quale visualizza i corsi di formazione esistenti.	56
3.29 Il codice TS del file app.module.ts, necessario per il funzionamento del calendario.	56

3.30 Il codice HTML del file fullcalendar.component.html.	56
3.31 Il codice TypeScript del file education.component.ts, necessario per il funzionamento della logica di business del calendario.	57
3.32 Il codice HTML del file education.component.html, in particolare la tabella di PrimeNG.	58
3.33 Il codice TS del file course.resolver.ts, contenente l'implementazione di un servizio Angular.	58
3.34 Il codice TS del file admin-user-guard.ts, contenente l'implementazione di un servizio Angular.	59