UNIVERSITÀ
DI PARMA

Dipartimento di Scienze Matematiche, Fisiche e Informatiche

Corso di Laurea [Triennale in Informatica]

Titolo in Italiano

Titolo in Inglese

Candidato:
Dennis Turco

Relatore:
Prof. Nome Cognome

Correlatori:
Prof. co-Nome co-Cognome
Prof. co-Nome2 co-Cognome2

Anno Accademico 2020–2023

Dedica

Indice

Introduzione **1**

1 Le caratteristiche dell'OnBoarding **4**

2 Il modello dell'OnBoarding nell'azienda **5**

3 Le tecnologie utilizzate **6**
3.1 Tecnologie e Servizi 6
3.1.1 ASP.NET MVC 6.0 6
3.1.2 Autenticazione con "Identity" 7
3.1.3 GitHub 8
3.1.4 Git 8
3.2 Linguaggi 9
3.2.1 SQL 9
3.2.2 C# 9
3.2.3 HTML, CSS, JavaScript 9

4 Il progetto e il suo sviluppo **10**
4.1 Database 10
4.1.1 Progettazione 10
4.1.2 Scrittura nel linguaggio SQL 12
4.1.3 Interrogazioni al Database 15
4.2 Scrittura del codice C# 16
4.3 Layout della piattaforma 19

5 Risultati e sviluppi futuri **20**

Conclusione **21**

Bibliografia **22**

Elenco delle figure

4.1 progettazione del database 11
4.2 Tree structure del database 12

Elenco degli algoritmi

1 algoritmo di hashing per criptare le password usato dal servizio Identity 8
2 classe per ottenere la stringa di connessione al database 17
3 esempio funzione per l'esecuzione di una query da codice tramite il framework Dapper 18

Elenco delle tabelle

Introduzione

Nel seguente elaborato tratterò una relazione relativa al progetto di tirocinio svolto presso un'azienda software house "ISolutions" di Noceto.

Il progetto si concentra sulla creazione di una piattaforma di e-learning per la gestione del processo di OnBoarding aziendale. L'obiettivo è di fornire ai nuovi dipendenti un'esperienza guidata, strutturata e personalizzata attraverso la piattaforma web.

Attualmente, l'azienda gestisce tra 20 e 30 processi di OnBoarding dei nuovi dipendenti ogni anno, grazie al personale HR, utilizzando un foglio di calcolo Excel per la gestione dei dati e del completamento dei vari task. Tuttavia questo metodo si basa su un funzionamento manuale di aggiornamento dei task completati dai vari dipendenti e richiede una supervisione umana costante, il che lo rende inefficiente e non scalabile. Inoltre, le informazioni raccolte attraverso il modulo post-OnBoarding non sono sempre esaustive e dettagliate. A causa dell'espansione dell'azienda, questo sistema sta diventando sempre più inefficiente e rischioso. Per questo motivo risulta importante l'implementazione di un software dedicato al processo di OnBoarding dei dipendenti, con lo scopo di garantire una riduzione del rischio di errori e di migliorare l'esperienza generale. Inoltre, il software deve permettere di automatizzare alcune attività ripetitive, liberando tempo prezioso per il team HR, che potrà concentrarsi su attività di maggiore valore aggiunto per l'azienda. Infine, il nuovo sistema di OnBoarding ha la necessità di consentire l'acquisizione e l'archiviazione in maniera più sicura e organizzata dei dati dei dipendenti, rispettando le normative sulla privacy e semplificando le attività di audit interno.

Il processo di OnBoarding avrà una durata di circa un mese per uno sviluppatore junior e due settimane per uno sviluppatore senior. La piattaforma guiderà i nuovi dipendenti attraverso i vari task previsti in modo strutturato e personalizzato, fornendo loro gli strumenti necessari per acquisire le cono-

scenze e le competenze richieste per il loro ruolo.
Inoltre, la piattaforma fornirà un modulo per la raccolta di feedback post-OnBoarding. Il modulo sarà strutturato in modo da raccogliere informazioni dettagliate e utili per migliorare continuamente il processo di OnBoarding. Questo feedback sarà utilizzato per aggiornare e migliorare costantemente la piattaforma.
Infine, poiché l'azienda è certificata ISO 9001 (certificazione della qualità), la piattaforma fornirà un'ulteriore valutazione del processo di OnBoarding. Dopo che il dipendente ha completato il processo, verrà chiesto di esprimere un giudizio attraverso un modulo dedicato. Ciò permetterà all'azienda di comprendere se il processo di OnBoarding sta funzionando correttamente e se ci sono aree che possono essere migliorate.
In sintesi, la piattaforma di e-learning per la gestione del processo di OnBoarding aziendale rappresenta un'importante innovazione per l'azienda. Grazie alla sua efficienza e scalabilità, la piattaforma migliorerà l'esperienza del dipendente e ridurrà il carico di lavoro del personale HR. Inoltre, la raccolta di feedback dettagliati post-OnBoarding e la valutazione attraverso il modulo dedicato forniranno all'azienda le informazioni necessarie per migliorare costantemente il processo di OnBoarding.

Il processo pricincipale di OnBoarding aziendale deve prevedere le seguenti fasi.

- Introduzione (˜10 tasks);
- Setup della work station (˜20 tasks);
- Documentazione, sia amministrativa che tecnica (˜30 tasks);
- Video introduttivi, tecnici e orientati ai prodotti sviluppati (˜10 tasks);
- Overview dell'organizzazione aziendale e degli strumenti utilizzati (˜10 tasks);
- Hands On degli applicativi aziendali (˜20 tasks);
- Formazione sui principali linguaggi di programmazione utilizzati (˜20 tasks);
- Overview struttura codice delle soluzioni software (˜10 tasks);
- Test finale con revisione e valutazione.

Sono previsti inoltre alcuni step preliminari, in carico al team HR, per la predispozione di quanto necessario (creazione utenza, preparazione pc etc. . .), e una fase finale di retrospettiva per raccogliere feedback riguardo il processo dal neo assunto.

Alcune desiderate del progetto:

- Integrazione autenticazione con modulo di login e gestione livelli di autorizzazione;
- Pannello amministrativo per il team HR per gestire (creazione, modifica, eliminazione) dei vari tasks;
- Tracciamento tempo impiegato sui vari task e reportistica per team HR;
- Possibilità di avviare, sospendere e saltare un task;
- Integrazione fase finale di test, revisione e valutazione.

Il progetto di creazione della piattaforma di e-learning per la gestione del processo di OnBoarding aziendale è stato realizzato come web application .NET MVC 6.0 utilizzando molteplici tecnologie e linguaggi di programmazione tra cui: C#, HTML, Css, Javascript, SQL.

- Capitolo 1 → Le caratteristiche dell'OnBoarding.
- Capitolo 2 → Il modello dell'OnBoarding nell'azienda.
- Capitolo 3 → Le tecnologie utilizzate.
- Capitolo 4 → Il progetto ed il suo sviluppo.
- Capitolo 5 → Risultati e sviluppi futuri.

Capitolo 1

Le caratteristiche dell'OnBoarding

Capitolo 2

Il modello dell'OnBoarding nell'azienda

Capitolo 3

Le tecnologie utilizzate

Il progetto di creazione della piattaforma di e-learning per la gestione del processo di OnBoarding aziendale è stato realizzato utilizzando molteplici tecnologie, servizi e linguaggi:

3.1 Tecnologie e Servizi

1. ASP.NET MVC 6.0;
2. Autenticazione;
3. GitHub;
4. Git;

3.1.1 ASP.NET MVC 6.0

ASP.NET Core MVC è un ricco framework creato da Microsoft per la creazione di applicazioni web e API utilizzando il modello di progettazione Model-View-Controller.

- Model: Il backend che contiene tutta la logica dei dati;
- View: interfaccia utente frontend o grafica (GUI);
- Controller: Il "cervello" dell'applicazione che controlla come vengono visualizzati i dati.

3.1.2 Autenticazione con "Identity"

Siccome il progetto in questione riguarda la creazione di una piattaforma web, è stato necessario implementare il servizio di Autenticazione attraverso il framework ASP.NET Core Identity, al fine di otterenere non soltanto la possibilità di permettere le azioni di login e registrazione da parte degli utenti, ma anche di farlo ottenendo una sicurezza dei dati sensibili degli utenti.

In particolare la forma di Autenticazione utilizzata nella piattaforma è una "Forms Authentication", ovvero, è quel tipo di autenticazione nella quale l'utente deve fornire le proprie credenziali attraverso un form dedicato.

ASP.NET Core Identity è un API che supporta funzionalità di login lato UI. Gestisce utenti, password, dati del profilo, ruoli, email di conferma e molto altro. In particolare, gli utenti possono creare un account con le informazioni di accesso memorizzate in Identity o possono utilizzare un provider di accesso esterno (es. Facebook, Google, ecc. . .).

Nota: L'algoritmo utilizzato dal servizio Identity per criptare le password è PBKDF2 (Password-Based Key Derivation Function 2):

Algoritmo 1 algoritmo di hashing per criptare le password usato dal servizio Identity

```
public static string HashPassword(string password)
{
    byte[] salt;
    byte[] bytes;
    if (password == null)
    {
        throw new ArgumentNullException("password");
    }
    using (Rfc2898DeriveBytes rfc2898DeriveByte = new
    Rfc2898DeriveBytes(password, 16, 1000))
    {
        salt = rfc2898DeriveByte.Salt;
        bytes = rfc2898DeriveByte.GetBytes(32);
    }
    byte[] numArray = new byte[49];
    Buffer.BlockCopy(salt, 0, numArray, 1, 16);
    Buffer.BlockCopy(bytes, 0, numArray, 17, 32);
    return Convert.ToBase64String(numArray);
}

```

3.1.3 GitHub

Il progetto durante il suo sviluppo è stato salvato in un'apposità repository privata su GitHub. GitHub è stato fondamentale per tenere una traccia storica di tutte le modifiche apportate nel corso del tempo, in risposta all'esecuzione dei vari task assegnati.
Esso non si è limitato ad essere un semplice strumento a d'uso esclusivamente individuale, ma ha permesso la condivisione del lavoro sia con il tutor aziendale che con il personale aziendale.

3.1.4 Git

In parallelo all'utilizzo di GitHub è stato impiegato il software di controllo di versione Git con l'obbiettivo di agevolare l'aggiornamento dei file del progetto dalla workspace locale verso la repository privata che ospita il progetto, tramite appositi commit.

3.2 Linguaggi

1. SQL;
2. C#;
3. HTML, CSS, JavaScript;

3.2.1 SQL

In particolare, il linguaggio SQL è stato utilizzato per la creazione del database del sito web. Il database contiene tutte le informazioni necessarie per la corretta gestione della piattaforma, tra cui i dati relativi agli utenti registrati, ai corsi e alle categorie dei corsi.

3.2.2 C#

Il linguaggio C# è stato utilizzato per la gestione della parte backend del sito, creando un dialogo tra le informazioni contenute nel database e l'interfaccia utente. L'accesso ai dati del database tramite il linguaggio C# è stato possibile grazie al framework "Dapper", che consente di eseguire interrogazioni in modo efficace ed efficiente. In questo modo, il sito è più responsivo anche in caso di grandi quantità di dati da gestire, eliminando elementi come le procedure, che risultano meno manutenibili e più complicate da gestire a causa di una complicazione del codice nella parte backend per il loro utilizzo.

Il linguaggio C# è stato anche utilizzato per la gestione degli eventi, la navigazione tra le pagine e i controlli di vario genere. Il progetto di creazione della piattaforma di e-learning per la gestione del processo di OnBoarding aziendale ha richiesto un'attenta pianificazione e la scelta di tecnologie adeguate per la realizzazione di un prodotto di alta qualità. L'utilizzo di molteplici tecnologie ha permesso di ottenere un prodotto completo e funzionale, in grado di soddisfare le esigenze dell'azienda e dei nuovi dipendenti.

3.2.3 HTML, CSS, JavaScript

Per la gestione dell'interfaccia utente, sono stati utilizzati il linguaggio HTML, il linguaggio CSS e il linguaggio JavaScript. Il linguaggio HTML è stato utilizzato per la creazione della struttura del sito, il linguaggio CSS per la gestione dello stile e il linguaggio JavaScript per la gestione degli eventi.

Capitolo 4

Il progetto e il suo sviluppo

4.1 Database

Una parte sostanziale del nostro impegno aziendale per portare a compimento il progetto si è concentrata in modo dettagliato sulla creazione del database, ritenuto elemento cruciale per assicurare il pieno funzionamento della piattaforma.

La creazione del database è stata articolata in diverse fasi chiave:

- Progettazione della struttura del database, un processo attentamente studiato.
- Scrittura del database in linguaggio SQL, una tappa essenziale.
- Implementazione delle interrogazioni al database nella sezione di backend, facendo uso della libreria Dapper.

Nota: il database è stato creato e gestito unicamente in locale per questioni di semplicità e per l'esecuzione di test sulla correttezza della struttura e dell'implementazione del database stesso.

4.1.1 Progettazione

La fase di progettazione 4.1 del database ha occupato un ruolo fondamentale nel corso di questo processo, coinvolgendo un'analisi costante e una riflessione profonda. Il nostro obiettivo principale era plasmare un database estremamente completo, in grado di soddisfare appieno le esigenze della piattaforma di e-learning. In aggiunta, ci siamo concentrati su:

- Migliorare la leggibilità del database;

- Rendere più agevole la manutenzione;
- Fornire ampie possibilità di estensione del sistema.

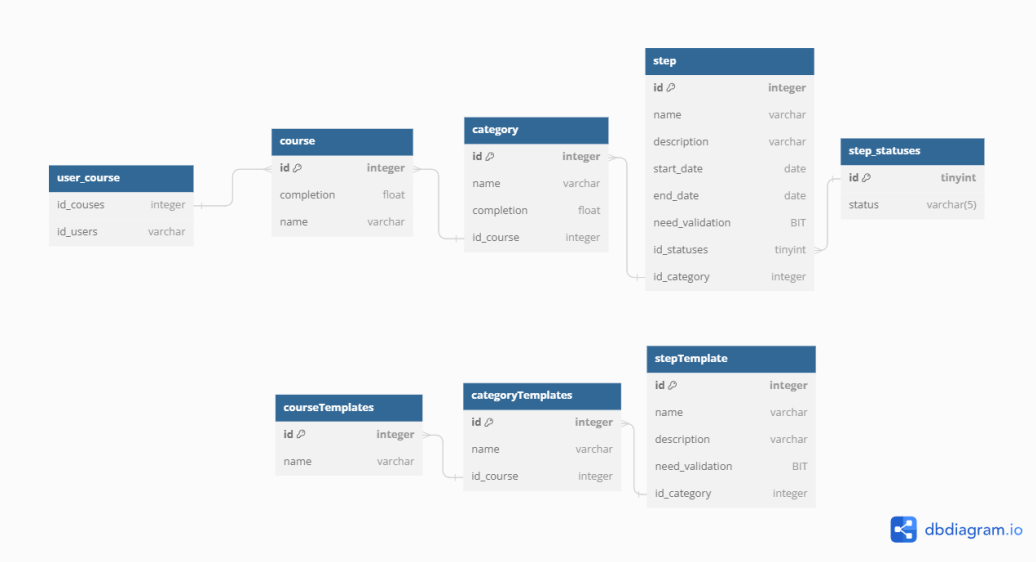

Figura 4.1: progettazione del database

Nota: la progettazione mostrata non tiene traccia delle tabelle utente, perchè vengono gestite automaticamente dal progetto .NET grazie al servizio di autenticazione già incluso (libreria `Microsoft.AspNet.Identity.EntityFramework;`).

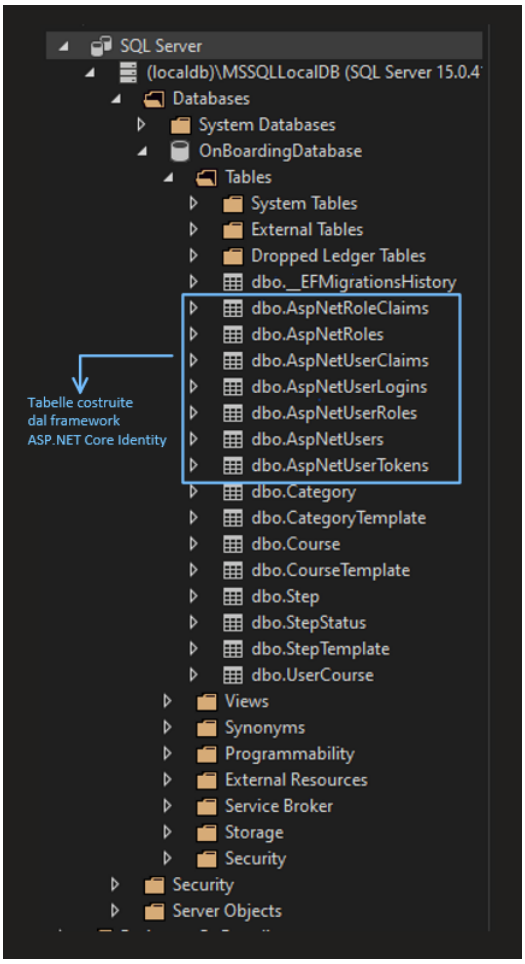

Figura 4.2: Tree structure del database

4.1.2 Scrittura nel linguaggio SQL

Il risultato finale di questa fase è il seguente (non sono riportate le tabelle generate dal servizio di Autenticazione, perchè gestite automaticamente dalla libreria a disposizione):

```
CREATE TABLE [dbo].[Course] (
    [Id]          INT             IDENTITY (1, 1) NOT NULL,
    [Name]        NVARCHAR (255)  NOT NULL,
    [Completion] REAL             DEFAULT ((0)) NULL,
    [StartDate]  DATETIME2 (7)   NULL,
    [EndDate]    DATETIME2 (7)   NULL,
    CONSTRAINT [PK_Course] PRIMARY KEY CLUSTERED ([Id] ASC)
);
```

```
9
10 CREATE TABLE [dbo].[Category] (
11     [Id]         INT            IDENTITY (1, 1) NOT NULL,
12     [Name]       NVARCHAR (255) NOT NULL,
13     [Completion] REAL           DEFAULT ((0)) NULL,
14     [IdCourse]   INT            NOT NULL,
15     CONSTRAINT [PK_Category] PRIMARY KEY CLUSTERED ([Id] ASC),
16     CONSTRAINT [FK_Category_Course_IdCourse]
17   FOREIGN KEY ([IdCourse]) REFERENCES [dbo].[Course] ([Id])
18   ON DELETE CASCADE
19    );
20    GO
21    CREATE NONCLUSTERED INDEX [IX_Category_IdCourse]
22      ON [dbo].[Category]([IdCourse] ASC);
23
24 -- 3 stati:
25   -- 1. done
26   -- 2. todo
27   -- 3. check
28 CREATE TABLE [dbo].[StepStatus] (
29   [Id]     INT            IDENTITY (1, 1) NOT NULL,
30   [Status] NVARCHAR (10) NOT NULL,
31   CONSTRAINT [PK_StepStatus] PRIMARY KEY CLUSTERED ([Id] ASC)
32 );
33
34 CREATE TABLE [dbo].[Step] (
35   [Id]             INT            IDENTITY (1, 1) NOT NULL,
36   [Name]           NVARCHAR (255) NOT NULL,
37   [Description]    NVARCHAR (MAX) NULL,
38   [StartDate]      DATETIME2 (7)  NULL,
39   [EndDate]        DATETIME2 (7)  NULL,
40   [Lock]           BIT            DEFAULT ((1)) NULL,
41   [NeedValidation] BIT            DEFAULT ((0)) NULL,
42   [IdStatus]       INT            DEFAULT ((1)) NULL,
43   [IdCategory]     INT            NOT NULL,
44   CONSTRAINT [PK_Step] PRIMARY KEY CLUSTERED ([Id] ASC),
45   CONSTRAINT [FK_Step_StepStatus_IdStatus]
46 FOREIGN KEY ([IdStatus]) REFERENCES [dbo].[StepStatus] ([Id])
47 ON DELETE CASCADE,
48   CONSTRAINT [FK_Step_Category_IdCategory]
49 FOREIGN KEY ([IdCategory]) REFERENCES [dbo].[Category] ([Id])
50 ON DELETE CASCADE
51 );
```

```
GO
CREATE NONCLUSTERED INDEX [IX_Step_IdCategory]
  ON [dbo].[Step]([IdCategory] ASC);
GO
CREATE NONCLUSTERED INDEX [IX_Step_IdStatus]
  ON [dbo].[Step]([IdStatus] ASC);

CREATE TABLE [dbo].[UserCourse] (
  [UserId]   NVARCHAR (450) NOT NULL,
  [CourseId] INT            NOT NULL,
  CONSTRAINT [FK_UserCourse_Course_CourseModel]
FOREIGN KEY ([CourseId]) REFERENCES [dbo].[Course] ([Id])
ON DELETE CASCADE,
  CONSTRAINT [FK_UserCourse_AspNetUsers_UserId]
FOREIGN KEY ([UserId]) REFERENCES [dbo].[AspNetUsers] ([Id])
ON DELETE CASCADE
);
GO
CREATE NONCLUSTERED INDEX [IX_UserCourse_CourseModel]
    ON [dbo].[UserCourse]([CourseId] ASC);
GO
CREATE NONCLUSTERED INDEX [IX_UserCourse_UserId]
    ON [dbo].[UserCourse]([UserId] ASC);

CREATE TABLE [dbo].[CourseTemplate] (
    [Id]             INT            IDENTITY (1, 1) NOT NULL,
    [Name]           NVARCHAR (255) NOT NULL,
    [Creator]        NVARCHAR (255) NOT NULL,
    [CreationDate]   DATETIME2 (7)  NULL,
    [LastUpdateDate] DATETIME2 (7)  NULL,
    CONSTRAINT [PK_CourseTemplate] PRIMARY KEY CLUSTERED ([Id] ASC)
);

CREATE TABLE [dbo].[CategoryTemplate] (
    [Id]                INT         IDENTITY (1, 1) NOT NULL,
    [name]              NCHAR (255) NOT NULL,
    IDCourseTemplate INT         NOT NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC),
    FOREIGN KEY (IDCourseTemplate)
 REFERENCES [dbo].[CourseTemplate] ([Id])
 ON DELETE CASCADE
);

```

```
95  CREATE TABLE [dbo].[StepTemplate] (
96      [Id]                  INT             IDENTITY (1, 1) NOT NULL,
97      [Name]                NVARCHAR (255) NOT NULL,
98      [Description]         NVARCHAR (MAX) NOT NULL,
99      [NeedValidation]      BIT             DEFAULT ((0)) NULL,
100     [IDCategoryTemplate] INT             NOT NULL,
101     CONSTRAINT [PK_StepTemplate] PRIMARY KEY CLUSTERED ([Id] ASC),
102     CONSTRAINT [FK_StepTemplate_CategoryTemplate_IDCategoryTemplate
        ]
103  FOREIGN KEY ([IDCategoryTemplate]) REFERENCES [dbo].[
        CategoryTemplate] ([Id])
104  ON DELETE CASCADE
105 );
106 GO
107 CREATE NONCLUSTERED INDEX [IX_StepTemplate_IDCategoryTemplate]
108     ON [dbo].[StepTemplate]([IDCategoryTemplate] ASC);
```

Codice 4.1: traduzione della progettazione del database nel linguaggio SQL

4.1.3 Interrogazioni al Database

Inizialmente, le interrogazioni al database erano state sviluppate attraverso stored procedure. Tra i numerosi vantaggi proposti, l'idea principale era ottenere:

- Riutilizzo del codice;
- Semplificazione della manutenzione;
- Prestazioni migliorate;

Tuttavia, durante lo sviluppo dell'applicativo, grazie al suggerimento di alcuni membri dell'azienda, si è preferito sostituire le procedure con interrogazioni dirette dal codice tramite il framework Dapper, al fine di poter migliorare:

- Prestazioni in termini di tempo;
- Semplificare il debugging del codice;
- Centrallizzare l'intera logica in un unico posto.

All'interno della piattaforma si possono distinguere 3 macro categorie di interrogazioni per:

- Inserimento dei dati per aggiungere:

  - corsi;
  - categorie (sotto tipo dei corsi);
  - step (sotto tipo delle categorie);
  - corsi template;
  - categorie template (sotto tipo dei corsi template);
  - step template (sotto tipo delle categorie template).

- Eliminazione dei dati per togliere:
  - corsi;
  - categorie (sotto tipo dei corsi);
  - step (sotto tipo delle categorie);
  - corsi template;
  - categorie template (sotto tipo dei corsi template);
  - step template (sotto tipo delle categorie template).

- Lettura dei dati per permettere di ottenere tutte le informazioni necessarie per la generazione corretta della pagina dinamica dato un determinato utente connesso;

4.2 Scrittura del codice C#

Nel contesto dello sviluppo della parte back-end del progetto, è stato optato l'impiego del linguaggio di programmazione C#. Questa scelta si è rivelata fondamentale per la gestione della logica e il comportamento del sito web. L'obbiettivo principale è stato garantire una gestione efficiente del database direttamente attraverso il codice, sfruttando appieno le potenzialità del framework Dapper.
Utilizzando C# e il framework Dapper, è stato possibile realizzare un corretto collegamento tra dati e utenti. Questo ha reso possibile la generazione dinamica e accurata delle pagine web, consentendo un'esperienza utente ottimale.

Per eseguire delle query al database è stato necessario creare una connessione dal codice C# al database attraverso la libreria `Microsoft.Data.SqlClient;`

Algoritmo 2 classe per ottenere la stringa di connessione al database

```
using Microsoft.Data.SqlClient;
using Microsoft.EntityFrameworkCore;

namespace OnBoarding.Services
{
  public class ConnectionService
  private SqlConnection _connection = new SqlConnection();
  private SqlCommand _command = new SqlCommand();

  public static IConfiguration? Configuration { get; set; }

  public string GetconnectionString()
  {
    var builder = new ConfigurationBuilder().SetBasePath(Directory.
    GetCurrentDirectory()).AddJsonFile("appsettings.json");

    Configuration = builder.Build();
    return Configuration.GetConnectionString("
    ApplicationDBContextConnection");
  }


  public SqlConnection GetConnection()
  {
    return _connection;
  }

  public SqlCommand GetCommand()
  {
    return _command;
  }

  }
}

```

In questo modo, una volta che è stata stabilita una connessione, è stato reso possibile il processo di scrittura di query. A titolo esemplificativo, si può menzionare la query nella funzione denominata “GetCourses” presente

all'interno del file "CourseManager.cs". Questa query, quando viene fornito l'ID dell'utente come input, ha lo scopo principale di recuperare e restituire l'insieme completo dei corsi associati all'utente specificato.

Algoritmo 3 esempio funzione per l'esecuzione di una query da codice tramite il framework Dapper

```
1  public List<CourseModel> GetCourses(string userid)
2  {
3    using (_connection = new SqlConnection(connectionService.
       GetconnectionString()))
4    {
5      _connection.Open();
6
7      string sql =
8        @"SELECT DISTINCT Course.*
9        FROM Course, AspNetUsers, UserCourse
10       WHERE UserCourse.UserId = @userId
11       AND UserCourse.CourseId = Course.Id";
12     var coursesList = _connection.Query<CourseModel>(sql, new {
       userId = userid }).ToList();
13
14     _connection.Close();
15
16     return coursesList;
17   }
18 }
19
```

Si osserva che ciascun utente può essere associato a più corsi, stabilendo così una relazione uno a molti. Pertanto, è fondamentale che il valore restituito dalla funzione sia di tipo `List<CourseModel >`, in quanto questa struttura dati può contenere più di un corso associato a un utente specifico. Inoltre, grazie all'uso di Dapper, è possibile effettuare un'interrogazione al database mediante una stringa appositamente creata (come mostrato nelle righe 7-11 del codice). Successivamente, è possibile ottenere il risultato dell'interrogazione corrispondente al parametro "userid" fornito alla funzione e al campo "userId" nella tabella "Course" (riga 12 del codice).
Notare che siccome la funzione ritorna in questo caso una lista, il risultato della query dovrà essere convertito in una collezione di oggetti dello stesso tipo (attraverso alla funzione di libreria `ToList()`).

4.3 Layout della piattaforma

Capitolo 5

Risultati e sviluppi futuri

Conclusione

Conclusione che riassume il lavoro svolto ed eventuali lavori futuri.

Ringraziamenti