POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Sviluppo di un portale web per la gestione delle carriere dei PhD del DAUIN

Supervisori

Prof. Fulvio VALENZA

Dott. Ric. Davide PIUMATTI

Candidato

Luca Daniele PREVATO

DICEMBRE 2025

Dedicato a Elisabetta, mia moglie e da sempre mia amica più cara.

Sviluppo di un portale web per la gestione delle carriere dei PhD del DAUIN

Luca Daniele Prevato

Abstract

Si vuole realizzare un portale web che consenta la gestione delle valutazioni annuali dei PhD, raccogliendo le informazioni di interesse per la valutazione (inserite dallo studente, dal tutore o dai database centrali di ateneo) e gli esiti della valutazione della commissione. La visione delle valutazioni agli studenti deve essere anonima, le valutazioni annuali devono essere visibili per tutta la durata del PhD. Inoltre, il portale deve gestire le commissioni di valutazione i cui membri possono variare nel tempo.

Sommario

Elenco delle figure

Elenco delle tabelle

Acronyms

AJAX	Asynchronous JavaScript and XML.
API	Application Programming Interface.
AOT	Ahead-of-Time (Compilation).
ASP	Active Server Pages.
ASP.NET	Active Server Pages .NET.
BCL	Base Class Library.
CGI	Common Gateway Interface.
CLR	Common Language Runtime.
DAUIN	Dipartimento di Automatica e Informatica.
DTO	Data Transfer Object.
EF	Entity Framework.
EF Core	Entity Framework Core.
HMVC	Hierarchical Model-View-Controller.
HTML	HyperText Markup Language.
HTTP	HyperText Transfer Protocol.
IdP	Identity Provider.
IIS	Internet Information Services.
IRIS	Institutional Research Information System.
JIT	Just-In-Time (Compilation).
JSON	JavaScript Object Notation.
JSP	JavaServer Pages.

MVA	Model-View-Adapter.
MVC	Model-View-Controller.
MVP	Model-View-Presenter.
MVVM	Model-View-ViewModel.
ORM	Object-Relational Mapping.
PhDManV2	PhD Manager Versione 2.
RDBMS	Relational Database Management System.
REST	Representational State Transfer.
SSMS	SQL Server Management Studio.
SPA	Single Page Application.
SP	Service Provider.
SQL	Structured Query Language.
SSO	Single Sign-On.
SAML	Security Assertion Markup Language.
SSOT	Single Source of Truth.
T-SQL	Transact-SQL.
URL	Uniform Resource Locator.
WWW	World Wide Web.

Capitolo 1

Evoluzione e stato attuale dello sviluppo web (2025)

1.1 Introduzione

Da quando venne inventato oltre 30 anni fa, il **World Wide Web (WWW)** si è evoluto da un semplice sistema di condivisione di documenti a un vasto ecosistema interconnesso che pervade quasi ogni aspetto della vita moderna; la comunicazione, il commercio, l'intrattenimento e l'enterprise computing sono oggigiorno radicati all'interno del web. Le sue tecnologie e paradigmi di programmazione si sono fortemente evoluti per soddisfare le sempre nuove richieste riguardo la scalabilità, l'interattività e l'esperienza utente.

Questo capitolo fornirà dettagli riguardo **l'evoluzione della programmazione web**, partendo dalle sue origini di pagine ipertestuali statiche fino a complesse applicazioni data-driven, complesse e dinamiche. Si inizierà parlando delle origini del web, dei sui standard e tecnologie iniziali che l'hanno reso possibile. Successivamente verranno analizzate le principali transizioni tecnologiche, come lo scripting server-side, l'introduzione di **modelli di progettazione architetturale** come **MVC (Model-View-Controller)** e la proliferazione di **framework e librerie** che permettono lo sviluppo di applicazioni web su larga scala.

Un'attenzione particolare verrà data all'**ecosistema web Microsoft**, in particolare ad **ASP.NET** e alla sua evoluzione moderna in **Pagine Razor**, che esemplifica le più moderne best practices relative alla programmazione per componenti, alla produttività dello sviluppo e alla separazione delle preoccupazioni. Questo capitolo, inoltre, introdurrà il concetto di gestione dei dati e come si sia reso necessario lo sviluppo di sistemi efficienti e consistenti.

Infine, il capitolo si conclude con un'analisi delle **tecnologie attuali** come il **cloud computing**, i **microservizi**, **Blazor** e le **integrazioni AI**, che plasmano la direzione della programmazione web moderna. Fornendo questa panoramica questo capitolo pone l'applicazione web sviluppata in un più ampio contesto storico e tecnologico del web.

1.2 Origini del Web

Le origini del web risalgono ai primi anni 90 quando **Tim Berners-Lee**, un ricercatore del **CERN**, sviluppò un metodo per semplificare la condivisione di dati e informazioni scientifiche attraverso sistemi di computer. [1] Il suo lavoro portò alla creazione del **World Wide Web**, un sistema ipertestuale distribuito basato su tre tecnologie base: **HTML (HyperText Markup Language)**, **HTTP (HyperText Transfer Protocol)** e **URL (Uniform Resource Locators)**. [2] Queste tecnologie hanno fornito un framework universale e indipendente dalla piattaforma per condividere documenti, permettendo uno scambio di informazioni su scala globale. [3]

La prima generazione di siti web, comunemente riferiti come **Web 1.0** consistevano di **pagine statiche** ed erano semplici documenti HTML inviati dai web server senza alcun tipo di interazione da parte dell'utente. Queste pagine dovevano essere aggiornate a mano ed erano quindi inefficienti per applicazioni che prevedevano operazioni da parte dell'utente o modifiche frequenti. Nonostante queste limitazioni questi siti statici hanno introdotto i concetti rivoluzionari di collegamento ipertestuale e accesso universale alle informazioni.

Con l'espansione del web, la necessità di generazione di **contenuti dinamici** diventò chiara. Gli utenti volevano inviare dati, fare ricerche e avere risultati personalizzati; tutte cose che le pagine statiche in HTML non permettevano di fare. Questo ha portato alla nascita di tecnologie

di **scripting lato server** come **CGI (Common Gateway Interface)** nel 1993 che permettevano ai server web di eseguire script esterni (spesso scritti in Perl) e generare contenuti dinamici in tempo reale. Poco dopo vennero sviluppate altre tecnologie come **PHP (Hypertext Preprocessor)** nel 1995, **ASP (Active Server Pages)** nel 1996 e **JSP (JavaServer Pages)** nel 1999, introducendo ambienti integrati di scripting che combinavano la logica server con dei template HTML. Questo segnò il passaggio dal **Web 1.0** al **Web 2.0**, focalizzato sull'interattività e partecipazione da parte dell'utente e generazione di contenuti personalizzati.

Tecnologia	Anno	Caratteristiche	Limiti
CGI	1993	Script esterni eseguiti dal server	Poco efficiente
PHP	1995	Scripting embedded nell'HTML	Codice poco strutturato
ASP	1996	Script lato server Microsoft	Dipendenza da Windows
JSP	1999	Java + template HTML	Verbosità Java

Tabella 1.1: Confronto tecnologie server-side storiche

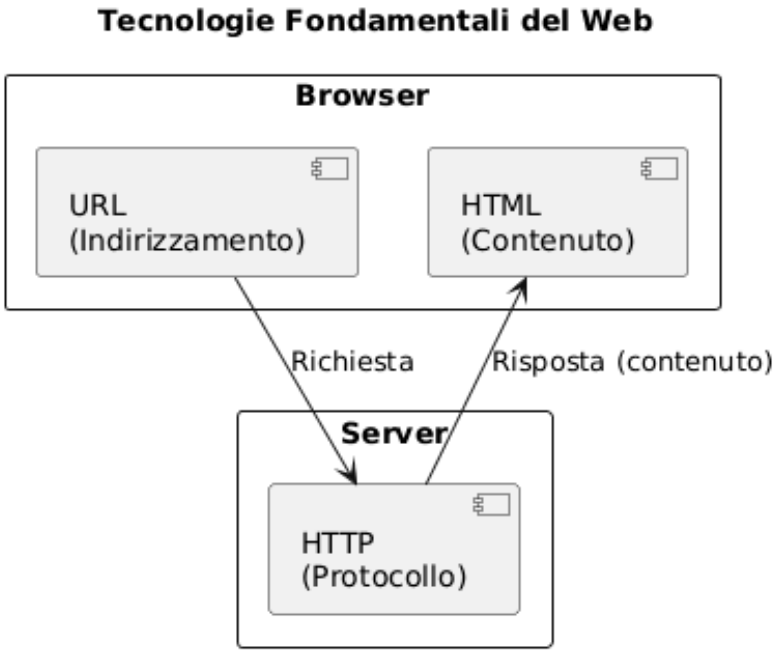

Figura 1.1: Tecnologie base nel Web (HTML, HTTP, URL)

1.3 Evoluzione Delle Applicazioni Web Dinamiche

I contenuti web dinamici, nati nel periodo dai tardi anni 90 ai primi anni 2000, hanno profondamente modificato il modo in cui gli sviluppatori dovevano concepire lo sviluppo web. Il web dinamico aumentava sia la complessità che le dimensioni dei siti web, gli sviluppatori quindi necessitavano di metodologie strutturate per organizzare il codice, separare le preoccupazioni e gestire le connessioni tra i diversi componenti delle applicazioni.

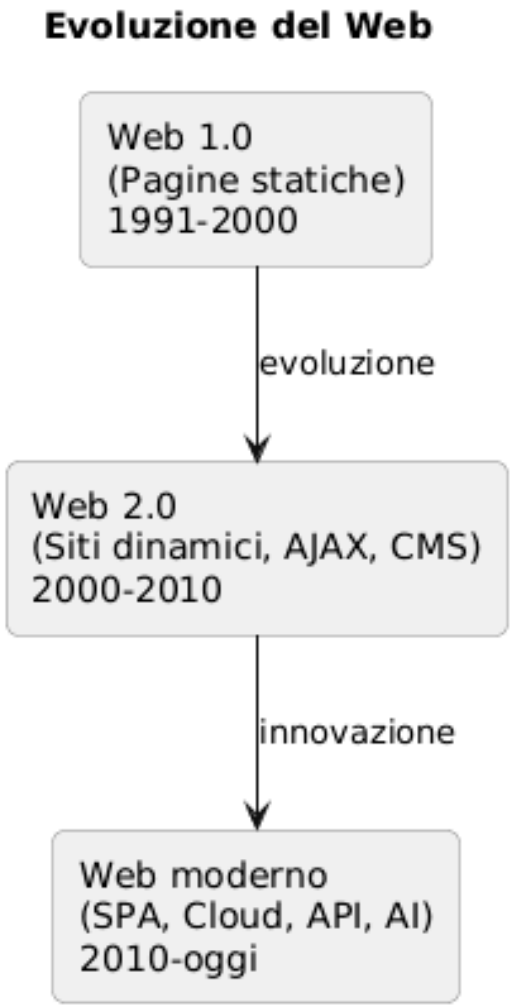

Figura 1.2: Linea temporale dell'evoluzione del Web.

Da queste necessità nacque l'architettura **Model-View-Controller (MVC)** che separa l'applicazione in tre sezioni distinte: il **Modello (Model)**, responsabile dei dati e logica di business; la **Vista (View)**

che governa l'interfaccia utente e la presentazione; e il **Controllore (Controller)** che gestisce gli input e coordina tutte le interazioni tra la Vista e il Modello. [4] Successivamente si è evoluto il modello MVC dando vita a varianti come il **Model-View-Controller gerarchico (HMVC)**, il **Model-View-Adapter (MVA)**, il **Model-View-Presenter (MVP)**, il **Model-View-Viewmodel (MVVM)** e altri che adattavano MVC a contesti differenti. [5] Il modello MVC divenne la base concettuale per molti framework web, permettendo di progettare applicazioni che fossero scalabili e mantenibili.

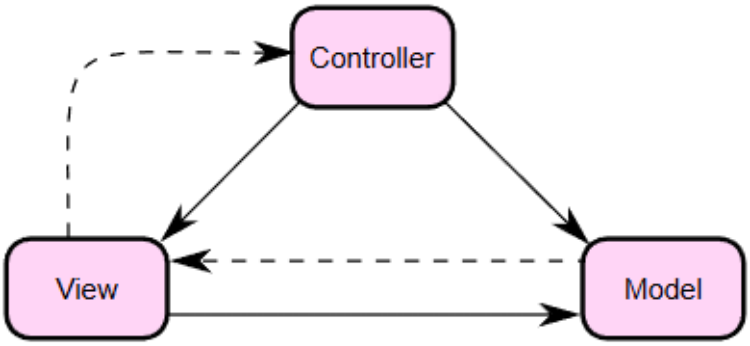

Figura 1.3: Modello MVC.

Pattern	Punti di forza	Debolezze
MVC	Separazione componenti	Non ideale per UI complesse
MVP	Testabilità elevata	Molta logica nel Presenter
MVVM	Ottimo per data binding	Sovraccarico concettuale

Tabella 1.2: Confronto tra pattern architetturali

Per soddisfare queste nuove esigenze in quegli anni emersero dei nuovi framework, atti a semplificare lo sviluppo delle applicazioni e a velocizzare i tempi di consegna; Microsoft **ASP.NET** (2002), il framework open-source **Ruby on Rails** (2005) e **Django** (2005) per Python adottarono il paradigma MVC (o una sua variante come nel caso di Django[6]) con lo scopo di rendere il codice più modulare e ridurre al minimo le attività ripetitive.

Nel frattempo, **JavaScript** che in origine era utilizzato solo per interazioni limitate client-side, si è evoluto ed è diventato un pilastro fondamentale per lo sviluppo web moderno. Un momento cruciale fu l'introduzione di **AJAX (Asynchronous JavaScript and XML)** nel 2005, che consentiva alle pagine web di aggiornare in maniera asincrona

porzioni specifiche delle pagine senza dover ricaricare l'intero documento. [7]

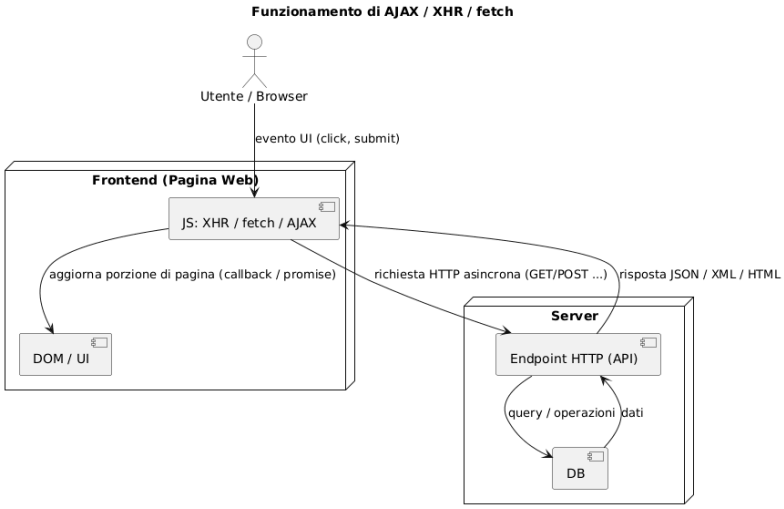

Figura 1.4: Schema del funzionamento di AJAX/XHR/fetch

Questa innovazione migliora sensibilmente la performance e l'esperienza utente, permettendo la realizzazione di interfacce web dinamiche e elaborate. L'adozione di **AJAX** ha spianato la strada verso le **Single Page Applications (SPA)** o applicazioni su singola pagina, nelle quali il client e il server comunicano prevalentemente tramite **API RESTful** scambiandosi dati strutturati in formati come il **JSON**. [8]

Questo cambiamento architetturale ha portato a un profondo cambiamento nella filosofia di sviluppo: il web non era più una mera raccolta di pagine e documenti, ma si era trasformato in una piattaforma interattiva capace di fornire applicazioni complesse. L'unione di potenti framework, comunicazione asincrona e architetture orientate ai servizi sono le basi del web interattivo, modulare e moderno che conosciamo oggi.

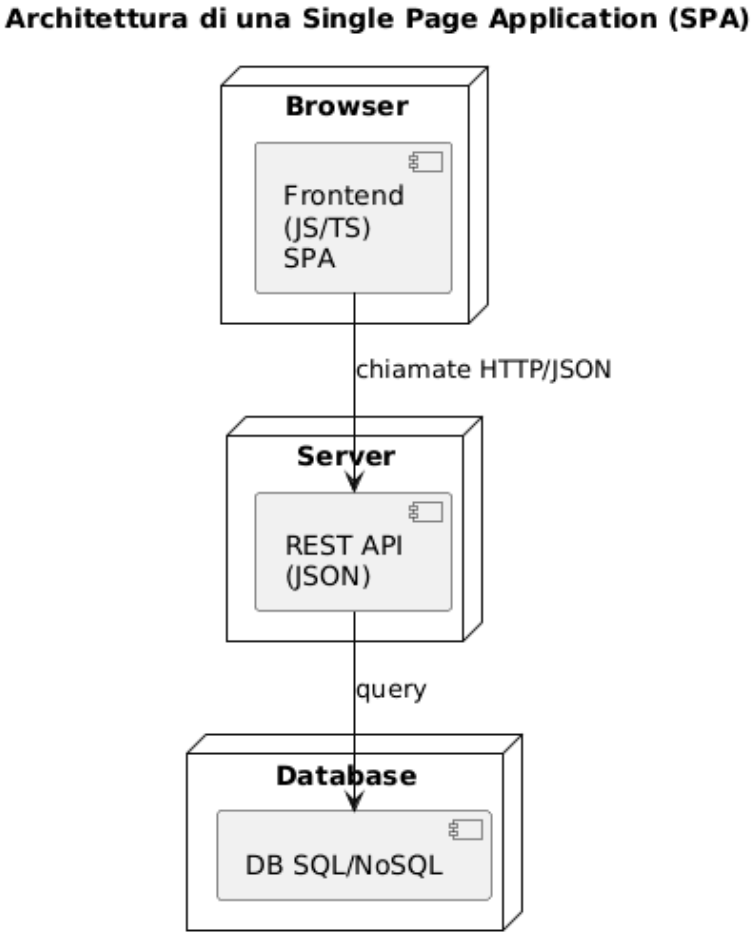

Figura 1.5: Architettura di una SPA

1.4 ASP.NET e l'Ecosistema Microsoft

Il contributo di Microsoft nell'evoluzione della programmazione web è stato sicuramente ampio e importante. La prima tecnologia web di rilievo, **Active Server Pages (ASP)**, fu introdotta alla fine degli anni 90 e permise agli sviluppatori di includere script server-side direttamente nelle pagine HTML. Sebbene questo semplificasse la creazione di pagine dinamiche, dall'altro lato portava a un forte accoppiamento del codice che diventava così più difficile da mantenere.

L'introduzione di **ASP.NET** nel 2002 rappresentò una completa riconsiderazione dell'approccio usato in precedenza; sviluppato sul **Framework .NET**, ASP.NET era **compilato**, fortemente tipizzato e completamente integrato nel più ampio ecosistema .NET. [9] Questo cambiamento portò un nuovo livello di robustezza e scalabilità alle applicazioni web, rendendo possibile lo sviluppo di soluzioni a livello aziendale più

veloci, più sicure e più facili da gestire.

Negli anni seguenti ASP.NET continuò ad evolversi, introducendo nuovi paradigmi come **ASP.NET MVC** e **Web API** per allinearsi con le più moderne architetture di sviluppo software. Un traguardo importante venne raggiunto con la pubblicazione di **.NET Core**, una reimplementazione **multipiattaforma e open-source** del framework, che ha reso le applicazioni sviluppate in ASP.NET portabili in diversi sistemi operativi e ambienti cloud. [10]

In mezzo ai vari paradigmi ASP.NET, le **Pagine Razor**, introdotte con ASP.NET Core, sono emerse come soluzione elegante per sviluppare applicazioni focalizzate sulle pagine. La **sintassi Razor**, combinando il codice HTML e quello C# in un singolo file, fornisce agli sviluppatori una metodologia ordinata e produttiva per creare contenuti dinamici. [11] Se confrontato col tradizionale **Web Forms**, le Pagine Razor presentano una netta separazione tra logica e presentazione, facilitano la testabilità e riducono significativamente il codice boilerplate (ovvero quelle sezioni di codice che vengono ripetute in più punti con poche o nessuna variazione).

ASP.NET Razor al momento è tra i framework più sofisticati nell'ambito dello sviluppo web, ed è frutto di un lavoro di perfezionamento da parte di Microsoft delle tecnologie web.

1.5 Gestione dei Dati nelle Applicazioni Web

I dati sono sempre stati il cuore pulsante delle applicazioni web. Con l'aumento della complessità dei sistemi una gestione efficiente dei dati è diventata un componente essenziale delle architetture web. Prima dell'avvento dei database relazionali, l'archiviazione dei dati veniva spesso implementata utilizzando file di testo o strutture proprietarie, che erano lenti, non strutturati e soggetti a errori.

L'introduzione del **modello relazionale** da parte di **E. F. Codd** nel 1970 ha rivoluzionato la gestione dei dati fornendo un sistema formale e logico per organizzare e recuperare le informazioni. [12] I dati ora venivano organizzati in tabelle e correlati tramite chiavi e vincoli e interrogati usando **SQL (Structured Query Language)**. [13] Questa innovazione fu una pietra miliare nei moderni sistemi di database e ancora oggi è lo standard.

Nell'ambiente web le applicazioni seguono tipicamente il modello **client-server**, nel quale l'applicazione web funge da client che comunica con il database sul server back-end. Scrivere query SQL direttamente nel codice dell'applicativo divenne rapidamente macchinoso, e per semplificare questo processo vennero introdotti i framework **ORM (Object-Relational Mapping)**.

Entity Framework (EF), la soluzione ORM di Microsoft, permette agli sviluppatori di manipolare le entità dei database direttamente dal codice dell'applicativo. [14] Questa astrazione permette di eliminare la necessità di gestire manualmente SQL garantendo una migliore manutenibilità, portabilità e sicurezza dei tipi. Entity Framework si integra perfettamente con **ASP.NET** e le **pagine Razor** permettendo agli sviluppatori di concentrarsi sulla logica di business piuttosto che sulle complessità dell'accesso al database. Insieme, il modello relazionale, SQL e le tecnologie ORM costituiscono la spina dorsale dello sviluppo web basato sui dati, consentendo alle applicazioni di gestire e conservare le informazioni in modo efficiente su sistemi su larga scala.

1.6 SQL Server e il suo ruolo nello sviluppo web moderno

Microsoft SQL Server è uno dei **sistemi di gestione di database relazionali (RDBMS)** più utilizzati in ambito aziendale, noto per le sue prestazioni, affidabilità e profonda integrazione con le altre tecnologie Microsoft.

Alla base SQL Server è composto da alcuni macro componenti: il **motore di database**, responsabile dell'archiviazione dei dati e dall'elaborazione delle query, **SQL Server Management Studio (SSMS)**, strumento grafico per l'amministrazione del database, e **Transact-SQL (T-SQL)**, un'estensione proprietaria di SQL Sviluppata da Microsoft che supporta la programmazione procedurale, le transazioni e operazioni avanzate sui dati. [15]

L'architettura di SQL Server enfatizza la **sicurezza**, **scalabilità** e **ottimizzazione delle prestazioni**. Funzionalità come la **sicurezza a livello di riga**, la **crittografia**, le **stored procedures** e l'**indicizzazione** consentono un controllo dettagliato sull'accesso ai dati e sulle prestazioni del sistema. Inoltre, SQL Server si integra direttamente con **Entity**

Framework, permettendo una comunicazione fluida tra il data layer e il layer applicativo nel progetti ASP.NET.

L'incremento di utilizzo del **cloud computing** ha richiesto un nuovo approccio nella gestione dei database; SQL Server, tramite **Azure SQL Database**, ha fornito una versione basata sul cloud, completamente gestita, e che offre scalabilità, disponibilità globale e elasticità. Per questi motivi è una soluzione potente e flessibile per tutte quelle applicazioni web che necessitano di elevata disponibilità e gestione sicura dei dati in ambienti distribuiti.

1.7 Tendenze Attuali e Prospettive Future nella Programmazione Web

L'ambito della programmazione web continua a evolversi rapidamente, modificato da nuovi paradigmi e innovazioni tecnologiche. Uno dei più importanti cambiamenti degli ultimi anni è stata l'adozione diffusa del **cloud computing**. Piattaforme come **Microsoft Azure**, **Amazon Web Services (AWS)** e **Google Cloud Platform (GCP)** ora ospitano una parte sostanziosa delle applicazioni web fornendo un'infrastruttura scalabile e servizi avanzati per quanto riguarda l'archiviazione, la rete e i calcoli. [16] L'incremento di popolarità dei **microservizi** e **architetture serverless** ha ulteriormente modificato come vengono progettate e pubblicate le applicazioni web. Al posto di sistemi monolitici, gli sviluppatori realizzano le applicazioni come una collezione di servizi più piccoli e distribuiti indipendentemente gli uni dagli altri. Questo approccio migliora la resistenza ai guasti, la scalabilità, la manutenibilità, allineandosi perfettamente con le metodologie DevOps e le pipeline di integrazione continua/distribuzione continua (CI/CD).

Il framework **Blazor** ha introdotto un nuovo modo di sviluppare applicazioni web usando C# invece di JavaScript. [17] Gli sviluppatori con Blazor, sfruttando WebAssembly, possono scrivere applicazioni full-stack in .NET con il front-end e il back-end unificati in un unico modello di programmazione; questo approccio non solo aumenta la coerenza nello sviluppo ma riduce anche la complessità.

In futuro ci si aspetta che l'**intelligenza artificiale (AI)** e il **Machine learning (ML)** avranno un ruolo sempre più centrale nello sviluppo web. [18] Le applicazioni possono essere oggi più **consapevoli del contesto**,

personalizzate e **autonome**, capaci di imparare dal comportamento dell'utente e di adattarsi in tempo reale. Questi sviluppi, insieme alla continua crescita dell'infrastruttura cloud e ai framework multi piattaforma, stanno plasmando una nuova era di **sistemi web intelligenti e adattivi** che definiranno la prossima generazione delle esperienze digitali.

1.8 Conclusioni

La storia della programmazione web riflette una più ampia evoluzione dell'informatica in generale: dalla mera rappresentazione di informazioni statiche a sistemi interattivi, guidati dai dati e intelligenti. Ogni fase di questo percorso ha introdotto nuovi paradigmi che hanno espanso le potenzialità del web: l'HTML ha standardizzato la distribuzione dei contenuti, lo scripting lato server ha introdotto un comportamento dinamico, i framework MVC hanno strutturato le applicazioni e il cloud computing ha ridefinito la scalabilità e l'accessibilità. Oggigiorno tecnologie come **ASP.NET Razor** e **SQL Server** rappresentano il culmine di decenni di progressi; combinano la robustezza di framework maturi con la flessibilità richiesta dallo sviluppo moderno. L'applicazione sviluppata in questa tesi sfrutta queste tecnologie per dimostrare dell'architettura modulare, forte consistenza dei dati e un'efficiente interazione utente.

Comprendere questa evoluzione storica e tecnologica fornisce informazioni su come sono nati gli attuali framework web, perché sono strutturati in questo modo e come continuano a evolversi verso un ecosistema digitale ancora più connesso, intelligente e fluido.

Per concludere la tesi è strutturata come segue: Nel Capitolo 2 descriverò nel dettaglio tutte le tecnologie usate per realizzare il portale. Il Capitolo 3 analizzerò le specifiche funzionali dell'applicativo, descrivendo il contesto, i desiderata e le funzionalità principali richieste. All'interno del Capitolo 4 descriverò il progetto realizzato, descrivendo l'architettura dell'applicazione, la tecnica di hosting, le tecniche di autenticazione e autorizzazione, il modello dati e le funzionalità. Nel Capitolo 5 illustrerò i risultati ottenuti rispetto sia ai requisiti funzionali che a quelli tecnici. Infine il Capitolo 6 analizzerò i possibili sviluppi futuri e miglioramenti del portale.

Capitolo 2

Tecnologie Usate

2.1 Introduzione

Lo sviluppo di un'applicazione web moderna dipende da una selezione attenta di tecnologie moderne che garantiscano robustezza, scalabilità, manutenibilità e sicurezza. L'applicazione sviluppata per questo progetto di tesi è stata scritta usando una combinazione di tecnologie che appartengono all'ecosistema Microsoft e standard aperti per l'autenticazione e lo scambio di dati. Nello specifico è stato utilizzato **.NET 9** come piattaforma di sviluppo, **C#** come linguaggio di programmazione principale, **Blazor Server** per il livello dell'interfaccia utente, Microsoft **SQL Server** per la gestione dei dati, **IIS (Internet Information Services)** per l'hosting e la gestione operativa e **Shibboleth** per l'identità federata e il single sign-on. Inoltre, l'applicazione utilizza API REST che restituiscono payload JSON per dati dinamici (per esempio i dati degli studenti di dottorato) che sono integrati tramite servizi asincroni e modelli tipizzati in C#.

Ognuna di queste tecnologie contribuisce a un differente strato architetturale: **.NET** e **C#** forniscono il modello computazionale e il linguaggio, **Blazor Server** gestisce la presentazione e il comportamento interattivo, **SQL Server** fornisce capacità di archiviazione persistenti e transazionali, **IIS** funge da piattaforma di hosting e superficie operativa, e **Shibboleth** applica l'identità federata a livello di autenticazione. Questo capitolo descriverà ogni elemento dello stack, spiegando i concetti chiave (per esempio SSO e SAML) e descriverà come i componenti sono integrati tra di loro.

2.2 La Piattaforma .NET 9 e il Linguaggio C#

2.2.1 Panoramica

La piattaforma **.NET** si è evoluta da un framework escusivamente per sistemi Windows a un runtime e SDK unificato, multi piattaforma e open-source. **.NET 9** rappresenta l'evoluzione a lungo termine più recente della linea .NET unificata, apportando ottimizzazioni runtime, miglioramenti cloud-native e un supporto più completo per carichi di lavoro moderni come servizi containerizzati e componenti di base AI/ML [19, 20]

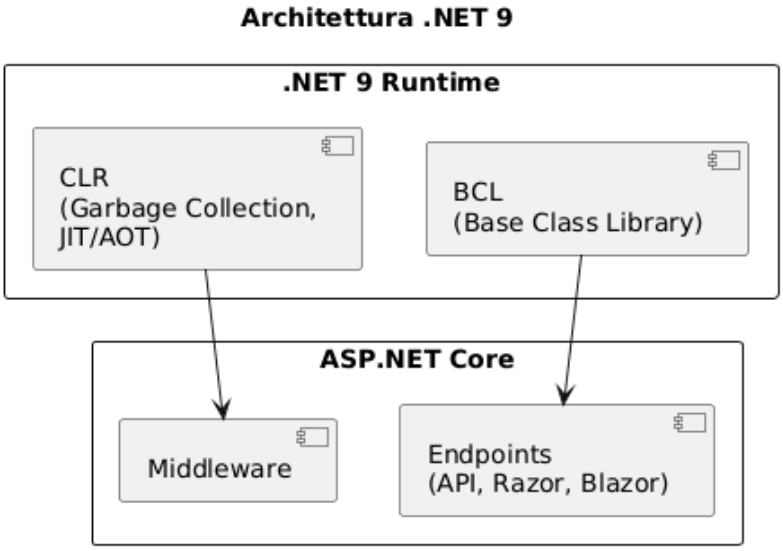

Figura 2.1: Architettura logica di .NET 9

La decisione di usare **.NET 9** per questo progetto è stata presa a causa di diverse motivazioni tecniche e pratiche: supporto degli strumenti in Visual Studio e nella CLI (interfaccia della riga di comando), la possibilità di pubblicare l'applicazione in diversi ambienti (Windows, Linux, container, ecc...) e una totale integrazione con ASP.NET Core e Entity Framework Core.

2.2.2 Architettura Runtime e componenti

Al centro della piattaforma si trova il **Common Language Runtime (CLR)**, che esegue codice gestito, fornisce la garbage collection, applica la sicurezza dei tipi e supporta la gestione delle eccezioni e dei thread.

Complementare al CLR, la **Base Class Library (BCL)** espone un ricco set di API per l'I/O, il networking, la crittografia, le raccolte, il threading e la globalizzazione. Sopra queste fondamenta di poggia **ASP.NET Core**, il framework web usato per ospitare gli endpoint HTML, i middleware e le applicazioni web. Il SDK .NET e la toolchain del compilatore (Roslyn per C#), forniscono servizi in fase di compilazione, come per esempio gli analizzatori, la generazione di codice e la pipeline di pubblicazione. **.NET 9** potenzia ulteriormente questi componenti con le opzioni di compilazione Ahead-of-Time (AOT), euristiche JIT a livelli migliorate e runtime trimming che riducono le dimensioni dei binari per le distribuzioni containerizzate. [21, 22]

2.2.3 Caratteristiche del linguaggio e modello di programmazione (C#)

Il linguaggio **C#** è stato impiegato in modo estensivo all'interno del progetto per l'implementazione dei servizi lato server, la definizione dei layer di accesso ai dati e la costruzione dei componenti Blazor. C# è un linguaggio di programmazione staticamente tipizzato e orientato agli oggetti, con funzionalità moderne che supportano modelli di programmazione asincroni e reattive, caratteristiche fondamentali per applicazioni web responsive.

Le funzionalità più rilevanti che sono state sfruttate in questo progetto comprendono:

- **Async/await**: usato in modo estensivo per eseguire operazioni di I/O non bloccanti, in particolare nelle chiamate HTTP verso API REST e nell'accesso al database tramite EF Core, migliorando la scalabilità e il throughput del sistema. [23]

- **LINQ (Language Integrated Query)**: utilizzato per esprimere trasformazioni e filtri su collezioni in memoria e query EF Core, aumentando la chiarezza del codice e riducendo la necessità di generare SQL boilerplate.

- **Records e immutabilità**: gli oggetti di trasferimento dati (DTO) e i view model sono stati modellati come record quando era richiesta semantica di valore e immutabilità.

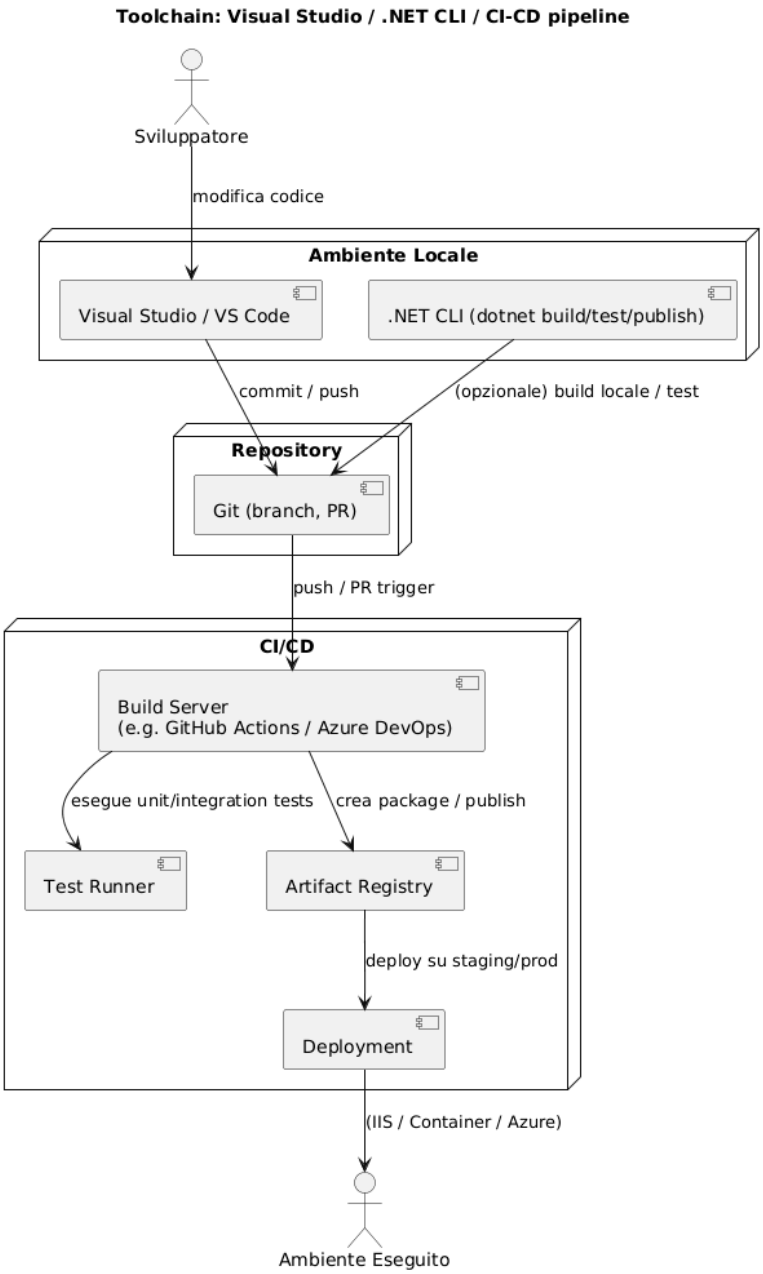

Figura 2.2: Schema della Toolchain Visual Studio/CLI/pipeline di build

- **Dependency injection**: ASP.NET Core integra il contenitore DI che garantisce una gestione pulita delle risorse utilizzandolo per registrare i servizi con diversi cicli di vita (singleton, scoped, transient).

Tutte queste caratteristiche permettono l'adozione di uno stack mono linguaggio che gestisca l'accesso ai dati, la logica di business e il comportamento dell'interfaccia utente, riducendo la complessità, il contesto cognitivo necessario e la manutenibilità del progetto.

Tabella 2.1: Principali componenti dello stack .NET 9

Componente	Descrizione	Note
CLR	Esecuzione codice gestito, GC, JIT/AOT	Fondamentale per performance e affidabilità
BCL	Librerie di base per I/O, networking, collezioni, sicurezza	Riduce codice personalizzato
ASP.NET Core	Middleware pipeline, routing, API, Razor, Blazor	Layer principale per hosting dell'applicazione
EF Core	ORM, migrazioni, LINQ, tracking	Facilita accesso ai dati, attenzione alle query complesse
Toolchain (.NET SDK)	Compilazione, analizzatori, CLI, hot reload	Migliora produttività e debugging
Kestrel / Hosting	Server HTTP e integrazione con IIS	Scelta consigliata per ambienti enterprise
Dependency Injection	Iniezione dipendenze, gestione lifetimes	Migliora testabilità e modularità

2.2.4 Considerazioni su prestazioni, compilazione AOT e architetture cloud-native

.NET 9 introduce la compilazione AOT (Ahead-of-Time) e ulteriori ottimizzazioni del runtime che migliorano sensibilmente i tempi di avvio e riducono il consumo di memoria, vantaggi particolarmente rilevanti per applicazioni Blazor Server ospitate lato server e microservizi eseguiti sotto orchestrazione containerizzata. [20, 21].

La compilazione AOT può anche essere applicata in modo selettivo (per esempio nel caso di carichi di lavoro sensibili al cold-start), mantenendo contemporaneamente i vantaggi della compilazione Just-in-Time per il codice più attivo.

Inoltre .NET 9 potenzia l'integrazione con gli strumenti di diagnostica e telemetria (come EventCounters, OpenTelemetry) facilitando sia il monitoraggio dei sistemi in produzione che l'ottimizzazione delle prestazioni nei deployment cloud.

2.2.5 Esperienza di sviluppo e strumenti

Visual Studio e l'interfaccia della riga di comando (CLI) di .NET costituiscono una toolchain integrata per lo sviluppo di applicazioni Blazor Server; questi strumenti supportano in modo nativo attività fondamentali come la modifica del codice, il debug interattivo, il supporto all'hot reload e la pubblicazione su ambienti locali o in cloud.

In particolare, l'hot reload per componenti Razor e Blazor consente di aggiornare l'interfaccia utente e la logica applicativa senza dover ricompilare o riavviare l'intera applicazione; questo approccio accelera notevolmente lo sviluppo applicativo favorendo iterazioni rapide e una maggiore fluidità nel processo di design e refactoring, in quanto le modifiche al progetto sono immediatamente riscontrabili nel browser senza dover riavviare l'applicazione. [24].

L'interfaccia della riga di comando di .NET offre inoltre comandi per la gestione del ciclo di vita dell'applicazione, inclusi `dotnet build`, `dotnet run`, `dotnet publish` e `dotnet watch`, facilitando l'integrazione con ambienti di sviluppo automatizzati grazie anche alla creazione di script.

2.3 Blazor Server e architettura UI dell'applicazione

2.3.1 Perché Blazor Server?

Blazor è un framework moderno per interfacce utente che consente agli sviluppatori di creare interfacce web interattive e complesse utilizzando C# e Razor al posto di JavaScript [25]. Esistono due principali modelli di hosting: Blazor WebAssembly (client-side) e Blazor Server (server-side).[26]

Per questa applicazione è stato scelto Blazor Server per i seguenti motivi:

- **Esecuzione centralizzata**: la logica di business viene eseguita sul server, dove ha accesso diretto alle risorse (database, servizi enterprise), evitando l'esposizione di logica o segreti lato client.

- **Requisiti minimi lato client**: il carico iniziale sul client è ridotto poiché non è necessario scaricare WebAssembly; è sufficiente disporre di un browser moderno e una connessione compatibile con SignalR.
- **Modello di sicurezza semplificato**: le operazioni sensibili rimangono tutte lato server, riducendo così le possibilità d'attacco lato client.
- **Stack C# unificato**: grazie allo stack tecnologico unificato gli sviluppatori hanno la possibilità di riutilizzare modelli, logiche e servizi tra i lato server e client.

Queste caratteristiche rispettano perfettamente i vincoli di progetto, il deployment istituzionale, l'autenticazione forte (in questo caso Shibboleth) e la centralizzazione dei dati, favorendo un'esecuzione lato server [27].

Caratteristica	Blazor Server	Blazor WebAssembly
Esecuzione	Sul server	Nel browser
Reattività	Dipende da latenza	Molto alta
Payload iniziale	Basso	Alto (WASM + DLL)
Sicurezza	Molto alta	Espone il codice al client
Scalabilità	Richiede molte connessioni	Scalabile lato client

Tabella 2.2: Differenze tra Blazor Server e Blazor WebAssembly

2.3.2 Circuiti SignalR e gestione dello stato

Blazor Server mantiene una connessione persistente e bidirezionale tra il browser e il server tramite **SignalR**. Sul server ogni client connesso è associato con un **circuito** che immagazzina lo stato dei componenti e gestisce l'invio degli eventi. Mentre questo permette interattività in tempo reale, sottintende che ogni utente consuma memoria sul server e potenzialmente thread/risorse a seconda del carico di lavoro. L'applicazione per essere scalabile deve:

- minimizzare lo stato del circuito e mantenere i componenti leggeri
- usare operazioni asincrone per le operazioni legate all'I/O per evitare la saturazione dei thread
- delegare operazioni pesanti o di lunga durata a servizi in background (IHostedService / code) invece di bloccare i gestori degli eventi dell'interfaccia utente

Un uso attento dei servizi con ciclo di vita scoped e dei pattern di rilascio delle risorse (disposal) previene la perdita di stato tra circuiti e riduce il consumo non necessario di memoria. [27].

2.3.3 Modello dei componenti, ciclo di vita e Razor

L'interfaccia utente Blazor è composta da **componenti Razor** riutilizzabili, file '.razor' che uniscono HTML e C# usando direttive Razor; per citarne alcuni '@page' indica che il componente è una pagina, '@inject' per includere servizi esterni nel componente, '@bind' per associare una variabile a un componente dell'interfaccia utente e '@code' che indica il blocco che contiene il codice C#. [28] I componenti espongono parametri, callback di eventi e metodi del ciclo di vita come OnInitializedAsync, OnParametersSetAsync e OnAfterRenderAsync, consentendo un controllo preciso sull'inizializzazione, la gestione dei parametri e il comportamento successivo al rendering. Nell'applicazione, i compiti della UI sono frammentati in componenti piccoli e testabili (navigazione, liste, dettagli, finestre modali) per migliorare la manutenibilità e la riusabilità. [29].

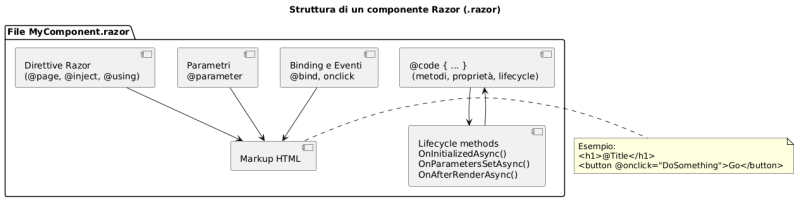

Figura 2.3: Struttura di un componente Razor

2.3.4 Integrazione di Sicurezza e Autenticazione

Blazor Server è perfettamente integrato con lo stack di autenticazione e autorizzazione di ASP.NET Core; l'applicazione usa un middleware di autenticazione per accettare le informazioni principali fornite dal livello di hosting (IIS con integrazione con Shibboleth). L'autorizzazione viene poi applicata tramite regole e attributi basati su ruoli e claim, sia sui singoli componenti che sugli endpoint. Dato il fatto che Blazor Server esegue tutta la logica lato server, i controlli di autorizzazione sono eseguiti prima delle operazioni sensibili, contribuendo così a mantenere uno standard di sicurezza elevato.

2.3.5 Compromessi tra performance e scalabilità

Blazor Server permette un'elevata produttività nello sviluppo e vantaggi di sicurezza ma necessita di specifiche scelte architetturali per un alta concorrenzialità:

- **Densità di connessioni**: ogni circuito SignalR ha associata un'impronta di memoria; per gestire un elevato numero di utenti può essere necessario adottare lo scaling orizzontale (istanze server multiple) e configurare sessioni sticky o uno stato distribuito dei circuiti (tramite backplane Redis).
- **Sensibilità alla latenza**: poiché gli eventi dell'interfaccia utente comportano un round-trip verso il server, la latenza geografica può influire negativamente sull'esperienza utente.
- **Rendering ottimizzato**: l'uso di override del metodo *ShouldRender* e una granularità fine dei componenti riducono i diff dell'interfaccia utente non necessari e il traffico di rete.

Nel caso d'uso specifico di questa applicazione, queste ottimizzazioni architetturali non si sono rese necessarie; l'applicazione è di tipo istituzionale ed è rivolta a un numero limitato di utenti, che per la maggior parte è localizzata in una zona geografica ristretta. Questo ha permesso di evitare configurazioni complesse come lo scaling orizzontale o i backplane, mantenendo comunque un'esperienza utente reattiva e fluida.

2.4 Microsoft SQL Server e il Livello Dati

2.4.1 Modello relazionale e garanzie transazionali

Microsoft SQL Server è un RDBMS maturo che supporta transazioni ACID, query complesse, strategie di indicizzazione, stored procedure e funzionalità avanzate di disponibilità (per esempio Always On Availability Groups). È particolarmente adatto a deployment istituzionali che richiedono una robusta integrità dei dati, strumenti per il backup e ripristino e un'amministrazione centralizzata. [30]

2.4.2 Integrazione tramite Entity Framework Core

L'applicazione usa **Entity Framework Core** (EF Core) come livello ORM per mappare le entità C# in tabelle relazionali. EF Core fornisce funzionalità di migrazione, tracciamento delle modifiche, query LINQ-to-Entities e la possibilità di utilizzare query SQL pure quando si rende necessario per operazioni critiche a livello prestazionale.[31]. I vantaggi includono una tipizzazione forte nelle operazioni sul database, riduzione del codice boilerplate e transazioni integrate con il ciclo di vita delle richieste in ASP.NET Core.

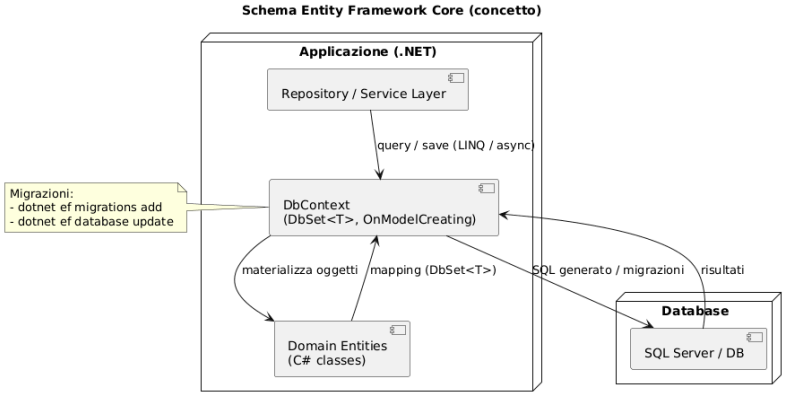

Figura 2.4: Schema Entity Framework Core

2.4.3 Sicurezza, prestazioni e funzionalità operative

SQL Server offre supporto integrato per la crittografia (Transparent Data Encryption, Always Encrypted), il controllo degli accessi basato sui ruoli, l'auditing e la gestione granulare dei permessi. Per le prestazioni possono

essere adottate delle strategie di indicizzazione (clustered, non-clustered, columnstore) e di OLTP in-memory quando appropriato. Dal punto di vista operativo i job di SQL Server Agent e i piani di manutenziona automatizzano backup, indicizzazione e verifiche di consistenza, elementi fondamentali per la preparazione per l'ambiente di produzione.

Tabella 2.3: Funzionalità avanzate di Microsoft SQL Server

Funzionalità	**Descrizione**	**Vantaggi / Note**
Transazioni ACID	Integrità dei dati e gestione concorrente	Essenziale nelle operazioni critiche
T-SQL	Estensione procedurale del linguaggio SQL	Utile per stored procedure e batch server-side
Sicurezza	Row/column security, TDE, Always Encrypted	Supporta compliance e protezione dati sensibili
Indicizzazione	Indici rowstore/columnstore	Migliora drasticamente performance di query
Query Optimizer	Scelta automatica piani di esecuzione	Fondamentale per performance; usare EXPLAIN
In-memory OLTP	Tabelle in memoria per OLTP ad alte prestazioni	Riduce latenza; utile per carichi elevati
Availability Groups	Replica e failover ad alta disponibilità	Minimizza downtime e protegge i dati

2.5 Internet Information Services (IIS)

2.5.1 Ruolo come Hosting e reverse proxy

IIS viene usato come front-end di hosting per l'applicazione Blazor Server. Riceve il traffico HTTP(S) in ingresso, termina la connessione TLS, applica eventualmente filtri alle richieste e inoltra le richieste al server Kestrel tramite il modulo ASP.NET Core (ANCM). L'uso di IIS offre vantaggi amministrativi come la gestione dei certificati e le funzionalità avanzate di log e monitoraggio, particolarmente utili negli ambienti basati su Windows.[32]

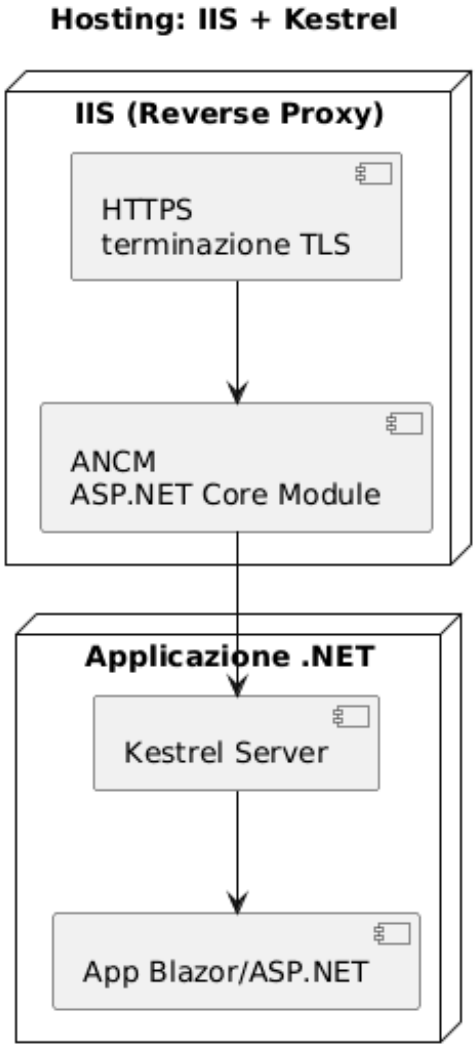

Figura 2.5: Diagramma architettura di hosting IIS + Kestrel

2.5.2 Pool di applicazioni e isolamento

IIS utilizza pool di applicazioni per isolare i processi; la configurazione dei pool controlla la concorrenza, le politiche di riciclo e l'identità del processo nel quale l'applicazione viene eseguita. Un'adeguata configurazione del pool garantisce che un'istanza difettosa di un'applicazione non interferisca con altri siti ospitati sullo stesso server.

2.5.3 Monitoraggio e diagnostica

La registrazione di IIS (log W3C), i log degli eventi di Windows e i contatori delle prestazioni sono stati utilizzati per monitorare i pattern delle richieste, l'utilizzo della CPU e della memoria. Inoltre, gli endpoint di diagnostica di ASP.NET Core e il sistema di logging sono stati integrati nella telemetria centralizzata per abilitare la manutenzione proattiva e la pianificazione della capacità.

2.6 Shibboleth, SAML e Autenticazione Federata

2.6.1 SSO e identità federata

Single Sign-On (SSO) significa che un utente si autentica una volta e ottiene accesso a un certo numero di applicazioni senza il bisogno di riautenticarsi. L'**identità federata** estende questo concetto oltre i confini organizzativi, abilitando relazioni di fiducia tra provider di identità (IdP) e provider di servizi (SP). Quando le identità vengono gestite centralmente da un'istituzione, il Single Sign-On (SSO) e la federazione riducono la proliferazione di credenziali, migliorano l'usabilità per l'utente e favoriscono l'applicazione centralizzata delle policy.

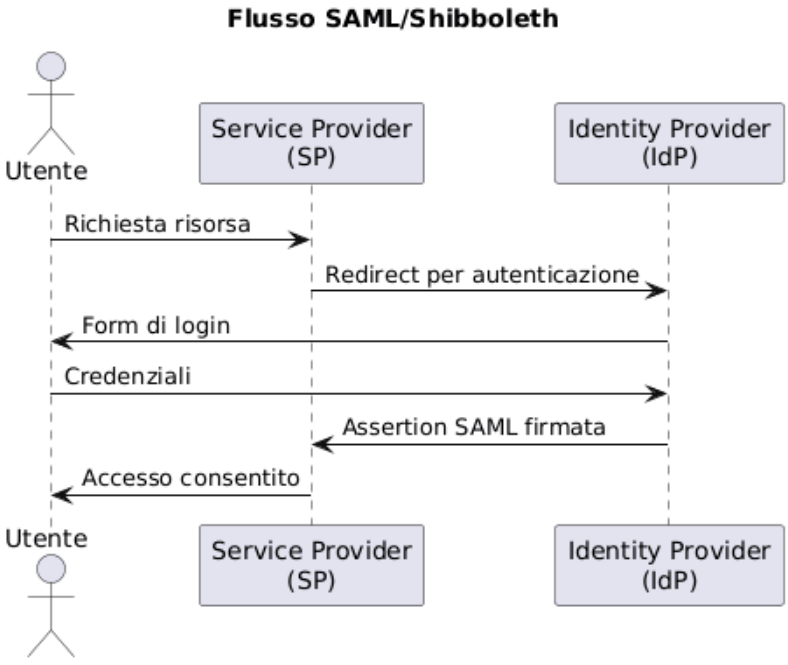

Figura 2.6: Flusso SAML/Shibboleth

2.6.2 SAML 2.0 e Shibboleth

Il **Security Assertion Markup Language (SAML) 2.0** è uno standard basato su XML per lo scambio di asserzioni di autenticazione e autorizzazione tra IdP e SP. **Shibboleth** è un'implementazione open-source di identità federata basata su SAML ampiamente usata in istituzioni accademiche e di ricerca. Il flusso tipico di SAML consiste in:

1. Un utente richiede l'accesso a una risorsa protetta al Service Provider
2. IL Service Provider reindirizza l'utente all'IdP per l'autenticazione
3. L'IdP autentica l'utente, per esempio tramite credenziali istituzionali, e emette un'asserzione SAML firmata
4. L'utente è reindirizzato al Service Provider, che valida la firma dell'asserzione SAML ed estrae le claim (attributi)
5. Il Service Provider stabilisce una sessione locale oppure associa l'asserzione a un account utente interno e concede l'accesso

Questo meccanismo garantisce che le credenziali siano gestite esclusivamente dall'IdP autorevole, consentendo ai Service Provider di fidarsi dell'esito dell'autenticazione e di utilizzare gli attributi utente secondo necessità. [33]

2.6.3 Integrazione in questo progetto

Nell'architettura implementata, Shibboleth funge da ponte esterno SP/IdP davanti all'applicazione ospitata su IIS. Il software Shibboleth Service Provider, implementato a livello di ateneo, valida le asserzioni SAML e inserisce gli attributi utente nel contesto della richiesta (tramite intestazioni HTTP). Il middleware di autenticazione di ASP.NET Core è configurato per accettare queste intestazioni come principal e per mappare gli attributi in claim utilizzati dall'applicazione per le decisioni di autorizzazione. Questo approccio riduce al minimo la gestione diretta delle credenziali da parte dell'applicazione e sfrutta la gestione centralizzata delle identità istituzionali.

2.6.4 Privacy e rilascio degli attributi

La gestione dell'identità federata richiede politiche rigorose di rilascio degli attributi per la tutela della privacy degli utenti. Il provider di identità (IdP) emette solamente gli attributi necessari (per esempio matricola, indirizzo email e ruolo istituzionale); le regole di mappatura e le policy locali garantiscono che queste informazioni personali siano gestite in conformità con le normative istituzionali.

2.7 API REST e REST APIs e Scambio Dati JSON

2.7.1 Principi REST e utilizzo HTTP

L'applicazione utilizza endpoint RESTful che forniscono dati relativi a dottorandi e informazioni istituzionali associate. REST è uno stile architetturale basato su interazioni stateless e sull'uso di verbi HTTP standard (GET, POST, PUT, DELETE) per manipolare risorse identificate da URI [34]. L'assenza di stato semplifica lo scaling orizzontale e la cache, risultando ben allineata con l'infrastruttura HTTP. In questo caso specifico le API utilizzate sono in sola lettura e forniscono in modo asincrono i dati.

2.7.2 Payload JSON e deserializzazione tipizzata

Le API restituiscono payload JSON che vengono analizzati e deserializzati in modelli C# fortemente tipizzati utilizzando la libreria System.Text.Json. La tipizzazione rigorosa consente controlli in fase di compilazione e semplifica la logica applicativa successiva. Ad esempio, un record semplificato di dottorando può essere rappresentato come:

```
public DidDottorato(string titolo, string descrizione,
int cicloAppartenenza, int annoDott, string tematica,
Argomento arg, string dataInizioAttivita,
string dataPresuntaFineAttivita, string tipologiaBorsa,
int matricolaTutore, List<DidUtente> cotutori,
string corsoDiStudio, string dipartimento)
```

HttpClient viene utilizzato per inviare richieste asincrone; le risposte vengono validate (codici di stato, gestione degli errori), deserializzate e memorizzate nel database per ridurre il carico sulle API.

2.7.3 Livello di servizio e associazione all'interfaccia utente

Nel modello Blazor Server, classi di servizio dedicate incapsulano le chiamate alle API e la logica di caching. I servizi vengono iniettati nei componenti e sono responsabili della trasformazione dei DTO in modelli

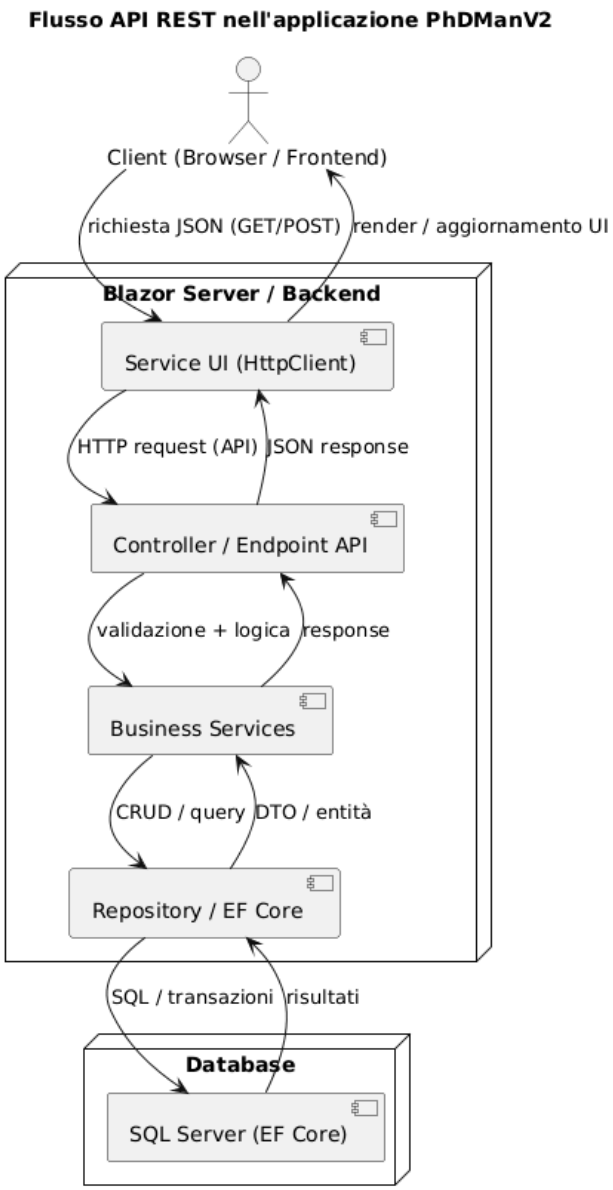

Figura 2.7: Schema di funzionamento delle API REST

di dominio o di visualizzazione per l'interfaccia. Questa separazione segue i principi SOLID e rende semplice l'aggiunta di test unitari per la logica di trasformazione dei dati.

2.8 Conclusioni

La combinazione di **.NET 9**, **C#**, **Blazor Server**, Microsoft **SQL Server**, **IIS**, **Shibboleth**, e API REST costituisce una base moderna, sicura e manutenibile per l'applicazione sviluppata in questa tesi. Ogni tecnologia svolge un ruolo specifico: **.NET 9** fornisce un runtime ad alte prestazioni e un ecosistema linguistico avanzato; Blazor Server consente un'interfaccia utente unificata in C# con interattività in tempo reale; SQL Server offre persistenza transazionale dei dati; IIS garantisce hosting e funzionalità operative di livello enterprise; Shibboleth supporta il Single Sign-On federato nei contesti accademici; REST/JSON abilita l'integrazione disaccoppiata con le fonti dati istituzionali. Insieme, queste tecnologie permettono di costruire un'architettura di sistema sicura, testabile e facilmente manutenibile, in linea con le migliori pratiche attuali per ambienti enterprise e accademici.

Capitolo 3

Specifiche funzionali dell'applicativo PhdMan v2

3.1 Introduzione e specifiche del progetto

Il sistema **PhDManV2** (abbreviazione di PhD Manager Versione 2) è stato concepito per fornire una piattaforma per gestire l'intero processo annuale di revisione degli studenti di dottorato.

Il processo attuale, nonostante sia ben collaudato, presenta diverse limitazioni causate dall'uso di procedure manuali, uso di fogli Excel e raccolta di dati da una serie di siti e piattaforme esterne; proprio questa frammentazione di informazioni su più sistemi (portale della didattica, cruscotto ScuDo, IRIS e PhDMan) crea inefficienze, duplicazione dei dati e difficoltà nella gestione del processo nel suo complesso.

L'obiettivo principale del progetto è quindi **digitalizzare e automatizzare il ciclo di revisione** garantendo la tracciabilità in ogni stato, consistenza dei dati reperiti dalle varie fonti e visibilità dei dati dipendente dal ruolo dell'utente.

PhDManV2 permetterà la gestione di tutte le attività legate al processo di revisione, dalla raccolta delle presentazioni dei dottorandi e la valutazione dei tutori alla valutazione della commissione e la preparazione dei dati per la decisione finale del Collegio, in un ambiente unico, sicuro e trasparente.

3.2 Contesto Operazionale

Il processo di revisione di un dottorando tipicamente ha luogo una volta all'anno, ma le tempistiche possono variare in quanto la data di inizio degli anni accademici dei dottorandi può essere diversa. In prossimità della fine di ogni singolo anno accademico viene avviato il processo di revisione che coinvolgerà più attori: lo studente di dottorato, il tutore e gli eventuali cotutori, i membri delle Commissioni, i membri del Collegio e il Coordinatore del Corso di Dottorato. Nel sistema inoltre deve essere possibile gestire più revisioni in contemporanea, relative a dottorandi di diversi cicli e/o anni, mantenendo la corretta associazione con la commissione e le valutazioni assegnate.

3.3 Obiettivi Funzionali Principali

PhDManV2 è concepito come una piattaforma digitale integrata che trasforma la gestione delle valutazioni degli studenti di Dottorato da una procedura frammentata e coordinata manualmente in un processo tracciabile e basato sui dati; i suoi obiettivi rispecchiano la necessità di semplificare il flusso operativo e fornire informazioni affidabili e aggiornate per il processo decisionale.

3.3.1 Centralizzazione delle Informazioni

Uno degli obiettivi principali di PhDManV2 è la **centralizzazione di tutti i dati e documenti** legati al processo annuale di revisione dei Dottorandi. Attualmente le informazioni sono contenute in sistemi differenti (il portale della didattica, il cruscotto ScuDo, IRIS e Pauper) e queste vengono estratte e combinate manualmente. Questo porta a inconsistenze, duplicazione delle attività e a inefficienze.

Il nuovo sistema dovrà:

- Consolidare in un unico repository i dati di tutti gli attori coinvolti (studenti, tutori, cotutori, commissari e membri del collegio) e di tutte le valutazioni.
- Eliminare ogni richiesta di dati ridondanti, sfruttando invece sistemi esistenti del Politecnico di Torino attraverso API o importazione di dati da Database esterni.

- Mantenere un record univoco (**single source of truth o SSOT**) per ogni studente di Dottorato che evolverà nel tempo, mostrando la progressione nel corso degli anni e dei cicli di revisione.
- Fornire agli utenti autorizzati informazioni aggiornate e consistenti in qualsiasi momento senza la necessità di ricercare le informazioni su altre piattaforme.

3.3.2 Integrazione con Fonti Dati Istituzionali

PhDManV2 dovrà operare come parte di un più largo **ecosistema di funzionalità** del Politecnico di Torino. Lo scopo è leggere a aggiornare le informazioni rilevanti da altri database istituzionali per assicurarsi l'allineamento dei dati e evitare discrepanze.

Nello specifico:

- da **Pauper/Gesd** il sistema recupererà i dati anagrafici istituzionali dello studente (matricola da studente, matricola da dottorando, indirizzo email, tipologia di dottorato) e tutti i dati legati al dottorato (come per esempio tutore, cotutori, anno, ciclo, argomento).
- da **IRIS** verranno importati i dati riguardo le pubblicazioni e gli indicatori bibliometrici come le classifiche delle riviste e l'indicatore R.
- dal sistema ScuDo otterrà i dati riguardo le ore di lezione seguite, l'elenco dei corsi tenuti e le loro rispettive ore.

L'integrazione verrà effettuata tramite chiamate ad API REST o query a database istituzionali nel rispetto di tutte le regole di sicurezza e accesso.

3.3.3 Pagine Personalizzate e Accesso Basato sui Ruoli

Data la diversità di tipologie di utenti il sistema dovrà mostrare **cruscotti personalizzati** che riflettano le necessità specifiche per ogni categoria.

- Il cruscotto del **Dottorando** mostrerà i risultati degli anni passati e lo stato attuale del processo di revisione.
- Il cruscotto del **tutore** e dei cotutori mostrerà un elenco dei dottorandi assegnati con i dati relativi alle valutazioni passate, lo stato

della valutazione attuale, la relazione caricata e eventuali note da parte dello studente.

- Il cruscotto della **Commissione** mostrerà l'elenco degli studenti assegnati, i loro dati (tutti quelli già visibili al tutore) e la valutazione del tutore.
- Il cruscotto del **Collegio** mostra tutti i dati disponibili alle Commissioni con in più le valutazioni delle stesse.
- Il **Coordinatore del Corso di Dottorato** e il suo vice avranno accesso a tutta una serie di funzionalità per gestire il sistema come creare le Commissioni e assegnar loro i dottorandi, gestire i membri del Collegio e vedere i dati completi di tutti i dottorandi.

Ogni cruscotto dovrà mostrare i dati in modo conciso ed efficace, fornendo solo le informazioni rilevanti per facilitare la lettura dei dati e il processo decisionale.

3.3.4 Conservazione dei Dati

Il sistema sarà anche un **archivio a lungo termine** del percorso accademico di ogni studente di dottorato. Questo significa che ogni valutazione, giudizio e decisione dal primo all'ultimo anno, resterà accessibile nel sistema.

La piattaforma:

- terrà traccia di tutte le valutazioni passate di ogni studente.
- mostrerà una panoramica delle prestazioni, permettendo sia allo studente che ai revisori di valutare i progressi nel tempo.
- dovrà mettere a disposizione funzionalità per la reportistica e per l'esportazione dei dati per facilitare la valutazione del dottorando.

Questa possibilità di controllare lo storico della carriera dello studente aiuterà ad individuare situazioni reiterate (per esempio "warning" frequenti) e supporterà il processo di gestione del programma di dottorato.

3.3.5 Configurabilità e Flessibilità

Siccome i programmi di dottorato potrebbero cambiare, il processo di valutazione e i criteri di giudizio potrebbero anch'essi modificarsi nel

tempo. PhDManV2 deve quindi offrire una configurabilità amministrativa, permettendo agli utenti autorizzati di adattare il sistema senza la necessità di interventi tecnici.

Alcuni di questi elementi sono:

- La composizione e i ruoli delle Commissioni (con date di validità dei componenti).
- La composizione del Collegio.
- Le tempistiche delle varie fasi di valutazione con date di apertura e chiusura delle varie fasi, differenziate per dottorando e per Commissione.

Questa flessibilità permette una gestione a lungo termine e un'adattabilità della piattaforma a futuri cambiamenti organizzativi o di regolamenti.

3.3.6 Miglioramento dell'Esperienza Utente

Infine, PhDManV2 sarà progettato in base a principi di **usabilità e accessibilità**. Data la diversità dei suoi utenti, dallo studente di dottorato ai docenti, l'interfaccia dovrà essere intuitiva, responsive e conforme agli standard di accessibilità.

Gli obiettivi principali, in termini di usabilità, includono:

- Un layout chiaro e intuitivo, pensato per ridurre al minimo il tempo necessario per apprenderne le funzionalità.
- Percorsi di navigazione chiari all'interno della piattaforma.
- Compatibilità con dispositivi mobili per consentire agli utenti di operare sulla piattaforma con qualsiasi dispositivo.

3.4 Attori e Responsabilità

Il sistema prevede diverse tipologie di utenti, ognuno con un insieme predefinito di permessi e attività da svolgere.

3.4.1 Studente di Dottorato

I dottorandi rappresentano il punto di partenza del processo. Attraverso la piattaforma gli studenti potranno:

- Caricare la **presentazione annuale** (in formato pdf o PowerPoint).
- Indicare in un campo testuale eventuali commenti o annotazioni.
- Visualizzare e verificare la correttezza dei loro dati attuali riguardo le ore di lezione seguite, i corsi tenuti e le pubblicazioni.
- Vedere tutte le valutazioni indicate come "pubbliche" e il responso della valutazione finale.
- Leggere lo storico di tutte le valutazioni passate date sulla piattaforma.

3.4.2 Tutore e Co-tutori

Il tutore e gli eventuali co-tutori sono i responsabili scientifici dello studente di dottorato.

Le attività che possono svolgere all'interno del sistema sono:

- Inserire la loro valutazione testuale sulle prestazioni dell'anno dello studente.
- Scaricare il materiale caricato e le annotazioni del dottorando.
- Visualizzare le valutazioni pubbliche e il responso finale per tutti gli studenti supervisionati.
- Avere a portata di mano le valutazioni degli anni passati degli studenti supervisionati.

È importante sottolineare che il tutore e i co-tutori non hanno accesso alle valutazioni della Commissione, con l'eccezione di quelle indicate come "pubbliche".

3.4.3 Commissione

Le Commissioni sono composte da tre membri (uno dei quali viene indicato come presidente) e sono responsabili della valutazione dei dottorandi basandosi sulla presentazione caricata dal dottorando e sui suoi dati. Ogni Commissione potrà:

- Scaricare la presentazione del dottorando, leggere le eventuali annotazioni e leggere la valutazione del tutore.

- Valutare l'operato annuale del dottorando sia tramite campi strutturati che di testo libero; questi campi riguardano vari aspetti del lavoro atteso da parte del dottorando.

- Assegnare una **raccomandazione finale** per i membri del collegio che può essere "ammesso", "ammesso con warning" o "respinto".

3.4.4 Coordinatore del Corso di Dottorato

Il coordinatore e il suo vice supervisionano l'intero processo di valutazione dei dottorandi.

Attraverso la piattaforma hanno la possibilità di:

- Creare nuove commissioni, modificare quelle esistenti, modificare i membri che le compongono e modificare l'assegnazione dei dottorandi a ognuna.

- Assegnare nuovi membri al Collegio dei docenti o rimuoverne.

- Aprire o chiudere le fasi in cui i dottorandi possono caricare le loro presentazioni annuali e quelle in cui i tutori, cotutori e commissioni possono dare i loro giudizi.

- Visualizzare i dati di tutti i dottorati.

- Visualizzare e scaricare report.

3.4.5 Collegio dei Docenti

Il Collegio dei Docenti ha in carico la fase finale del processo. Dopo aver ricevuto tutte le valutazioni, il Collegio discuterà e formalizzerà la decisione finale per ognuno degli studenti. Con l'ausilio della piattaforma i membri possono analizzare tutti i dati a disposizione e registrare la loro decisione, la quale diventa parte integrante della storia permanente delle valutazioni dello studente.

3.4.6 Gestore del Sistema

Il gestore del sistema garantisce il corretto funzionamento della piattaforma. Tra le sue responsabilità c'è la gestione dei ruoli utente, il controllo la sicurezza informatica, delle prestazioni del sistema e mantenere i log e i backup in linea con le normative sulla privacy.

3.5 Descrizione del Processo di Revisione

Il processo annuale di revisione è diviso in 5 fasi principali:

3.5.1 Fase di Preparazione

Il Coordinatore (o il vice) definisce i periodi per la revisione per ogni gruppo di studenti basati sulle date di inizio di dottorato individuali.

Inoltre, definirà i membri delle commissioni e i dottorandi che dovranno essere valutati da queste, basandosi su criteri specifici come l'affinità tematica e su vincoli come l'assenza di pubblicazioni in comune tra il docente e il dottorando.

3.5.2 Caricamento delle Relazioni e Valutazioni da Parte dei Tutori

All'approssimarsi del periodo di revisione il Coordinatore comunica ai dottorandi e ai relativi tutori il periodo di tempo in cui potranno procedere con la loro parte del processo di revisione.

Nella data di inizio il Coordinatore abiliterà i dottorandi all'inserimento; da quel momento questi potranno caricare le loro presentazioni e i tutori potranno esprimere la loro valutazione.

Terminato il periodo per il caricamento, il Coordinatore provvederà con l'eliminazione del permesso all'inserimento.

3.5.3 Revisione da parte della Commissione

Le Commissioni avranno a disposizione sul portale il materiale necessario per poter organizzare le sessioni orali dei dottorandi. Al termine della discussione un membro della commissione (tipicamente il presidente) esprimerà il giudizio collettivo nel sistema.

Il form di valutazione dei revisori comprenderà:

- Campi strutturati che riguardano i vari aspetti del lavoro del dottorando (*Piano di ricerca*, *Slides*, *Presentazione Orale*, *Pubblicazioni*, *Insegnamenti* e *Corsi*) con valutazioni che possono avere valori predefiniti (*Ok*, *Problemi*, *Non applicabile*).
- Due campi testuali: uno pubblico, che sarà visibile anche al dottorando e al tutore/cotutori, e l'altro privato, visibile solo dal Collegio.
- Un campo strutturato con la valutazione globale (*Scarso*, *Discreto*, *Medio*, *Buono*, *Eccellente*)
- Un campo con una valutazione finale consigliata (*Ammesso*, *Ammesso con Riserva*, *Respinto*)

3.5.4 Delibera del Collegio di Docenti

Il Collegio dei Docenti avrà a disposizione una **vista aggregata** che combina tutti i dati disponibili che includono per esempio pubblicazioni, ore di insegnamento, corsi seguiti, presentazione caricata dal dottorando, valutazione del tutore e quella della commissione. Durante questo incontro il Collegio darà uno tra tre possibili responsi:

- **Ammesso**: lo studente può accedere all'anno successivo (se al primo o secondo anno) o all'esame finale (se del terzo anno).
- **Ammesso con Riserva**: lo studente non avrà immediatamente accesso all'anno successivo ma sarà sottoposto a un'ulteriore valutazione dopo 6 mesi.
- **Respinto**: lo studente non potrà accedere all'anno successivo ma dovrà ripetere l'anno attuale.

La decisione finale verrà salvata nel sistema e sarà immediatamente reperibile da tutte le parti in causa.

3.5.5 Chiusura e Archiviazione

Una volta terminato il ciclo di revisione, tutti i dati vengono resi disponibili in base ai criteri di visibilità. Il sistema conserverà l'**intera cronologia** delle valutazioni di ogni dottorando, offrendo una visione a lungo termine dei progressi accademici e dei risultati ottenuti.

3.6 Requisiti Tecnici e di Sicurezza

Il sistema sarà un'**applicazione web responsive**, accessibile da pc e mobile, con autenticazione tramite i sistemi di **Single Sign-On** dell'ateneo. Il sistema deve implementare un controllo degli accessi granulare, garantendo che ciascun utente visualizzi solo le informazioni pertinenti al proprio ruolo e al periodo di valutazione. Tutti i dati sensibili saranno protetti tramite crittografia e connessioni HTTPS sicure. Le revisioni chiuse non possono essere modificate, ma restano accessibili per consultazione e finalità statistiche. Tutti i trattamenti di dati personali devono essere conformi al Regolamento Generale sulla Protezione dei Dati (GDPR) dell'UE e alle politiche interne dell'Ateneo.

3.7 Conclusioni

PhDManV2 è progettato per modernizzare e semplificare la gestione delle revisioni annuali dei progressi dei dottorandi. La piattaforma proposta ridurrà significativamente il carico di lavoro manuale, migliorerà la qualità e la reperibilità dei dati e fornirà al Collegio dei Docenti strumenti analitici e di monitoraggio avanzati.

Capitolo 4

Analisi Funzionale e Tecnica di PhDManV2

4.1 Panoramica

PhDManV2 (abbreviazione di PhD Manager Versione 2) è una piattaforma web progettata per supportare la valutazione del rendimento annuale dei dottorandi. Questa è realizzata usando Blazor Server nell'ecosistema .NET, l'applicazione utilizza rendering lato server, la comunicazione in tempo reale e un modello dati solido per offrire un'esperienza reattiva e scalabile. Il sistema è progettato per operare all'interno dell'infrastruttura istituzionale integrandosi con il provider di identità federata Shibboleth e supportando il deployment come applicazione in IIS.

4.2 Architettura dell'applicazione

L'applicazione adotta un'architettura a livelli che separa le responsabilità in tra domini. Questa separazione migliora la manutenibilità e consente agli sviluppatori di operare sul sistema in termini modulari.

- Il **livello di presentazione (presentation layer)** è composto da componenti Razor renderizzati tramite Blazor Server; questi componenti, che incapsulano la logica dell'interfaccia utente, sono organizzati per ruolo e funzionalità. Ad esempio DottorandoPage.razor gestisce le interazioni degli utenti, mentre RevisioneTutore.razor quelle dei tutor.

- Il **livello della logica di business (business logic layer)** include classi di supporto che astraggono operazioni comuni come la gestione dei cookie, la lettura dei dati utente dai sistemi centrali di ateneo o la gestione dei ruoli, per permettere la riusabilità e semplificare i test.

- Il **livello di accesso ai dati (data access layer)** è implementata tramite Entity Framework Core. La classe `PhdReviewContext` definisce lo schema del database e fornisce accesso a entità come `Dottorando`, `AnnoDottorato`, `Commissione` e `Collegio`

4.3 Hosting e Routing

PhDManV2 è progettata per essere pubblicata come sottoapplicazione all'interno di una più ampia infrastruttura web del dipartimento. Questa scelta riflette la necessità di integrare la piattaforma con servizi esistenti (come per esempio il provider di autenticazione). L'hosting dell'applicazione nel sotto-percorso `/PhDManV2` introduce diverse considerazioni tecniche legate al routing, alla risoluzione delle risorse e alla compatibilità con Internet Information Services (IIS).

Per supportare questo modello di pubblicazione, l'applicazione usa il middleware `UsePathBase` fornito da ASP.NET Core. Questo middleware permette all'applicazione di interpretare tutte le richieste in arrivo come relative a uno specifico percorso base. In pratica questo significa che una richiesta a `https://dauin-webapp.polito.it/phdmanv2/MainPage` verrà internamente interpretata semplicemente come una chiamata a `/MainPage`, preservando l'integrità della logica di routing e al tempo stesso mantenendo la compatibilità con l'ambiente di hosting.

Questa configurazione non influisce solo sul routing delle pagine Razor ma anche sulla risoluzione delle risorse statiche, come i file JavaScript, i fogli di stile CSS e le immagini. Tutte le URL alle risorse devono essere precedute col path base, per garantire un corretto funzionamento: per esempio, l'hub Blazor SignalR, che abilita la comunicazione in tempo reale tra client e server, viene mappato su `/PhDManV2/_blazor` invece che sul percorso predefinito `/_blazor`. Nel caso in cui non venisse tenuto in considerazione questo adattamento si verificherebbero errori di connessione interrotta, stili mancanti e componenti non accessibili.

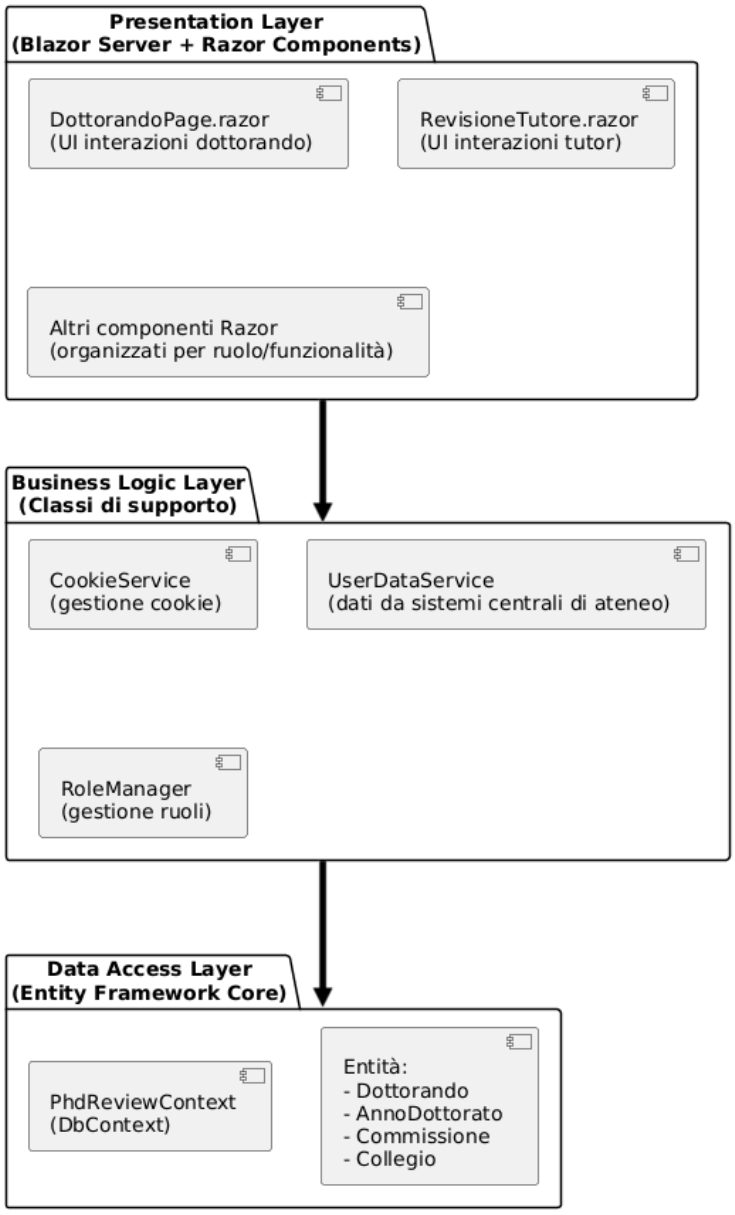

Figura 4.1: Architettura a livelli: flusso tra i componenti dell'interfaccia utente, i servizi e il database.

Questa metodologia di hosting influenza anche il flusso di autenticazione; l'applicazione si affida a Shibboleth per la gestione federata delle identità e per questo motivo deve essere raggiungibile tramite una struttura di URL coerente. La distribuzione come sotto-percorso garantisce che i reindirizzamenti di autenticazione, gli ambiti dei cookie e l'analisi delle intenzioni operino all'interno di uno spazio di nomi preciso, riducendo le probabilità di conflitti con altre applicazioni ospitate sullo stesso dominio.

Dal punto di vista della distribuzione, l'hosting sotto IIS richiede una

configurazione accurata del pool di applicazioni, dei binding del sito e delle regole di riscrittura. La sottoapplicazione deve essere registrata come applicazione all'interno del sito IIS con un percorso fisico che faccia riferimento all'output pubblicato del progetto Blazor Server. È inoltre necessario configurare i permessi di accesso alle risorse richieste e personalizzare il file web.config per garantire la compatibilità con il modello di rendering lato server di Blazor.

Path Logico	URL Fisica	Descrizione
/login	/PhDManV2/login	Punto di ingresso per l'autenticazione
/dottorandoPage	/PhDManV2/dottorandoPage	Cruscotto dottorando
/revisioneTutore	/PhDManV2/revisioneTutore	Interfaccia di valutazione del tutore
/revisioneCommissione	/PhDManV2/revisioneCommissione	Interfaccia di valutazione della Commissione
/revisioneCollegio	/PhDManV2/revisioneCollegio	Interfaccia di valutazione del Collegio
/dottorandiAdmin	/PhDManV2/dottorandiAdmin	Pannello amministratore per la gestione dei dottorandi
/_blazor	/PhDManV2/_blazor	Hub Blazor SignalR per la acomunicazione in real time
/css/app.css	/PhDManV2/css/app.css	Risorsa statica: foglio di stile principale
/imgs/logo.png	/PhDManV2/imgs/logo.png	Risorsa statica: Logo dell'applicazione

Tabella 4.1: Esempi di Configurazione di Routing

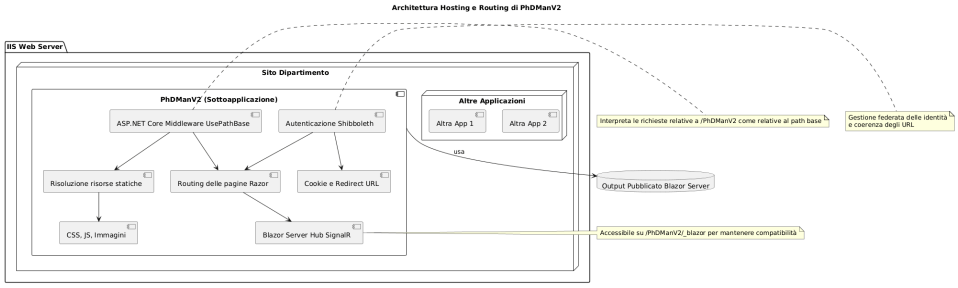

Figura 4.2: Topologia di hosting: mostra IIS, la struttura della sottoapplicazione e il flusso di routing dalle richieste esterne agli endpoint interni.

4.4 Autenticazione e Autorizzazione

L'autenticazione e l'autorizzazione sono una parte cruciale dell'architettura di qualsiasi piattaforma per garantire che gli utenti siano correttamente identificati e venga loro dato accesso solo alle risorse necessarie. PhDManV2 adotta un approccio ibrido che combina la gestione federata delle identità con un'autorizzazione basata sui claims, adatto alle esigenza sia dell'ambiente di produzione che di quello di sviluppo.

4.4.1 Gestione dell'Identità

In ambiente di produzione la piattaforma si integra con Shibboleth, un sistema di autenticazione federata ampiamente usato in istituzioni universitarie. Quando un utente accede all'applicazione Shibboleth inserisce gli attributi di identità negli header HTTP della richiesta; questi attributi includono tipicamente dati istituzionali come la matricola e l'email, e dati anagrafici come nome e cognome. Un componente middleware personalizzato legge questi header e costruisce un'identità basata sui claims, che viene poi utilizzata per tutta la sessione per applicare i controlli di accesso e personalizzare l'esperienza utente.

Nell'ambiente di sviluppo, dove gli header di Shibboleth non sono disponibili, l'applicazione simula l'autenticazione tramite variabili di sistema. Questo permette allo sviluppatore di testare comportamenti specifici per un ruolo senza affidarsi all'infrastruttura esistente o a dati reali. Le variabili vengono usate per emulare profili utente differenti, inclusi gli studenti, tutori e amministratori.

Questa strategia di autenticazione duale permette di ottenere flessibilità durante lo sviluppo e di mantenere un'elevata sicurezza e una solida integrazione in produzione.

4.4.2 Controllo degli Accessi Basato sui Ruoli

Una volta che l'utente è autenticato la piattaforma applica autorizzazioni basate sui ruoli per determinare quali componenti e quali azioni saranno accessibili. I Ruoli sono definiti nella configurazione del sistema e includono `dottorando`, `tutore`, `commissario`, `collegio` e `amministratore`. Ogni ruolo è associato con un set di permessi che governano l'accesso ai componenti Razor, gli endpoint delle API e gli elementi dell'interfaccia utente.

L'autorizzazione viene applicata tramite l'attributo `[Authorize(Roles = "...")]` di ASP.NET Core, utilizzato nei componenti Razor e nelle azioni dei controller; questo garantisce che gli utenti non possano eseguire operazioni o accedere a risorse che non siano comprese nel loro ambito assegnato.

Ad esempio solo gli utenti col ruolo `commissario` possono accedere alla pagina `/revisioneCommissione`, mentre solo gli utenti `amministratore` possono gestire le assegnazioni delle commissioni o esportare i dati di valutazione.

Il sistema supporta inoltre una risoluzione dinamica dei ruoli, operazione che viene fatta durante la fase di login. Dopo aver letto gli attributi di identità, il middleware verifica sul database i ruoli dell'utente basandosi sulla matricola; questi ruoli vengono poi eventualmente aggiunti come claim al cookie di autenticazione, permettendo un'applicazione coerente dei controlli di accesso in tutta l'applicazione.

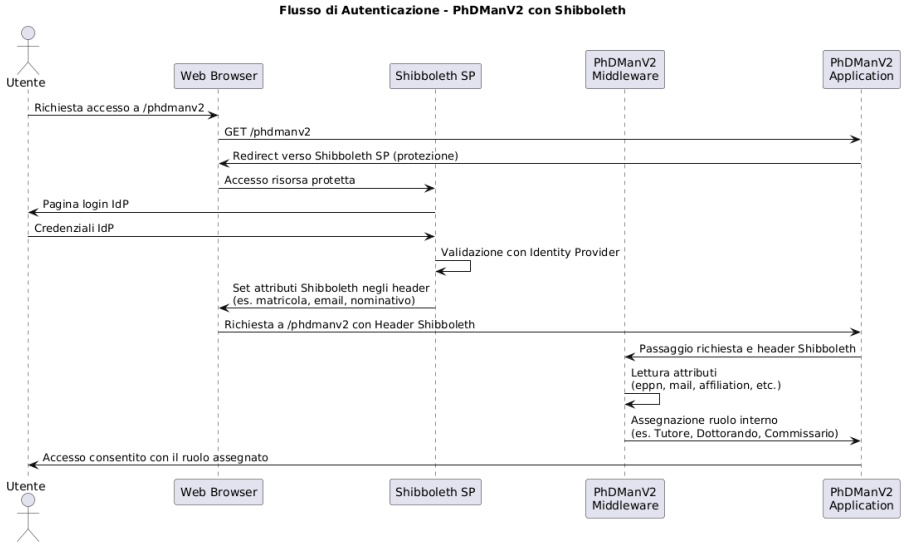

Figura 4.3: Flusso di autenticazione e autorizzazione.

Tabella 4.2: Ruoli e Permessi

Ruolo	Componenti e Pagine Accessibili	Permessi Concessi

`dottorando`	`/dottorandoPage`
	`/login`
	`/logout`
		Verificare i propri dati
		Upload della presentazione
		Vedere la valutazione finale

`tutore`	`/revisioneTutore`	Sottomettere la valutazione, Vedere i dati dello studente Vedere la presentazione
`cotutore`[1]	`/revisioneTutore`	Sottomettere la valutazione, Vedere i dati dello studente Vedere la presentazione
`commissario`	`/revisioneCommissione`	Vedere i dati e la presentazione dello studente Leggere la valutazione del tutore Compilare un form strutturato di valutazione
`collegio`	`/revisioneCollegio`	Accedere a tutti i dati dello studente Vedere tutte le valutazioni passate Dare la valutazione finale

`amministratore`	`/dottorandiAdmin`
	`/commissioneAdmin`
	`/collegioAdmin`
		Gestire gli utenti
		assegnare i ruoli
		esportare dati
		abilitare e disabilitare le valutazionie

[01]Inizialmente il ruolo di Cotutore doveva avere accesso in sola lettura alla valutazione del tutore. Nella versione finale dell'applicazione è stata data anche a loro la possibilità di valutare, rendendo quindi i due ruoli solo nominalmente differenti.

4.5 Funzionalità specifiche per ruolo

La piattaforma è progettata nell'ottica di un modello multi ruolo, nella quale a ciascun tipo di utente vengono concessi accesso a interfacce dedicate e a un insieme specifico di funzionalità. Questo garantisce che gli utenti visualizzino solo le informazioni e i controlli pertinenti, riducendo il carico cognitivo e minimizzando il rischio di azioni non autorizzate.

L'applicazione supporta cinque ruoli principali: **dottorandi**, **tutore/cotutore**, membri della Commissione, membri del Collegio e amministratori.

4.5.1 Studenti di Dottorato

Gli studenti usando la piattaforma potranno consultare tutti i dati che riguardano il loro dottorato, come il titolo del corso, il tutore e i cotutori, l'anno attuale di corso e il ciclo di appartenenza. Inoltre potranno verificare i loro dati per quanto riguarda le pubblicazioni, le ore di lezione seguite (sia quelle riferite alle soft skill che alle hard skill) e i corsi tenuti (con le relative ore). Questi dati possono essere aggiornati in modo asincrono tramite un pulsante che effettua una richiesta a delle API REST messe a disposizione dalla Direzione ISIAD del Politecnico. A regime il cruscotto mostrerà l'elenco degli anni passati, ognuno con le relazioni caricate, tutte le valutazioni e il giudizio finale. Per quanto riguarda l'anno in corso, dal momento in cui il dottorando verrà abilitato a farlo, l'interfaccia permetterà di caricare il file di relazione in formato PDF o Powerpoint, con un limite di dimensione di 50MB; inoltre potranno compilare un campo note per evidenziare eventuali discrepanze riscontrate nei loro dati o fornire informazioni di contesto ai revisori.

Una volta terminate le valutazioni, gli studenti potranno vedere le valutazioni della Commissione (configurate come pubbliche) e il responso finale del Collegio.

4.5.2 Tutori e Cotutori

I Tutori e i cotutori sono responsabili della valutazione qualitativa del lavoro dello studente; la loro valutazione si basa sul lavoro complessivo svolto dal dottorando durante l'anno e la finestra temporale in cui potranno valutarne il lavoro è la stessa che avrà lo studente per caricare

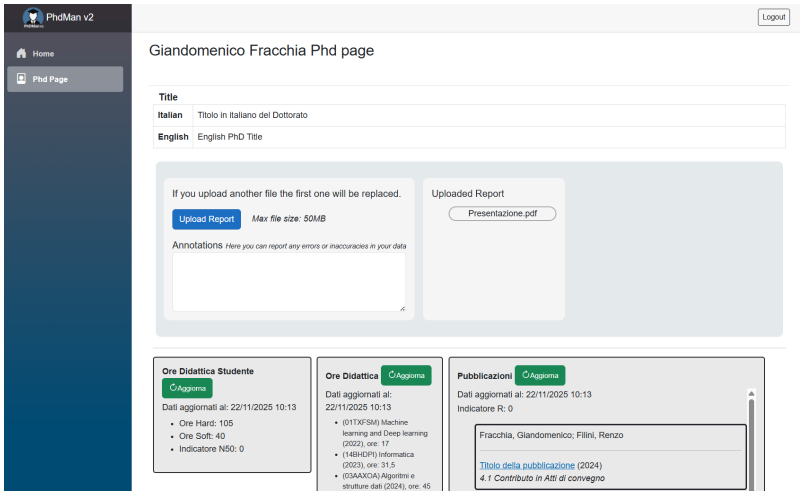

Figura 4.4: Esempio di pagina di un dottorando.

la relazione. La loro interfaccia comprende una lista filtrata di tutti gli studenti assegnati, con relativi dati riguardanti il loro dottorato. Sono visibili tutti i dati che già erano visibili al dottorando, con l'aggiunta del proprio giudizio per gli anni passati. Per l'anno in corso inoltre è presente un form per esprimere la propria valutazione; questa valutazione non è visibile al dottorando e viene salvata nell'entità TutoriGiudizio. I tutori e cotutori potranno aggiungere e modificare la loro valutazione durante tutta la finestra temporale e la valutazione è unica.

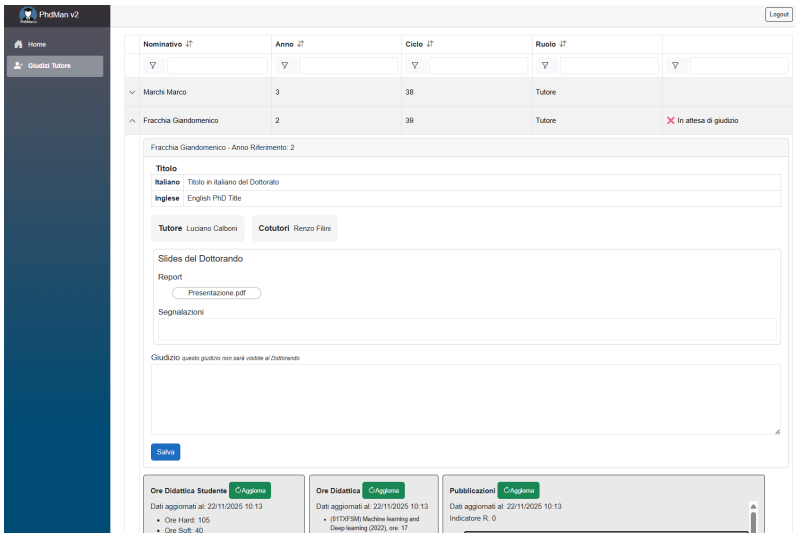

Figura 4.5: Esempio di pagina dei tutori.

4.5.3 Commissione

I membri della Commissione, dal momento in cui vengono abilitati a farlo, effettuano valutazioni utilizzando moduli generati dinamicamente; questi moduli sono basati su template memorizzati nella tabella CommissioneGiudizioDatiTemplate, che definiscono le etichette, i tipi di input, i valori di default e le regole di validazione di ciascun campo. L'interfaccia supporta sia valutazioni qualitative che categoriche, e i controlli di validazione garantiscono che i campi configurati come obbligatori siano compilati prima dell'invio. La valutazione è unica per tutta la Commissione e ogni membro può, a valle del colloquio con lo studente, modificare i giudizi. Le valutazioni inviate vengono salvate nella tabella CommissioneGiudizioDati e sono collegate ai dati del rispettivo anno del dottorando e alla commissione corrispondente. Questo approccio strutturato garantisce coerenza tra le valutazioni e supporta analisi e reportistica future.

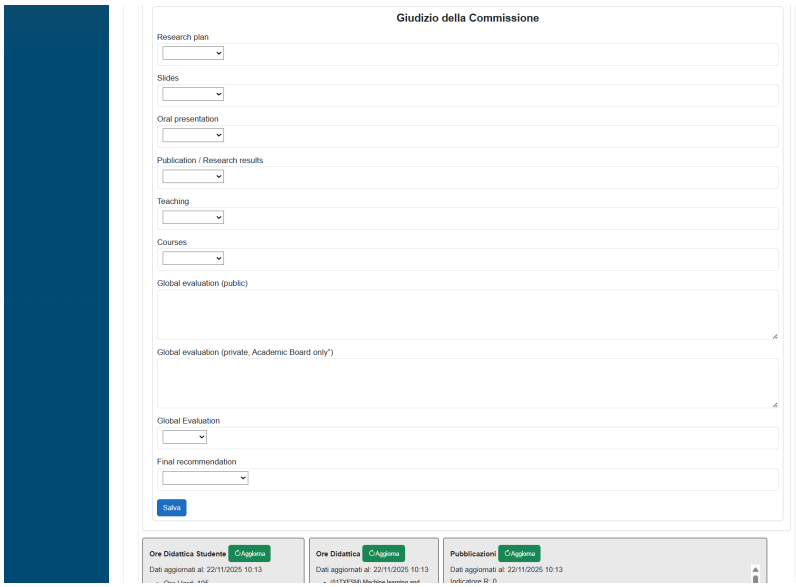

Figura 4.6: Esempio di pagina della Commissione.

4.5.4 Collegio dei Docenti

Il Collegio dei Docenti, o più semplicemente Collegio, emette il giudizio finale basato sulle valutazioni del tutore e della commissione. La loro interfaccia include tutti i dati del dottorando, la presentazione caricata e le annotazioni, il feedback del tutore e le valutazioni della commissione. Il Collegio può dare la valutazione finale tramite un form strutturato che

include da data di riunione, la valutazione finale ("ammesso", "ammesso con riserva" o "respinto") e le eventuali motivazioni. Queste informazioni sono salvate nella tabella CollegioGiudizio e la valutazione finale sarà visibile nel form dello studente e del tutore.

Figura 4.7: Dettaglio della pagina del Collegio.

4.5.5 Amministratori

Gli amministratori gestiscono e supervisionano l'intero processo di valutazione. Vengono indicati come amministratori il Coordinatore del Corso di Dottorato e il suo vice. Le loro interfacce includono:

CollegioAssegna.razor

Pagina per gestire i componenti del collegio. Ogni componente è assegnato in una certa fascia temporale ed è abilitato come membro del collegio solo tra quelle date.

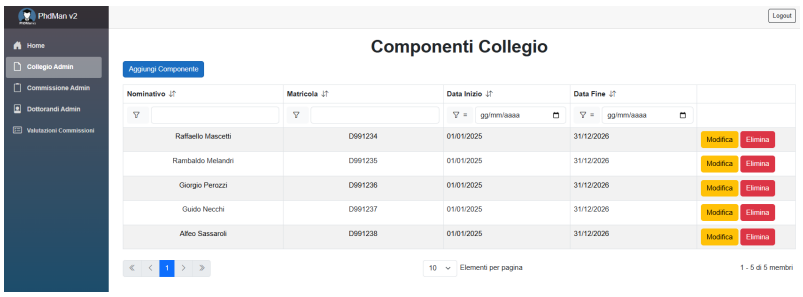

Figura 4.8: Esempio di pagina di assegnazione del Collegio.

commissioneAdmin.razor

Pagina per la creazione delle commissioni, dell'assegnazione dei membri e dei dottorandi alla stessa. Ogni commissione ha un identificativo (modificabile) e può essere abilitata o disabilitata ad esprimere il giudizio. I componenti hanno un ruolo e date di validità in cui possono

operare. Gli amministratori possono modificare in qualsiasi momento queste assegnazioni in ogni dettaglio. È inoltre possibile *nascondere* una commissione nel caso in cui non serva più averla a disposizione in pagina; rimane comunque possibile ripristinarle cliccando su "mostra commissioni nascoste" e cliccando su "mostra commissione". Nella stessa pagina è possibile esportare in un file csv l'elenco di tutti gli indirizzi email dei membri attivi di tutte le commissioni, nel caso sia necessario inviare comunicazioni.

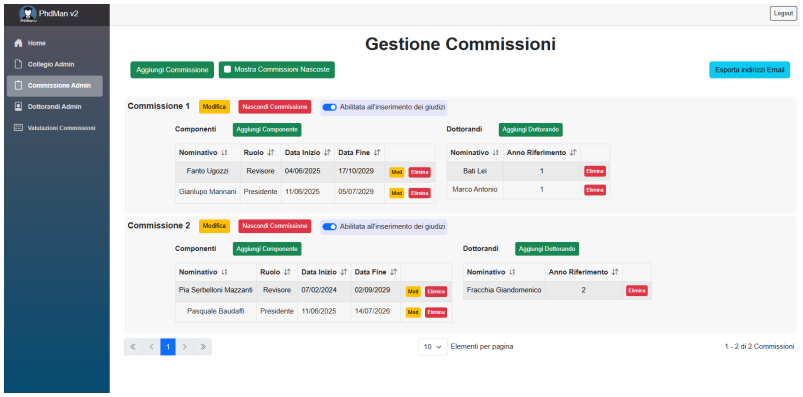

Figura 4.9: Esempio di pagina di assegnazione delle Commissioni.

dottorandiAdmin.razor

In questa pagina gli amministratori hanno l'elenco completo dei dottorandi attivi. Le informazioni immediatamente disponibili sono il nominativo, le matricole, il ciclo di appartenenza, le date di inizio e fine ciclo, se è stato già caricato la presentazione per l'anno attuale e se lo studente è abilitato all'inserimento della presentazione. Nel dettaglio di ogni studente è possibile vedere, per ogni anno di dottorato, il titolo del dottorato, il tutore, i cotutori; poi, se presenti, è possibile vedere la presentazione con le annotazioni, il giudizio del tutore, della commissione e del collegio. Alla fine del dettaglio è possibile vedere, ed eventualmente aggiornare, i dati riguardo le ore di didattica da studente, i corsi tenuti con le rispettive ore e i dati sulle pubblicazioni. Inoltre, è possibile aggiornare in maniera asincrona i dati di tutti i dottorandi tramite una API REST fornita dalla direzione ISIAD; data la mole di dati letti l'operazione richiede un certo tempo ed è stato necessario renderla un'operazione manuale eseguita dagli amministratori.

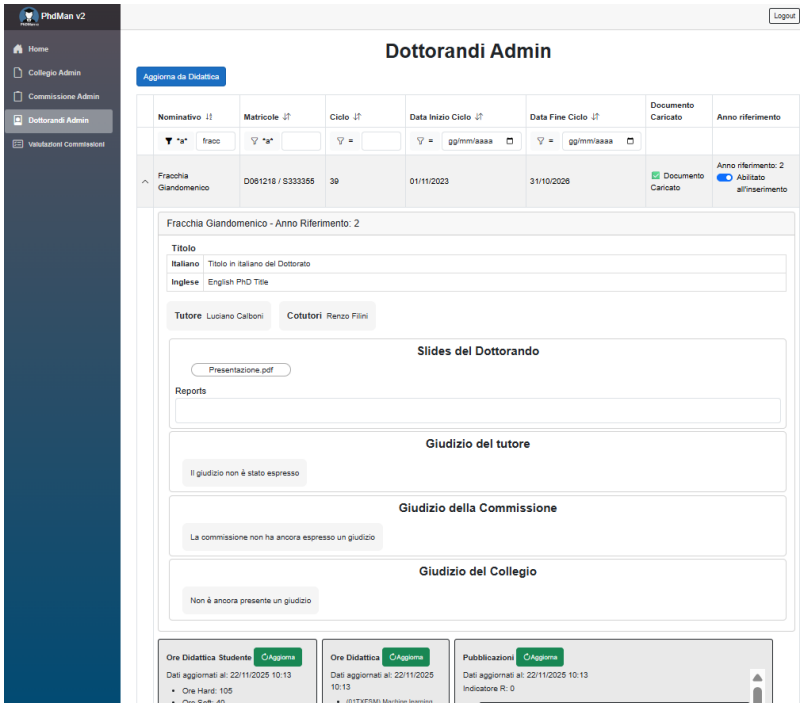

Figura 4.10: Esempio di pagina di gestione dei dottorandi.

dottGiudiziComm.razor

In questo cruscotto è disponibile un dettaglio di tutte le valutazioni date dalle commissioni a ogni dottorando in un'unica tabella. Questo dettaglio è esportabile in excel tramite una funzione apposita. Nel dettaglio di ogni studente sono presenti gli stessi dati della pagina dottorandi.admin

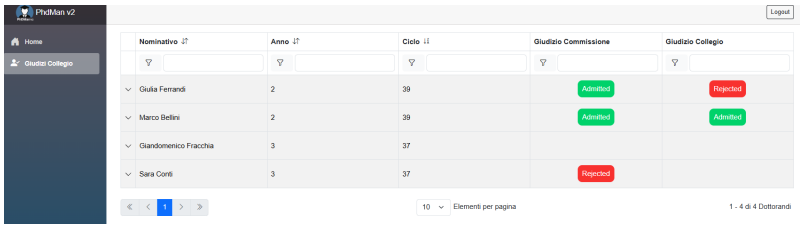

Figura 4.11: Esempio di pagina di riepilogo delle valutazioni delle commissioni.

4.6 Modello dei Dati e Persistenza

La spina dorsale della piattaforma PhDManV2 è il suo modello dati, che è stato accuratamente progettato per riflettere la complessità del processo di valutazione dei dottorati. Implementato usando **Entity Framework Core**, il modello rappresenta le relazioni tra i dottorandi, i loro cicli di valutazione, i documenti caricati e tutti gli attori coinvolti nel processo di revisione. Questa struttura garantisce coerenza dei dati, tracciabilità e estensibilità. Il modello sfrutta il mapping object-relational (ORM) per fornire un'interfaccia fortemente tipizzata e facilmente manutenibile al database SQL Server sottostante.

4.6.1 Entità Principali

Al centro del modello c'è l'entità AnnoDottorato, che rappresenta un singolo anno di valutazione dello studente di dottorato. Ogni record include metadati come l'anno di riferimento, data di inizio e fine, le annotazioni da parte dello studente e, una volta finalizzata, il giudizio finale. L'entità tiene traccia anche se questo è l'anno attivo del dottorato o è un anno precedente e se lo studente è abilitato all'inserimento della relazione.

Ogni record di AnnoDottorato è associato a un singolo record di Dottorando che contiene le informazioni anagrafiche (come nome e cognome), accademiche (le matricole e l'email), il tema di ricerca, il numero di ciclo e gli indicatori bibliometrici. Questa separazione tra i dati statici dello studente (nell'entità Dottorando) e i dati legati alla valutazione (nell'entità AnnoDottorato) consente al sistema di monitorare il progresso longitudinale mantenendo in maniera chiara i confini tra identità e performance.

4.6.2 Gestione dei documenti

Le presentazioni caricate dagli studenti di dottorato sono archiviate nella tabella AnnoDottoratoDocumenti, che supporta caricamenti multipli per ciclo di valutazione. Ogni documento è salvato come blob binario (byte[] Data) insieme a metadati come:

- NomeFile: il nome del file originale.

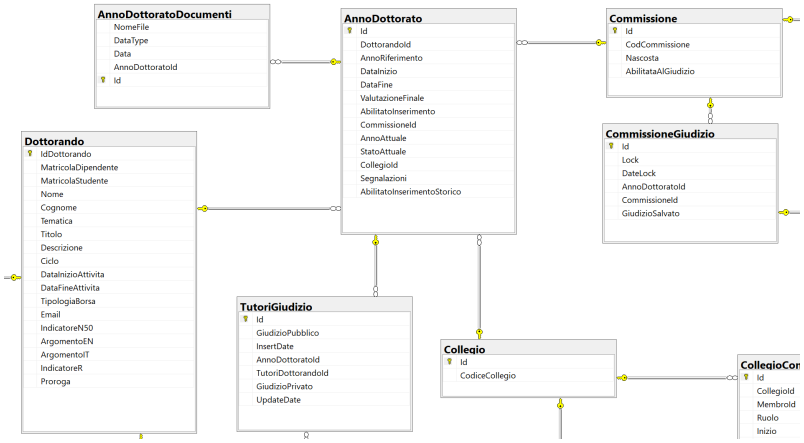

Figura 4.12: Diagramma entità-relazione delle entità principali.

- MimeType: il tipo di contenuto (per esempio application/pdf).
- DataCaricamento: da data e ora di upload della presentazione.

Questa struttura garantisce che i documenti vengano archiviati in modo sicuro all'interno del database, evitando la dipendenza da un file system esterno o da servizi di cloud storage. Inoltre, semplifica le procedure di backup e di ripristino poiché tutti i dati sono centralizzati all'interno del contesto del database. Per prevenire abusi e garantire prestazioni ottimali, il sistema impone una dimensione massima dei file di 50 MB e limita i tipi di file accettati ai soli pdf e presentazioni Powerpoint. Questi vincoli vengono applicati sia lato client tramite la validazione dell'input, sia lato server tramite middleware e validazione del modello.

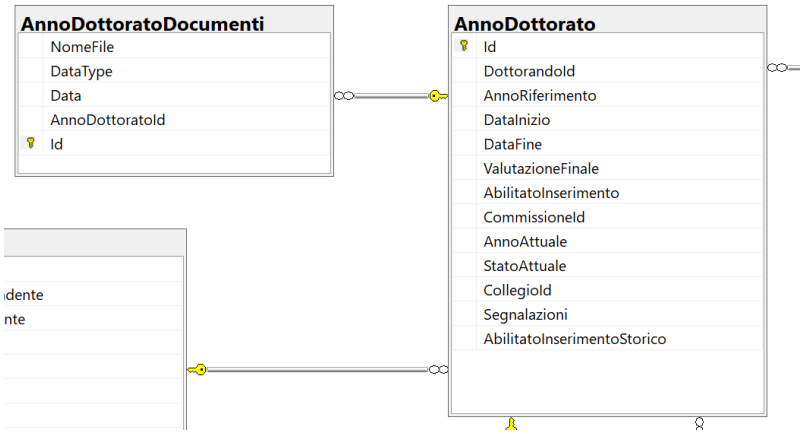

Figura 4.13: Diagramma entità-relazione della tabella dei documenti.

4.6.3 Tutori e il loro Giudizio

La gestione dei tutori è modellata attraverso tre entità interconnesse:

- Tutori: contiene le anagrafiche di tutti i tutori e cotutori, ognuno con la matricola, nominativo e indirizzo email.
- TutoriDottorando: una tabella di giunzione che collega i tutori ai dottorandi, assegnando loro il ruolo specifico (tutore o cotutore).
- TutoriGiudizio: questa entità contiene il giudizio del tutore per uno specifico record di AnnoDottorato.

Questa struttura garantisce che i feedback dei tutor siano contestualizzati sia rispetto all'anno di valutazione sia alla specifica relazione tutor–dottorando. Supporta inoltre la presenza di più tutor per ciascuno studente e tiene traccia dei loro ruoli e delle responsabilità di valutazione nel tempo.

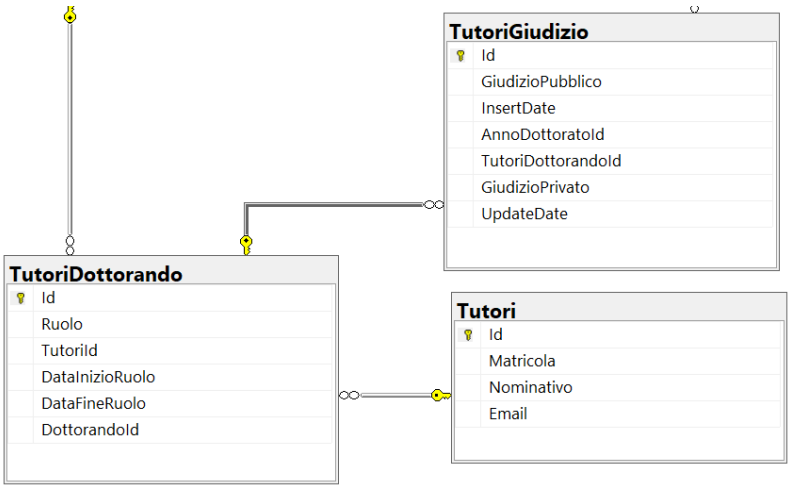

Figura 4.14: Diagramma entità-relazione delle tabelle dei Tutori.

4.6.4 Giudizio della Commissione

Il processo di valutazione della commissione viene rappresentata da cinque entità chiave:

- Commissione: definisce una specifica commissione assegnata a uno o più dottorandi. Include il nome della commissione, un flag per indicare se in questo momento è abilitata per dare i giudizi e un flag per indicare se la è attiva o disattivata.

- CommissioneMembri: la lista completa dei possibili membri delle commissioni. Comprende la matricola e il nominativo.
- CommissioneComponenti: associa i singoli membri di CommissioneMembri a una commissione. Ogni record comprende il ruolo del membro (Presidente o Revisore) e le date di inizio e fine validità dell'incarico.
- CommissioneGiudizio: contiene i dati strutturati dei giudizi dati da una commissione per uno specifico AnnoDottorato. Contiene anche un flag che indica se il giudizio è stato salvato.
- CommissioneGiudizioDati: i record rappresentano i singoli campi del form di valutazione della commissione. Ogni record include:
  - CodiceDato: un codice che identifica univocamente un singolo campo.
  - Nome: nome del campo che verrà visualizzato.
  - Descrizione: eventuale descrizione che comparirà sotto il campo nel form.
  - Tipologia: indica il tipo di campo che verrà visualizzato (per esempio una campo di testo o una combo box).
  - Valore: il valore salvato.
  - ValoriCombo: contiene una lista dei possibili valori di campi strutturati che prevedono delle scelte (come combo box, liste di radio button o liste di checkbox)
  - OrdineInPagina: indica la posizione in cui verrà visualizzato il campo.
  - LivelloVisibilita: un flag per impostare il campo come pubblico (se uguale a 1) o privato (se uguale a 0). I campi indicati come "privati" saranno visibili solo ad altri membri della stessa commissione, ai membri del collegio e agli amministratori, quelli "pubblici" saranno visibili anche ai dottorandi e ai tutori/cotutori.
  - IsFinalRaccomandation: è un flag per indicare quale campo è l'esito finale della commissione; è un campo con scopo funzionale per alcuni report degli amministratori.
  - Obbligatorio: questo flag permette di verificare se il campo è obbligatorio oppure no.

I record di CommissioneGiudizioDati vengono generati automaticamente basandosi sulla tabella CommissioneGiudizioDatiTemplate. Le due tabelle condividono la struttura e questo tipo di gestione permette di configurare il form di valutazione senza dover intervenire sul codice.

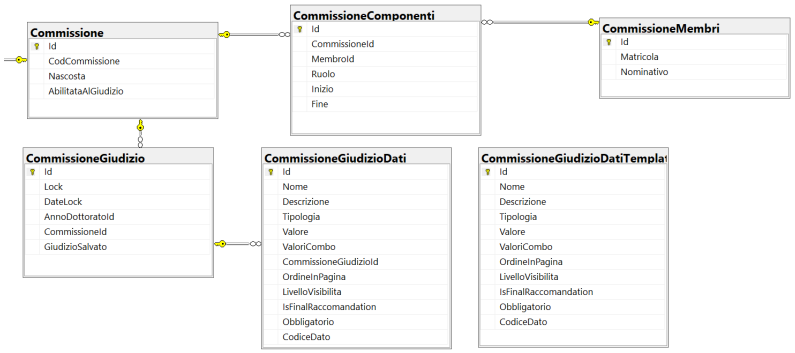

Figura 4.15: Diagramma entità-relazione delle tabelle delle Commissioni.

4.6.5 Giudizio del Collegio

L'entità Collegio e le tabelle collegate modellano la fase finale del processo di valutazione, in cui i membri del collegio dei docenti emettono una raccomandazione formale. Questo comprende quattro entità principali:

- Collegio: rappresenta il collegio dei docenti assegnato ai giudizi dei dottorandi.
- CollegioMembri: contiene l'elenco completo dei possibili membri del collegio; ogni record è composto da matricola e nominativo.
- CollegioComponenti: collega i singoli membri di CollegioMembri a un determinato collegio. Ogni record include anche le date di inizio e fine dell'incarico.
- CollegioGiudizio: qui è salvato il giudizio finale del collegio relativa a un determinato record di AnnoDottorato; include:
  - ValoreGiudizio: esito finale dell'anno di dottorato (Ammesso, Ammesso con riserva, Non ammesso).
  - MotivazioneGiudizio: eventuale motivazione del giudizio.
  - DataRiunioneCollegio: data in cui si è riunito il collegio e ha espresso la valutazione.

– InsertDate: data e ora del salvataggio del record.

Questa struttura garantisce che le decisioni del collegio siano tracciabili, datate e collegate al corretto ciclo di valutazione. Supporta inoltre il controllo degli accessi basato sui ruoli e i meccanismi di protezione contro modifiche concorrenti.

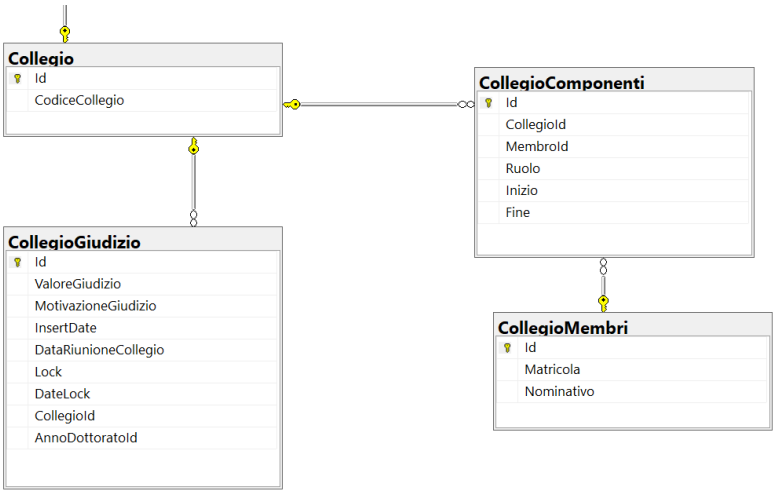

Figura 4.16: Diagramma entità-relazione delle tabelle del Collegio.

4.6.6 Integrazione del ciclo di vita della valutazione

I giudizi delle commissioni e del collegio sono collegate alla tabella AnnoDottorato, formando un ciclo di valutazione completo:

1. Il dottorando carica la relazione sul suo lavoro nell'anno corrente di dottorato (AnnoDottoratoDocumenti)

2. I tutor forniscono il loro feedback sull'operato del dottorando (TutoriGiudizio)

3. I membri della commissione basandosi sul colloquio, sulla presentazione e sul giudizio del tutore, esprimono il loro giudizio (CommissioneGiudizio)

4. Il collegio dei docenti emette la valutazione finale con l'ausilio di tutti i dati raccolti durante il processo.(CollegioGiudizio)

Ogni fase è salvata con una datazione precisa, associata a un ruolo specifico e memorizzata in un'entità dedicata, garantendo piena tracciabilità e la possibilità di ricostruire lo storico degli eventi.

4.7 Middleware e Logging

Per supportare l'autenticazione, la diagnostica e l'audit, la piattaforma include un componente middleware personalizzato e un'infrastruttura di logging strutturato; questi elementi svolgono un ruolo fondamentale nel garantire trasparenza, tracciabilità e stabilità operativa.

4.7.1 Middleware personalizzato

Il middleware ha il compito di intercettare le richieste HTTP in arrivo ed estrarre le informazioni di identità dagli header Shibboleth. In produzione, legge attributi come email, nome, cognome e matricola, utilizzandoli per costruire un'identità basata sui claims. In ambiente di sviluppo, simula l'autenticazione tramite variabili d'ambiente, consentendo agli sviluppatori di testare il comportamento specifico dei vari ruoli.

Oltre all'autenticazione, il middleware registra i metadati delle richieste, tra cui i parametri di query, i percorsi richiesti e lo stato di autenticazione. Gestisce inoltre i reindirizzamenti per gli utenti non autenticati e applica le politiche di accesso basate sui claims di ruolo.

4.7.2 Logging strutturato con Serilog

L'applicazione utilizza Serilog per il logging strutturato; i log vengono scritti giornalmente e includono timestamp, livello di gravità, messaggi e dettagli delle eccezioni. Questa configurazione consente agli amministratori di monitorare il comportamento del sistema, rilevare anomalie ed eseguire analisi approfondite in caso di errori critici.

Oltre ai log basati su file, gli errori critici vengono memorizzati nella tabella LogErrori all'interno del database. Ciò garantisce che i record degli errori vengano preservati anche nel caso in cui l'accesso ai file sia limitato o non disponibile.

I log vengono inoltre utilizzati per tracciare le attività degli utenti, come i tentativi di accesso, la risoluzione dei ruoli e il download dei documenti; questi record supportano la conformità alle politiche istituzionali e possono essere utilizzati a fini di audit.

4.8 Distribuzione e Configurazione degli Ambienti

La piattaforma è progettata per operare in modo affidabile sia negli ambienti di sviluppo che in quelli di produzione. La sua strategia di configurazione garantisce che il comportamento specifico di ciascun ambiente sia chiaramente definito e facilmente manutenibile.

4.8.1 Sviluppo vs. Produzione

In ambiente di sviluppo, l'applicazione utilizza variabili d'ambiente per simulare identità e ruoli utente. Ciò consente agli sviluppatori di testare i flussi di autenticazione, il controllo degli accessi basato sui ruoli e il comportamento dell'interfaccia utente, senza dover dipendere da provider di identità esterni. Sono inoltre abilitate informazioni di errore dettagliate e strumenti di debugging per facilitare la risoluzione dei problemi.

In produzione, l'applicazione si integra con Shibboleth per l'autenticazione federata e applica le politiche HTTPS e HSTS, utilizza cookie sicuri e disattiva la visualizzazione dettagliata degli errori; i ruoli utente vengono derivati dagli header Shibboleth e il controllo degli accessi è applicato in modo rigoroso.

4.8.2 Distribuzione su IIS e Hosting come Sottoapplicazione

L'applicazione è distribuita sotto IIS come sotto-applicazione con il percorso `/PhDManV2`. Ciò richiede una configurazione specifica:

- La direttiva `UsePathBase("/PhDManV2")` garantisce che i percorsi di routing e delle risorse vengano risolti correttamente.
- Le risorse statiche (CSS, JS, immagini) vengono servite dal sottopercorso.
- L'hub Blazor SignalR è mappato su `/PhDManV2/_blazor`.

Questo modello di distribuzione consente alla piattaforma di coesistere con altri servizi preesistenti, mantenendo al tempo stesso una separazione logica delle risorse e un routing indipendente.

4.8.3 Sicurezza e Configurazioni Avanzate

Sono state implementate diverse misure di sicurezza per proteggere i dati degli utenti e garantire la conformità:

- Le politiche CORS limitano l'accesso ai domini considerati attendibili.
- I token antiforgery sono abilitati per prevenire gli attacchi CSRF.
- Sono imposti limiti alla dimensione delle richieste per evitare abusi (ad es. attacchi denial-of-service tramite caricamenti eccessivamente grandi).
- I cookie di autenticazione sono configurati con un flag di sicurezza e politiche di scadenza.

Queste configurazioni vengono applicate in modo uniforme tra i vari ambienti e sono documentate per garantire una manutenzione semplificata.

Capitolo 5

Risultati

5.1 Introduzione

Questo capitolo ha come obiettivo quello di illustrare i risultati ottenuti nella realizzazione dell'applicazione PhDManV2, verificare la corrispondenza tra le specifiche tecniche e funzionali definite durante la fase di progettazione e il progetto implementato. L'analisi non si limita a fornire un elenco di funzionalità, ma mira a mostrare come ogni prerequisito è stato tradotto in un risultato concreto, evidenziando la coerenza complessiva dell'architettura e la qualità dell'esperienza utente offerta dal sistema.

5.2 Risultati rispetto ai requisiti funzionali

Uno degli obiettivi principali del progetto era la **centralizzazione delle informazioni** sulla carriera di dottorato degli studenti; il portale permette di raccogliere e gestire in un unico ambiente i dati provenienti da fonti eterogenee: inserimenti manuali da parte degli studenti, vari giudizi durante le fasi del processo e informazioni da diverse fonti istituzionali. Tutto questo ha ridotto la frammentazione dei dati e ha facilitato la reperibilità degli stessi, garantendo una visione d'insieme coerente del percorso accademico di ogni studente di dottorato.

Un altro requisito riguardava **l'integrazione con le fonti dati istituzionali**; questo obiettivo è stato raggiunto grazie all'uso di API REST e servizi asincroni che consentono lo scambio di informazioni tra i sistemi centrali e il portale tramite lo scambio sicuro di dati in formato JSON.

L'integrazione è stata progettata in modo tipizzato, basandosi sulle potenzialità di Entity Framework Core di assicurare consistenza e affidabilità nelle operazioni di lettura e scrittura. In questo modo il portale non si limita ad essere un contenitore isolato ma può diventare parte integrante del sistema informativo di ateneo.

La **personalizzazione delle pagine e l'accesso basato sui ruoli** rappresentano un altro risultato; ogni attore coinvolto nel processo, studenti, tutori, commissioni, collegio e amministratori, hanno interfacce dedicate progettate per soddisfare le necessità specifiche del loro ruolo. L'autenticazione federata, implementata tramite Shibboleth e SAML 2.0, garantisce l'accesso sicuro in conformità con le policy del Politecnico di Torino, mentre l'autorizzazione basata sui ruoli garantisce che ogni utente abbia accesso solo alle funzionalità rilevanti alle proprie responsabilità. Questo approccio ha reso l'applicazione non solamente sicura ma anche intuitiva e adeguata alle pratiche operative.

La **preservazione dei dati** è stata implementata per garantire la disponibilità delle valutazioni annuali lungo tutto il percorso di dottorato. Ogni valutazione rimane accessibile e consultabile, garantendo una totale tracciabilità del percorso accademico. Il ciclo di valutazione, dalla preparazione, al caricamento della presentazione, ai giudizi del tutore, commissione e collegio e all'archiviazione finale, è stato implementato in un flusso chiaro e ben strutturato che riflette appieno le procedure istituzionali.

La **configurabilità e flessibilità** del sistema sono evidenziate dalla possibilità di modificare la composizione delle commissioni e del collegio nel tempo attraverso interfacce dedicate che permettono al Coordinatore di assegnare o eliminare membri da esse. Questa caratteristica risponde ad una necessità reale, in quanto le commissioni devono essere composte ad-hoc per ogni dottorando rispettando criteri di pertinenza degli argomenti, per evitare conflitti di interesse, e possono cambiare nel tempo. Il portale fornisce uno strumento semplice e immediato per gestire questi cambiamenti, riducendo la complessità di gestione.

Infine, il requisito riguardo il miglioramento dell'**esperienza utente** è stato raggiunto grazie all'adozione di Blazor Server e la sua gestione dello stato con SignalR. Le interfacce sono interattive e fluide, con aggiornamenti in tempo reale che riducono i tempi di attesa percepita che contribuiscono a rendere l'applicazione funzionale e semplice da usare.

5.3 Risultati rispetto ai requisiti tecnici e di sicurezza

Dal punto di vista tecnico, l'applicazione ha soddisfatto gli obiettivi nell'ottica dell'**autenticazione e autorizzazione**. L'integrazione con Shibboleth e SAML 2.0 ha permesso di implementare un sistema Single Sign-On pienamente conforme agli standard di identità federata, garantendo contemporaneamente un controllo granulare degli accessi. Ogni operazione è legata al ruolo dell'utente assicurando che le informazioni sensibili siano protette e che tutte le funzionalità siano utilizzate soltanto da chi ne ha il diritto.

La **persistenza e gestione dei dati** è garantita dall'utilizzo di Microsoft SQL Server, che assicura transazioni ACID e meccanismi avanzati di sicurezza. L'uso di Entity Framework Core ha semplificato la gestione delle entità e ha reso più agevole lo sviluppo, riducendo il rischio di errore e aumentando la manutenibilità del codice. L'integrazione tra il livello dati e quello applicativo è risultata coerente e fluida, confermando la solidità dell'architettura scelta.

Il sistema di **middleware e logging** ha contribuito a migliorare la robustezza complessiva dell'applicativo; l'uso di middleware personalizzati permette di gestire eccezioni e flussi applicativi, mentre l'integrazione con Serilog consente dei log strutturati e dettagliati. Grazie a questo approccio è stato possibile monitorare il comportamento del sistema in tempo reale e di tracciare eventuali anomalie, facilitando le attività di manutenzione e garantendo elevati livelli di affidabilità.

Per quanto riguarda l'**hosting e la distribuzione**, l'applicativo è stato pubblicato su Internet Information Services (IIS) come sottoapplicazione, sfruttando i meccanismi di isolamento offerti dagli application pool. La configurazione dell'ambiente di produzione ha incluso parametri di sicurezza e monitoraggio assicurando che il sistema sia pronto a operare in un contesto reale e a gestire carichi di lavoro significativi.

5.4 Sintesi dei Risultati

In conclusione, lo sviluppo di **PhDManV2** ha portato alla realizzazione di un portale web che risponde pienamente ai requisiti funzionali e tecnici definiti in fase di progettazione. L'applicativo consente una gestione completa e integrata delle carriere dei dottorandi, garantendo sicurezza, affidabilità e un'esperienza utente di qualità. La corrispondenza tra specifiche e implementazione dimostra la validità delle scelte tecnologiche effettuate e la coerenza dell'approccio progettuale, ponendo le basi per un utilizzo efficace e duraturo del sistema all'interno del contesto accademico.

Tabella 5.1: Tracciamento dei requisiti, dell'implementazione e dei risultati di PhDManV2

Requisito	**Implementazione**	**Risultato**
Centralizzazione delle informazioni	Integrazione di dati dei dottorati, degli studenti, dei tutori dai database istituzionali tramite API REST	Portale unico che raccoglie e presenta in modo chiaro e coerente tutte le informazioni necessarie
Integrazione con fonti dati istituzionali	Servizi asincroni, payload JSON tipizzati e sincronizzazione con viste SQL Server	Aggiornamento on-demand e consistenza dei dati con i sistemi centrali di ateneo
Pagine personalizzate e accesso basato sui ruoli	Autenticazione federata con Shibboleth/SAML 2.0; autorizzazione tramite RBAC (Role-Based Access Control)	Interfacce dedicate per studenti, tutori, commissioni, collegio e amministratori; accesso sicuro e differenziato
Conservazione dei dati	Gestione del ciclo di vita della valutazione (preparazione, caricamento, revisione, delibera, archiviazione)	Le valutazioni annuali sono disponibili e tracciabili per tutta la durata del percorso di dottorato
Configurabilità e flessibilità	Interfacce di amministrazione per modificare composizione delle commissioni e assegnazioni del collegio	Sistema adattabile alle variazioni organizzative e alle esigenze istituzionali
Miglioramento dell'esperienza utente	Blazor Server con gestione dello stato tramite SignalR; interfacce interattive e responsive	Navigazione fluida, aggiornamenti in tempo reale e riduzione della latenza percepita
Autenticazione e autorizzazione	Single Sign-On con Shibboleth; rilascio controllato degli attributi; controlli granulari sui ruoli	Accesso sicuro e conforme alle policy di ateneo; protezione delle informazioni sensibili
Persistenza e gestione dei dati	SQL Server con transazioni ACID; integrazione con Entity Framework Core	Affidabilità, sicurezza e manutenibilità del livello dati
Middleware e logging	Middleware personalizzati per gestione eccezioni; logging strutturato con Serilog	Monitoraggio avanzato e tracciabilità completa delle operazioni
Hosting e distribuzione	Distribuzione su IIS come sottoapplicazione; configurazioni di sicurezza e isolamento tramite application pool	Ambiente di produzione stabile, sicuro e pronto a gestire carichi significativi

Capitolo 6

Sviluppi futuri

6.1 Introduzione

Lo sviluppo di **PhDManV2** ha portato alla realizzazione di un portale completo e funzionale per la gestione delle carriere dei dottorandi, tuttavia, come ogni sistema informativo, l'applicativo può essere arricchito con ulteriori funzionalità che ne aumentino l'efficacia, la flessibilità e il valore operativo. In questo capitolo vengono delineati i possibili sviluppi futuri, pensati per rispondere alle esigenze emergenti degli attori coinvolti e per garantire una maggiore automazione e trasparenza dei processi.

6.2 Pagina di gestione delle comunicazioni

Un primo ambito di evoluzione riguarda la **gestione delle comunicazioni interne**; attualmente il portale centralizza i dati e i giudizi, ma non offre strumenti integrati per l'invio di notifiche. Si propone quindi di introdurre una pagina dedicata agli amministratori per l'invio di e-mail ai diversi attori (studenti, tutori, commissioni, collegio). Tale funzionalità dovrebbe consentire:

- la selezione del tipo di comunicazione (sollecito o semplice notifica).
- la verifica dello storico delle notifiche inviate in precedenza, così da evitare duplicati.
- la possibilità di filtrare i destinatari in base a criteri specifici, ad esempio:
  - tutori e/o commissioni che non hanno ancora espresso un giudizio.

  - dottorandi che non hanno caricato la presentazione in prossimità della scadenza.

Questa evoluzione renderebbe il portale non solo uno strumento per la gestione dei dati, ma anche un canale di comunicazione istituzionale, riducendo la necessità di strumenti esterni e garantendo tracciabilità completa.

6.3 Gestione dei periodi di inserimento e revisione

Un ulteriore sviluppo riguarda la possibilità di **impostare intervalli temporali** per l'apertura e la chiusura delle diverse fasi del processo. Gli amministratori potrebbero definire, tramite interfacce dedicate, i periodi in cui:

- gli studenti possono caricare le presentazioni.
- i tutori possono inserire le proprie revisioni.
- le commissioni possono esprimere i giudizi.

Questa funzionalità permetterebbe di automatizzare la gestione delle scadenze, riducendo il rischio di errori e garantendo uniformità nelle procedure. Inoltre, l'integrazione con il sistema di notifiche consentirebbe di avvisare automaticamente gli attori all'apertura o alla chiusura dei periodi.

6.4 Sistema di notifiche automatiche

Accanto alle notifiche manuali, si propone l'introduzione di un **sistema di e-mail automatiche**. Il portale potrebbe inviare comunicazioni predefinite in corrispondenza di eventi significativi, come:

- apertura di un nuovo periodo di inserimento.
- scadenza imminente per il caricamento delle presentazioni.
- mancata compilazione di giudizi da parte di tutori o commissioni.
- pubblicazione di una delibera del collegio.

Questa automazione ridurrebbe il carico amministrativo e garantirebbe che tutti gli attori siano costantemente informati, migliorando la puntualità e la trasparenza del processo.

6.5 Pagina di consultazione dei log

Un'altra evoluzione importante riguarda la **consultazione dei log** in tempo reale. Attualmente, il sistema registra le operazioni tramite Serilog, ma l'accesso ai log è pensato per fini tecnici. Una pagina dedicata, accessibile agli amministratori, permetterebbe di:

- visualizzare in diretta le operazioni compiute dagli utenti.
- filtrare i log per tipologia di evento (accessi, caricamenti, giudizi, notifiche).
- esportare i dati per analisi successive.

Questa funzionalità aumenterebbe la trasparenza e fornirebbe uno strumento utile per monitorare l'andamento del sistema e individuare rapidamente eventuali anomalie.

6.6 Ulteriori possibili sviluppi

Oltre alle funzionalità sopra descritte, si possono immaginare ulteriori evoluzioni coerenti con l'architettura del portale:

- **Dashboard di sintesi**: una pagina con grafici e indicatori che mostrino lo stato complessivo delle valutazioni, il numero di giudizi mancanti e le scadenze imminenti.
- **Supporto multilingua**: estensione dell'interfaccia per consentire l'uso del portale anche da parte di utenti internazionali, migliorando l'accessibilità.
- **Sistema di versionamento documentale**: gestione delle diverse versioni delle presentazioni caricate dai dottorandi, con possibilità di consultare lo storico delle modifiche.

6.7 Conclusioni

Gli sviluppi futuri delineati in questo capitolo mirano a trasformare PhDManV2 da semplice portale di gestione delle valutazioni a piattaforma completa di supporto ai processi accademici. L'introduzione di strumenti di comunicazione, automazione delle scadenze, consultazione dei log e funzionalità avanzate di monitoraggio renderebbe il sistema ancora più efficace, riducendo il carico amministrativo e migliorando la qualità dell'esperienza per tutti gli attori coinvolti.

Appendice A

A.1 Codice sorgente rilevante

In questa sezione verranno riportati frammenti significativi di codice per la comprensione delle scelte architetturali e implementative effettuate nello sviluppo dell'applicazione **PhDManV2**.

A.1.1 Esempio di componente Razor

Di seguito è mostrato un estratto semplificato della struttura di un componente Razor utilizzato nel portale.

```
@page "/dottorandoPage"
@inject UserDataService UserData
@inject AuthenticationStateProvider
    AuthenticationStateProvider
@attribute [Authorize(Roles = "dottorando")]
<PageTitle>Dottorato</PageTitle>

<Grid @ref="tabAnni"
TItem="AnnoDottorato"
Class="table"
Data="TabData"
AllowFiltering="false"
Responsive="true"
AllowSorting="false"
lang="it"
AllowDetailView="false"
AllowRowClick="false"
PageSizeSelectorVisible="true"
AllowPaging="false"
PageSize="100"
PageSizeSelectorItems="@(new int[] { 10,25,50,75,100 })"
PaginationItemsTextFormat="{0} - {1} di {2} Dottorandi"
```

```
ItemsPerPageText="Elementi per pagina">

    <GridColumns>
        ...codice delle colonne della tabella...
    </GridColumns>
</Grid>

@code {
    private PhdReviewContext dataContext = default!;

    private Grid<AnnoDottorato> tabAnni = default!;

    private ClaimsPrincipal? user;
    protected override async Task OnInitializedAsync()
    {
        dataContext = DbFactory.CreateDbContext();

        var authState = await AuthenticationStateProvider.
    GetAuthenticationStateAsync();
        user = authState.User;

        MatricolaLoggata = user.FindFirst("Matricola")?.Value
    ;
        getTabData();

        nome = "";
        cognome = "";
        if (user.Identity.Name.StartsWith("S") && int.
    TryParse(user.Identity.Name.Substring(1), out _))
        {
            Dottorando d = dataContext.Dottorando.
    FirstOrDefault(d => d.MatricolaStudente == user.Identity.
    Name);

            if (d != null)
            {
                nome = d.Nome;
                cognome = d.Cognome;
            }
        }
        else
        {
            nome = user.FindFirst(ClaimTypes.GivenName)?.
    Value;
```

```
60              cognome = user.FindFirst(ClaimTypes.Surname)?.
    Value;
61          }
62          nome = FunzioniGeneriche.ToTitleCase(nome);
63          cognome = FunzioniGeneriche.ToTitleCase(cognome);
64      }
65
66      private void getTabData()
67      {
68          TabData = dataContext.AnnoDottorato.Include(i => i.
    Dottorando)
69          .Where(w => w.Dottorando.MatricolaDipendente ==
    MatricolaLoggata || w.Dottorando.MatricolaStudente ==
    MatricolaLoggata)
70          .Include(i => i.Documenti)
71          .OrderByDescending(o => o.AnnoRiferimento)
72          .AsNoTracking()
73          .ToList();
74      }
75  }
```

A.1.2 Estratto del servizio applicativo

Esempio di classe del *Business Logic Layer*:

```
1  public class DataRetriever
2  {
3      internal enum TipoChiamataApi { dottorando, dottorato,
    ore_didattica_studente, ore_didattica, pubblicazioni }
4
5      private static HttpClient httpClient = new HttpClient();
6
7      internal static async Task<string> ChiamataApi(
    TipoChiamataApi tipo, string matricola)
8      {
9          string res = "";
10         try
11         {
12             var options = new RestClientOptions("https://
    didattica.polito.it")
13             {
14                 ThrowOnAnyError = true,
15                 Timeout = new TimeSpan(0, 0, 10000)
```

```
16              };
17
18              var client = new RestClient(options);
19              var request = new RestRequest("https://didattica.
    polito.it/pls/portal30/sviluppo.pkg_json_scudo.data",
    Method.Post);
20              request.AddHeader("X-API-Key",
    "**********************");
21              request.AddHeader("Content-Type", "text/x-json");
22              request.AddStringBody("{\"usecache\":\"N\", \"dip
    \": \"DAUIN\", \"data\": \"" + tipo.ToString() + "\", \"
    utente\": \"" + matricola + "\"}", DataFormat.None);
23
24              var response = await client.ExecuteAsync(request)
    ;
25
26              // Leggo la risposta
27              string responseContent = response.Content;
28              Console.WriteLine($"Status Code: {response.
    StatusCode}");
29              Console.WriteLine("Response:");
30              Console.WriteLine(responseContent);
31              res = responseContent;
32          }
33          catch(Exception ex)
34          {
35              res = $"Error:{ex.Message}:{ex.InnerException}";
36          }
37          return res;
38      }
39  }
```

A.1.3 Estratto middleware personalizzato (autenticazione Shibboleth)

```
1  public class UserContextService
2  {
3      private readonly IHttpContextAccessor
       _httpContextAccessor;
4      private readonly IWebHostEnvironment _env;
5
6      public UserContextService(IHttpContextAccessor accessor,
       IWebHostEnvironment env)
7      {
8          _httpContextAccessor = accessor;
9          _env = env;
10     }
11
12     public (string Email, string Name, string Surname, string
       EmployeeNumber) GetUserData()
13     {
14         var context = _httpContextAccessor.HttpContext!;
15         string email = "", eppn = "", givenName = "", surname
       = "", employeeNumber = "", eppnp = "";
16
17         var headers = context.Request.Headers;
18         eppn = headers.TryGetValue("eppn", out var eppnVal) ?
       eppnVal.FirstOrDefault() ?? "" : "";
19         givenName = headers.TryGetValue("givenName", out var
       gnVal) ? gnVal.FirstOrDefault() ?? "NoData" : "NoData";
20         surname = headers.TryGetValue("sn", out var snVal) ?
       snVal.FirstOrDefault() ?? "NoData" : "NoData";
21         employeeNumber = headers.TryGetValue("employeeNumber
       ", out var enVal) ? enVal.FirstOrDefault() ?? "NoData" : "
       NoData";
22         eppnp = headers.TryGetValue("eppnp", out var epVal) ?
       epVal.FirstOrDefault() ?? "NoData" : "NoData";
23         email = string.IsNullOrEmpty(eppn) ? eppnp : eppn;
24
25         return (email, givenName, surname, employeeNumber);
26     }
27 }
```

A.2 Modello dei dati — Modelli principali

Di seguito sono riportati alcuni dei modelli dati utilizzati nell'applicazione.

```
public class Dottorando
{
    [Key]
    public int IdDottorando { get; set; }

    [Required]
    public string MatricolaDipendente { get; set; }
    public string MatricolaStudente { get; set; }
    public string Nome { get; set; }
    public string Cognome { get; set; }

    [NotMapped]
    public string Nominativo { get => (Cognome + " " + Nome).
   ToTitleCase(); }
    [NotMapped]
    public string Matricole { get => MatricolaDipendente + "
   / " + MatricolaStudente; }
    public string Tematica { get; set; }
    public string Titolo { get; set; }
    public string ArgomentoIT { get; set; } = "";
    public string ArgomentoEN { get; set; } = "";
    public string Descrizione { get; set; }
    public int Ciclo { get; set; }
    public DateTime DataInizioAttivita { get; set; }
    public DateTime DataFineAttivita { get; set; }
    [NotMapped]
    public DateOnly DataInizioCiclo {
        get{
            return DateOnly.FromDateTime(DataInizioAttivita);
        }
    }
    [NotMapped]
    public DateOnly DataFineCiclo
    {
        get
        {
            return DateOnly.FromDateTime(DataFineAttivita);
        }
    }
```

```
38
39      public string TipologiaBorsa { get; set; }
40      public double IndicatoreN50 { get; set; } = 0.0;
41      public double IndicatoreR { get; set; } = 0.0;
42      public string Email { get; set; }
43      public string Proroga { get; set; }
44
45      public IEnumerable<AnnoDottorato> AnnoDottorato { get;
        set; } = new List<AnnoDottorato>();
46      public IEnumerable<TutoriDottorando> TutoriDottorando {
        get; set; } = new List<TutoriDottorando>();
47
48      public IEnumerable<DottorandoDatiDaDidattica>
        DatiDaDidattica { get; set; } = new List<
        DottorandoDatiDaDidattica>();
49
50
51      public Dottorando()
52      { }
53
54      public Dottorando(string matricolaDipendente, string
        matricolaStudente, string nome, string cognome, string
        tematica, string argomentoIta, string argomentoEn, string
        titolo, string descrizione, int ciclo, DateTime
        dataInizioAttivita, DateTime dataFineAttivita, string
        tipologiaBorsa, string proroga, double n50, double indiR,
        string email)
55      {
56          MatricolaDipendente = matricolaDipendente;
57          MatricolaStudente = matricolaStudente;
58          Nome = nome;
59          Cognome = cognome;
60          Tematica = tematica;
61          ArgomentoIT = string.IsNullOrEmpty(argomentoIta) ? ""
         : argomentoIta;
62          ArgomentoEN = string.IsNullOrEmpty(argomentoEn) ? ""
        : argomentoEn;
63          Titolo = titolo;
64          Descrizione = descrizione;
65          Ciclo = ciclo;
66          DataInizioAttivita = dataInizioAttivita;
67          DataFineAttivita = dataFineAttivita;
68          TipologiaBorsa = tipologiaBorsa;
69          IndicatoreN50 = n50;
70          IndicatoreR = indiR;
```

```
71          Email = email;
72          Proroga = proroga;
73      }
74  }
```

```
1  public class AnnoDottorato
2  {
3      [Key]
4      public int Id { get; set; }
5      public string AnnoRiferimento { get; set; }
6      public DateTime DataInizio { get; set; }
7      public DateTime DataFine { get; set; }
8
9      public string Segnalazioni { get; set; } = string.Empty;
10
11     [Comment("Abilitato all'inserimento dei dati/documenti
       per la revisione")]
12     public bool AbilitatoInserimento { get; set; }
13
14     [Comment("Indica se a un certo punto è stato abilitato
       all'inserimento")]
15     public bool AbilitatoInserimentoStorico { get; set; } =
       false;
16
17     [Comment("Stato della richiesta. TOSTART, CARICA_FILE,
       VAL_TUTORE, VAL_COMMISSIONE, VAL_COLLEGIO, VAL_END")]
18     public string StatoAttuale { get; set; } = string.Empty;
19
20     [Comment("Admitted, Admitted with warning e Rejected")]
21     public string? ValutazioneFinale { get; set; }
22
23     // da qui si dovrà fare riferimento ai dati del
       dottorando su pauper, ai dati delle pubblicazioni su iris
       e i dati della didattica da gesd
24
25     [Comment("Solo uno deve essere abilitato ad essere l'anno
       attuale")]
26     public bool AnnoAttuale { get; set; } = false;
27
28     public Dottorando Dottorando {  get; set; }
29     public int DottorandoId { get; set; }
30     public List<AnnoDottoratoDocumenti> Documenti { get; set;
       } = new List<AnnoDottoratoDocumenti>();
```

```
31
32      public List<TutoriGiudizio> TutoriGiudizio { get; set; }
        = new List<TutoriGiudizio>();
33
34      public Commissione? Commissione { get; set; }
35      public int? CommissioneId { get; set; }
36
37      public Collegio? Collegio { get; set; }
38      public int? CollegioId { get; set; }
39
40      public AnnoDottorato()
41      {
42
43      }
44
45      public AnnoDottorato(string annoRiferimento, DateTime
        dataInizio, DateTime dataFine, string segnalazioni, bool
        abilitatoInserimento, string? valutazioneFinale, int
        dottorandoId, bool annoAttuale)
46      {
47          AnnoRiferimento = annoRiferimento;
48          DataInizio = dataInizio;
49          DataFine = dataFine;
50          Segnalazioni = segnalazioni;
51          AbilitatoInserimento = abilitatoInserimento;
52          ValutazioneFinale = valutazioneFinale;
53          DottorandoId = dottorandoId;
54          AnnoAttuale = annoAttuale;
55      }
56  }
```

```
1  public class TutoriDottorando
2  {
3      [Key]
4      public int Id { get; set; }
5
6      [Comment("Tutore o cotutore")]
7      public string Ruolo { get; set; }
8      public Tutori Tutori { get; set; }
9
10     public int TutoriId { get; set; }
11
12     public Dottorando Dottorando { get; set; }
```

```
13      public int DottorandoId { get; set; }
14
15      public DateTime DataInizioRuolo { get; set; } = DateTime.
    Now;
16      public DateTime DataFineRuolo { get; set; } = DateTime.
    MaxValue;
17
18      public IEnumerable<TutoriGiudizio> Giudizi { get; set; }
19
20      public TutoriDottorando() {
21          Ruolo = "";
22
23      }
24
25      public TutoriDottorando(RuoloTutCotut ruolo, int tutoriId
    , int dottorandoId, DateTime dataInizioRuolo, DateTime
    dataFineRuolo)
26      {
27          Ruolo = ruolo.ToString();
28          TutoriId = tutoriId;
29          DottorandoId = dottorandoId;
30          DataInizioRuolo = dataInizioRuolo;
31          DataFineRuolo = dataFineRuolo;
32      }
33  }
```

A.3 Codice del DbContext: `PhdReviewContext`

```
1  public class PhdReviewContext : DbContext
2  {
3      public PhdReviewContext (DbContextOptions<
       PhdReviewContext> options)
4          : base(options){}
5
6      public DbSet<AnnoDottorato> AnnoDottorato { get; set; }
7      public DbSet<AnnoDottoratoDocumenti>
       AnnoDottoratoDocumenti { get; set; }
8      public DbSet<Collegio> Collegio { get; set; }
9      public DbSet<CollegioGiudizio> CollegioGiudizio { get;
       set; }
10     public DbSet<CollegioComponenti> CollegioComponenti { get
       ; set; }
11     public DbSet<CollegioMembri> CollegioMembri { get; set; }
12     public DbSet<Commissione> Commissione { get; set; }
13     public DbSet<CommissioneComponenti> CommissioneComponenti
       { get; set; }
14     public DbSet<CommissioneGiudizio> CommissioneGiudizio {
       get; set; }
15     public DbSet<CommissioneGiudizioDati>
       CommissioneGiudizioDati { get; set; }
16     public DbSet<CommissioneGiudizioDatiTemplate>
       CommissioneGiudizioDatiTemplate { get; set; }
17     public DbSet<CommissioneMembri> CommissioneMembri { get;
       set; }
18     public DbSet<Dottorando> Dottorando { get; set; }
19     public DbSet<DottorandoDatiDaDidattica>
       DottorandoDatiDaDidattica { get; set; }
20     public DbSet<LogErrori> LogErrori { get; set; }
21     public DbSet<Tutori> Tutori { get; set; }
22     public DbSet<TutoriDottorando> TutoriDottorando { get;
       set; }
23     public DbSet<TutoriGiudizio> TutoriGiudizio { get; set; }
24     public DbSet<Email> Email { get; set; } = default!;
25     public DbSet<Ruoli> Ruoli { get; set; }
26     public DbSet<RuoliUtenti> RuoliUtenti { get; set; }
27     public DbSet<UserData> UserData { get; set; }
28
29     protected override void OnModelCreating(ModelBuilder
       modelBuilder)
```

```
30      {
31          modelBuilder.Entity<AnnoDottorato>().ToTable("
    AnnoDottorato");
32          modelBuilder.Entity<AnnoDottoratoDocumenti>().ToTable
    ("AnnoDottoratoDocumenti");
33          modelBuilder.Entity<Collegio>().ToTable("Collegio");
34          modelBuilder.Entity<CollegioGiudizio>().ToTable("
    CollegioGiudizio").Property(e => e.InsertDate).
    ValueGeneratedOnAdd().HasDefaultValueSql("GETDATE()");
35          modelBuilder.Entity<CollegioComponenti>().ToTable("
    CollegioComponenti");
36          modelBuilder.Entity<CollegioMembri>().ToTable("
    CollegioMembri");
37          modelBuilder.Entity<Commissione>().ToTable("
    Commissione");
38          modelBuilder.Entity<CommissioneGiudizio>().ToTable("
    CommissioneGiudizio");
39          modelBuilder.Entity<CommissioneGiudizioDati>().
    ToTable("CommissioneGiudizioDati");
40          modelBuilder.Entity<CommissioneGiudizioDatiTemplate
    >().ToTable("CommissioneGiudizioDatiTemplate");
41          modelBuilder.Entity<CommissioneComponenti>().ToTable
    ("CommissioneComponenti");
42          modelBuilder.Entity<CommissioneMembri>().ToTable("
    CommissioneMembri");
43          modelBuilder.Entity<Dottorando>().ToTable("Dottorando
    ");
44          modelBuilder.Entity<DottorandoDatiDaDidattica>().
    ToTable("DottorandoDatiDaDidattica");
45          modelBuilder.Entity<Tutori>().ToTable("Tutori");
46          modelBuilder.Entity<TutoriDottorando>().ToTable("
    TutoriDottorando");
47          modelBuilder.Entity<TutoriGiudizio>().ToTable("
    TutoriGiudizio");
48          modelBuilder.Entity<LogErrori>().ToTable("LogErrori")
    ;
49          modelBuilder.Entity<Ruoli>().ToTable("Ruoli");
50          modelBuilder.Entity<RuoliUtenti>().ToTable("
    RuoliUtenti");
51          modelBuilder.Entity<UserData>().ToTable("UserData");
52      }
53  }
```

Bibliografia

[1] Tim Berners-Lee, Robert Cailliau, Jean-François Groff e Berner Luotonen. «The World-Wide Web». In: *Computer Networks and ISDN Systems* 25.4-5 (1991), pp. 454–459. DOI: 10.1016/0169-7552(92)90005-V (cit. a p. 2).

[2] Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. Dissertation. University of California, Irvine, 2000 (cit. a p. 2).

[3] Tim Berners-Lee. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. Harper San Francisco, 1999 (cit. a p. 2).

[4] Martin Fowler. *Patterns of Enterprise Application Architecture*. Pearson Education, Inc, 2003. ISBN: 0-321-12742-0 (cit. a p. 5).

[5] Martin Fowler. *The evolution of MVC and other UI architectures*. 2006. URL: http://martinfowler.com/eaaDev/uiArchs.html (cit. a p. 5).

[6] F. Marcone. *Il paradiigma MVC in Django*. 2009. URL: https://www.html.it/pag/17904/il-paradigma-mvc-in-django/ (cit. a p. 5).

[7] Jesse James Garrett. «Ajax: A New Approach to Web Applications». In: *Adaptive Path Essays* (2005). URL: https://adaptivepath.org/ideas/ajax-new-approach-web-applications/ (cit. a p. 6).

[8] Leonard Richardson e Sam Ruby. «RESTful Web Services». In: *O'Reilly Media* (2008) (cit. a p. 6).

[9] Dino Esposito. *Programming Microsoft ASP.NET*. Microsoft Press, 2002 (cit. a p. 7).

[10] Adam Freeman. *Pro ASP.NET Core 6*. 9th. Apress, 2022 (cit. a p. 8).

[11] Phil Haack e Scott Guthrie. «Introducing ASP.NET Razor». In: *Microsoft Developer Network (MSDN)* (2010). URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor (cit. a p. 8).

[12] E. F. Codd. «A Relational Model of Data for Large Shared Data Banks». In: *Communications of the ACM* 13.6 (1970), pp. 377–387. DOI: 10.1145/362384.362685 (cit. a p. 8).

[13] C. J. Date. *An Introduction to Database Systems*. 8th. Addison-Wesley, 2004 (cit. a p. 8).

[14] Julia Lerman e Rowan Miller. *Programming Entity Framework: Code First*. O'Reilly Media, 2019 (cit. a p. 9).

[15] Microsoft Corporation. *SQL Server Documentation*. 2023. URL: https://learn.microsoft.com/en-us/sql/sql-server/ (cit. a p. 9).

[16] Thomas Erl. *Cloud Computing: Concepts, Technology & Architecture*. Prentice Hall, 2016 (cit. a p. 10).

[17] Daniel Roth e Steve Sanderson. *Blazor in Action*. Manning Publications, 2022 (cit. a p. 10).

[18] Stuart Russell e Peter Norvig. «Artificial Intelligence: A Modern Approach». In: *Pearson* (2021) (cit. a p. 10).

[19] Microsoft. *What's new in .NET 9: Overview*. 2024. URL: https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-9/overview (cit. a p. 13).

[20] Syncfusion. *What's New in .NET 9: A Developer's Perspective*. 2024. URL: https://www.syncfusion.com/blogs/post/whats-new-in-dotnet-9 (cit. alle pp. 13, 16).

[21] Microsoft. *What's new in .NET 9: Runtime*. 2024. URL: https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-9/runtime (cit. alle pp. 14, 16).

[22] ByteHide. *Everything New in .NET 9: The Ultimate Developer's Guide*. 2024. URL: https://www.bytehide.com/blog/everything-new-in-net-9-the-ultimate-developers-guide (cit. a p. 14).

[23] Microsoft. *C# Language Specification*. 2023. URL: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/ (cit. a p. 14).

[24] A. Freeman. *Pro ASP.NET Core 6*. ASP.NET Core and Blazor architecture and patterns. Apress, 2021 (cit. a p. 17).

[25] Microsoft. *ASP.NET Core Blazor - Overview*. 2024. URL: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-9.0 (cit. a p. 17).

[26] N. Kramer. *Blazor Basics for Beginners*. 2024. URL: https://daily.dev/blog/blazor-basics-for-beginners (cit. a p. 17).

[27] Microsoft. *Blazor Hosting Models*. 2024. URL: https://learn.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-9.0 (cit. alle pp. 18, 19).

[28] Microsoft. *Razor Pages architecture and concepts in ASP.NET Core*. 2025. URL: https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-9.0&tabs=visual-studio (cit. a p. 19).

[29] Microsoft. *Project structure for Blazor apps*. 2024. URL: https://learn.microsoft.com/en-us/dotnet/architecture/blazor-for-web-forms-developers/project-structure (cit. a p. 19).

[30] Microsoft. *SQL Server Documentation*. 2024. URL: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16 (cit. a p. 21).

[31] Microsoft. *Entity Framework Core Documentation*. 2024. URL: https://learn.microsoft.com/ef/core (cit. a p. 21).

[32] Microsoft. *Internet Information Services (IIS) Documentation*. 2024. URL: https://learn.microsoft.com/en-us/iis/ (cit. a p. 23).

[33] Shibboleth Consortium. *Shibboleth Documentation*. 2024. URL: https://shibboleth.net/documentation/ (cit. a p. 26).

[34] R. T. Fielding. «Architectural Styles and the Design of Network-based Software Architectures». Defines REST architectural principles. Tesi di dott. University of California, Irvine, 2000 (cit. a p. 27).

Ringraziamenti

Questa tesi non sarebbe stata possibile senza l'aiuto e il supporto di alcune persone: in primo luogo mia moglie Elisabetta, che mi ha spronato a completare questo percorso e mi ha supportato (e sopportato) in questi mesi di sviluppo dell'applicazione e di stesura della tesi; grazie amore mio.
In secondo luogo i miei genitori, grazie ai quali ho potuto cominciare questo percorso, per l'aiuto dato negli anni, e che hanno atteso fino ad oggi per vedermelo terminare.
Un grazie anche ai miei suoceri per il sostegno e il supporto dato in questi mesi di stesura della tesi.
Voglio anche ringraziare i miei colleghi del Labinf che hanno mostrato disponibilità e collaborazione durante il mio lavoro sulla tesi, rendendo possibile conciliare tutto questo con il lavoro.
Ringrazio anche Fulvio e Davide per il supporto fornito in questi mesi di lavoro.
Infine, voglio ringraziare tutti gli amici e i compagni di studi con i quali ho condiviso il percorso e tutti coloro che negli anni hanno reso tutto questo possibile.