



# 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. Roberto Alfieri**

CORRELATORI:  
**Prof. co-Nome co-Cognome**  
**Prof. co-Nome2 co-Cognome2**

*Dedica*

# Indice

<b>Introduzione</b>	<b>1</b>
<b>1 Le caratteristiche dell'OnBoarding</b>	<b>4</b>
<b>2 Il modello dell'OnBoarding nell'azienda</b>	<b>5</b>
<b>3 Le tecnologie utilizzate</b>	<b>6</b>
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
<b>4 Il progetto e il suo sviluppo</b>	<b>10</b>
4.1 Database . . . . .	10
4.1.1 Progettazione . . . . .	10
4.1.2 Scrittura nel linguaggio SQL . . . . .	12
4.1.3 Interrogazioni al Database . . . . .	12
4.2 Scrittura del codice C# . . . . .	14
4.3 Layout della piattaforma . . . . .	19
<b>5 Risultati e sviluppi futuri</b>	<b>21</b>
<b>Conclusione</b>	<b>22</b>
<b>Bibliografia</b>	<b>23</b>
<b>A Scrittura del Database</b>	<b>24</b>

# 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 per il collegamento al database . . . . .	15
3	esempio funzione per l'esecuzione di una query da codice tramite il framework Dapper . . . . .	16
4	esempio sezione codice Javascript per la creazione di un grafico a barre per la rappresentazione dei dati . . . . .	19

# **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 principale 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 predisposizione 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 ottenere 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.

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

---

```
1 public static string HashPassword(string password)
2 {
3     byte[] salt;
4     byte[] bytes;
5     if (password == null)
6     {
7         throw new ArgumentNullException("password");
8     }
9     using (Rfc2898DeriveBytes rfc2898DeriveByte = new
Rfc2898DeriveBytes(password, 16, 1000))
10    {
11        salt = rfc2898DeriveByte.Salt;
12        bytes = rfc2898DeriveByte.GetBytes(32);
13    }
14    byte[] numArray = new byte[49];
15    Buffer.BlockCopy(salt, 0, numArray, 1, 16);
16    Buffer.BlockCopy(bytes, 0, numArray, 17, 32);
17    return Convert.ToBase64String(numArray);
18 }
19
```

---

### 3.1.3 GitHub

Il progetto durante il suo sviluppo è stato salvato in un'apposita 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 resto del personale.

### 3.1.4 Git

In parallelo all'utilizzo di GitHub è stato impiegato il software di controllo di versione Git con l'obiettivo 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

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. 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 lavoro impiegato 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 per l'implementazione della struttura.
- 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. L'obiettivo principale è stato quello di plasmare un database estremamente completo, in grado di soddisfare appieno le esigenze della piattaforma di e-learning, ponendosi lo scopo aggiuntivo di:

- Migliorare la leggibilità del database;



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

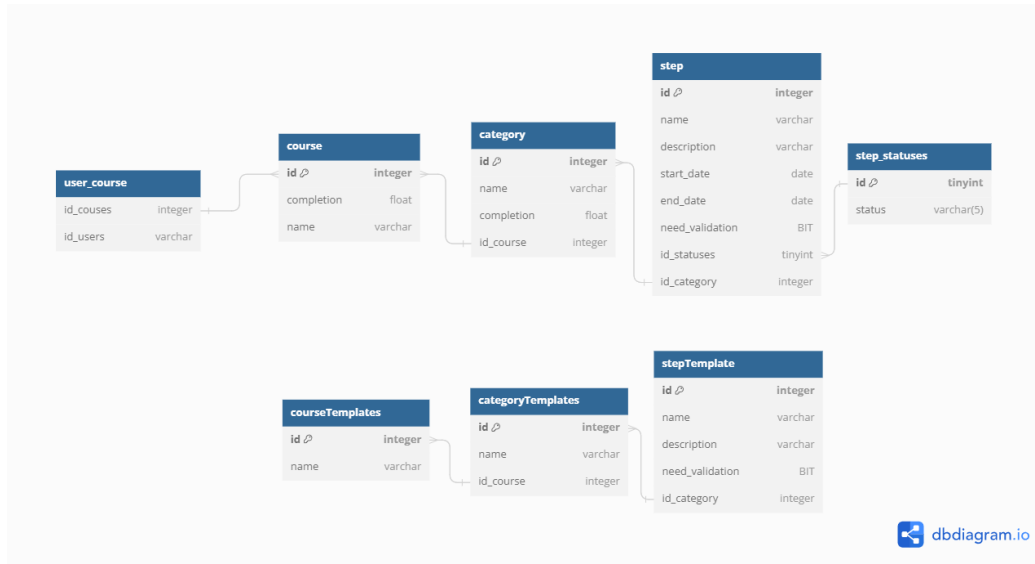


Figura 4.1: progettazione del database

*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`).

Si noti, inoltre, che non è stato creato un collegamento tra le tabelle “\*Templates” con il resto del tabelle perchè esse vengono gestite in maniera del tutto indipendente con le altre informazioni e relazioni presenti.

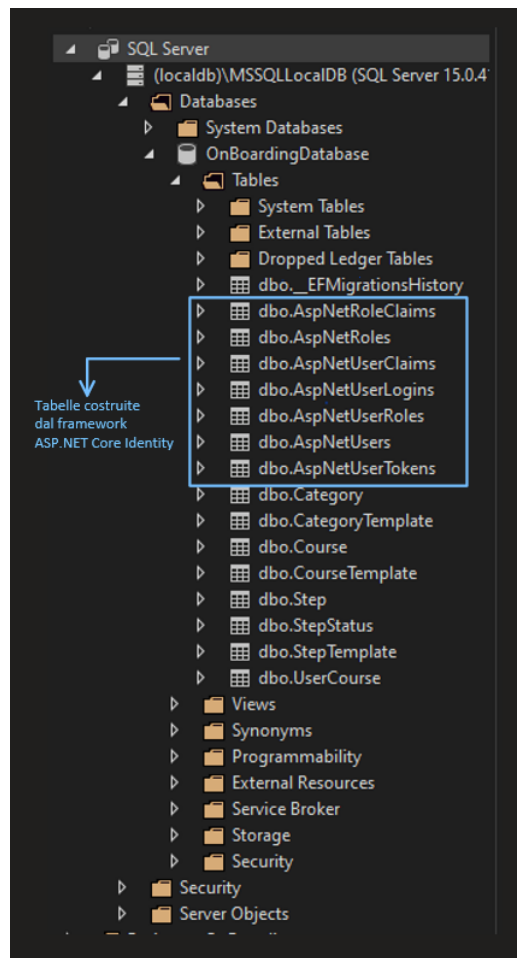


Figura 4.2: Tree structure del database

### 4.1.2 Scrittura nel linguaggio SQL

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

### 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 riguardanti:
  - utenti
  - corsi;
  - categorie (sotto gruppo dei corsi);
  - step (sotto gruppo delle categorie);
  - corsi template;
  - categorie template (sotto gruppo dei corsi template);
  - step template (sotto gruppo delle categorie template).
- Eliminazione dei dati riguardanti:
  - utenti
  - corsi;
  - categorie (sotto gruppo dei corsi);
  - step (sotto gruppo delle categorie);
  - corsi template;
  - categorie template (sotto gruppo dei corsi template);
  - step template (sotto gruppo 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#

La parte del codice back-end rappresenta il nucleo vitale dell'intero progetto, che ne garantisce il corretto funzionamento. Questa componente è stata completamente sviluppata utilizzando il linguaggio di programmazione C#.

L'obiettivo 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 per il collegamento al database

---

```
1 using Microsoft.Data.SqlClient;
2 using Microsoft.EntityFrameworkCore;
3
4 namespace OnBoarding.Services
5 {
6     public class ConnectionService
7     private SqlConnection _connection = new SqlConnection();
8     private SqlCommand _command = new SqlCommand();
9
10    public static IConfiguration? Configuration { get; set; }
11
12    public string GetconnectionString()
13    {
14        var builder = new ConfigurationBuilder().SetBasePath(Directory.
15            GetCurrentDirectory()).AddJsonFile("appsettings.json");
16
17        Configuration = builder.Build();
18        return Configuration.GetConnectionString("
19            ApplicationDBContextConnection");
20    }
21
22    public SqlConnection GetConnection()
23    {
24        return _connection;
25    }
26
27    public SqlCommand GetCommand()
28    {
29        return _command;
30    }
31 }
32 }
33
```

---

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(connectionStringService.
4         GetConnectionString()))
5     {
6         _connection.Open();
7
8         string sql =
9             @"SELECT DISTINCT Course.*
10             FROM Course,AspNetUsers,UserCourse
11             WHERE UserCourse.UserId = @userId
12             AND UserCourse.CourseId = Course.Id";
13         var coursesList = _connection.Query<CourseModel>(sql, new {
14             userId = userid }).ToList();
15
16         _connection.Close();
17
18         return coursesList;
19     }
20 }
```

---

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 `ToList()`).

Un punto di particolare rilevanza che merita menzione è rappresentato dallo sviluppo di una classe denominata “StatisticsController.cs”. Il suo scopo principale consiste nell’elaborare in modo esaustivo e dettagliato tutte le statistiche associate a un utente specifico. Questa classe si avvale della sofisticata libreria `ClosedXML.Excel`; per generare dati statistici inerenti al completamento dei corsi che sono stati non solo avviati, ma anche portati a termine dall’utente in questione. È essenziale sottolineare che questa funzionalità è rigorosamente riservata agli utenti che detengono il privilegiato ruolo di “Admin”.

Un aspetto di notevole rilevanza è stato il processo di gestione dei dati statistici all’interno della pagina denominata “/Views/Statistics/Index.cshtml”. Questa pagina ha una funzione fondamentale poiché consente agli admin di visualizzare l’insieme completo delle statistiche. Ciò è reso possibile grazie alla capacità di ordinare agevolmente le informazioni presenti all’interno delle tabelle. Questa caratteristica permette agli utenti di organizzare i dati in base alle loro specifiche esigenze e criteri di ricerca.

Inoltre, per rendere l’esperienza ancora più interattiva e informativa, è stata inclusa una sezione di codice dedicata direttamente all’interno della pagina stessa, utilizzando il tag `<script>...</script>`. Questo approccio ha consentito di incorporare efficacemente dei grafici che offrono una rappresentazione visiva dei dati statistici. Questi grafici sono stati sviluppati in modo da offrire quattro diverse visualizzazioni, ognuna delle quali fornisce un’analisi dettagliata dei dati statistici, permettendo agli utenti di comprendere meglio i pattern e le tendenze sottostanti.

- due grafici a barre;
  - Il primo rappresenta la somma totale dei corsi suddivisi per stato di completamento (done, running, todo);
  - Il secondo rappresenta il numero di corsi completati in relazione al mese dell’anno.
- due grafici a torta;
  - Il primo rappresenta il tempo medio di completamento necessario per terminare uno specifico corso, suddiviso per corso;
  - Il secondo rappresenta il tempo medio di completamento necessario per terminare uno specifico corso, suddiviso per utenti.

L’implementazione del codice per la creazione dei grafici è stata effettuata mediante l’uso del linguaggio di programmazione JavaScript. Un aspetto interessante da sottolineare è che i dati numerici necessari per la creazione dei

grafici non sono stati ottenuti direttamente tramite il codice JavaScript, ma sono stati acquisiti attraverso apposite funzioni getter scritte in codice C#. Questo approccio è stato adottato per garantire l'accuratezza e la coerenza dei dati, nonché per sfruttare le funzionalità di gestione dei dati messe a disposizione dal linguaggio C#. In sostanza, il processo è stato il seguente: il codice JavaScript si è interfacciato con il codice C# attraverso apposite funzioni getter, ottenendo così i dati numerici necessari per alimentare i grafici.



**Algoritmo 4** esempio sezione codice Javascript per la creazione di un grafico a barre per la rappresentazione dei dati

---

```
1 <script>
2   // ...
3
4   /*----- BAR CHART
   -----*/
5   var xValues = ["To Do", "Running", "Done"];
6   var yValues = [@todo, @running, @done, 0];
7   var barColors = ["red", "#e5e600", "green"];
8
9   new Chart("BarChart", {
10     type: "bar",
11     data: {
12       labels: xValues,
13       datasets: [{
14         backgroundColor: barColors,
15         data: yValues
16       }]
17     },
18     options: {
19       legend: { display: false },
20       title: {
21         display: false,
22         text: ""
23       }
24     }
25   });
26
27   // ...
28 </script>
29
```

---

## 4.3 Layout della piattaforma

Il layout della piattaforma è stato attentamente progettato e realizzato con l'obiettivo di creare un servizio che sia estremamente intuitivo e facile da utilizzare, sia per gli utenti che per gli amministratori responsabili della gestione degli accessi e del progresso degli utenti durante il loro percorso nell'espe-

rienza. Grazie a questa attenzione al design e all'usabilità, è stato possibile assicurarsi che l'intera esperienza sia accessibile e fruibile in modo agevole e soddisfacente per tutti i suoi utilizzatori.

## Capitolo 5

### Risultati e sviluppi futuri

# Conclusione

Conclusione che riassume il lavoro svolto ed eventuali lavori futuri.

# Ringraziamenti

# Appendice A

## Scrittura del Database

```
1 CREATE TABLE [dbo].[Course] (  
2     [Id]          INT          IDENTITY (1, 1) NOT NULL,  
3     [Name]        NVARCHAR (255) NOT NULL,  
4     [Completion]   REAL          DEFAULT ((0)) NULL,  
5     [StartDate]   DATETIME2 (7) NULL,  
6     [EndDate]     DATETIME2 (7) NULL,  
7     CONSTRAINT [PK_Course] PRIMARY KEY CLUSTERED ([Id] ASC)  
8 );  
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 );
52 GO
53 CREATE NONCLUSTERED INDEX [IX_Step_IdCategory]
54 ON [dbo].[Step]([IdCategory] ASC);
55 GO
56 CREATE NONCLUSTERED INDEX [IX_Step_IdStatus]
57 ON [dbo].[Step]([IdStatus] ASC);
58
59 CREATE TABLE [dbo].[UserCourse] (
60     [UserId] NVARCHAR (450) NOT NULL,
61     [CourseId] INT NOT NULL,
62     CONSTRAINT [FK_UserCourse_Course_CourseModel]
63 FOREIGN KEY ([CourseId]) REFERENCES [dbo].[Course] ([Id])
64 ON DELETE CASCADE,
65     CONSTRAINT [FK_UserCourse_AspNetUsers_UserId]
66 FOREIGN KEY ([UserId]) REFERENCES [dbo].[AspNetUsers] ([Id])
67 ON DELETE CASCADE
68 );
69 GO
70 CREATE NONCLUSTERED INDEX [IX_UserCourse_CourseModel]
71 ON [dbo].[UserCourse]([CourseId] ASC);
72 GO

```

```

73 CREATE NONCLUSTERED INDEX [IX_UserCourse_UserId]
74     ON [dbo].[UserCourse]([UserId] ASC);
75
76 CREATE TABLE [dbo].[CourseTemplate] (
77     [Id] INT IDENTITY (1, 1) NOT NULL,
78     [Name] NVARCHAR (255) NOT NULL,
79     [Creator] NVARCHAR (255) NOT NULL,
80     [CreationDate] DATETIME2 (7) NULL,
81     [LastUpdateDate] DATETIME2 (7) NULL,
82     CONSTRAINT [PK_CourseTemplate] PRIMARY KEY CLUSTERED ([Id] ASC)
83 );
84
85 CREATE TABLE [dbo].[CategoryTemplate] (
86     [Id] INT IDENTITY (1, 1) NOT NULL,
87     [name] NCHAR (255) NOT NULL,
88     IDCourseTemplate INT NOT NULL,
89     PRIMARY KEY CLUSTERED ([Id] ASC),
90     FOREIGN KEY (IDCourseTemplate)
91 REFERENCES [dbo].[CourseTemplate] ([Id])
92 ON DELETE CASCADE
93 );
94
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 ]
104 FOREIGN KEY ([IDCategoryTemplate]) REFERENCES [dbo].[
105 CategoryTemplate] ([Id])
106 ON DELETE CASCADE
107 );
108 GO
109 CREATE NONCLUSTERED INDEX [IX_StepTemplate_IDCategoryTemplate]
110     ON [dbo].[StepTemplate]([IDCategoryTemplate] ASC);

```

Codice A.1: traduzione della progettazione del database nel linguaggio SQL