



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

# Indice

<b>Acronimi</b>	<b>ii</b>
<b>1 Introduzione</b>	<b>1</b>
<b>2 Contesto di riferimento</b>	<b>2</b>
2.1 Modello three-tier . . . . .	2
2.2 Framework utilizzati . . . . .	5
2.2.1 Angular e PrimeNG . . . . .	6
2.2.2 Typescript . . . . .	10
2.2.3 .NET Framework, .NET Core e .NET 8 . . . . .	11
2.2.4 ASP.NET . . . . .	12
2.2.5 Swagger . . . . .	13
<b>3 Prototipo</b>	<b>15</b>
3.1 Database . . . . .	16
3.2 Back-end . . . . .	17
3.3 Front-end . . . . .	25
3.3.1 PrimeNG . . . . .	33
3.3.2 Servizi Angular . . . . .	34
<b>4 Conclusioni</b>	<b>36</b>
4.1 Ringraziamenti . . . . .	36
<b>Elenco delle figure</b>	<b>38</b>
<b>Bibliografia</b>	<b>40</b>

# Acronimi

**API** Application Programming Interface. 14, 16, 23–25, 38

**ASP.NET** Active Server Pages.NET. 12–14, 16, 17

**CLI** Command Line Interface. 9, 26, 27, 31, 39

**CLR** Common Language Runtime. 12

**CSS** Cascading Style Sheets. 28, 30, 39

**DB** database. 3, 4

**DBMS** Database Management System. 4

**DOM** Document Object Model. 7

**FCL** Framework Class Library. 11, 12

**GUI** Graphical User Interface. 3

**HTML** HyperText Markup Language. 28, 29, 31, 39

**HTTP** HyperText Transfer Protocol. 19

**I/O** Input/Output. 12

**IDE** Integrated Development Environment. 13

**JS** JavaScript. 4, 6, 7, 10, 11

**JSON** JavaScript Object Notation. 24, 38

**MVC** Model-View-Controller. 4, 5, 7, 12, 38

**NPM** Node Package Manager. 25

**NVM** Node Version Manager. 25, 26, 38

**ORM** Object-Relational Mapping. 16, 18

**PrimeNG** PrimeFaces Next Generation. 6, 9, 30, 33, 34

**SQL** Structured Query Language. 3

**TS** TypeScript. 4, 6, 7, 10, 11, 28, 29, 32–35, 39

**UI** User Interface. 6, 9

**URL** Uniform Resource Locator. 14, 26

# Capitolo 1

## Introduzione

Tutte le aziende, in particolare quelle di una certa complessità, hanno la 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 dipendenti 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 dispositivi mobili per l'accesso ai servizi aziendali.

L'azienda presso la quale ho svolto il mio tirocinio è Gruppo SIGLA, un'azienda di più di 100 persone, di cui molte impiegate in via temporanea presso clienti in Italia e all'estero. Gruppo SIGLA 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 ed 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, ed è sviluppato 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.

Questo documento descrive la mia attività di tirocinio ed è organizzato nei seguenti capitoli. Il Capitolo 2 presenta il contesto di riferimento, in particolare si parla del progetto scelto, 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 e introduce le scelte fatte per la sua implementazione. 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 framework<sup>1</sup> e gli strumenti già esistenti [1] [2], delineando le ragioni dietro la loro scelta e illustrando come si integrano per supportare l'intera struttura dell'applicazione.

### 2.1 Modello three-tier

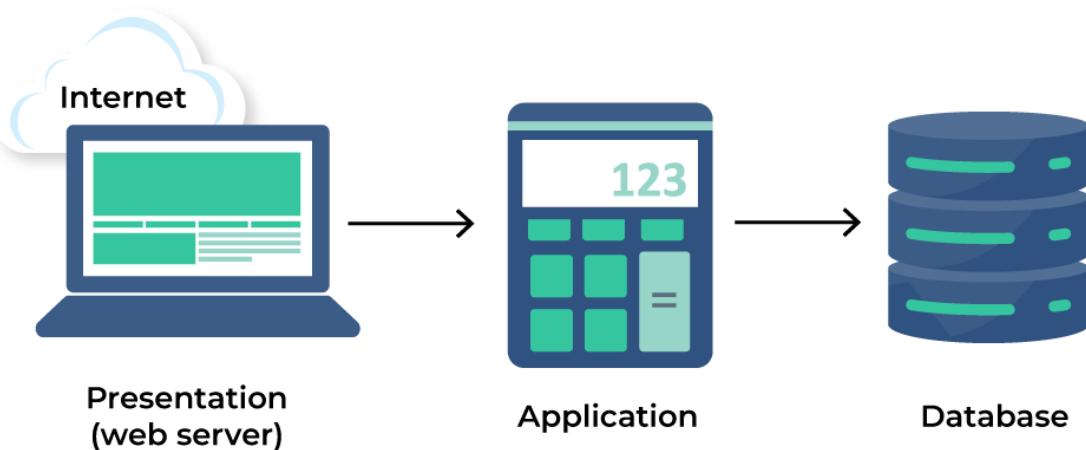


Figura 2.1: Schema di un'applicazione a tre livelli.

Il modello three-tier (a tre livelli) è un'architettura software consolidata, di tipo client-server, in cui l'interfaccia utente, l'elaborazione logica e la gestione dei dati sono svil-

<sup>1</sup>In informatica, un framework è un'architettura logica di supporto su cui un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore.

luppati 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 consiste nel fatto che i livelli sono sviluppati e mantenuti in modo indipendente, con la conseguenza che, in caso di qualsiasi modifica, viene coinvolto solamente il livello modificato. 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 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, su un'applicazione desktop o su un'interfaccia grafica (GUI), a seconda delle esigenze del progetto. Le applicazioni del livello di presentazione, nel caso del web, sono solitamente sviluppate utilizzando HTML, CSS e JavaScript mentre, nel caso delle applicazioni desktop, possono essere scritte in un'ampia varietà di linguaggi a seconda della piattaforma. Recentemente i framework per lo sviluppo front-end, come Angular, React e Vue.js, sono diventati sempre più popolari. In questo progetto di tirocinio è stato scelto il framework Angular, come verrà spiegato più approfonditamente nella sezione 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 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 query<sup>2</sup> 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

---

<sup>2</sup> Una richiesta di informazioni o di azioni da parte di un database; solitamente scritta in un linguaggio di interrogazione, come ad esempio SQL.

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 API<sup>3</sup> (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, detto anche livello di accesso ai dati (database) o back-end, le informazioni elaborate dall'applicazione vengono archiviate e gestite, in modo da 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.



Figura 2.2: Logo di MySQL.

L'architettura a tre livelli viene spesso paragonata all'architettura Model-View-Controller (MVC): 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 dell'applicazione in diversi livelli, specificando una comunicazione ben definita tra di loro, come già discusso in precedenza.

Invece, l'architettura MVC si concentra di più sulla separazione dell'applicazione all'interno di un livello in tre componenti distinti: Modello, Vista e Controllore, come si può vedere nella figura 2.3. 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'input dall'utente, a interagire con il Modello per elaborare i dati e ad aggiornare la Vista per visualizzarne i risultati.

---

<sup>3</sup>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.

## MVC Architecture

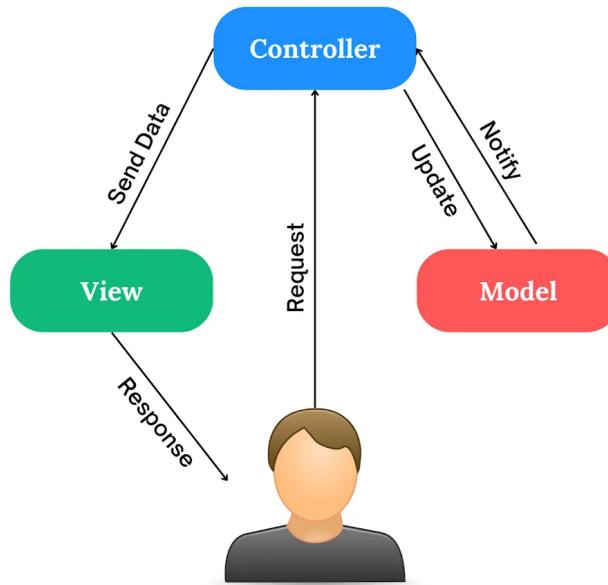


Figura 2.3: Schema astratto dell'architettura MVC.

## 2.2 Framework utilizzati

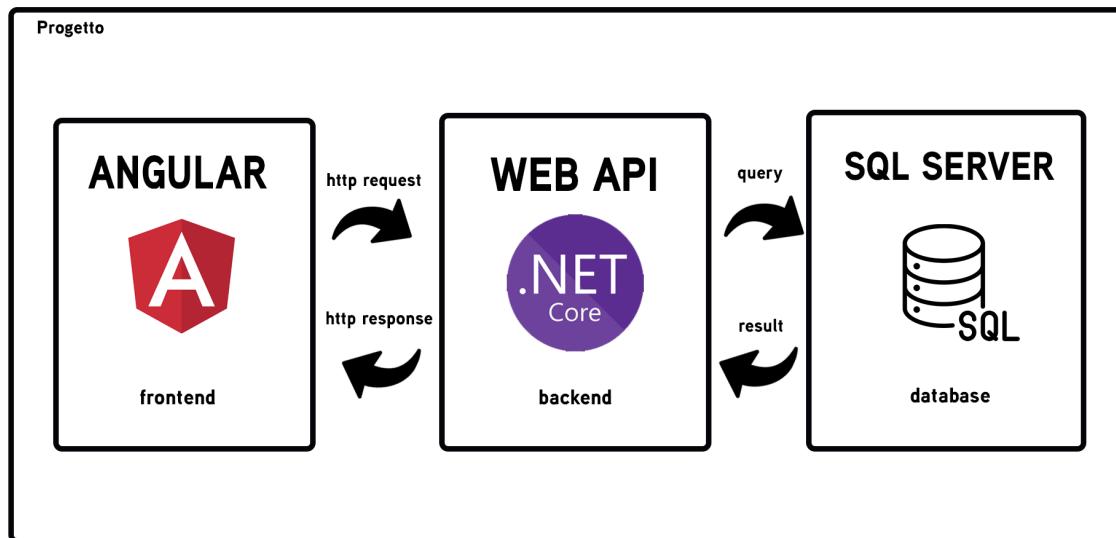


Figura 2.4: Schema dei framework utilizzati.

Un framework è un'astrazione che unisce codice comune tra progetti software con scopi simili, fornendo una funzionalità generale, che può essere modificata o estesa

per risolvere un problema specifico. Più in generale, i framework per il web, incrementano l'efficienza e le prestazioni dello sviluppo, fornendo una struttura coerente, affinché gli sviluppatori non debbano continuamente scrivere il codice partendo da zero. I framework sono quindi ottimi quando si tratta di risparmiare tempo, perché offrono ai programmati una serie di funzionalità extra che possono essere aggiunte al software, senza richiedere molto sforzo.

Nel caso di questo progetto di tirocinio, lo scopo principale è stato quello di implementare nuovi componenti e funzionalità per un'applicazione web già esistente, utilizzando i framework già adottati nel progetto stesso, senza dover scrivere il codice partendo 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 repository dell'intero progetto è composta da più di 55 mila file, con il rischio di perdersi nella moltitudine di elementi. È stato comunque possibile integrare le nuove funzionalità grazie ai framework di sviluppo utilizzati, sia per il back-end che per il front-end.

### 2.2.1 Angular e PrimeNG

Questa sezione descrive uno dei framework per il front-end più utilizzati al mondo, Angular [3], insieme al relativo framework open source<sup>4</sup> PrimeNG [4], che fornisce uno dei set di componenti UI più completi per Angular.



Figura 2.5: Logo di Angular.



Figura 2.6: Logo di PrimeNG.

Angular, denominato anche Angular 2+ (ossia una versione più moderna di AngularJS), è un framework JavaScript scritto e basato su TypeScript, per lo sviluppo delle interfacce delle applicazioni. Sviluppato principalmente dal team Angular di Google e da una comunità di programmati e aziende esterne, è una riscrittura completa del framework originale (AngularJS). Attualmente, Angular è mantenuto da Google.

In quanto framework, Angular fornisce una struttura standard per gli sviluppatori. Consente di creare applicazioni di grandi dimensioni, facilmente mantenibili.

La principale funzionalità di Angular è quella di fornire blocchi per costruire le applicazioni web, denominati componenti, grazie ai quali gli sviluppatori possono im-

<sup>4</sup>Un 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.

postare rapidamente applicazioni web scalabili e mantenibili. I programmati hanno la libertà di sviluppare applicazioni che possono essere eseguite su più piattaforme, inclusi dispositivi mobili, desktop e web. Essendo progettato per essere modulare, permette di integrare o rimuovere facilmente le funzionalità di cui si ha bisogno o meno.

Alcune caratteristiche **vantaggiose** di Angular sono:

- Document Object Model: Angular sfrutta il DOM per gestire le pagine web.
- TypeScript (TS): 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. TypeScript è altamente consigliato, ma non vincolante per sviluppare con Angular.
- Data Binding: 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 questa contiene una grande quantità di dati. Nel caso di Angular viene utilizzato un data binding bidirezionale, infatti lo stato del modello riflette eventuali modifiche apportate agli elementi dell'interfaccia corrispondente. Al contrario, lo stato della Vista riflette eventuali modifiche apportate allo stato del Modello e questa funzionalità consente al framework di collegare il DOM ai dati del Modello attraverso il Controllore.
- Testing: Angular usa un framework denominato Jasmine per il test delle applicazioni, ma non sarà oggetto di discussione in questo documento.

Passando all'architettura, è importante ricordare che Angular è un framework MVC a tutti gli effetti e offre una guida chiara su come strutturare applicazioni web.

I blocchi fondamentali di un'applicazione Angular sono sei, introdotti successivamente:

1. Moduli: una web app Angular possiede un modulo root (denominato AppModule) che fornisce il meccanismo di bootstrap<sup>5</sup> per avviare l'applicazione. Ogni modulo può avere uno o più moduli figlio.

---

<sup>5</sup> In informatica, il boot (o bootstrap, o più raramente booting) o avvio è, in generale, l'insieme dei processi che vengono eseguiti da un computer dall'accensione fino al completo caricamento in memoria primaria del programma.

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.
3. Modelli (o Template): contengono il codice della struttura iniziale dell'applicazione, in figura 2.7 si può vedere un esempio di template Angular;
4. Metadati: comunicano ad Angular come processare e decorare le classi, 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 introdotta con il decoratore '@Injectable'.
6. Dependency Injection: pattern di progettazione che permette di mantenere le classi dei componenti precise ed 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 [5].

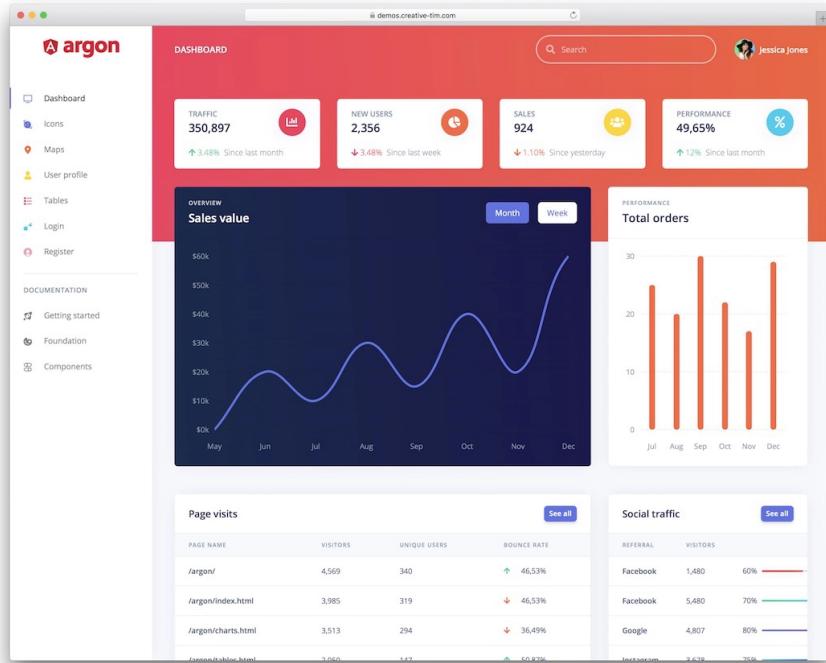


Figura 2.7: Argon, un template Angular.

Angular ha anche degli **svantaggi**, tra cui ricordiamo una ripida curva di apprendimento, capacità di Search Engine Optimization limitate, difficoltà nelle migrazioni alle versioni successive e verbosità.

Una ulteriore funzionalità degna di nota di Angular è la sua Command Line Interface (CLI), che 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. Tra le aziende più famose che utilizzano Angular ci sono Google, Microsoft, Nike, Forbes, Upwork, HBO, Sony, General Motors, etc.

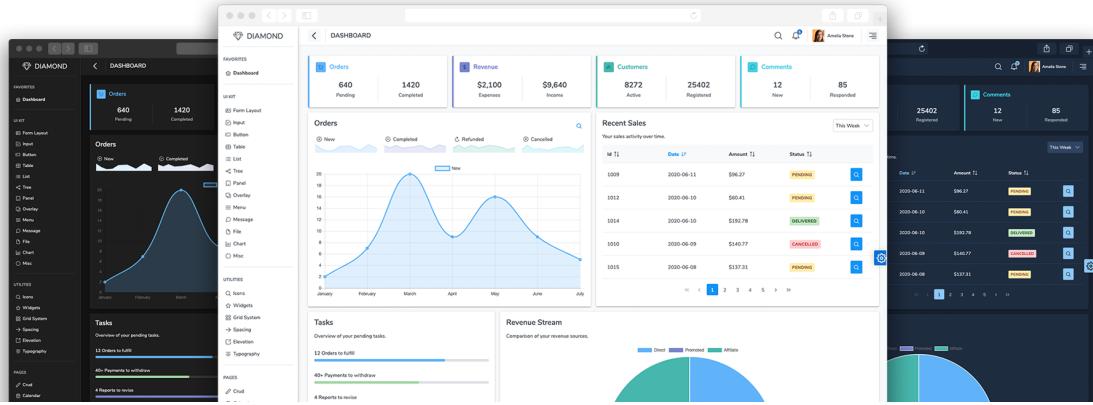


Figura 2.8: Diamond, uno dei template di PrimeNG.

PrimeFaces Next Generation (PrimeNG) è una collezione ricca di componenti UI per Angular, i cui widget sono open source e gratuiti. Offre oltre 80 componenti UI, come ad esempio tabelle, grafici, dialoghi, etc. PrimeNG supporta inoltre vari temi, permettendo agli sviluppatori di scegliere tra una vasta gamma. La figura 2.8 mostra un esempio di template.

Molte 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 le aziende che usano PrimeNG troviamo Ericsson, Siemens, Lufthansa, UniCredit, Ford, Volvo, Nvidia, eBay, Mercedes-Benz, Audi, HP, Scania, Cisco, Nike, Volkswagen.

## 2.2.2 Typescript

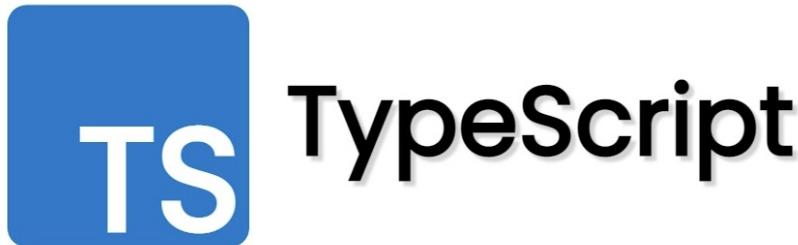


Figura 2.9: Logo di TypeScript.

TypeScript (TS) [6] è un linguaggio di programmazione open source sviluppato e mantenuto da Microsoft. È un superset di JavaScript, infatti offre tutte le funzionalità di quest'ultimo, aggiungendone di nuove e più avanzate, come la possibilità di tipare staticamente il linguaggio. Nello specifico, siccome TypeScript ‘conosce’ il linguaggio JavaScript [7], in molti casi genererà i tipi inferendoli<sup>6</sup> dal codice. Per esempio, inizializzando una variabile con un valore arbitrario, TypeScript userà il valore come tipo (Figura 2.10).

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

Figura 2.10: Inizializzazione del classico ‘Hello World!’.

Sapendo come funziona JavaScript, TypeScript può costruire un sistema di tipi che accetta il codice JavaScript. Infatti, è così che TypeScript è in grado di capire che ‘helloWorld’ è una stringa nell’esempio in figura 2.10. Visual Studio Code, sviluppato da Microsoft, usa TypeScript per facilitare l’esperienza della scrittura del codice.

<sup>6</sup>L’inferenza è un processo di deduzione logica che permette di ottenere nuove informazioni a partire da quelle già note.

Inoltre, TypeScript è stato progettato per lo sviluppo di applicazioni di grandi dimensioni e viene transcompilato<sup>7</sup> in JavaScript.

Pubblicato nel 2012 da Microsoft, voleva migliorare JavaScript in modo che gli sviluppatori potessero lavorare su applicazioni di grandi dimensioni, cosa difficile con JavaScript. Importanti e numerose sono le funzionalità introdotte da TypeScript:

- Controllo di tipo statico: TypeScript introduce il controllo di tipo statico, che può rilevare errori a tempo di compilazione, piuttosto che durante l'esecuzione.
- Oggetti e classi: una delle caratteristiche principali della programmazione orientata agli oggetti.
- Moduli: possono aiutare ad organizzare e incapsulare il codice.
- Strumenti avanzati: autocompletamento, navigazione e refactoring<sup>8</sup>.
- Scalabilità: grazie a funzionalità quali tipi, classi e moduli, TypeScript è particolarmente adatto per applicazioni di grandi dimensioni.

Tra le aziende più famose che utilizzano TypeScript ricordiamo Google, Slack, Medium, Doordash, bitpanda e Techstack.

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



Figura 2.11: Logo di .NET Framework.

.NET [8] è un framework software sviluppato da Microsoft, che include una vasta libreria di classi, denominata Framework Class Library e fornisce interoperabilità tra diversi linguaggi di programmazione [9]. Inizialmente rilasciato nel 2002 [10], .NET nasce come software proprietario, ma negli anni successivi Microsoft ha rilasciato le

<sup>7</sup>Compilare 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.

<sup>8</sup>Nell'ingegneria del software, il refactoring è una tecnica strutturata per modificare la struttura interna di porzioni di codice senza modificarne il comportamento esterno.

versioni open source. .NET [11] è usato da milioni di sviluppatori in tutto il mondo per creare applicazioni per una moltitudine di piattaforme e dispositivi.

Successivamente a .NET Framework, Microsoft ha rilasciato .NET Core, una versione più leggera e veloce della precedente, che può essere eseguita su Windows, Linux e macOS, grazie a una nuova architettura modulare e flessibile. Infatti, è possibile scegliere direttamente i componenti di cui si necessita e implementarli subito, senza dover installare il framework completo. .NET ha una vasta comunità di sviluppatori che contribuiscono alla creazione di strumenti, librerie e documentazione, il che rende più semplice per gli sviluppatori ottenere aiuto e trovare soluzioni ai problemi.

In breve, i due elementi principali dell'architettura di .NET sono il Common Language Runtime (CLR) e la Framework Class Library (FCL).

La FCL fornisce l'interfaccia grafica, l'accesso ai dati, connettività ai database, crittografia, supporto per lo sviluppo di applicazioni web, algoritmi numerici e funzioni per le comunicazioni di rete.

Il CLR, invece, è il ‘motore’ dell’esecuzione di .NET: 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 supervisione del CLR.

Inoltre, tra le varie funzionalità di .NET, troviamo anche una vasta libreria per problemi comuni di programmazione e una macchina virtuale che gestisce l’esecuzione di programmi scritti specificatamente per .NET.

## 2.2.4 ASP.NET

Una delle principali tecnologie di .NET è Active Server Pages.NET (ASP.NET) [12], utilizzata ampiamente in questo progetto di prova finale. Anche ASP.NET è un framework di sviluppo web. Attualmente, la versione più recente è la 4.8.1, rilasciata nel 2022. [13]

Le funzionalità chiave di ASP.NET includono [14]:

- Model-View-Controller (MVC): già discusso nella sezione 2.1;
- C#<sup>9</sup>: uno degli aspetti distintivi di ASP.NET è la sua integrazione con il linguaggio di programmazione C# per la programmazione lato server. ASP.NET sfrutta a pieno le capacità di C#;

---

<sup>9</sup>Un 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 di programmazione orientata ai componenti.

- Multi-piattaforma: ASP.NET offre ampio supporto per Windows, Linux e macOS e facilita l'adozione di pratiche di sviluppo moderne, come le architetture a microservizi e la containerizzazione<sup>10</sup> [15];
- Supporto Integrated Development Environment (IDE): l'ambiente di sviluppo ASP.NET è integrato con Visual Studio.

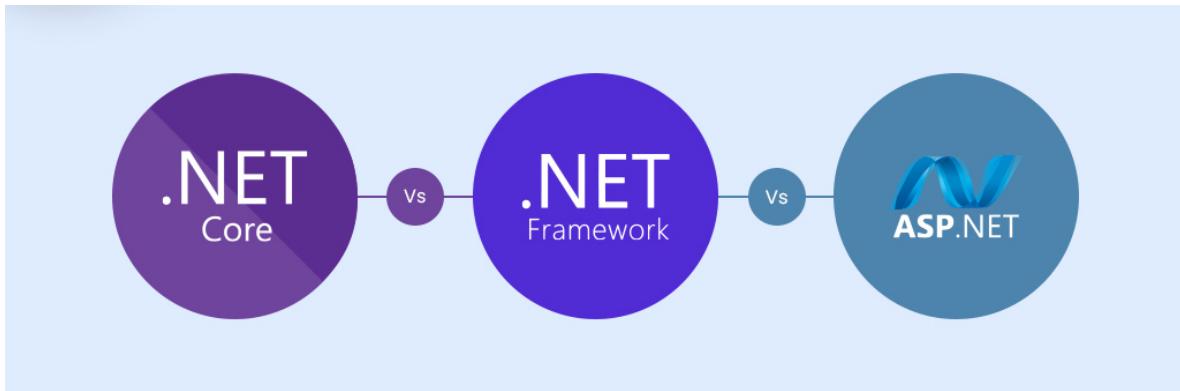


Figura 2.12: Principali tecnologie Microsoft .NET.

Concludendo, ASP.NET offre un framework completo e versatile per lo sviluppo web. 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.

## 2.2.5 Swagger

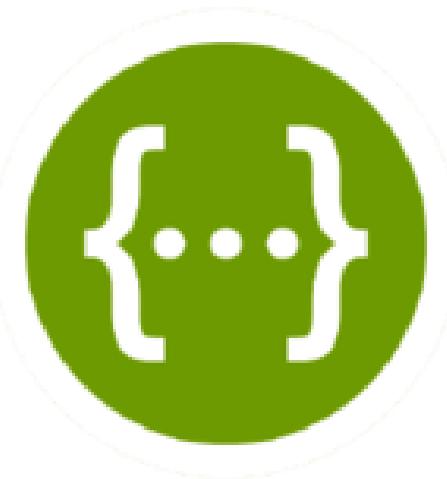


Figura 2.13: Logo di Swagger.

<sup>10</sup>Dall'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.

Swagger è un insieme di strumenti per sviluppatori di API, originariamente specificato e sviluppato da SmartBear Software. [16] È stato usato per testare il corretto funzionamento del back-end. Il suo vantaggio principale sta nella possibilità di essere utilizzato direttamente dal browser [17], senza dover installare alcun software aggiuntivo: è sufficiente digitare un indirizzo URL apposito 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 come si può vedere in figura 2.14.

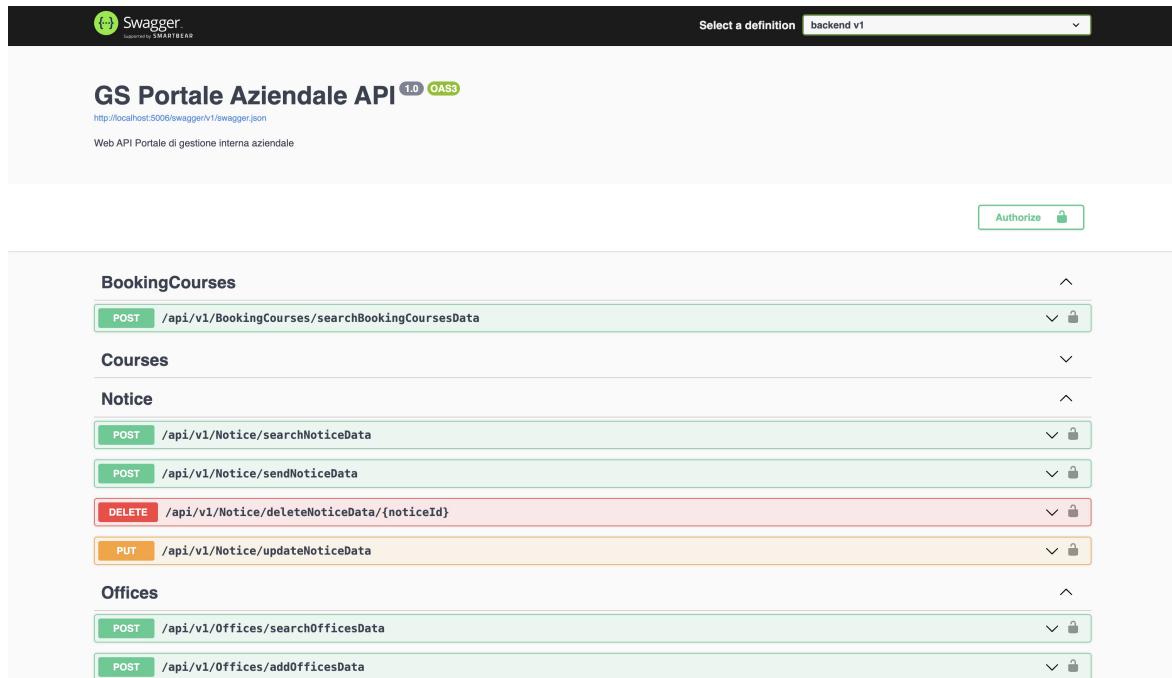


Figura 2.14: Interfaccia grafica di Swagger, relativa al progetto di tirocinio.

# Capitolo 3

## Prototipo

Questo capitolo descrive il progetto di prova finale realizzato durante il tirocinio, presso Gruppo SIGLA. In particolare, vengono discussi i tre livelli di sviluppo di un'applicazione web (il database, il back-end e il front-end) e le scelte progettuali fatte per realizzare il prototipo.

La parte di autenticazione all'applicazione web è stata gestita da Gruppo SIGLA tramite un sistema di autenticazione di terze parti, in modo da poter gestire gli accessi e le autorizzazioni degli utenti in modo centralizzato, usando i loro server esterni. Con questa decisione, il progetto non ha avuto la necessità di sviluppare login e registrazione degli utenti, ma si è concentrato sulle funzionalità di tracciamento e feedback delle attività formative erogate dall'azienda.

The screenshot shows a web application interface for managing courses. At the top, there is a navigation bar with a SAKAI logo, a main menu, and a user profile icon. The main content area is divided into two sections: 'Gestione corsi (Admin)' and 'Elenco corsi creati'.  
**Gestione corsi (Admin):** This section contains a form for adding a new course. It includes fields for 'Nome corso', 'Numero postazioni', 'Selezione tipologia corso', and 'Data inizio corso'. A 'Salva' button is located next to the date field. Below the form is a green 'Aggiungi corso' button.  
**Elenco corsi creati:** This section displays a table of created courses. The columns are 'Nome corso', 'Partecipanti max', 'Tipologia', 'Data inizio', 'Modifica', and 'Elimina'. Two entries are listed:

- Angular Modulo 1: 10 participants, Udemy, 01/01/2024. Actions: Modifica (blue edit icon), Elimina (blue trash icon).
- test: 22 participants, Angular University, 22/01/2024. Actions: Modifica (blue edit icon), Elimina (blue trash icon).

A footer at the bottom indicates 'Showing 1 to 2 of 2 entries' and includes a 'Course Search' input field, a PrimeNG logo, and a page number indicator '1'.

Figura 3.1: Prototipo finale.

La figura 3.1 mostra una delle pagine del prototipo.

## 3.1 Database

Innanzitutto è stato necessario progettare il database, nello specifico, la creazione di tabelle contenenti le tuple di prova per effettuare il test del prototipo. Come da richiesta di Gruppo SIGLA, non è stata realizzata la tabella per la gestione degli utenti, in quanto il progetto di tirocinio si è concentrato principalmente sulle funzionalità dell'amministratore per aggiungere, modificare ed eliminare le attività formative, erogate dall'azienda.

Grazie alle API di ASP.NET, è stato possibile creare la tabella del database direttamente dal codice del back-end. Quindi, è stato utilizzato un Object-Relational Mapping (ORM), una tecnica che si usa per convertire i dati tra un database relazionale e lo heap<sup>1</sup> di un linguaggio di programmazione orientato agli oggetti [18].

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<sup>2</sup>. Le sue colonne sono:

- Version (integer), versione del database
- AppliedOn (datetime), data di applicazione della migrazione
- Description (text), descrizione della migrazione

Per visualizzare e gestire il database, è stato utilizzato DBeaver, un software gratuito e open source per la gestione di database relazionali. La scelta di usare DBeaver è dovuta alla sua semplicità d'uso e al fatto che sia sviluppato dalla comunità open source [19].

---

<sup>1</sup> Lo heap è una regione di memoria in cui vengono allocati gli oggetti istanziati durante l'esecuzione di un programma.

<sup>2</sup> 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.



Figura 3.2: Logo di DBeaver.

## 3.2 Back-end

Questa sezione descrive come ho usato ASP.NET per lo sviluppo del back-end del prototipo.

Lo sviluppo principale è stato fatto appoggiandosi all'applicazione web già esistente, sulla quale Gruppo SIGLA 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] e le colonne vengono definite come proprietà della classe stessa, coi rispettivi metodi getter<sup>3</sup> e setter<sup>4</sup>. La figura 3.3 mostra il codice C# del file CoursesData.cs.

---

<sup>3</sup> Un getter è un metodo, encapsulato in una proprietà, che **legge** il valore di un campo privato di una classe.

<sup>4</sup> Un setter è un metodo, encapsulato in una proprietà, che **scrive** il valore di un campo privato di una classe.

```

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

```

Figura 3.3: Codice in C# del file CoursesData.cs.

La classe CoursesMigration, che estende la classe del contesto Migration, si occupa di creare la tabella del database, tramite un ORM [20] e i metodi appositi della libreria FluentMigrator [21]. In figura 3.3 si può vedere il listato del codice relativo alla creazione del database.

```

1 Create.Table(${nameof(CoursesData)})"
2     .WithColumn(${nameof(CoursesData.CoursesId)})"
3         ↪ .AsInt32() .PrimaryKey().Identity()
4     .WithColumn(${nameof(CoursesData.CoursesName)})"
5         ↪ .AsBoolean() .NotNullable().WithDefaultValue(false)
6     .WithColumn(${nameof(CoursesData.CoursesCapacity)})"
7         ↪ .AsInt16() .NotNullable().WithDefaultValue(10)
8     .WithColumn(${nameof(CoursesData.CoursesType)})"
9         ↪ .AsString() .NotNullable()
10    .WithColumn(${nameof(CoursesData.CoursesDate)})"
11        ↪ .AsString() .NotNullable();

```

Figura 3.4: Frammento di codice C# del file 4\_CoursesMigration.cs.

Il listato si trova all'interno del metodo Up() della classe CoursesMigration.

Siccome i metodi Up() e Down() sono ereditati dalla libreria FluentMigrator, è necessario fare un override<sup>5</sup>, riscrivendoli.

Successivamente, ho inizializzato delle tuple di prova all'interno di una lista denominata coursesRows e le ho inserite nella tabella appena creata tramite un ciclo di scorrimento. In figura 3.5 si può vedere il listato appena discusso.

```

1 foreach(var course in coursesRows)
2     Insert.IntoTable(${nameof(CoursesData)}").Row(course);

```

Figura 3.5: Frammento di codice C# del file 4\_CoursesMigration.cs.

La classe CoursesController, invece, estende la classe ControllerBase. In particolare, grazie agli attributi [HttpPost(" ")] e [ProducesResponseType(400)], gestisce le richieste e le risposte HTTP, smistandole nei metodi responsabili di ognuna delle operazioni (aggiunta, modifica, ricerca o eliminazione). Per esempio, il metodo AddCoursesToDb crea un oggetto contenente le informazioni necessarie all'esecuzione e lo passa come parametro al metodo Execute che eseguirà una delle quattro azioni, a seconda del tipo dell'oggetto in input. In figura 3.6 si può vedere il listato del metodo appena discusso.

---

<sup>5</sup> Nella programmazione ad oggetti override è l'operazione di riscrittura di un metodo ereditato.

```
1 [HttpPost("addCoursesData")]
2 [ProducesResponseType(200)]
3 [ProducesResponseType(400)]
4 [ProducesResponseType(500)]
5 public async Task<IActionResult> AddCoursesToDb([FromBody]
6     ↪ CoursesData[] Courses) {
7     var CoursesInfo = new AddCoursesToDbExecuteInfo {Courses
8         ↪ = Courses};
9     var result = await
10        ↪ this.coursesDataRepo.Execute(CoursesInfo);
11    if(result.IsSuccess)
12        return Ok();
13    else
14        throw result.Error.SourceException;
15 }
```

Figura 3.6: Metodo AddCoursesToDb() in C#, del file CoursesController.cs.

I metodi relativi alle altre operazioni sono analoghi al codice in figura 3.6.

Il metodo Execute(...) è all'interno della classe CoursesDataExecutor, che si occupa di eseguire l'operazione corretta, a seconda del tipo dell'oggetto (info) che riceve in input. In figura 3.7 si può vedere il listato del codice che gestisce lo smistamento delle operazioni.

```
1 public async Task<IExecuteResult> Execute(DatabaseSettings
    ↪ settings, IDbConnection connection, IDbTransaction
    ↪ transaction, IExecuteInfo info)
2 {
3     switch(info) {
4         case AddCoursesToDbCreateInfo i:
5             return await AddCoursesExecute(settings,
    ↪ connection, transaction, i);
6         case DeleteCoursesDataCreateInfo i:
7             return await DeleteCoursesDataExecute(settings,
    ↪ connection, transaction, i);
8         case UpdateCoursesDataCreateInfo i:
9             return await UpdateCoursesDataExecute(settings,
    ↪ connection, transaction, i);
10        default:
11            throw new NotSupportedException($"Execute with
    ↪ transaction for type
    ↪ {info.GetType().FullName} not supported");
12    }
13 }
```

Figura 3.7: Metodo C# Execute() contenente lo switch relativo alle operazioni.

Nel caso in cui info sia di tipo AddCoursesToDbCreateInfo, verranno eseguiti i metodi AddCoursesExecute e Add, relativi all'aggiunta di una riga alla tabella dei corsi, come si può vedere nei frammenti di codice delle figure 3.8 e 3.9.

```

1 private async Task<IExecuteResult>
  ↪ AddCoursesExecute(DatabaseSettings settings,
  ↪ IDbConnection connection, IDbTransaction transaction,
  ↪ AddCoursesToDbExecuteInfo info) {
2   foreach(var Course in info.Courses) {
3     _ = await Add(settings, connection, transaction,
  ↪ Course);
4   }
5
6   return await IExecuteResult.From(\texttt{true});
7 }
8 private async Task<CoursesData> Add(DatabaseSettings
  ↪ settings, IDbConnection connection, IDbTransaction
  ↪ transaction, CoursesData item)
9 {
10  Console.WriteLine("item: ",item);
11  var sql = $"INSERT INTO CoursesData (CoursesName,
  ↪ CoursesCapacity, CoursesType, CoursesDate) VALUES
  ↪ (@{nameof(CoursesData.CoursesName)}},
  ↪ @{nameof(CoursesData.CoursesCapacity)},
  ↪ @{nameof(CoursesData.CoursesType)},
  ↪ @{nameof(CoursesData.CoursesDate)})";
12
13  var r = await connection.ExecuteAsync(sql, item,
  ↪ transaction);
14  return item;
15 }
```

Figura 3.8: Metodi C# per l'inserimento dei corsi nel database.

In particolare, notiamo che `AddCoursesToDbExecuteInfo()` chiama `Add()` e in quest'ultimo vengono effettivamente implementate le query per interrogare il database.

Infine, è degno di nota anche il file `CoursesDataGetListExecutor.cs`, che si occupa di raccogliere la lista dei corsi di formazione, tramite il metodo `GetList()`, chiamato dalla classe `CoursesController` descritta in precedenza.

(Listato 3.9).

```

1 public async Task<IEnumerable<CoursesData>>
    ↪ GetList(DatabaseSettings settings, IDbConnection
    ↪ connection, Maybe<CoreDynamicQueryPart> query)
2 {
3     var (countSql, sql, parameters) =
        ↪ query.EnsureOrderBy(nameof(CoursesData.CoursesId))
        ↪ .ComposeWithCount<DapperQueryParametersBuilder,
        ↪ DynamicParameters>($"SELECT * $FROM
        ↪ {nameof(CoursesData)}",
        ↪ nameof(CoursesData.CoursesId), settings);
4     var result = await
        ↪ connection.QueryAsync<CoursesData>(sql: sql,
        ↪ param: parameters);
5     var count = await connection.QuerySingleAsync<int>(sql:
        ↪ countSql, param: parameters);
6     return result.Select(x => { x.Count = count; return x;
    ↪ });
7 }
```

Figura 3.9: Implementazione del metodo GetList.cs del file C# CoursesDataGetListExecutor.cs.

Una volta implementato il back-end, per verificare il suo corretto funzionamento, ho utilizzato una API apposita, Swagger, già discussa nel Capitolo 2. Nelle figure 3.10 e 3.11 si possono vedere alcuni esempi di utilizzo.

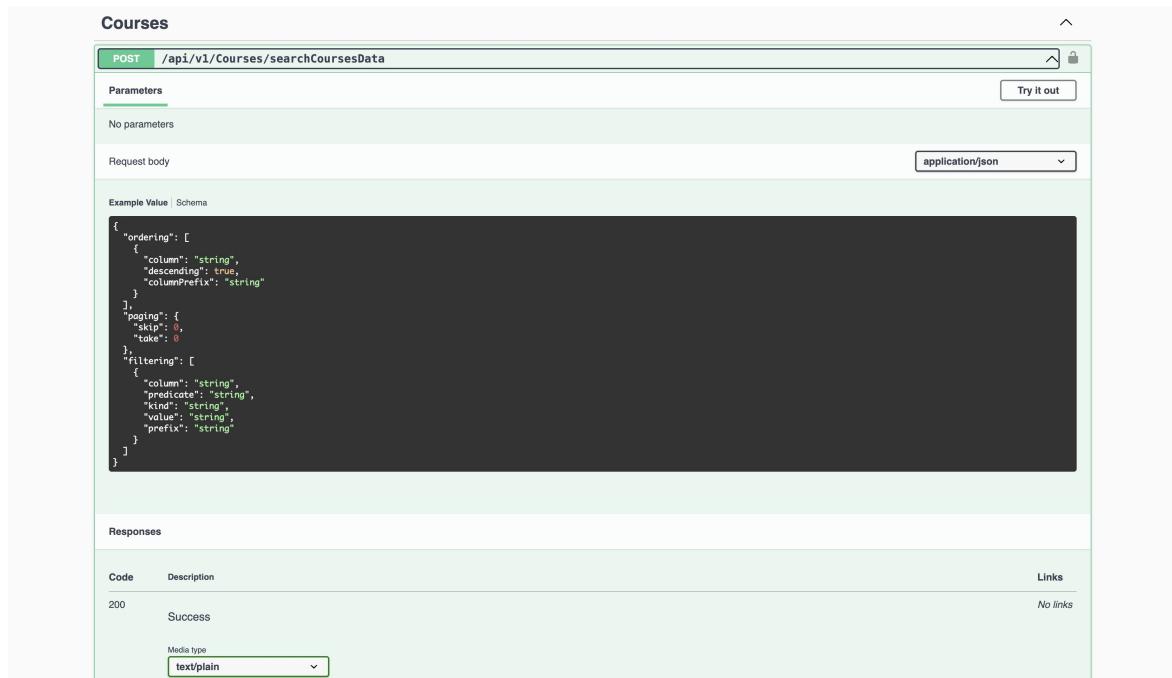


Figura 3.10: Esempio di esecuzione di una API per searchCoursesData, il corpo della richiesta è in formato JSON.

<p>200 Success</p> <p>Media type  <input checked="" type="button" value="text/plain"/> <span style="border: 1px solid #ccc; padding: 2px;">▼</span></p> <p>Controls Accept header.</p> <p><a href="#">Example Value</a>   <a href="#">Schema</a></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"/> <span style="border: 1px solid #ccc; padding: 2px;">▼</span></p> <p>Controls Accept header.</p> <p><a href="#">Example Value</a>   <a href="#">Schema</a></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 3.11: 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 Front-end

Questa sezione descrive come ho usato Angular per lo sviluppo del front-end del prototipo. Innanzitutto è stato necessario installare Node Package Manager (NPM) e Node Version Manager (NVM), i gestori di pacchetti rispettivamente per Angular e

```

portale-aziendale/front-end/angular on P gestione-formazione [!] via ⚡ orbstack via @ v21.6.1
> ng serve
Node.js version v21.6.1 detected.
Odd numbered Node.js versions will not enter LTS status and should not be used for production. For more information
, please see https://nodejs.org/en/about/previous-releases/.
One or more browsers which are configured in the project's Browserslist configuration will be ignored as ES5 output
is not supported by the Angular CLI.
Ignored browsers: kaios 2.5, op_mini all
Generating browser application bundles (phase: setup)...    TypeScript compiler options "target" and "useDefineFor
rClassFields" are set to "ES2022" and "false" respectively by the Angular CLI. To control ECMA version and features
use the Browserslist configuration. For more information, see https://angular.io/guide/build#configuring-browser-co
mpatibility
✓ Browser application bundle generation complete.

Initial Chunk Files | Names      | Raw Size
vendor.js          | vendor     |  8.61 MB |
styles.css, styles.js | styles     | 666.89 KB |
main.js            | main       | 401.43 KB |
polyfills.js       | polyfills  | 364.55 KB |
scripts.js         | scripts    | 58.77 KB |
runtime.js         | runtime    |  6.51 KB |

| Initial Total | 10.07 MB

Build at: 2024-02-12T14:53:00.290Z - Hash: 16bf1ad9f76e83a - Time: 3300ms

Warning: /Users/enrico/Desktop/tirocinio-tesi/portale-aziendale/front-end/angular/node_modules/angular-oauth2-oidc/
fesm2020/angular-oauth2-oidc.mjs depends on 'fast-sha256'. CommonJS or AMD dependencies can cause optimization bail
outs.
For more info see: https://angular.io/guide/build#configuring-commonjs-dependencies

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

✓ Compiled successfully.

```

Figura 3.13: Screenshot di un terminale dopo aver eseguito `ng serve`.

Node.js. In figura 3.12 si può vedere il listato del codice necessario per installarli, da riga di comando:

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

Figura 3.12: Comando per il terminale per scaricare e installare NVM.

Successivamente, sempre da riga di comando, ho impostato la versione di NVM necessaria per questo progetto, la numero 16, e poi ho installato le dipendenze necessarie, tramite il comando `npm install` [22].

A questo punto, ho compilato e avviato il front-end tramite i comandi appositi `ng build` e `ng serve` in modo da poter accedere all’interfaccia grafica del prototipo, semplicemente accedendo ad un URL su un qualsiasi browser. Nella figura 3.13 è mostrato l’output di un esempio di esecuzione del comando `ng serve` sul terminale.

Per la parte di sviluppo in Angular, ho utilizzato la sua CLI, che mi ha permesso di creare nuovi componenti, direttamente da terminale, con un solo comando. Per installarla è bastato eseguire `npm install -g @angular/cli` da riga di comando. Se

un nuovo componente viene creato senza errori, verrà creata una nuova cartella, chiamata con il nome del componente, all'interno del percorso `src/app/components`. La cartella contiene fin dall'inizio tutti i file necessari per far funzionare il nuovo componente, come si può vedere in figura 3.14.

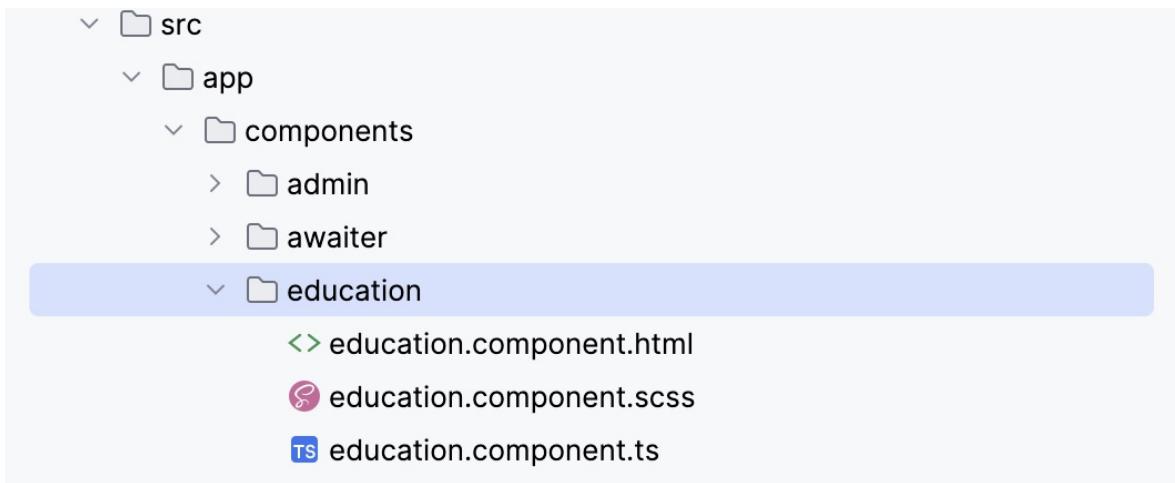


Figura 3.14: Struttura della cartella del componente `education`, creata tramite la CLI di Angular.

Una volta creati i componenti, ho sviluppato il file `app-routing.module.ts`, che contiene la route guard di Angular.

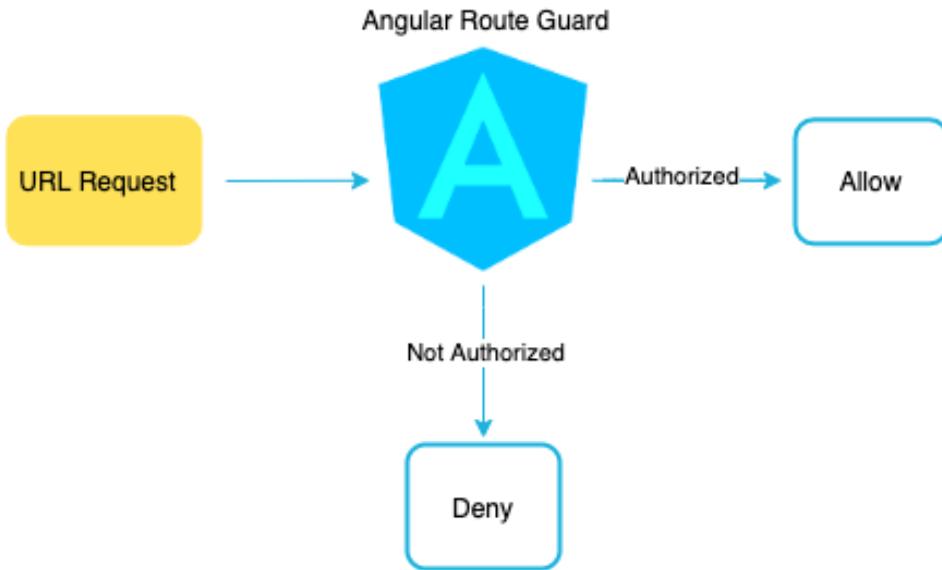


Figura 3.15: Schema di instradamento di Angular.

Una route guard in Angular è un meccanismo utilizzato per proteggere l'intradamento dell'applicazione e limitare l'accesso alle pagine in base alle autorizzazioni dell'utente. Le guardie consentono di gestire la navigazione dell'utente in modo sicuro, garantendo l'accesso alle pagine riservate solo agli utenti autorizzati (figura 3.15).

Una route guard è una classe TypeScript che implementa l'interfaccia canActivate, i cui metodi vengono eseguiti quando un utente tenta di accedere a una determinata rotta dell'applicazione. Il metodo canActivate, ad esempio, viene eseguito quando un utente tenta di accedere a una route specifica. La route guard può quindi verificare se l'utente ha l'autorizzazione per accedere alla route e restituire un valore booleano che indica se l'accesso deve essere consentito o meno. Se la route guard restituisce true, l'utente può accedere, se invece restituisce false, l'utente viene reindirizzato a una pagina di errore o alla pagina di login. In figura 3.16 si può vedere il listato di un frammento di codice di una route guard sviluppata per questo progetto di tirocinio.

```

1 export class AdminUserGuard implements canActivate {
2   constructor(private oauthService: CustomOAuthService) {}
3
4   async canActivate(
5     route: ActivatedRouteSnapshot,
6     state: RouterStateSnapshot
7   ): Promise<boolean | UrlTree> {
8     const isAdminUser = await
9       ↪ this.oauthService.isAdminUser();
10    if (!isAdminUser)
11      ↪ console.error("You should be 'admin-user' to
12        ↪ activate this route!");
13    return isAdminUser;
14  }
15}
```

Figura 3.16: Metodo canActivate del file admin-user.guard.ts.

Una volta sviluppato il codice in figura 3.16 è stato necessario implementare il codice relativo al file app-routing.module.ts e l'instradamento dell'utente è completato. Il risultato è mostrato nelle seguenti figure.

Il codice di ogni componente è diviso nei tre file seguenti:

- ‘name’ .component.html, che si occupa di gestire la parte HTML del componente, è quindi il template dello stesso;
- ‘name’ .component.ts, che si occupa di gestire la parte TypeScript del componente;
- ‘name’ .component.css, che in questo caso è vuoto, siccome il CSS di ogni componente è stato gestito tramite il file styles.css.

## MAIN MENU

**🔔 Notifiche**

- Prenota
- Iscriviti
- Formazione
- Panoramica uffici (A)
- Gestione uffici (A)
- Gestione notifiche (A)
- Gestione formazione (A)

## MAIN MENU

**🔔 Notifiche**

- Prenota
- Iscriviti

Figura 3.18: Vista dell'utente base.

Figura 3.17: Vista dell'admin.

In particolare, il file HTML contiene lo ‘scheletro’ del componente da visualizzare e alcuni tag speciali di Angular come, ad esempio, `ng-template` `pTemplate` nel listato 3.19, che si occupa di uniformare i dati del back-end con le relative caselle di una tabella HTML.

```

1 <ng-template pTemplate="body" let-coursesData>
2   <tr class="p-selectable-row"
3     ↪ [pRowToggler]="coursesData.coursesId">
4     <td>
5       <button type="button" pButton pRipple
6         class="p-button-text p-button-rounded
7           ↪ p-button-plain"
8         [icon]="isRowExpanded(coursesData.coursesId)
9           ↪ ? 'pi pi-chevron-down' : 'pi
10           ↪ pi-chevron-right'"
11         (click)="toggleRow(coursesData.coursesId)">
12       </button>
13     </td>
14     <td>
15       <span class="p-column-title">Nome corso</span>
16       {{coursesData.coursesName}}
17     </td>
18   </tr>
19 </ng-template>
```

Figura 3.19: Frammento di codice HTML del file `education.component.html`.

Per la parte TypeScript del componente, invece, è stato necessario considera-

re sia il livello di presentazione, che il livello di database, in modo da gestire e presentare le informazioni correttamente.

```

1 addEducation(){
2   this.coursesData$.subscribe((data) => {
3     if(data) {
4       data.forEach((course) => {
5         console.log(course);
6     });
7   });

```

Figura 3.20: Metodo addEducation() del file education.component.ts.

Come mostrato in figura 3.20, il codice si occupa di aggiungere le informazioni trovate dal database (salvate in coursesData\$()) al front-end, in modo da essere visualizzate dall'utente.

```

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

```

Figura 3.21: CSS del file styles.css, per gestire il CSS di tutti i componenti.

La figura 3.21 mostra lo stile dei componenti. Ho deciso di usare un tema predefinito di PrimeNG, importato nel file styles.css, come si può vedere nella figura. Nonostante la leggera perdita di personalizzazione, la scelta di utilizzare un tema predefinito mi ha permesso di ridurre i tempi di sviluppo, in quanto è stato necessario solamente importare un tema già pronto e non scrivere codice CSS partendo da zero, cosa spesso complicata.

Altri file degni di nota sono:

- courses.actions.ts: contiene tutti gli export delle azioni relative ai corsi di formazione;
- courses.effects.ts: si occupa principalmente di mappare le informazioni in entrata o in uscita dal componente;
- courses.reducer.ts: contiene alcuni metodi ausiliari per il corretto funzionamento della logica di business, in particolare per filtrare o manipolare le informazioni dal o verso il back-end;

- courses.state.ts: contiene la struttura di base dello stato dei corsi di formazione;
- courses.selectors.ts: che contiene gli export necessari per il funzionamento di ogni metodo filtro o di stato dei corsi di formazione.

Inoltre, dopo aver completato i file citati sopra, ho aggiunto un componente Angular per visualizzare un calendario visibile a larghezza intera, come si può vedere nella figura 3.24 della pagina dell'applicazione web. Così facendo, l'utente può visualizzare i corsi di formazione esistenti e scegliere il corso di formazione più adatto alle proprie esigenze.

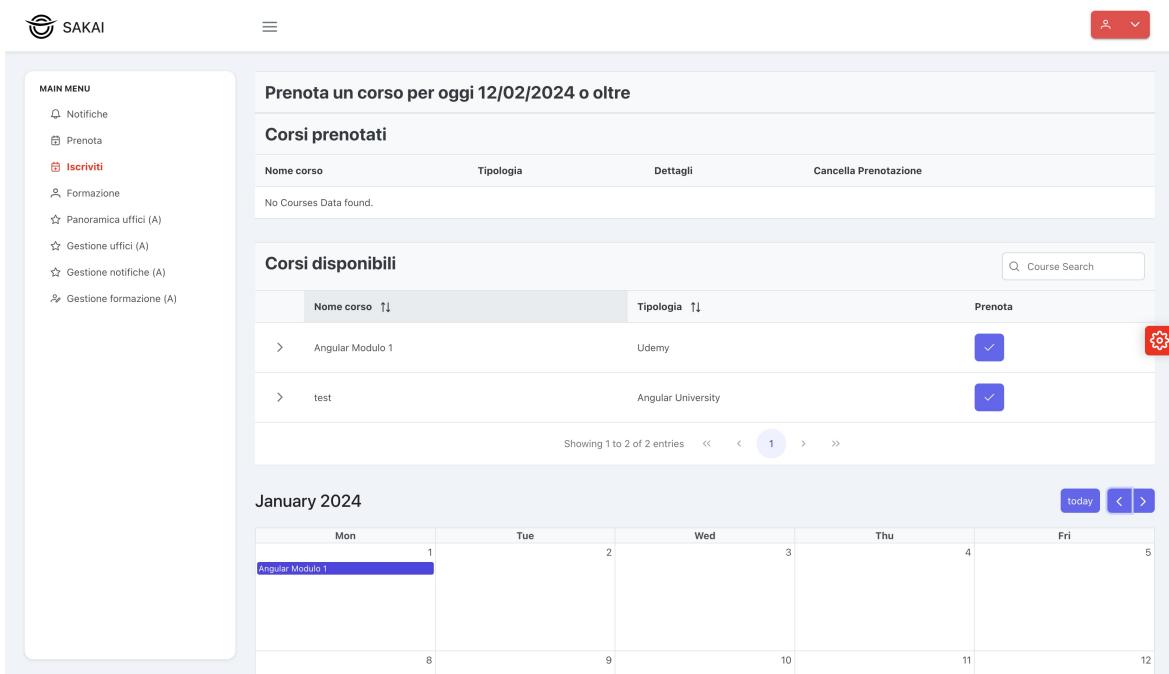


Figura 3.22: Vista iniziale di una pagina web del prototipo, con il componente fullcalendar.

Dopo averlo installato, sempre tramite la CLI di Angular e aver configurato i file app.module.ts e app.component.ts, il calendario può funzionare correttamente [23], come si può vedere in figura 3.24, semplicemente aggiungendo il tag HTML apposito, mostrato nella figura 3.23:

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

Figura 3.23: Tag HTML del file education.component.html.

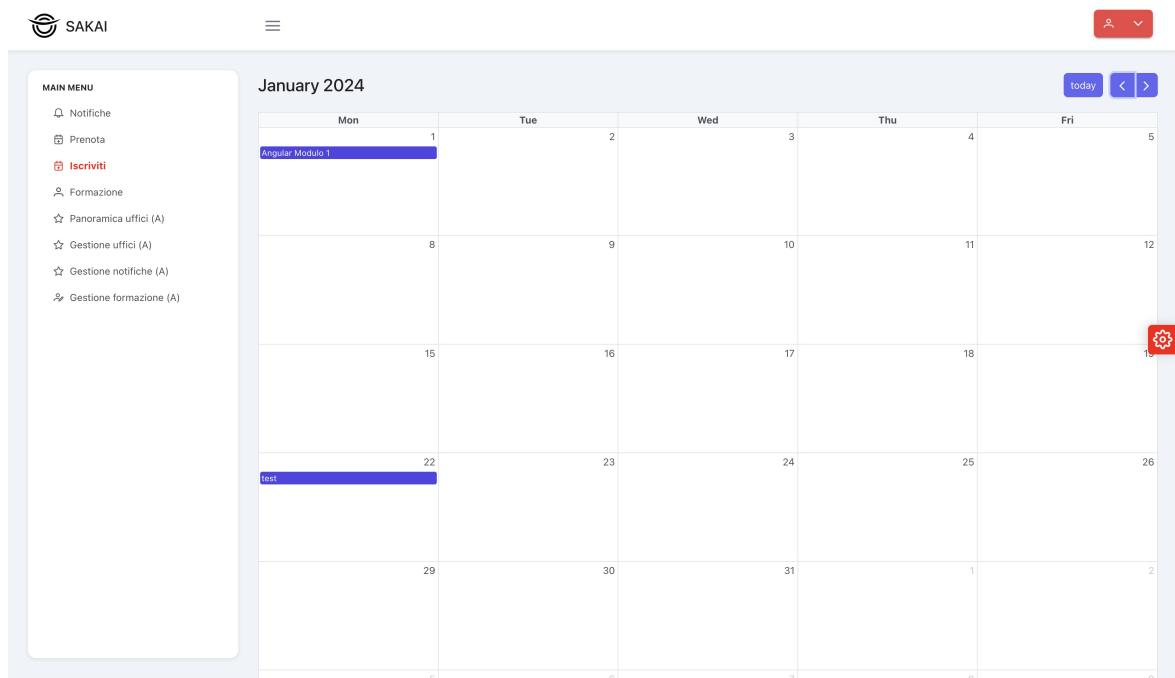


Figura 3.24: Componente fullcalendar.

In figura 3.25 si può vedere il listato del codice TypeScript, per gestire la logica di business del componente fullcalendarComponent.

```

1  ngOnInit(): void {
2      this.coursesData$ =
3          ↪ this.store.select(selectCoursesData).pipe(startWith(
4              ↪ this.route.snapshot.data.CoursesData));
5      this.coursesData$.pipe(
6          filter(data => !data)
7      ).subscribe((data) => {
8          this.totalRecords$ = this.coursesData$.pipe(map((x) =>
9              ↪ (x ? (x[0] ? x[0].count : 0) : 0)));
10         let tmp = [];
11         data.forEach((course) => {
12             let day:string = course.coursesDate[0] +
13                 ↪ course.coursesDate[1];
14             let month:string = course.coursesDate[3] +
15                 ↪ course.coursesDate[4];
16             let year:string = course.coursesDate[6] +
17                 ↪ course.coursesDate[7] +
18                 ↪ course.coursesDate[8] +
19                 ↪ course.coursesDate[9];
20             tmp.push({ title: course.coursesName, date: year +
21                 ↪ '-' + month + '-' + day }); // yyyy-mm-dd
22             console.log(tmp);
23         });
24         this.calendarOptions = {
25             ...this.calendarOptions,
26             events: tmp
27         };
28     })
29 }

```

Figura 3.25: TypeScript del file education.component.ts, necessario per il funzionamento della logica di business del calendario.

Lo sviluppo degli altri metodi e componenti è coerente agli esempi mostrati in precedenza e non verrà descritto in questo documento. Infine, nella fase finale del tirocinio, ho aggiunto un componente per la gestione delle iscrizioni da parte degli utenti base. Anche in questo caso il codice è analogo ai file discussi in precedenza e non verrà descritto.

### 3.3.1 PrimeNG

Grazie a PrimeNG, ho potuto implementare un'interfaccia grafica funzionale e minimale. Innanzitutto, è stato necessario installare i giusti pacchetti e le giuste dipendenze. Successivamente, la scelta dei componenti utilizzati e descritti in prece-

denza, 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 (figura 3.26);
- 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.

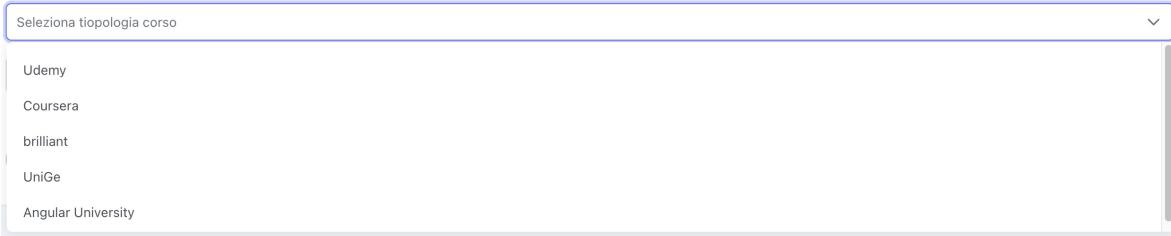


Figura 3.26: Menù a tendina per la selezione della tipologia di un corso.

### 3.3.2 Servizi Angular

I servizi Angular sono tra le funzionalità più potenti e flessibili del framework, in quanto permettono di creare e gestire la logica di business in modo efficiente e modulare [24]. Durante questo tirocinio, 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 (figura 3.27 e 3.28), si possono vedere due esempi di come è stato usato un servizio Angular per la gestione dei corsi di formazione:

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

Figura 3.27: TypeScript del file course.resolver.ts, contenente l'implementazione di un servizio Angular.

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

Figura 3.28: TypeScript del file admin-user-guard.ts, contenente l'implementazione di un servizio Angular.

# **Capitolo 4**

## **Conclusioni**

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

Dopo aver studiato e approfondito l'architettura del modello three-tier e la documentazione necessaria per lo sviluppo, posso ritenermi soddisfatto dell'esperienza svolta. Il tirocinio ha fornito un'ottima occasione per approfondire le mie capacità in alcuni linguaggi e impararne altri. L'applicazione web finale è risultata moderna e intuitiva, al passo 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.

Gli obiettivi futuri di Gruppo SIGLA, per questo prototipo, includono il login interno (utilizzando Keycloak[25] e OAuth2[26]), l'aggiunta della tabella degli utenti e la possibilità di integrare una funzione per gestire i turni lavorativi, in modo che l'azienda possa controllare il personale interamente dall'applicazione web.

### **4.1 Ringraziamenti**

Ringrazio tutte le persone che mi sono state accanto in questi anni di 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;

la mia ragazza, per avermi supportato nelle giornate più impegnative e nei momenti più stressanti;

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 disponibili 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. . . . .	2
2.2	Logo di MySQL. . . . .	4
2.3	Schema astratto dell'architettura MVC. . . . .	5
2.4	Schema dei framework utilizzati. . . . .	5
2.5	Logo di Angular. . . . .	6
2.6	Logo di PrimeNG. . . . .	6
2.7	Argon, un template Angular. . . . .	8
2.8	Diamond, uno dei template di PrimeNG. . . . .	9
2.9	Logo di TypeScript. . . . .	10
2.10	Inizializzazione del classico ‘Hello World!’. . . . .	10
2.11	Logo di .NET Framework. . . . .	11
2.12	Principali tecnologie Microsoft .NET. . . . .	13
2.13	Logo di Swagger. . . . .	13
2.14	Interfaccia grafica di Swagger, relativa al progetto di tirocinio. . . . .	14
3.1	Prototipo finale. . . . .	15
3.2	Logo di DBeaver. . . . .	17
3.3	Codice in C# del file CoursesData.cs. . . . .	18
3.4	Frammento di codice C# del file 4_CoursesMigration.cs. . . . .	19
3.5	Frammento di codice C# del file 4_CoursesMigration.cs. . . . .	19
3.6	Metodo AddCoursesToDb() in C#, del file CoursesController.cs. . . . .	20
3.7	Metodo C# Execute() contenente lo switch relativo alle operazioni. . . . .	21
3.8	Metodi C# per l'inserimento dei corsi nel database. . . . .	22
3.9	Implementazione del metodo GetList.cs del file C# CoursesDataGetListExecutor.cs.	23
3.10	Esempio di esecuzione di una API per searchCoursesData, il corpo della richiesta è in formato JSON. . . . .	24
3.11	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’. . . . .	25
3.13	Screenshot di un terminale dopo aver eseguito ng serve. . . . .	26
3.12	Comando per il terminale per scaricare e installare NVM. . . . .	26

3.14 Struttura della cartella del componente education, creata tramite la CLI di Angular. . . . .	27
3.15 Schema di instradamento di Angular. . . . .	27
3.16 Metodo canActivate del file admin-user.guard.ts. . . . .	28
3.17 Vista dell'admin. . . . .	29
3.18 Vista dell'utente base. . . . .	29
3.19 Frammento di codice HTML del file education.component.html. . . . .	29
3.20 Metodo addEducation() del file education.component.ts. . . . .	30
3.21 CSS del file styles.css, per gestire il CSS di tutti i componenti. . . . .	30
3.22 Vista iniziale di una pagina web del prototipo, con il componente fullcalendar.	31
3.23 Tag HTML del file education.component.html. . . . .	31
3.24 Componente fullcalendar. . . . .	32
3.25 TypeScript del file education.component.ts, necessario per il funzionamento della logica di business del calendario. . . . .	33
3.26 Menù a tendina per la selezione della tipologia di un corso. . . . .	34
3.27 TypeScript del file course.resolver.ts, contenente l'implementazione di un servizio Angular. . . . .	34
3.28 TypeScript del file admin-user-guard.ts, contenente l'implementazione di un servizio Angular. . . . .	35

# Bibliografia

- [1] N. Community, “Ngrx docs,” *ngrx*, 2023.
- [2] ReactiveX, “Reactivex docs,” *reactivex*, 2022.
- [3] Google, “Angular docs,” *Angular Docs*, 2024.
- [4] PrimeFaces, “Primeng docs,” *PrimeNG Docs*, 2023.
- [5] Google, “Introduction to services and dependency injection,” *Angular Docs*, 2023.
- [6] “TypeScript.” <https://en.wikipedia.org/wiki/TypeScript#References>.
- [7] C. Murphy, “What is typescript and why you should use it for your next project,” *Prismic*, 2023.
- [8] Microsoft, “Net docs,” *LearnMicrosoft*, 2023.
- [9] Wikipedia, “.net framework,” *Wikipedia*, 2023.
- [10] Wikipedia, “.net framework version history,” *Wikipedia*, 2023.
- [11] Microsoft, “Net videos,” *MicrosoftChannel*, 2023.
- [12] Microsoft, “Asp.net core docs,” *LearnMicrosoft*, 2023.
- [13] Wikipedia, “Asp.net,” *Wikipedia*, 2023.
- [14] Microsoft, “Asp.net documentation,” *LearnMicrosoft*, 2023.
- [15] Microsoft, “Introduction to identity on asp.net core,” *Prismic*, 2023.
- [16] Wikipedia, “Swagger (software),” *Wikipedia*, 2023.
- [17] SmartBear, “Api development for everyone,” *swagger.io*, 2023.
- [18] C. V. Abba, “What is an orm – the meaning of object relational mapping database tools,” *freeCodeCamp*, 2023.
- [19] Wikipedia, “Dbeaver,” *Wikipedia*, 2023.
- [20] Wikipedia, “Object–relational mapping,” *Wikipedia*, 2023.

- [21] F. M. Project, “Fluent migrations framework for .net,” *FluentMigratorGitHub*, 2018.
- [22] lukekarrys, “npm-install,” *npm Documentation*, 2023.
- [23] F. LLC, “Angular component,” *FullCalendar*, 2023.
- [24] Google, “Lesson 09: Angular services,” *Angular Docs*, 2023.
- [25] T. L. Foundation, “Keycloak docs,” *KeycloakDocumentation*, 2023.
- [26] OAuth, “Oauth 2.0,” 2023. <https://oauth.net/2/>.