

# PROGRAMMAZIONE AVANZATA IN C#

Appunti di informatica per la classe 4° - articolazione informatica

Prof. Gennaro Malafronte

---

A.S. 2022/2023



# Premessa e Copyright

Questi appunti costituiscono una guida per gli studenti del corso di Informatica degli studenti dell'Istituto Greppi. Talvolta alcuni esempi sono presi da Internet o da altre fonti. In tal caso è sempre riportato il link alla risorsa dalla quale l'esempio è stato preso.



Quest'opera è distribuita con Licenza [Creative Commons Attribuzione 4.0 Internazionale](#).

1.	Delegates in C# .....	18
1.1.	Primo esempio.....	18
1.2.	Invoking Delegate .....	19
1.3.	Multicast Delegates In C#.....	20
1.3.1.	Covariance and Contravariance In C# .....	21
1.3.2.	Covariance With Delegates .....	22
1.3.3.	Contravariance With Delegates.....	22
1.4.	Points to Remember.....	22
2.	Lambda Expressions In C# .....	23
2.1.	Lambda examples .....	23
2.1.1.	Function Delegate and its use with lambda .....	28
2.1.2.	Action Delegate and its use with lambda .....	29
2.2.	What's a Closure?.....	31
2.3.	Points to Remember.....	32
3.	Events in C# .....	32
3.1.	What Are Events In C#?.....	32
3.1.1.	Primo esempio.....	33
3.1.2.	Un esempio articolato .....	35
3.2.	Points to Remember.....	36
4.	LINQ .....	37
4.1.	What is LINQ?.....	37
4.2.	Why LINQ?.....	38
4.2.1.	Example: Use for loop to find elements from the collection in C# 1.0 .....	38
4.2.2.	Example: Use Delegates to Find Elements from the Collection.....	39
4.2.3.	Advantages of LINQ .....	42
4.2.4.	LINQ API.....	42
4.2.5.	System.Linq.Enumerable .....	43
4.2.6.	System.Linq.Queryable.....	44
4.2.7.	Come usare LINQ? .....	46
4.2.7.1.	LINQ Query (Query Expression Syntax) .....	46
4.2.7.1.1.	Points to Remember .....	48
4.2.7.2.	LINQ Methods (Fluent) .....	48
4.2.7.3.	Standard Query Operators .....	50
4.2.7.3.1.	Standard Query Operators in Query Syntax.....	50
4.2.7.3.2.	Standard Query Operators in Method Syntax .....	50
4.2.7.4.	Alcuni esempi di codice LINQ .....	51
4.2.7.5.	Esercitazione – LINQ al Museo .....	58
4.2.7.5.1.	Svolgimento.....	59

4.2.7.6. Points to Remember.....	66
5. SQLite .....	66
5.1. SQLite and C#.....	66
6. EF Core model .....	66
6.1. EF Core Model Tutorial - Console App (utilizzo comandi .NET SDK CLI) .....	68
6.2.....	77
6.3. EF Core Model Tutorial 2 - Console App (utilizzo comandi .NET SDK CLI) .....	77
6.4. EF Core: DbContext.....	83
6.4.1. Design Patterns – Cenni.....	83
6.4.1.1. Repository pattern.....	84
6.4.1.2. Unity of Work Pattern.....	84
6.4.2. DbContext Lifetime, Configuration, and Initialization.....	84
6.5. Creazione di modelli di dati con EF Core.....	85
6.5.1. Creating and configuring a model.....	88
6.5.2. Entities .....	89
6.5.2.1. Entities Properties.....	90
6.5.2.1.1. Required and optional properties .....	90
6.5.3. Chiavi – Keys.....	91
6.5.3.1. Chiave primaria .....	91
6.5.3.2. Chiavi alternative.....	92
6.5.4. Indici .....	94
6.5.5. Associazioni – Relationships .....	95
6.5.5.1. Definition of Terms.....	95
6.5.5.2. Associazione uno a molti - 1:N .....	96
6.5.5.2.1. Conventions.....	96
6.5.5.2.1.1. Fully Defined Relationships .....	96
6.5.5.2.1.2. No Foreign Key Property – Shadow Foreign Key .....	97
6.5.5.2.1.3. Single Navigation Property .....	98
6.5.5.2.1.4. Limitations .....	98
6.5.5.2.1.5. Cascade Delete .....	98
6.5.5.2.2. Manual Configuration .....	98
6.5.5.2.3. Fluent Api – manual configuration.....	98
6.5.5.2.3.1. Single navigation property – manual configuration.....	99
6.5.5.2.3.2. Foreign Key – manual configuration .....	100
6.5.5.2.3.3. Shadow Foreign Key – manual configuration.....	101
6.5.5.2.3.4. Without navigation property – manual configuration.....	102
6.5.5.2.3.5. Principal Key of a Relationship – manual configuration.....	103
6.5.5.2.3.6. Required and optional relationships – Manual configuration .....	105

6.5.5.2.3.7. Cascade delete – manual configuration.....	105
6.5.5.2.4. Data Annotations – manual configuration.....	106
6.5.5.2.4.1. Data Annotations [ForeignKey] – manual configuration.....	106
6.5.5.2.4.2. Data Annotations [InverseProperty] – manual configuration.....	107
6.5.5.2.4.3. Data Annotations [Primary Key] – manual configuration .....	107
6.5.6. Caso di associazione molti a molti – N:M.....	108
6.5.7. Caso di associazione 1 a 1 – 1:1 .....	112
6.5.8. Generazione del database - Migrations .....	114
6.6. Tipi di dato .....	117
6.6.1. Data Annotations.....	117
6.6.2. Fluent API .....	118
6.7. Conversioni di valore .....	118
6.7.1. Built-in converters.....	119
6.7.2. Pre-defined conversions .....	121
6.8. Data seeding .....	121
6.8.1. Model seed data .....	121
6.9. Tipi di dato usati da SQLite .....	122
6.9.1. Storage Classes and Datatypes.....	123
6.10. Esercitazione con il database dei Romanzi .....	123
6.10.1. Svolgimento .....	125
6.10.1.1. Model.....	125
6.10.1.2. Data .....	126
6.10.1.3. Migration e creazione del database .....	127
6.10.1.4. Applicazione.....	127
6.11. Esercitazione con il database dell'Università .....	132
6.11.1. Svolgimento .....	132
6.11.1.1. Model.....	135
6.11.1.2. Data.....	137
6.11.1.3. Migration e creazione del database .....	139
6.11.1.4. Applicazione.....	140
6.12. EF Core 3.x e successive versioni.....	147
6.12.1. Corso su EF Core 3 .....	147
6.12.2. EF Core con altri database (MySQL, MariaDB, etc.).....	147
7. Multithreading.....	149
7.1. Thread e threading .....	149
7.1.1. What Are Threads in C#?.....	149
7.1.2. Processes vs Threads.....	150
7.1.3. Come usare il multithreading in .NET .....	152

7.1.4.	When To Use Thread In C#? .....	153
7.1.5.	C# Thread.....	153
7.1.5.1.	Difference Between Foreground And Background Thread In C# .....	154
7.1.5.2.	C# Start Thread With Parameters .....	155
7.1.5.3.	Thread Life Cycle In C# .....	157
7.1.6.	Sospensione e interruzione di thread .....	160
7.1.6.1.	Metodi Thread.Sleep, Thread.Interrupt .....	160
7.1.7.	Threadpool .....	163
7.1.7.1.	C# Thread Pool Queue.....	163
7.1.7.2.	Limitations To Thread Pool Queue.....	164
7.1.8.	Race Condition.....	164
7.1.8.1.	Shared Resources.....	164
7.1.8.2.	What Is Race Condition? .....	165
7.1.9.	Sezione critica .....	166
7.1.10.	Stati transienti ed interferenze.....	166
7.1.10.1.	Thread Safeness.....	168
7.1.10.2.	Dominio e Rango.....	169
7.1.10.3.	Condizioni di Bernstein.....	169
7.1.11.	C# Thread synchronization techniques .....	171
7.1.11.1.	C# Interlocked .....	171
7.1.11.2.	Data Partitioning.....	172
7.1.11.3.	Wait Based Synchronization.....	173
7.1.11.3.1.	Panoramica delle primitive di sincronizzazione in .NET .....	174
7.1.11.3.2.	Monitor Class Usage In C# (approfondimento) .....	175
7.1.11.3.2.1.	Exception Aware Monitor Usage (approfondimento).....	177
7.1.11.3.2.2.	C# Lock Keyword (il costrutto da usare per lock tra thread dello stesso processo)	
	178	
7.1.12.	Semafori .....	179
7.1.12.1.	Produttore consumatore.....	183
7.1.12.2.	Algoritmo del barbiere .....	185
7.1.13.	Deadlock .....	189
7.1.13.1.	Un esempio di deadlock: il barbiere sbagliato! .....	192
7.2.	Costrutti per la programmazione concorrente – uso di thread .....	194
7.2.1.	Costrutto fork/join.....	194
7.2.1.1.	Costrutto fork/join: esempio 1 .....	194
7.2.2.	Costrutto join(count) .....	196
7.2.2.1.	Costrutto join(count): esempio 1 .....	196
7.2.3.	Costrutto cobegin/coend .....	197

7.2.3.1. Costrutto cobegin/coend: esempio 1 .....	197
7.3. Task Parallel Library (TPL) .....	199
7.3.1. C# Task.....	199
7.3.1.1. C# Task: Programmazione asincrona e/o parallela basata su attività .....	200
7.3.1.1.1. Execution Model Of A Task .....	204
7.3.1.1.2. C# Task That Returns A Result .....	204
7.3.1.1.3. Adding continuation .....	205
7.3.1.1.4. Scheduling Different Continuation Tasks.....	207
7.3.2. Costrutti per la programmazione concorrente – uso di Task.....	208
7.3.2.1. Costrutto fork/join .....	208
7.3.2.2. Costrutto join(count).....	210
7.3.2.3. Costrutto cobegin/coend .....	211
7.3.3. C# Task: Parallelismo dei dati (Task Parallel Library) .....	212
7.3.3.1. Esempio 1 di Parallel.For: .....	212
7.3.3.2. Esempio 2 di Parallel.For: .....	213
7.3.3.3. Esempio 3 di Parallel.For .....	215
7.3.3.4. Esempio 4 di Parallel.For .....	222
7.3.3.5. Esempio 1 Parallel.ForEach.....	226
7.3.3.6. Esempio 5 di Parallel.For (con Thread-local data).....	227
7.3.3.7. Esempio 2 di Parallel.ForEach (con Thread-local data).....	228
7.3.4. Attached and Detached Child Tasks .....	230
7.3.4.1. Detached child tasks .....	231
7.3.4.2. Attached child tasks.....	232
7.3.5. Cancellazione di un’attività .....	234
7.3.5.1. Un esempio di cancellazione di attività .....	234
7.3.5.2. Cancellare un task e i suoi figli.....	235
7.3.5.3. Un esempio di cancellazione di attività con il Parallel.ForEach .....	239
7.3.6. Exception handling (Task Parallel Library) .....	240
7.3.6.1. Attached child tasks and nested AggregateExceptions.....	241
7.3.6.2. Exceptions from detached child tasks.....	242
7.3.7. Dataflow (Task Parallel Library – solo per approfondimento) .....	243
7.3.7.1. How to: Write Messages to and Read Messages from a Dataflow Block .....	243
7.3.7.2. How to: Implement a Producer-Consumer Dataflow Pattern .....	243
7.3.7.3. How to: Perform Action When a Dataflow Block Receives Data.....	244
7.3.7.4. Walkthrough: Creating a Dataflow Pipeline .....	244
7.3.8. Programmazione asincrona con async e await.....	244
7.3.8.1. Versione parallela, veloce ma non sempre ottimizzata .....	247
7.3.8.2. Non bloccare, ma attendere .....	248

7.3.8.3.	Iniziare più attività contemporaneamente.....	251
7.3.8.4.	Composizione di attività .....	252
7.3.8.5.	Attendere le attività in modo efficiente .....	253
7.3.9.	Flusso di controllo in programmi asincroni (C#).....	254
7.3.10.	Modelli di programmazione asincrona .....	257
7.3.10.1.	Task-based asynchronous pattern (TAP) .....	257
7.3.10.2.	Event-based Asynchronous Pattern (EAP) – (legacy model).....	257
7.3.11.	I/O di file asincrono .....	258
7.3.11.1.	Copiare file in modalità asincrona.....	258
7.3.11.2.	Processare a blocchi un file di testo .....	260
7.3.12.	Async/Await - Best Practices in Asynchronous Programming .....	261
8.	Introduzione alla programmazione di rete in .NET .....	262
8.1.	Introduzione .....	262
8.1.1.	Applicazioni Internet.....	262
8.1.2.	Identificazione delle risorse .....	262
8.1.2.1.	Authority specification .....	263
8.1.2.2.	Query specification .....	263
8.1.2.3.	The URL as a sub-type of the URI .....	263
8.1.2.4.	The System.Net.UriBuilder class .....	263
8.1.2.5.	Hosts – domain names and IPs .....	264
8.1.2.6.	The Network Reference Models .....	265
8.1.3.	List of Well-Known Ports .....	266
8.1.4.	HTTP – application to application communication .....	267
8.1.4.1.	Request/response .....	268
8.1.4.2.	HTTP sessions .....	268
8.1.4.3.	Request methods .....	268
8.1.4.4.	Status codes .....	269
8.1.4.5.	The HTTP message format .....	269
8.1.4.6.	HTTP in C#.....	270
8.1.5.	FTP and SFTP .....	270
8.1.6.	SMTP .....	270
8.1.7.	TCP .....	270
8.1.8.	UDP .....	270
8.2.	Richieste e risposte HTTP in .NET .....	271
8.2.1.	Generalità .....	271
8.2.1.1.	Gestione del DNS .....	271
8.2.1.2.	Costruzione di un oggetto URI .....	272
8.2.1.3.	Caricamento e download dei dati da un server Internet .....	273

8.2.2. HttpClient.....	273
8.2.2.1. Scaricare del testo dalla rete .....	273
8.2.2.2. Uso di HttpClient con proxy.....	274
8.2.2.3. Come eseguire più richieste web in parallelo tramite async e await (C#).....	276
8.2.2.4. Analizzare la risposta di una Get http mediante le properties della response ....	282
8.2.2.5. Esempi di utilizzo di HttpClient .....	285
8.2.2.6. Scaricamento e salvataggio di una pagina web .....	285
8.2.2.7. Lettura e scrittura di file scaricati da Internet con metodi asincroni .....	287
8.2.2.7.1. Riportare lo stato di avanzamento del download di un file .....	290
8.2.2.7.2. Gestione di file di grandi dimensioni .....	291
8.2.2.7.3. Un esempio di scaricamento di file di grandi dimensioni .....	292
8.2.3. Extension Methods e librerie .....	295
8.2.3.1. How to write custom Extension Methods? .....	295
8.2.3.1.1. To define and call the extension method .....	295
8.2.3.1.1.1. Esempio di Extension Method.....	296
8.2.3.1.1.2. Creazione di una libreria con metodi di estensione per la gestione del grado di parallelismo dei Task che elaborano una collection di dati.....	296
8.2.3.1.1.2.1. Step N1: creazione di un progetto di tipo libreria in Visual Studio .....	297
8.2.3.1.1.2.2. Step N2: utilizzo della libreria in un altro progetto di Visual Studio .....	298
8.2.3.1.1.3. Libreria per la gestione del Proxy con l'oggetto HttpClient .....	303
8.2.3.1.1.4. Utilizzo di un Assembly compilato di una libreria .....	308
8.2.3.1.1.5. Creazione di una libreria e pubblicazione su NuGet .....	309
8.2.4. JSON (JavaScript Object Notation) .....	310
8.2.4.1. Concetti di base.....	310
8.2.4.2. Working with Complex Types in JSON.....	311
8.2.4.2.1. Nested Objects.....	311
8.2.4.2.2. Nested Arrays .....	312
8.2.4.3. Comparison to XML.....	313
8.2.4.4. Utility and resources .....	314
8.2.4.5. Serializzazione e deserializzazione di oggetti JSON in .NET con System.Text.Json (la libreria di riferimento per i nuovi progetti).....	314
8.2.4.5.1. Deserialization behavior .....	318
8.2.4.5.2. Serialize to formatted JSON .....	319
8.2.4.5.3. HttpClient and HttpContent extension methods.....	319
8.2.4.5.4. Metodi di estensione per inviare e ricevere contenuto HTTP sotto forma di oggetti JSON .....	321
8.2.4.6. Serializzazione e deserializzazione con la libreria Newtonsoft.Json.....	321
8.2.4.7. JSON New Features in .NET 6 .....	322

8.2.4.8. Esercizi sulla serializzazione e deserializzazione di oggetti JSON .....	322
8.2.4.9. Esercizi su JSON .....	322
9. REST Api .....	327
9.1. REST 101 .....	327
9.1.1. Principi .....	327
9.1.2. Vincoli .....	327
9.1.3. Il principio fondamentale di REST: le risorse .....	328
9.1.4. REST Resource Naming Guide .....	329
9.1.4.1. What is a Resource? .....	329
9.1.4.2. Singleton and Collection Resources .....	329
9.1.4.3. Collection and Sub-collection Resources .....	329
9.1.4.4. URI .....	329
9.1.5. Best Practices .....	330
9.1.5.1. Use nouns to represent resources .....	330
9.1.5.2. document .....	330
9.1.5.3. collection .....	331
9.1.5.4. store .....	331
9.1.5.5. controller .....	331
9.1.5.6. Consistency is the key .....	332
9.1.5.6.1. Use forward slash (/) to indicate hierarchical relationships .....	332
9.1.5.6.2. Do not use trailing forward slash (/) in URIs .....	332
9.1.5.6.3. Use hyphens (-) to improve the readability of URIs .....	332
9.1.5.6.4. Do not use underscores ( _) .....	332
9.1.5.6.5. Use lowercase letters in URIs .....	333
9.1.5.6.6. Do not use file extensions .....	333
9.1.5.6.7. Never use CRUD function names in URIs .....	333
9.1.5.6.8. Use query component to filter URI collection .....	334
9.1.6. HTTP Methods .....	334
9.1.7. HATEOAS Driven REST APIs .....	337
9.1.8. Richardson Maturity Model .....	338
9.2. Consumare REST Api mediante l'oggetto HttpClient .....	339
9.2.1. Un esempio di programma che utilizza le API di Github .....	339
9.2.2. Principali classi di riferimento per la gestione delle richieste HTTP .....	341
9.2.3. Postman .....	342
9.2.4. Testing APIs with Postman .....	342
9.2.5. Impariamo ad usare Postman con la collection di Postman Echo .....	343
9.2.6. Usiamo la collection Postman API .....	344
9.2.7. Autorizzazione delle API in Postman .....	346

9.2.8.	Curl (solo per approfondimento).....	346
9.2.8.1.	Curl basics .....	346
9.2.8.2.	Http with curl.....	346
9.2.8.3.	Culr with TLS .....	346
9.2.9.	Mock Server di REST API.....	347
9.2.9.1.	Mockaroo .....	347
9.2.9.1.1.	Utilizzo di Mockaroo per generare DataSet a partire da uno schema .....	347
9.2.9.1.2.	Utilizzo di Mockaroo per la creazione di un Mock Server di REST API .....	350
9.2.9.1.3.	Scrittura di un client console per le Mock API di Mockaroo .....	354
9.2.10.	Postman Mock Server .....	361
9.2.10.1.	Utilizzo di un Postman Mock Server per il testing un'applicazione client .....	366
9.2.10.2.	Postman Mock Server privato .....	367
9.2.10.3.	Scrittura di un client console per le Mock API di Postman .....	368
9.3.	Esempi di servizi REST .....	376
9.3.1.	Weather API .....	376
9.3.1.1.	Un esempio di client console per il servizio OpenWeatherMap.....	376
9.3.1.1.1.	URL Encoding.....	376
9.3.1.1.2.	Recupero dei dati relativi al tempo atmosferico corrente.....	377
9.3.1.1.2.1.	OpenWeatherMap Geocoding service .....	377
9.3.1.1.2.2.	Recupero dei dati relativi alle condizioni meteo correnti .....	379
9.3.1.1.3.	Previsioni meteo a 7 giorni e previsioni dettagliate a 48 ore .....	383
9.3.2.	Bing Maps API.....	389
9.3.2.1.	Un esempio di client console per Bing Maps API .....	389
9.3.2.2.	Altri esempi d'uso delle REST Bing Maps API .....	395
9.3.2.2.1.	Find a Location by Point (inverse geocoding) .....	395
9.3.2.2.2.	Location Recognition .....	396
9.3.2.2.3.	Bing Maps Imagery API.....	396
9.3.2.2.4.	Bing Maps Routes API.....	396
9.3.2.2.4.1.	Calculate a Route.....	397
9.3.3.	Here Rest API.....	397
9.3.4.	Mediawiki API .....	397
9.3.4.1.	MediaWiki Action API .....	397
9.3.4.1.1.	Esempio: Ricerca fulltext in Wikipedia.....	397
9.3.4.1.2.	Esempio: Estrazione del summary di una pagina.....	399
9.3.4.1.3.	Analisi del JSON DOM con la classe JsonDocument.....	400
9.3.4.2.	MediaWiki REST API .....	402
9.3.4.2.1.	Esempio: trovare le pagine che trattano un determinato argomento .....	402
9.3.4.2.2.	Esempio: trovare il summary di una pagina contenente le parole da cercare.....	405

9.3.4.2.3. Esempio: trovare le sezioni di una pagina di Wikipedia .....	407
9.3.4.2.4. Da Wikitext a testo leggibile .....	411
10. Microsoft Azure .....	420
10.1. Sottoscrizione Studenti per Azure.....	420
10.2. Azure Cognitive Services.....	420
10.2.1. Speech to text.....	420
10.2.1.1. What is speech-to-text? .....	420
10.2.1.2. Get started with speech-to-text.....	420
10.2.2. Text to speech.....	424
10.2.2.1. Get started with text to speech .....	424
10.2.3. Intent Recognition.....	427
10.2.3.1. Pattern matching.....	427
10.2.3.2. LUIS .....	430
10.2.3.2.1. What is LUIS.....	430
10.2.3.2.2. How to create and manage LUIS resources .....	430
10.2.3.2.3. Get Started Creating App .....	430
10.2.3.2.4. Get started with intent recognition .....	434
10.3. Il progetto Alice Neural.....	443
10.3.1. Punto di partenza per Alice Neural .....	450
11. HttpListener .....	459
11.1.1. Links utili per HttpListener.....	460
11.2. Esempi di un Server che accetta una sola richiesta e stampa una pagina HTML.....	460
11.3. Classe HttpListenerContext .....	463
11.4. Classe HttpListenerRequest .....	463
11.5. Classe HttpListenerResponse.....	463
11.6. Certificati SSL per HttpListener .....	463
12. Evoluzione del linguaggio C#.....	463
12.1. C# 8 .....	463
12.1.1. Using declarations.....	463
12.1.2. Nullable reference types.....	463
12.1.3. Null-coalescing assignment .....	464
12.2. C# 9 .....	464
12.2.1. Record types.....	464
12.2.2. Top level statements.....	464
12.2.2.1. Only one top-level file.....	464
12.2.2.2. No other entry points .....	464
12.2.2.3. using directives.....	465
12.2.2.4. Global namespace.....	465

12.2.2.5. Namespaces and type definitions .....	465
12.2.2.6. args .....	466
12.2.2.7. await .....	466
12.2.2.8. Exit code for the process .....	466
12.2.2.9. Implicit entry point method.....	466
12.2.2.10. New General Structure of a C# Program.....	467
12.2.3. Struttura del Main (solo per approfondimento) .....	467
12.2.3.1. Main() and command-line arguments .....	467
12.2.3.1.1. Caratteristiche del Main .....	468
12.2.3.1.2. Valori di ritorno del Main.....	468
12.2.3.1.3. Command-Line Arguments .....	468
12.3. C# 10 .....	468
12.3.1. Record Structs .....	468
12.3.2. Global and implicit using.....	468
12.3.3. File-scoped namespaces .....	469
13. Sviluppo di RESTful API con Minimal API .NET .....	471
13.1. Getting started with ASP.NET Core Minimal API.....	471
13.1.1. What is minimal API? .....	471
13.1.2. Minimal API .NET a partire da un'App console .....	471
13.1.3. Minimal API .NET a partire da un template ASP.NET Core Web API.....	473
13.1.3.1. Esecuzione dell'App.....	474
13.1.3.2. Concetto di Certificato Digitale e di HTTPS .....	478
13.1.4. Chi esegue l'applicazione server?.....	479
13.1.4.1. Kestrel .....	480
13.1.4.2. IIS Express .....	480
13.1.5. La configurazione di lancio dell'App .....	481
13.1.6. Come funziona una minimal API? .....	481
13.1.6.1.1. Add documentation with Swagger .....	482
13.1.7. How to add routes and use other advanced commands .....	483
13.1.7.1. HTTP verbs in minimal API.....	483
13.1.8. Un semplice esempio di Minimal API: progetto PizzaStore.....	485
13.1.9. Esempio guidato di Minimal API: TodoAPI .....	489
13.1.9.1. Prima versione .....	490
13.1.9.2. Seconda versione: analizziamo Swagger .....	492
13.1.9.3. Terza versione: utilizziamo EntityFramework .....	493
13.1.9.3.1. Testing dell'applicazione con Swagger .....	496
13.1.9.3.2. Testing dell'applicazione con Postman .....	498
13.1.9.3.3. Return Values .....	499

13.1.9.4. Quarta versione: uso di Data Transfer Object (DTO) .....	499
13.1.10. Esempio guidato di Minimal API: PizzaStoreSQLite.....	501
13.2. DBMS relazionali client/server.....	508
13.2.1. Il database MySQL/MariaDb.....	508
13.2.1.1. Installazione del software MariaDb .....	508
13.2.1.2. Avvio di MariaDb.....	510
13.2.1.3. Stop di MariaDB .....	510
13.2.1.4. MySQL Monitor (interfaccia console).....	511
13.2.1.5. HeidiSQL (interfaccia grafica).....	512
13.2.1.6. MySQL Workbench (strumento di sviluppo avanzato).....	514
13.2.1.7. Cambio della password di root.....	515
13.2.1.8. Dove sono i dati?.....	515
13.2.1.9. MariaDb/MySQL con EF Core .....	515
13.2.2. Un esempio di Minimal API con MariaDB: Azienda API .....	515
13.2.2.1. Sviluppo del progetto .....	517
13.2.2.1.1. Creazione del Model .....	518
13.2.2.1.2. Collegamento dell'applicazione con il DBMS.....	520
13.2.2.1.3. Migration .....	522
13.2.2.1.4. View Model per i dati (Data Transfer Object – DTO) .....	522
13.2.2.1.5. Migrazione con Code First Approach (solo per sviluppo/testing) .....	524
13.2.2.1.6. Configurazione di Swagger con il View Model .....	524
13.2.2.1.7. Gestione di tanti endpoints .....	525
13.2.2.1.8. Il file Program.cs .....	525
13.2.2.1.9. Gestione degli Endpoints (Handlers) .....	526
13.2.2.1.10. Testing dell'app .....	534
13.2.3. Gestione degli account per MariaDb .....	536
13.2.3.1. Gestione degli account (diversi da root) per MariaDb .....	537
13.2.3.1.1. Gli account attivi sul DBMS .....	538
13.2.3.1.2. Creazione di un account .....	540
13.2.3.1.3. Rimozione di un account.....	540
13.2.3.2. Gestione dei permessi di un account .....	540
13.2.3.2.1. Flush .....	542
13.2.3.3. Creazione di un account specifico per una Web Application .....	542
13.2.3.3.1. Creazione del database mediante uno script SQL (in produzione) .....	543
13.2.3.3.2. Creazione del database mediante codice (per sviluppo e testing) .....	547
13.2.3.3.2.1. Esempio: Riscrittura del codice dell'esempio Azienda API con account specifico	
548	
13.3. Binding dei dati con le Minimal API .....	549

13.3.1. Custom Binding .....	553
13.3.1.1. TryParse.....	554
13.3.1.2. BindAsync.....	555
13.3.1.3. Binding failures .....	556
13.3.1.4. Binding Precedence.....	556
13.3.2. Customize JSON binding.....	557
13.4. Configuration in ASP.NET Core .....	558
13.5. Route Handlers.....	560
13.5.1. Lambda expression .....	560
13.5.2. Local Function .....	560
13.5.3. Instance method .....	560
13.5.4. Static method.....	560
13.5.5. Name routes and link generation .....	561
13.6. Responses.....	562
13.6.1. string return values.....	562
13.6.2. JSON return values .....	562
13.6.3. IResult return values .....	562
13.7. ASP.NET Core Service Lifetimes .....	563
13.7.1. Singleton .....	563
13.7.2. Scoped.....	563
13.7.3. Transient.....	563
13.8. Data Validation.....	565
13.8.1. FluentValidator.....	565
13.8.1.1. Installazione.....	566
13.8.1.2. Getting started with validation .....	566
13.8.1.3. Throwing Exceptions .....	567
13.8.1.4. Complex Properties .....	568
13.8.1.5. FluentValidator with ASP.NET Core.....	569
13.8.1.5.1. Un esempio di applicazione di FluentValidator .....	569
13.9. Messa in sicurezza delle REST API.....	575
13.9.1. Meccanismi di autenticazione e autorizzazione.....	575
13.9.1.1. Basic Authentication.....	576
13.9.1.1.1. Benefits.....	577
13.9.1.1.2. Drawbacks .....	577
13.9.1.2. API Key authentication/authorization .....	577
13.9.1.2.1. Benefits.....	578
13.9.1.2.2. Drawbacks .....	578
13.9.1.3. Cookie based authentication/authorization .....	579

13.9.1.4. Implementazione dei meccanismi di sicurezza in .NET Core.....	581
13.9.1.4.1. Esempio di Basic authentication/authorization .....	581
13.9.1.4.2. Esempio di ApiKey authentication/authorization.....	586
13.9.1.4.3. Esempio di Session based authentication/authorization.....	596
13.9.1.4.4. Esempio di cookie based authentication/authorization (senza sessioni) ....	602
13.9.1.4.5. PizzaStoreAPI - Un esempio completo di Web API con user management e key management .....	609
13.9.1.4.5.1. Setup del progetto a partire dall'esempio su GitHub .....	614
13.9.1.4.5.2. Struttura del progetto .....	623
13.9.1.4.5.2.1. Il data model.....	623
13.9.1.4.5.3. Deployment del progetto su Azure .....	635
13.9.1.4.5.3.1. Deployment su Azure mediante il wizard integrato di Visual Studio.....	636
13.9.1.4.5.3.2. Parte Opzionale – uso di SQL Server Express .....	644
14. Xamarin.....	649
14.1. Xamarin Forms .....	649
14.2. Xamarin Getting Started .....	649
14.2.1. Che cosa è Xamarin?.....	649
14.2.1.1. Funzionamento di Xamarin .....	649
14.2.1.2. Funzionalità aggiuntive .....	650
14.2.1.3. Xamarin.Android.....	650
14.2.1.4. Xamarin.iOS .....	650
14.2.1.5. Xamarin.Essentials .....	650
14.2.1.6. Xamarin.Forms.....	651
14.2.2. Installazione di Xamarin .....	651
14.2.3. Piattaforme supportate da Xamarin.Forms .....	651
14.2.4. Installazione dell'emulatore Android.....	652
14.2.4.1. Accelerazione hardware per le prestazioni dell'emulatore (Hyper-V e HAXM).....	652
14.2.5. Set Up Device for Development (Configurare il dispositivo per lo sviluppo) .....	652
14.2.6. Prima App Xamarin.Forms .....	652
14.2.7. Understanding Android API Levels (Informazioni sui livelli API Android) .....	652
14.2.8. Xamarin QuickStart .....	655
14.2.9. Xamarin Tutorials .....	655
14.2.10. Xamarin Forms XAML.....	656
14.2.11. 657	
14.3. App fundamentals .....	657
14.3.1.1. Navigazione.....	657
14.3.2. Xamarin.Essentials.....	658
14.3.3. Dati e servizi cloud .....	658

14.3.3.1. Servizi Azure .....	658
14.3.4. Esempi di applicazioni .....	659
15. .NET ecosystem .....	661
15.1. Learn .NET.....	661
15.1.1. .NET videos .....	661
15.2. .NET Standard.....	661
15.3. .NET Core .....	661
15.3.1. .NET Core command-line interface (CLI) tools .....	661

## 1. Delegates in C#

<https://kudchikarsk.com/delegates-and-events-in-csharp/>

<https://www.tutorialsteacher.com/csharp/csharp-delegates>

A function can have one or more parameters of different data types, but what if you want to pass a function itself as a parameter? How does C# handle the callback functions or event handler? The answer is - **delegate**.

A delegate is like a pointer to a function. It is a reference type data type and it holds the reference of a method. All the delegates are implicitly derived from System.Delegate class.

A delegate can be declared using **delegate** keyword followed by a function signature as shown below.

```
<access modifier> delegate <return type> <delegate_name>(<parameters>)
```

In C#, delegates form the basic building blocks for events. A delegate is a type that defines a method signature. In C# you can instantiate a delegate and let it point to another method. You can invoke the method through the delegate.

### 1.1. Primo esempio

```
class Program
{
    // declare delegate
    public delegate void Print(int value);

    static void Main(string[] args)
    {
        // Print delegate points to PrintNumber
        Print printDel = PrintNumber;

        // or
        // Print printDel = new Print(PrintNumber);

        printDel(100000);
        printDel(200);

        // Print delegate points to PrintMoney
        printDel = PrintMoney;

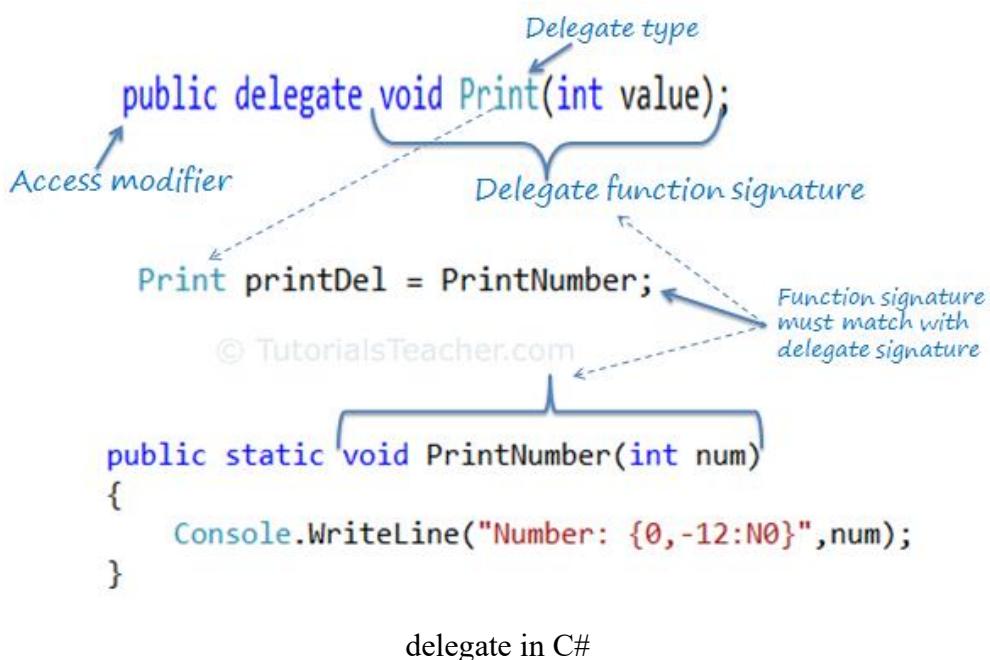
        printDel(10000);
        printDel(200);
        Console.ReadLine();
    }

    public static void PrintNumber(int num)
    {
        Console.WriteLine("Number: {0,-12:N0}", num);
    }

    public static void PrintMoney(int money)
    {
        Console.WriteLine("Money: {0:C}", money);
    }
}
```

In the above example, we have declared Print delegate that accepts *int* type parameter and returns void. In the Main() method, a variable of Print type is declared and assigned a PrintNumber method name. Now, invoking Print delegate will in-turn invoke PrintNumber method. In the same way, if the Print delegate variable is assigned to the PrintMoney method, then it will invoke the PrintMoney method.

The following image illustrates the delegate.



Optionaly, a delegate object can be created using the new operator and specify a method name, as shown below:

```
Print printDel = new Print(PrintNumber);
```

## 1.2. Invoking Delegate

The delegate can be invoked like a method because it is a reference to a method. Invoking a delegate will in-turn invoke a method which id referred to. The delegate can be invoked by two ways: using () operator or using the Invoke() method of delegate as shown below.

```
Print printDel = PrintNumber;
printDel.Invoke(10000);
```

//or  
printDel(10000);

**Secondo esempio:**

```
class Program
{
    public delegate double MathDelegate(double value1, double value2);

    public static double Add(double value1, double value2)
    {
        return value1 + value2;
    }
    public static double Subtract(double value1, double value2)
```

```

    {
        return value1 - value2;
    }

    public static void Main()
    {
        MathDelegate mathDelegate = Add;
        var result = mathDelegate(5, 2);
        Console.WriteLine(result);
        // output: 7

        mathDelegate = Subtract;
        result = mathDelegate(5, 2);
        Console.WriteLine(result);
        // output: 3

        Console.ReadLine();
    }
}

```

As you can see, we use the delegate keyword to tell the compiler that we are creating a delegate type. Instantiating delegates is easy with the automatic creation of a new delegate type.

You can also use the new keyword method of instantiating a delegate

```
MathDelegate mathDelegate = new MathDelegate(Add);
```

An instantiated delegate is an object; you can pass it around and give it as an argument to other methods.

### 1.3. Multicast Delegates In C#

Another great feature of delegates is that you can combine them together. This is called multicasting. You can use the + or += operator to add another method to the invocation list of an existing delegate instance. Similarly, you can also remove a method from an invocation list by using the decrement assignment operator (- or -=). This feature forms the base for events in C#. Below is a multicast delegate example.

```

class Program
{
    static void Hello(string s)
    {
        Console.WriteLine("  Hello, {0}!", s);
    }

    static void Goodbye(string s)
    {
        Console.WriteLine("  Goodbye, {0}!", s);
    }

    delegate void Del(string s);

    static void Main()
    {
        Del a, b, c, d, k;

        // Create the delegate object a that references
        // the method Hello:
        a = new Del(Hello);

        // Create the delegate object b that references
        // the method Goodbye:
        b = Goodbye;
    }
}

```

```

// The two delegates, a and b, are composed to form c:
c = a + b;
// allo stesso modo possiamo inizializzare una variabile delegate Del a null
// e usare += per aggiungere un metodo
k = null;
//aggiungo un metodo
k += a;
//aggiungo un altro metodo
k += b;
// Remove a from the composed delegate, leaving d,
// which calls only the method Goodbye:
d = c - a;

Console.WriteLine("Invoking delegate a:");
a("A");
Console.WriteLine("Invoking delegate b:");
b("B");
Console.WriteLine("Invoking delegate c:");
c("C");
Console.WriteLine("Invoking delegate d:");
d("D");
Console.WriteLine("Invoking delegate k:");
k("K");

/* Output:
Invoking delegate a:
Hello, A!
Invoking delegate b:
Goodbye, B!
Invoking delegate c:
Hello, C!
Goodbye, C!
Invoking delegate d:
Goodbye, D!
Invoking delegate k:
Hello, K!
Goodbye, K!
*/
Console.ReadLine();
}
}

```

All this is possible because delegates inherit from the System.MulticastDelegate class that in turn inherits from System.Delegate. Because of this, you can use the members that are defined in those base classes on your delegates.

For example, to find out how many methods a multicast delegate is going to call, you can use the following code:

```
int invocationCount = d.GetInvocationList().GetLength(0);
```

### 1.3.1. Covariance and Contravariance In C#

When you assign a method to a delegate, the method signature does not have to match the delegate exactly. This is called covariance and contravariance. Covariance makes it possible that a method has a return type that is more derived than that defined in the delegate. Contravariance permits a method that has parameter types that are less derived than those in the delegate type.

### 1.3.2. Covariance With Delegates

Here is an example of covariance,

```
class Program
{
    public delegate TextWriter CovarianceDel();

    public static StreamWriter MethodStream() { return null; }
    public static StringWriter MethodString() { return null; }

    static void Main()
    {
        CovarianceDel del;

        del = MethodStream;
        del = MethodString;

        Console.ReadLine();
    }
}
```

Because both StreamWriter and StringWriter inherit from TextWriter, you can use the CovarianceDel with both methods.

### 1.3.3. Contravariance With Delegates

Below is an example of contravariance.

```
class Program
{
    public static void DoSomething(TextWriter textWriter) { }
    public delegate void ContravarianceDel(StreamWriter streamWriter);

    static void Main()
    {
        ContravarianceDel del = DoSomething;

        Console.ReadLine();
    }
}
```

Because the method DoSomething can work with a TextWriter, it surely can also work with a StreamWriter. Because of contravariance, you can call the delegate and pass an instance of StreamWriter to the DoSomething method

You can learn more about this concept [here](#).

## 1.4. Points to Remember

1. Delegate is a function pointer. It is reference type data type.
2. Syntax: `public delegate void <function name>(<parameters>)`
3. A method that is going to assign to delegate must have same signature as delegate.
4. Delegates can be invoke like a normal function or `Invoke()` method.
5. Multiple methods can be assigned to the delegate using "+" operator. It is called multicast delegate.

Delegate is also used with [Event](#), [Anonymous method](#), [Func delegate](#), [Action delegate](#).

## 2. Lambda Expressions In C#

<https://www.tutorialsteacher.com/linq/linq-lambda-expression>

<https://kudchikarsk.com/delegates-and-events-in-csharp/#lambda-expressions-in-csharp>

Sometimes the whole signature of a method can be more code than the body of a method. There are also situations in which you need to create an entire method only to use it in a delegate.

For these cases, Microsoft added some new features to C#, 2.0 anonymous methods were added. In C# 3.0, things became even better when lambda expressions were added. Lambda expression is the preferred way to go when writing new code.

Below is an example of newer lambda syntax.

```
class Program
{
    public delegate double MathDelegate(double value1, double value2);

    public static void Main()
    {
        MathDelegate mathDelegate = (x, y) => x + y;
        var result = mathDelegate(5, 2);
        Console.WriteLine(result);
        // output: 7

        mathDelegate = (x, y) => x - y;
        result = mathDelegate(5, 2);
        Console.WriteLine(result);
        // output: 3

        Console.ReadLine();
    }
}
```

When reading this code, you can say go or goes to for the special lambda syntax. For example, the first lambda expression in the above example is read as “x and y goes to adding x and y”.

The lambda function has no specific name as the methods. Because of this, lambda functions are called anonymous functions. You also don't have to specify a return type explicitly. The compiler infers this automatically from your lambda. And in the case of the above example, the types of parameters x and y are also not specified explicitly.

You can create lambdas that span multiple statements. You can do this by adding curly braces around the statements that form the lambda as below example shows.

```
MathDelegate mathDelegate = (x, y) =>
{
    Console.WriteLine("Add");
    return x + y;
};
```

You can learn more about .NET built-in delegates [here](#).

### 2.1. Lambda examples

<https://www.tutorialsteacher.com/linq/linq-lambda-expression>

Si riportano di seguito alcuni esempi che spiegano il passaggio da funzione anonima a lambda expression

Si consideri il seguente esempio di delegato con funzione anonima:

```
namespace ArgomentiAvanzati
{
    public class Program
    {
        delegate bool IsTeenAger(Student stud);

        public static void Main()
        {
            IsTeenAger isTeenAger = delegate (Student s) { return s.Age > 12 && s.Age < 20; };

            Student stud = new Student() { Age = 25 };

            Console.WriteLine(isTeenAger(stud));
            Console.ReadLine();
        }
    }

    public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
    }
}
```

Lo stesso risultato si può ottenere con il seguente costrutto lambda

```
namespace ArgomentiAvanzati
{
    public class Program
    {
        delegate bool IsTeenAger(Student stud);

        public static void Main()
        {
            IsTeenAger isTeenAger = s => s.Age > 12 && s.Age < 20;

            Student stud = new Student() { Age = 25 };

            Console.WriteLine(isTeenAger(stud));
            Console.ReadLine();
        }
    }

    public class Student
    {
```

```

    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}

```

}

Spiegazione:

The Lambda expression evolves from anonymous method by first removing the delegate keyword and parameter type and adding a lambda operator =>.

```

delegate(Student s) { return s.Age > 12 && s.Age < 20; };
© TutorialsTeacher.com
↓
1 - Remove Delegate and Parameter Type and
add lambda operator =>
delegate(StIudent s)=> { return s.Age > 12 && s.Age < 20; };
↓
(s) => { return s.Age > 12 && s.Age < 20; };
Lambda Expression from Anonymous Method

```

The above lambda expression is absolutely valid, but we don't need the curly braces, return and semicolon if we have only one statement that returns a value. So we can eliminate it.

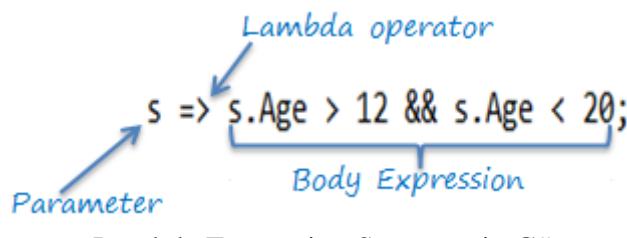
Also, we can remove parenthesis (), if we have only one parameter.

```

(s) => { return s.Age > 12 && s.Age < 20;} ;
© TutorialsTeacher.com
↓
2 - Remove curly bracket, return and semicolon
(s) => s.Age > 12 && s.Age < 20;
↓
3 - Remove Parenthesis around parameter if there
is only one parameter
s => s.Age > 12 && s.Age < 20;
Lambda Expression from Anonymous Method

```

Thus, we got the lambda expression: `s => s.Age > 12 && s.Age < 20` where `s` is a parameter, `=>` is the lambda operator and `s.Age > 12 && s.Age < 20` is the body expression:



Lambda Expression Structure in C#

You can wrap the parameters in parenthesis if you need to pass more than one parameter, as below:

```
namespace ArgomentiAvanzati
{
    public class Program
    {
        delegate bool IsYoungerThan(Student stud, int youngAge);

        public static void Main()
        {
            IsYoungerThan isYoungerThan = (s, youngAge) => s.Age < youngAge;

            Student stud = new Student() { Age = 25 };

            Console.WriteLine(isYoungerThan(stud, 26));
            Console.ReadLine();
        }
    }

    public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
    }
}
```

You can also give type of each parameters if parameters are confusing:

```
(Student s,int youngAge) => s.Age >= youngage;
```

It is not necessary to have at least one parameter in a lambda expression. The lambda expression can be specify without any parameter also.

```
() => Console.WriteLine("Parameter less lambda expression")
```

Ad esempio:

```
namespace ArgomentiAvanzati
{
    public class Program
    {
        delegate void Print();

        public static void Main()
        {
```

```

        Print print = () => Console.WriteLine("This is parameter less lambda
expression");

        print();
        Console.ReadLine();

    }

}

```

You can wrap expressions in curly braces if you want to have more than one statement in the body:

```

(s, youngAge) =>
{
    Console.WriteLine("Lambda expression with multiple statements in the body");

    Return s.Age >= youngAge;
}

```

Ad esempio:

```

namespace ArgomentiAvanzati
{
    public class Student
    {

        public int Id { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
    }

    public class Program
    {
        delegate bool IsYoungerThan(Student stud, int youngAge);

        public static void Main()
        {
            IsYoungerThan isYoungerThan = (s, youngAge) => {

                Console.WriteLine("Lambda expression with multiple statements in the body");

                return s.Age < youngAge;
            };

            Student stud = new Student() { Age = 25 };

            Console.WriteLine(isYoungerThan(stud, 26));
            Console.ReadLine();
        }
    }
}

```

You can declare a variable in the expression body to use it anywhere in the expression body, as below:

```

namespace ArgomentiAvanzati
{
    public class Program
    {
        delegate bool IsAdult(Student stud);

```

```

public static void Main()
{
    IsAdult isAdult = (s) => {
        int adultAge = 18;

        Console.WriteLine("Lambda expression with multiple statements in the body");

        return s.Age >= adultAge;
    };

    Student stud = new Student() { Age = 25 };

    Console.WriteLine(isAdult(stud));
    Console.ReadLine();
}

public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}
}

```

### 2.1.1. Function Delegate and its use with lambda

The .NET Framework has a couple of built-in delegates types that you can use when declaring delegates.

For the MathDelegate examples, you have used the following delegate:

```
public delegate double MathDelegate(double value1, double value2);
```

You can replace this delegate with one of the built-in types namely `Func<int, int, int>`.

like this,

```

class Program
{
    public static void Main()
    {
        //modalità alternativa per definire un delegato
        Func<int, int, int> mathDelegate = (x, y) =>
        {
            Console.WriteLine("Add");
            return x + y;
        };

        var result = mathDelegate(5, 2);
        Console.WriteLine(result);
        // output: 7

        mathDelegate = (x, y) => x - y; ;
        result = mathDelegate(5, 2);
        Console.WriteLine(result);
        // output: 3

        Console.ReadLine();
    }
}

```

```
}
```

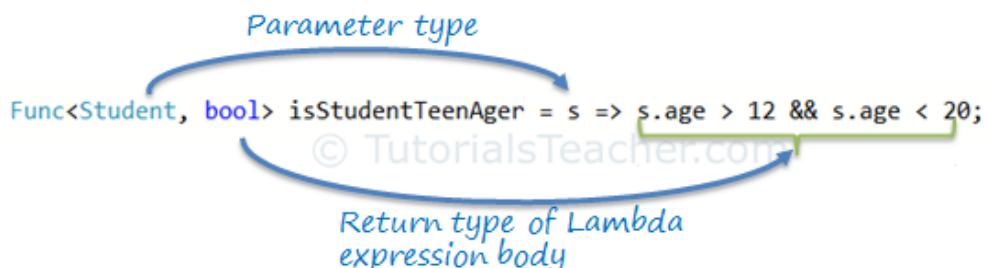
The `Func<...>` types can be found in the `System` namespace and they represent delegates that return a type and take 0 to 16 parameters. All those types inherit from `System.MulticastDelegate` so you can add multiple methods to the invocation list.

The lambda expression can be assigned to `Func<in T, out TResult>` type delegate. The last parameter type in a `Func` delegate is the return type and rest are input parameters. Visit [Func delegate](#) section of C# tutorials to know more about it.

Consider the following lambda expression to find out whether a student is a teenager or not.

```
Func<Student, bool> isStudentTeenAger = s => s.Age > 12 && s.Age < 20;  
  
Student std = new Student() { age = 21 };  
  
bool isTeen = isStudentTeenAger(std); // returns false
```

In the above example, the `Func` delegate expects the first input parameter to be of `Student` type and the return type to be boolean. The lambda expression `s => s.age > 12 && s.age < 20` satisfies the `Func<Student, bool>` delegate requirement, as shown below:



#### Func delegate with Lambda Expression

The `Func<>` delegate shown above, would turn out to be a function as shown below.

```
bool isStudentTeenAger(Student s)  
{  
    return s.Age > 12 && s.Age < 20;  
}
```

### 2.1.2. Action Delegate and its use with lambda

Unlike the `Func` delegate, an `Action` delegate can only have input parameters. Use the [Action delegate](#) type when you don't need to return any value from lambda expression.

If you want a delegate type that doesn't return a value, you can use the `System.Action` types. They can also take 0 to 16 parameters, but they don't return a value.

Here is an example of using the `Action` type,

```
class Program
```

```

{
    public static void Main()
    {
        Action<int, int> mathDelegate = (x, y) =>
        {
            Console.WriteLine(x + y);
        };

        mathDelegate(5, 2);
        // output: 7

        mathDelegate = (x, y) => Console.WriteLine(x - y);
        mathDelegate(5, 2);
        // output: 3

        Console.ReadLine();
    }
}

```

**Esempio con le lambda expressions:**

```

namespace ArgomentiAvanzati
{
    public class Program
    {

        public static void Main()
        {
            Action<Student> PrintStudentDetail = s => Console.WriteLine("Name: {0}, Age: {1}"
", s.Name, s.Age);

            Student std = new Student() { Name = "Bill", Age = 21 };

            PrintStudentDetail(std);
            Console.ReadLine();
        }
    }

    public class Student
    {

        public int Id { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
    }
}

```

Things start to become more complex when your lambda function starts referring to variables declared outside of the lambda expression or to this reference. Normally when control leaves the scope of the variable, the variable is no longer valid. But what if a delegate refers to a local variable. To fix this, the compiler generates code that makes the life of the captured variable at least as long as the longest-living delegate. This is called a closure.

You can learn more about closure [here](#).

## 2.2.What's a Closure?

A closure is a function that is bound to the environment in which it is declared. Thus, the function can reference elements from the environment within its body. In the case of a C# 2.0 anonymous method, the environment to which it is bound is its parenting method body. This means that local variables from the parenting method body can be referenced within the anonymous method's body. So, this code prints 0 to the console as expected:

```
class Program
{
    delegate void MyAction();

    static void Main(string[] args)
    {
        int x = 0;

        MyAction a = delegate { Console.WriteLine(x); };

        a();
        Console.ReadLine();
    }
}
```

Most developers don't have any problem with the code above. A local variable "x" is declared and initialized to 0. Then, a new delegate "a" of type Action is declared and assigned to an anonymous method that writes "x" to the console. Finally, "a" is called and the value of "x" (0) is printed to the console. The rub occurs when the code is changed like this:

```
class Program
{
    delegate void MyAction();

    static void Main(string[] args)
    {
        int x = 0;

        MyAction a = delegate { Console.WriteLine(x); };

        x = 1;

        a();

        Console.ReadLine();
    }
}
```

Now, "x" is reassigned to a value of 1 before "a" is called. What will be output to the console?

It turns out that the answer is 1, not 0. The reason for this is that the anonymous method is a closure and is bound to its parenting method body and the local variables in it. The important distinction is that it is bound to variables, not to values. In other words, the value of "x" is not copied in when "a" is declared. Instead, a reference to "x" is used so that "a" will always use the most recent value of "x". In fact, this reference to "x" will be persisted even if "x" goes out of scope. Consider this code:

```
class Program
{
    delegate void MyAction();

    static MyAction GetAction()
```

```

{
    int x = 0;

    MyAction a = delegate { Console.WriteLine(x); };

    x = 1;

    return a;
}

static void Main(string[] args)
{
    MyAction a = GetAction();

    a();

    Console.ReadLine();
}
}

```

That will still print 1 to the console even though "x" is out of scope by the time that "a" is called. So, how is this achieved? Well, the good news is that this is handled through compiler magic. There isn't any runtime support for closures. That means that you could use the same techniques to create a closure without using an anonymous method.

## 2.3. Points to Remember

1. Lambda Expression is a shorter way of representing anonymous method.
2. Lambda Expression syntax: *parameters => body expression*
3. Lambda Expression can have zero parameter.
4. Lambda Expression can have multiple parameters in parenthesis () .
5. Lambda Expression can have multiple statements in body expression in curly brackets {} .
6. Lambda Expression can be assigned to Func, Action or Predicate delegate.
7. Lambda Expression can be invoked in a similar way to delegate.

## 3. Events in C#

### 3.1. What Are Events In C#?

<https://kudchikarsk.com/delegates-and-events-in-csharp/#events-in-csharp>

<https://www.tutorialsteacher.com/csharp/csharp-event>

In general terms, an event is something special that is going to happen. For example, Microsoft launches events for developers, to make them aware about the features of new or existing products. Microsoft notifies the developers about the event by email or other advertisement options. So in this case, Microsoft is a publisher who launches (raises) an event and notifies the developers about it and developers are the subscribers of the event and attend (handle) the event.

Events in C# follow a similar concept. An event has a publisher, subscriber, notification and a handler. Generally, UI controls use events extensively. For example, the button control in a Windows form has

multiple events such as click, mouseover, etc. A custom class can also have an event to notify other subscriber classes about something that has happened or is going to happen. Let's see how you can define an event and notify other classes that have event handlers.

An event is nothing but an encapsulated delegate. As we have learned in the previous section, a delegate is a reference type data type. You can declare the delegate as shown below:

An event can be used to provide notifications. You can subscribe to an event if you are interested in those notifications. You can also create your own events and raise them to provide notifications when something interesting happens. The .NET Framework offers built-in types that you can use to create events. By using delegates, lambda expressions, and anonymous methods, you can create and use events in a comfortable way.

A popular design pattern in application development is that of publish-subscribe. You can subscribe to an event and then you are notified when the publisher of the event raises a new event. This is used to establish loose coupling between components in an application.

Delegate form the basis for the event system in C#.

An event is a special kind of delegate that facilitates event-driven programming. Events are class members that cannot be called outside of the class regardless of its access specifier. So, for example, an event declared to be public would allow other classes the use of += and -= on the event, but firing the event (i.e. invoking the delegate) is only allowed in the class containing the event. Let's see an example,

To declare an event, use the **event** keyword before declaring a variable of delegate type, as below:

```
public delegate void someEvent();  
  
public event someEvent someEvent;
```

Thus, a delegate becomes an event using the event keyword.

### 3.1.1. Primo esempio

```
namespace ArgomentiAvanzati  
{  
  
    //Define publisher class as Pub  
    public class Pub  
    {  
        //OnChange property containing all the  
        //list of subscribers callback methods  
        public event Action OnChange = delegate { };  
        public event Action<int> OnGetInput = delegate { };  
  
        public void Raise()  
        {  
            //Invoke OnChange Action  
            OnChange();  
        }  
  
        //un esempio che prende in input un parametro di tipo intero
```

```

        public void RaiseWithInput(int p)
    {
        OnGetInput(p);
    }
}
class Program
{
    static void Main(string[] args)
    {
        //Initialize pub class object
        Pub p = new Pub();

        //register for OnChange event - Subscriber 1
        p.OnChange += () => Console.WriteLine("Subscriber 1!");
        //register for OnChange event - Subscriber 2
        p.OnChange += () => Console.WriteLine("Subscriber 2!");

        //raise the event
        p.Raise();
        //After this Raise() method is called
        //all subscribers callback methods will get invoked

        //definiamo una variabile locale per dimostrare il concetto di closure
        int x = 5;

        //subscriber all'evento OnGetInput: accetta un input un numero intero
        p.OnGetInput += (q) => {
            Console.WriteLine("OnGetInput fired");
            q = x + 1;
            Console.WriteLine($"parametro locale q = {q}");
            Console.WriteLine($"riferimento esterno alla lambda expression x = {x}");
        };

        Console.WriteLine("Invochiamo RaiseWithInput");
        //qui passiamo un valore intero che corrisponderà alla q
        p.RaiseWithInput(6);

        Console.WriteLine("Press enter to terminate!");
        Console.ReadLine();
    }
}

```

Even though the event is declared public, it cannot be directly fired anywhere except in the class containing it. By using **event** keyword compiler protects our field from unwanted access. And, it does not permit the use of = (direct assignment of a delegate). Hence, your code is now safe from the risk of removing previous subscribers by using = instead of +=.

Also, you may have noticed special syntax of initializing the OnChange field to an empty delegate like this **delegate { }**. This ensures that our OnChange field will never be null. Hence, we can remove the null check before raising the event, if no other members of the class making it null.

When you run the above program, your code creates a new instance of Pub, subscribes to the event with two different methods and raises the event by calling p.Raise or p.RaiseWithInput. The Pub class is completely unaware of any subscribers. It just raises the event.

You can also read the article [Publish Subscribe Design Pattern In C#](#) for more in-depth knowledge of this concept.

### 3.1.2. Un esempio articolato

```
namespace EventiDemo
{
    public class Student
    {

        public int Id { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
    }

    public class Exam
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Score { get; set; }
    }

    public class StudentTutor
    {
        public event Action<Student> OnStudentReceived;
        public event Func<Student, Exam, int> OnStudentTakeExam;

        //metodo che solleva l'evento StudentReceived
        public void StudentReceived(Student s)
        {
            OnStudentReceived(s);
        }

        //metodo che solleva l'evento StudentTakeExam
        public void StudentTakeExam(Student s, Exam e)
        {
            //callback function che restituisce il punteggio dell'esame
            int score = OnStudentTakeExam(s, e);
            Console.WriteLine($"Lo studente {s.Name} ha sostenuto l'esame {e.Name} con un
punteggio di {score}");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            //creazione di un oggetto Publisher
            StudentTutor tutor = new StudentTutor();
            //creazione di un oggetto Publisher
            tutor.OnStudentReceived += (s) => Console.WriteLine($"Benvenuto {s.Name}");
            tutor.OnStudentTakeExam += Tutor_OnStudentTakeExam;

            //creazione di studenti ed esami
            Student marioRossi = new Student() { Id = 1, Name = "Mario Rossi", Age = 21 };
            Exam fisica1 = new Exam() { Id = 1, Name = "Fisica 1" };
            Exam analisi1 = new Exam() { Id = 2, Name = "Analisi 1" };

            //attivazione eventi
            tutor.StudentReceived(marioRossi);
            tutor.StudentTakeExam(marioRossi, fisica1 );
            tutor.StudentTakeExam(marioRossi, analisi1);
            tutor.StudentTakeExam(marioRossi, fisica1);
            tutor.StudentTakeExam(marioRossi, analisi1);
            tutor.StudentTakeExam(marioRossi, fisica1);
            tutor.StudentTakeExam(marioRossi, analisi1);
        }
    }
}
```

```

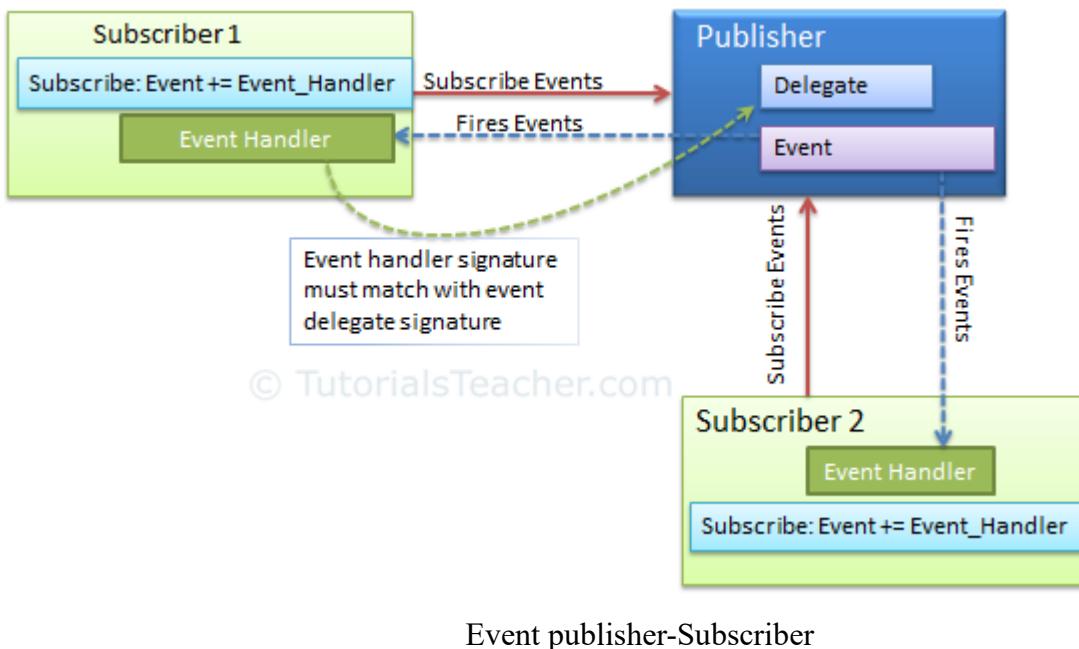
static Random gen = new Random();
//implementazione callback function corrispondenti agli eventi
private static int Tutor_OnStudentTakeExam(Student s, Exam e)
{
    return gen.Next(15, 31);
}

private static void Tutor_OnStudentReceived(Student s)
{
    Console.WriteLine($"Benvenuto {s.Name}");
}
}

```

All the subscribers must provided a handler function, which is going to be called when a publisher raises an event.

The following image illustrates an event model:



### 3.2. Points to Remember

1. Use event keyword with delegate type to declare an event.
2. Check event is null or not before raising an event.
3. Subscribe to events using "+=" operator. Unsubscribe it using "-=" operator.
4. Function that handles the event is called event handler. Event handler must have same signature as declared by event delegate.
5. Events can have arguments which will be passed to handler function.
6. Events can also be declared static, virtual, sealed and abstract.
7. An Interface can include event as a member.
8. Events will not be raised if there is no subscriber
9. Event handlers are invoked synchronously if there are multiple subscribers
10. The .NET framework uses an [EventHandler](#) delegate and an [EventArgs](#) base class.

## 4. LINQ

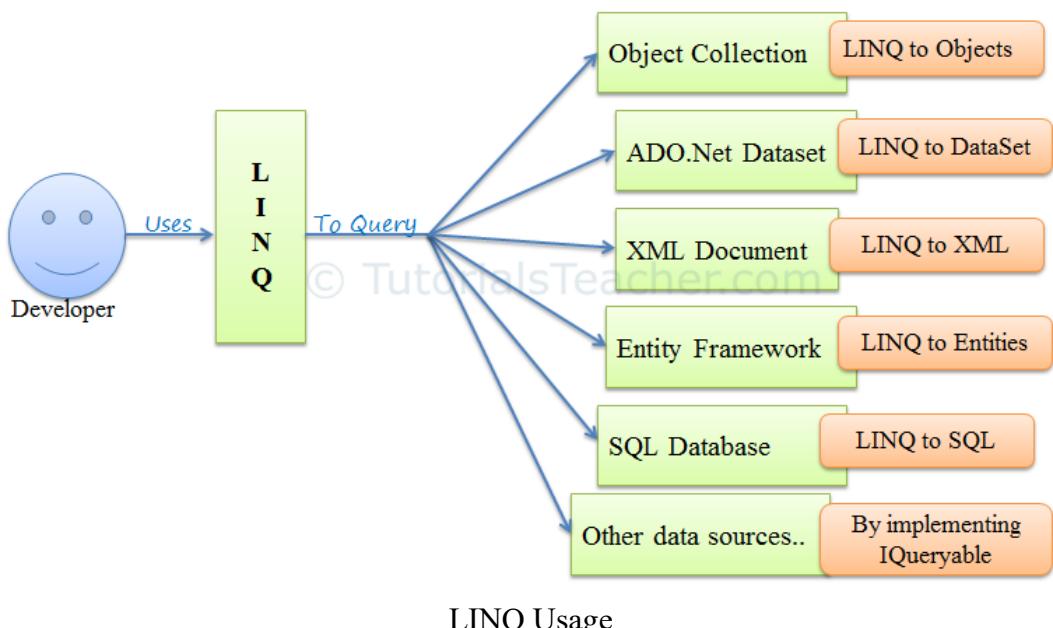
Language-Integrated Query (LINQ) is a powerful query language introduced with .Net 3.5.

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/>

### 4.1. What is LINQ?

LINQ (Language Integrated Query) is uniform query syntax in C# to retrieve data from different sources and formats. It is integrated in C#, thereby eliminating the mismatch between programming languages and databases, as well as providing a single querying interface for different types of data sources.

For example, SQL is a Structured Query Language used to save and retrieve data from a database. In the same way, LINQ is a structured query syntax built in C# to retrieve data from different types of data sources such as collections, ADO.Net DataSet, XML Docs, web service and MS SQL Server and other databases.



LINQ queries return results as objects. It enables you to use object-oriented approach on the result set and not to worry about transforming different formats of results into objects.



The following example demonstrates a simple LINQ query that gets all strings from an array which contains 'a'.

```
using System;
using System.Linq;

namespace ArgomentiAvanzati
{
    public class Program
    {
        public static void Main()
        {
```

```

    // Data source
    string[] names = { "Bill", "Steve", "James", "Mohan" };

    // LINQ Query
    var myLinqQuery = from name in names
                      where name.Contains('a')
                      select name;

    // Query execution
    foreach (var name in myLinqQuery)
        Console.WriteLine(name + " ");

    Console.ReadLine();
}

}

}

```

In the above example, string array names is a data source. The following is a LINQ query which is assigned to a variable myLinqQuery.

```

from name in names
where name.Contains('a')
select name;

```

The above query uses query syntax of LINQ. You will learn more about it in the [Query Syntax](#) chapter.

You will not get the result of a LINQ query until you execute it. LINQ query can be executed in multiple ways, here we used foreach loop to execute our query stored in myLinqQuery. The foreach loop executes the query on the data source and get the result and then iterates over the result set.

Thus, every LINQ query must query to some kind of data sources whether it can be array, collections, XML or other databases. After writing LINQ query, it must be executed to get the result.

## 4.2. Why LINQ?

To understand why we should use LINQ, let's look at some examples. Suppose you want to find list of teenage students from an array of Student objects.

Before C# 2.0, we had to use a 'foreach' or a 'for' loop to traverse the collection to find a particular object. For example, we had to write the following code to find all Student objects from an array of Students where the age is between 12 and 20 (for teenage 13 to 19):

### 4.2.1. Example: Use for loop to find elements from the collection in C# 1.0

```

using System;

namespace ArgomentiAvanzati
{
    class Student
    {
        public int StudentID { get; set; }
        public String StudentName { get; set; }
        public int Age { get; set; }

        public override string ToString()
    }
}

```

```

        {
            return String.Format($"[StudentID = {StudentID}, StudentName = {StudentName},
Age = {Age}]");
        }
    }

class Program
{
    static void Main(string[] args)
    {
        Student[] studentArray = {
            new Student() { StudentID = 1, StudentName = "John", Age = 18 },
            new Student() { StudentID = 2, StudentName = "Steve", Age = 21 },
            new Student() { StudentID = 3, StudentName = "Bill", Age = 25 },
            new Student() { StudentID = 4, StudentName = "Ram", Age = 20 },
            new Student() { StudentID = 5, StudentName = "Ron", Age = 31 },
            new Student() { StudentID = 6, StudentName = "Chris", Age = 17 },
            new Student() { StudentID = 7, StudentName = "Rob", Age = 19 },
        };

        Student[] students = new Student[10];

        int i = 0;

        foreach (Student std in studentArray)
        {
            if (std.Age > 12 && std.Age < 20)
            {
                students[i] = std;
                i++;
            }
        }

        //write result
        foreach (var studente in students)
        {
            Console.WriteLine(studente);
        }
        Console.ReadLine();
    }
}

```

}

Use of for loop is cumbersome, not maintainable and readable. C# 2.0 introduced delegate, which can be used to handle this kind of a scenario, as shown below.

#### 4.2.2. Example: Use Delegates to Find Elements from the Collection

```
using System;
```

```

namespace ArgomentiAvanzati
{
    delegate bool FindStudent(Student std);

    class StudentExtension
    {
        public static Student[] where(Student[] stdArray, FindStudent del)
        {
            int i = 0;
            Student[] result = new Student[10];
            foreach (Student std in stdArray)
                if (del(std))

```

```

        {
            result[i] = std;
            i++;
        }

        return result;
    }
}

class Student
{
    public int StudentID { get; set; }
    public String StudentName { get; set; }
    public int Age { get; set; }

    public override string ToString()
    {
        return String.Format($"[StudentID = {StudentID}, StudentName = {StudentName},
Age = {Age}]");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Student[] studentArray = {
            new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,
            new Student() { StudentID = 2, StudentName = "Steve", Age = 21 } ,
            new Student() { StudentID = 3, StudentName = "Bill", Age = 25 } ,
            new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 } ,
            new Student() { StudentID = 5, StudentName = "Ron" , Age = 31 } ,
            new Student() { StudentID = 6, StudentName = "Chris", Age = 17 } ,
            new Student() { StudentID = 7, StudentName = "Rob", Age = 19 } ,
        };

        Student[] students = StudentExtension.where(studentArray, delegate (Student std)
        {
            return std.Age > 12 && std.Age < 20;
        });
        //in alternativa al delegate si può usare una lambda
        //Student[] students = StudentExtension.where(studentArray,
        //        std => std.Age > 12 && std.Age < 20);
        //write result
        foreach (var studente in students)
        {
            Console.WriteLine(studente);
        }
        Console.ReadLine();
    }
}

```

So, with C# 2.0, you got the advantage of **delegate** in finding students with any criteria. You don't have to use a for loop to find students using different criteria. For example, you can use the same delegate function to find a student whose StudentId is 5 or whose name is Bill, as below:

```

Student[] students = StudentExtension.where(studentArray, delegate(Student std) {
    return std.StudentID == 5;
});

//Also, use another criteria using same delegate

```

```

Student[] students = StudentExtension.where(studentArray, delegate(Student std) {
    return std.StudentName == "Bill";
});

```

The C# team felt that they still needed to make the code even more compact and readable. So they introduced the extension method, lambda expression, expression tree, anonymous type and query expression in [C# 3.0](#). You can use these features of C# 3.0, which are building blocks of LINQ to query to the different types of collection and get the resulted element(s) in a single statement.

```

using System;
using System.Linq;

namespace ArgomentiAvanzati
{
    class Student
    {
        public int StudentID { get; set; }
        public String StudentName { get; set; }
        public int Age { get; set; }

        public override string ToString()
        {
            return String.Format($"[StudentID = {StudentID}, StudentName = {StudentName},
Age = {Age}]");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Student[] studentArray =
                new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,
                new Student() { StudentID = 2, StudentName = "Steve", Age = 21 } ,
                new Student() { StudentID = 3, StudentName = "Bill", Age = 25 } ,
                new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 } ,
                new Student() { StudentID = 5, StudentName = "Ron" , Age = 31 } ,
                new Student() { StudentID = 6, StudentName = "Chris", Age = 17 } ,
                new Student() { StudentID = 7, StudentName = "Rob" , Age = 19 } ,
            };

            // Use LINQ to find teenager students
            Student[] teenAgerStudents = studentArray.Where(s => s.Age > 12 && s.Age <
20).ToArray();

            // Use LINQ to find first student whose name is Bill
            Student bill = studentArray.Where(s => s.StudentName ==
"Bill").FirstOrDefault();

            // Use LINQ to find student whose StudentID is 5
            Student student5 = studentArray.Where(s => s.StudentID == 5).FirstOrDefault();

            //write result
            foreach (var studente in teenAgerStudents)
            {
                Console.WriteLine(studente);
            }
            Console.WriteLine("bill: "+bill);
            Console.WriteLine("student5: "+student5);
            Console.ReadLine();
        }
    }
}

```

```
    }  
}
```

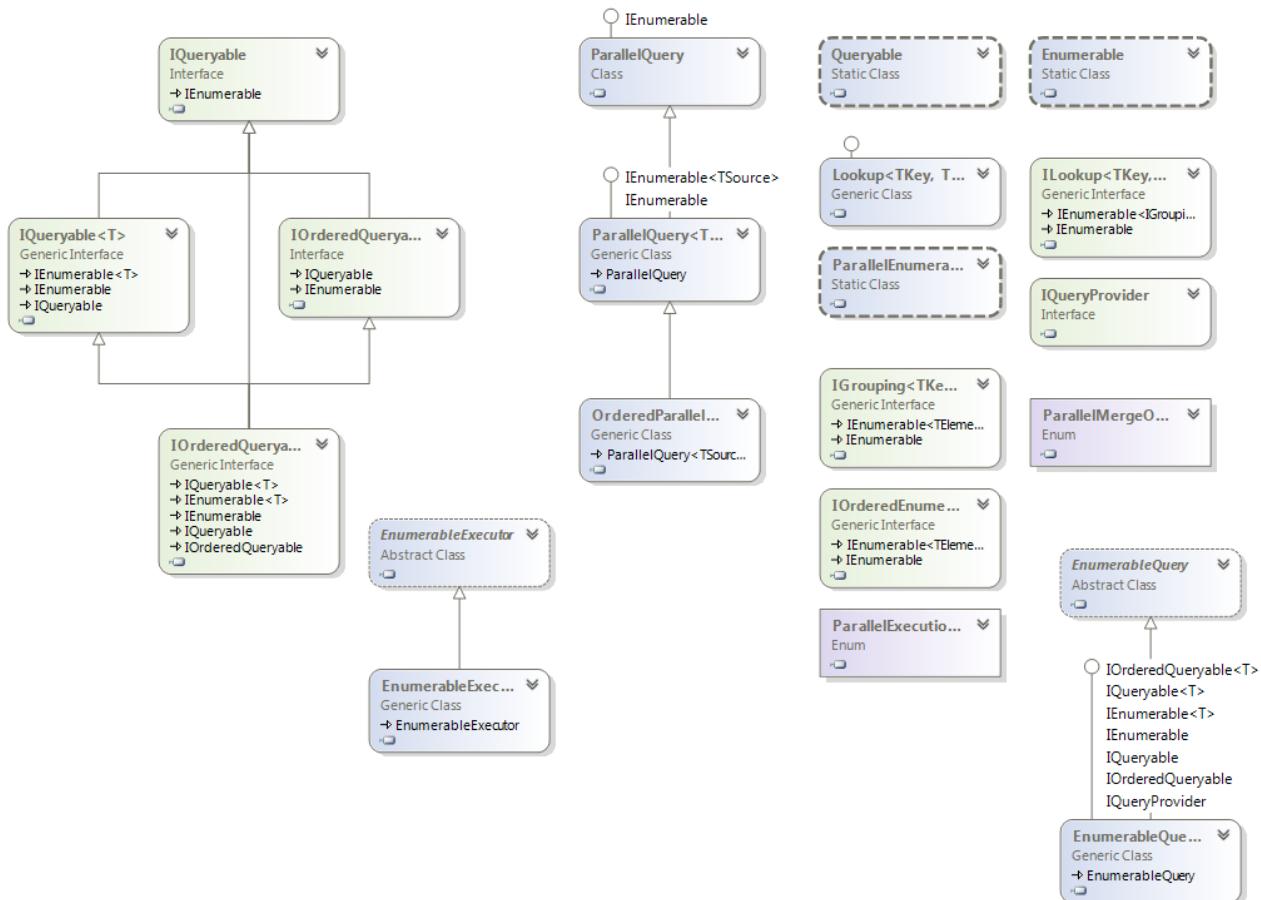
As you can see in the above example, we specify different criteria using LINQ operator and lambda expression in a single statement. Thus, LINQ makes code more compact and readable and it can also be used to query different data sources. For example, if you have a student table in a database instead of an array of student objects as above, you can still use the same query to find students using the [Entity Framework](#).

#### 4.2.3. Advantages of LINQ

- **Familiar language:** Developers don't have to learn a new query language for each type of data source or data format.
- **Less coding:** It reduces the amount of code to be written as compared with a more traditional approach.
- **Readable code:** LINQ makes the code more readable so other developers can easily understand and maintain it.
- **Standardized way of querying multiple data sources:** The same LINQ syntax can be used to query multiple data sources.
- **Compile time safety of queries:** It provides type checking of objects at compile time.
- **IntelliSense Support:** LINQ provides IntelliSense for generic collections.
- **Shaping data:** You can retrieve data in different shapes.

#### 4.2.4. LINQ API

We can write LINQ queries for the classes that implement [IEnumerable<T>](#) or [IQueryable<T>](#) interface. The [System.Linq](#) namespace includes the following classes and interfaces required for LINQ queries.



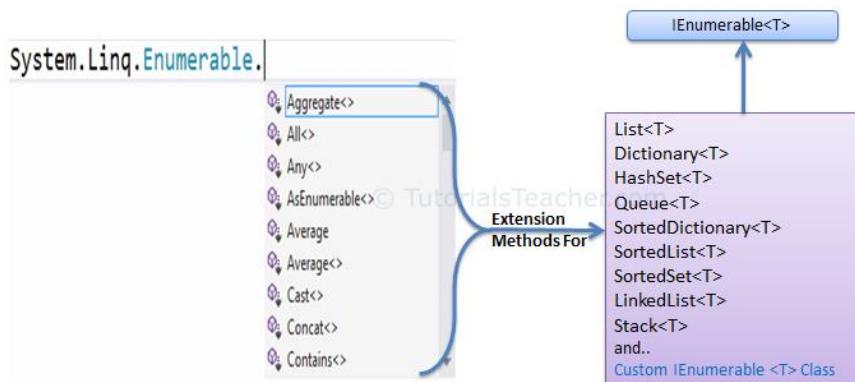
## LINQ API

LINQ queries uses extension methods for classes that implement `IEnumerable` or `IQueryable` interface.

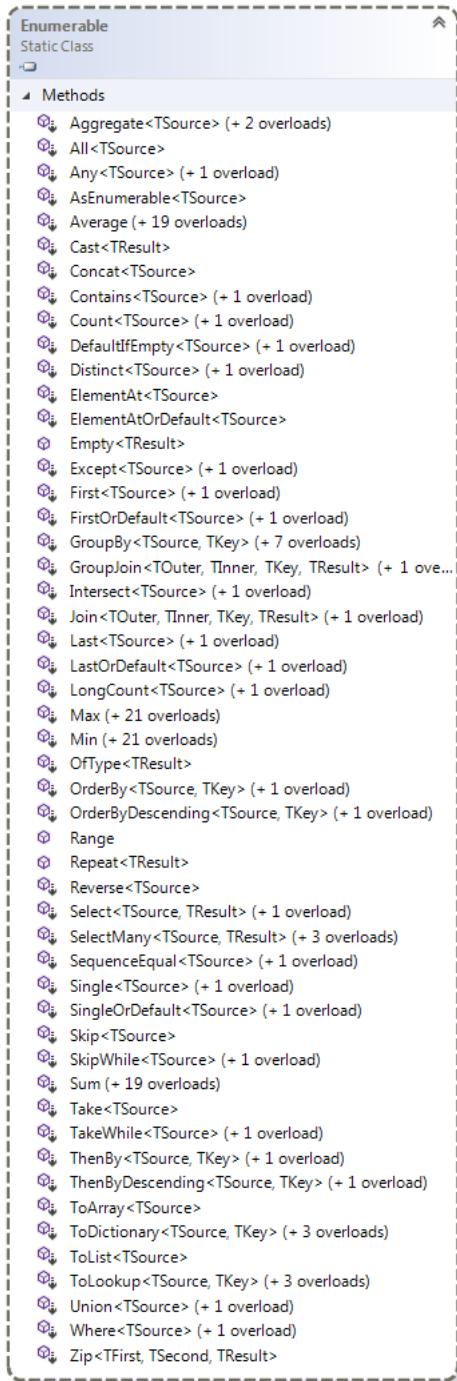
### 4.2.5. System.Linq.Enumerable

The `Enumerable` class includes extension methods for the classes that implement `IEnumerable<T>` interface, for example all the built-in collection classes implement `IEnumerable<T>` interface and so we can write LINQ queries to retrieve data from the built-in collections.

The following figure shows the extension methods included in `Enumerable` class that can be used with the generic collections in C#.



The following figure shows all the extension methods available in `Enumerable` class.



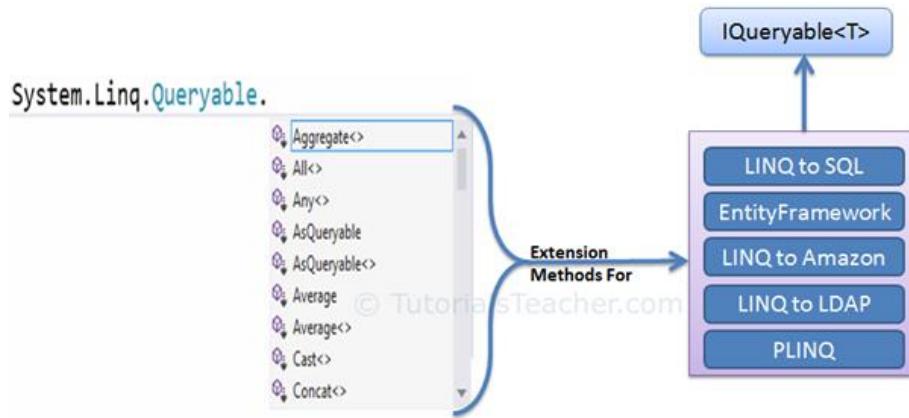
Enumerable Class

#### 4.2.6. System.Linq.Queryable

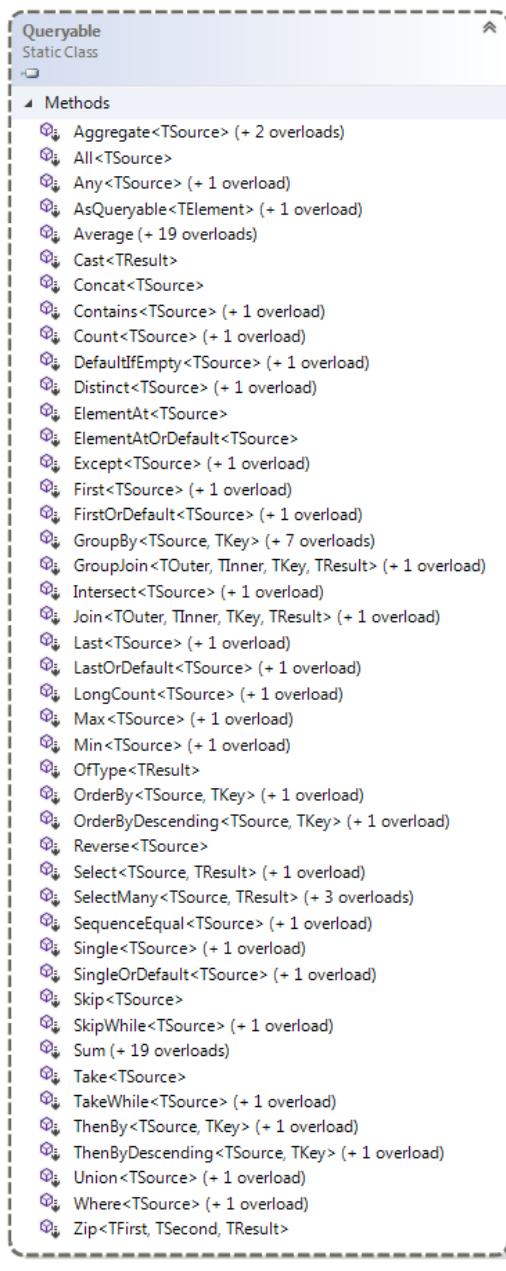
The [Queryable](#) class includes extension methods for classes that implement [IQueryable<T>](#) interface. The [IQueryable<T>](#) interface is used to provide querying capabilities against a specific data source where the type of the data is known. For example, Entity Framework api implements [IQueryable<T>](#) interface to support LINQ queries with underlaying databases such as MS SQL Server.

Also, there are APIs available to access third party data; for example, LINQ to Amazon provides the ability to use LINQ with Amazon web services to search for books and other items. This can be achieved by implementing the [IQueryable](#) interface for Amazon.

The following figure shows the extension methods available in the Queryable class can be used with various native or third party data providers.



The following figure shows the extension methods available in the `Queryable` class.



Queryable class

## 4.2.7. Come usare LINQ?

Ci sono due modi per usare LINQ: mediante l'uso delle "LINQ query" oppure mediante l'uso dei "LINQ methods".

### 4.2.7.1. LINQ Query (Query Expression Syntax)

Query syntax is similar to SQL (Structured Query Language) for the database. It is defined within the C#.

LINQ Query Syntax:

```
from <range variable> in <IEnumerable<T> or IQueryable<T> Collection>
```

```
<Standard Query Operators> <lambida expression>
```

```
<select or groupBy operator> <result formation>
```

The LINQ query syntax starts with from keyword and ends with select keyword. The following is a sample LINQ query that returns a collection of strings which contains a word "Tutorials".

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace ArgomentiAvanzati
{

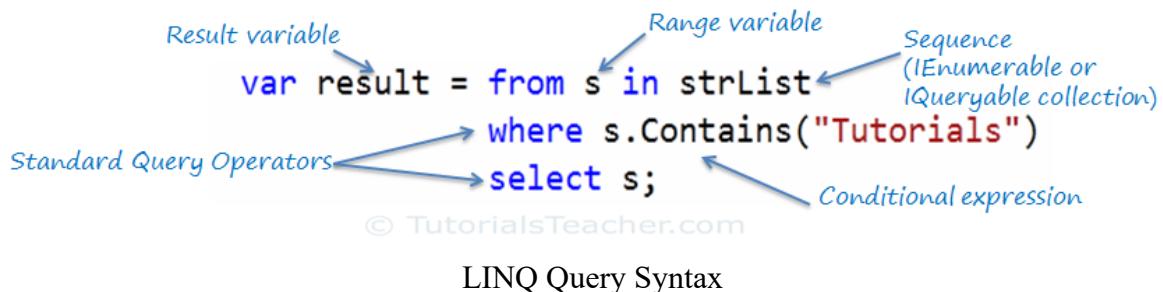
    class Program
    {
        public static void Main()
        {
            // string collection
            IList<string> stringList = new List<string>() {
                "C# Tutorials",
                "VB.NET Tutorials",
                "Learn C++",
                "MVC Tutorials" ,
                "Java"
            };

            // LINQ Query Syntax
            var result = from s in stringList
                         where s.Contains("Tutorials")
                         select s;

            foreach (var str in result)
            {
                Console.WriteLine(str);
            }

            Console.ReadLine();
        }
    }
}
```

The following figure shows the structure of LINQ query syntax.



Query syntax starts with a **From** clause followed by a **Range** variable. The **From** clause is structured like "**From** rangeVariableName **in** IEnumerablecollection". In English, this means, from each object in the collection. It is similar to a foreach loop: foreach(Student s in studentList).

After the From clause, you can use different Standard Query Operators to filter, group, join elements of the collection. There are around 50 Standard Query Operators available in LINQ. In the above figure, we have used "where" operator (aka clause) followed by a condition. This condition is generally expressed using lambda expression.

LINQ query syntax always ends with a Select or Group clause. The Select clause is used to shape the data. You can select the whole object as it is or only some properties of it. In the above example, we selected the each resulted string elements.

In the following example, we use LINQ query syntax to find out teenager students from the Student collection (sequence).

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace ArgomentiAvanzati
{
    public class Student
    {
        public int StudentID { get; set; }
        public string StudentName { get; set; }
        public int Age { get; set; }
    }

    class Program
    {
        public static void Main()
        {
            // Student collection
            IList<Student> studentList = new List<Student>() {
                new Student() { StudentID = 1, StudentName = "John", Age = 13} ,
                new Student() { StudentID = 2, StudentName = "Moin", Age = 21 } ,
                new Student() { StudentID = 3, StudentName = "Bill", Age = 18 } ,
                new Student() { StudentID = 4, StudentName = "Ram" , Age = 20} ,
                new Student() { StudentID = 5, StudentName = "Ron" , Age = 15 }
            };

            // LINQ Query Syntax to find out teenager students
            var teenAgerStudent = from s in studentList
                                  where s.Age > 12 && s.Age < 20
                                  select s;
            Console.WriteLine("Teen age Students:");
        }
    }
}
```

```

        foreach (Student std in teenAgerStudent)
        {
            Console.WriteLine(std.StudentName);
        }
        Console.ReadLine();
    }

}

```

#### 4.2.7.1.1. Points to Remember

1. As name suggest, **Query Syntax** is same like SQL (Structure Query Language) syntax.
2. Query Syntax starts with *from* clause and can be end with *Select* or *GroupBy* clause.
3. Use various other operators like filtering, joining, grouping, sorting operators to construct the desired result.
4. [Implicitly typed variable - var](#) can be used to hold the result of the LINQ query.

#### 4.2.7.2. LINQ Methods (Fluent)

Method syntax (also known as fluent syntax) uses extension methods included in the [Enumerable](#) or [Queryable](#) static class, similar to how you would call the extension method of any class.

The following is a sample LINQ method syntax query that returns a collection of strings which contains a word "Tutorials".

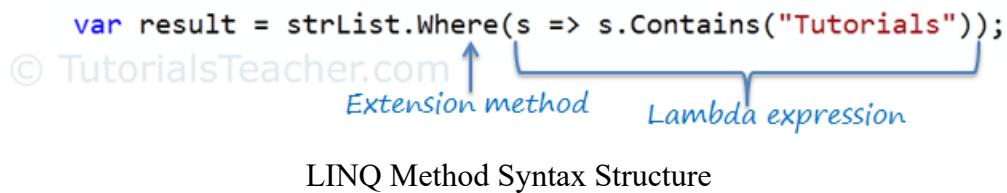
```

// string collection
IList<string> stringList = new List<string>() {
    "C# Tutorials",
    "VB.NET Tutorials",
    "Learn C++",
    "MVC Tutorials" ,
    "Java"
};

// LINQ Method Syntax
var result = stringList.Where(s => s.Contains("Tutorials"));

```

The following figure illustrates the structure of LINQ method syntax.



LINQ Method Syntax Structure

As you can see in the above figure, method syntax comprises of extension methods and Lambda expression. The extension method **Where()** is defined in the `Enumerable` class.

If you check the signature of the `Where` extension method, you will find the `Where` method accepts a [predicate](#) delegate as `Func<Student, bool>`. This means you can pass any delegate function that accepts a `Student` object as an input parameter and returns a Boolean value as shown in the below figure. The lambda expression works as a delegate passed in the `Where` clause.

```
var students = studentList.Where()
```

▲ 1 of 2 ▼ (extension) I Enumerable<Student> I Enumerable<Student>.Where(Func<Student, bool> predicate)

Filters a sequence of values based on a predicate.

*predicate: A function to test each element for a condition.*

Func delegate

### Func delegate in Where

The following example shows how to use LINQ method syntax query with the `IEnumerable<T>` collection.

```
// Student collection
IList<Student> studentList = new List<Student>() {
    new Student() { StudentID = 1, StudentName = "John", Age = 13} ,
    new Student() { StudentID = 2, StudentName = "Moin", Age = 21 } ,
    new Student() { StudentID = 3, StudentName = "Bill", Age = 18 } ,
    new Student() { StudentID = 4, StudentName = "Ram", Age = 20} ,
    new Student() { StudentID = 5, StudentName = "Ron" , Age = 15 }
};

// LINQ Method Syntax to find out teenager students
var teenAgerStudents = studentList.Where(s => s.Age > 12 && s.Age < 20)
    .ToList<Student>();
```

Altri esempi:

```
using System;
using System.Linq;
using System.Collections.Generic;

public class Program
{
    public static void Main()
    {
        // Student collection
        IList<Student> studentList = new List<Student>() {
            new Student() { StudentID = 1, StudentName = "John", Age = 13} ,
            new Student() { StudentID = 2, StudentName = "Moin", Age = 21 } ,
            new Student() { StudentID = 3, StudentName = "Bill", Age = 18 } ,
            new Student() { StudentID = 4, StudentName = "Ram", Age = 20} ,
            new Student() { StudentID = 5, StudentName = "Ron" , Age = 15 }
        };

        Func<Student, bool> isStudentTeenAger = s => s.Age > 12 && s.Age < 20;

        var teenAgerStudent = studentList.Where(isStudentTeenAger);

        Console.WriteLine("Teen age Students:");

        foreach (Student std in teenAgerStudent)
        {
            Console.WriteLine(std.StudentName);
        }

        // oppure
        var teenAgerStudents = from s in studentList
                               where isStudentTeenAger(s)
```

```
        select s;

    Console.WriteLine("Teen age Students:");

    foreach (Student std in teenAgerStudents)
    {
        Console.WriteLine(std.StudentName);
    }
    Console.ReadLine();
}

public class Student
{
    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public int Age { get; set; }
}
```

### **4.2.7.3. Standard Query Operators**

Standard Query Operators in LINQ are actually extension methods for the `IEnumerable<T>` and `IQueryable<T>` types. They are defined in the `System.Linq.Enumerable` and `System.Linq.Queryable` classes. There are over 50 standard query operators available in LINQ that provide different functionalities like filtering, sorting, grouping, aggregation, concatenation, etc.

#### 4.2.7.3.1. Standard Query Operators in Query Syntax

```
var students = from s in studentList  
Standard Query Operators → where s.age > 20  
→ select s;
```

## Standard Query Operators in Query Syntax

#### 4.2.7.3.2. Standard Query Operators in Method Syntax

```
var students = studentList.Where(s => s.age > 20).ToList<Student>();
```

© TutorialsTeacher.com  
*Extension Methods*

## Standard Query Operators in Method Syntax

Standard query operators in query syntax is converted into extension methods at compile time. So both are same.

Standard Query Operators can be classified based on the functionality they provide. The following table lists all the classification of Standard Query Operators:

<b>Filtering</b>	Where, OfType
<b>Sorting</b>	OrderBy, OrderByDescending, ThenBy, ThenByDescending,

	Reverse
<b>Grouping</b>	GroupBy, ToLookup
<b>Join</b>	GroupJoin, Join
<b>Projection</b>	Select, SelectMany
<b>Aggregation</b>	Aggregate, Average, Count, LongCount, Max, Min, Sum
<b>Quantifiers</b>	All, Any, Contains
<b>Elements</b>	ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault
<b>Set</b>	Distinct, Except, Intersect, Union
<b>Partitioning</b>	Skip, SkipWhile, Take, TakeWhile
<b>Concatenation</b>	Concat
<b>Equality</b>	SequenceEqual
<b>Generation</b>	DefaultEmpty, Empty, Range, Repeat
<b>Conversion</b>	AsEnumerable, AsQueryable, Cast, ToArray, ToDictionary,ToList

Gli esempi di tutte le funzioni presenti nella tabella precedente sono reperibili qui:

<https://www.tutorialsteacher.com/linq/linq-standard-query-operators>

#### 4.2.7.4. Alcuni esempi di codice LINQ.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LINQgym02
{
    class Student
    {
        public int StudentID { get; set; }
        public String StudentName { get; set; }
        public int Age { get; set; }
        public double MediaVoti { get; set; }

        public override string ToString()
        {
            return String.Format($"[StudentID = {StudentID}, StudentName = {StudentName}, Age = {Age}, MediaVoti = {MediaVoti}]");
        }
    }

    class Assenza
    {
```

```

        public int ID { get; set; }
        public DateTime Giorno { get; set; }
        public int StudentID { get; set; }
    }

    class Persona
    {
        public string Nome { get; set; }
        public int Eta { get; set; }

        public override string ToString()
        {
            return String.Format($"[Nome = {Nome}, Età = {Eta}]");
        }
    }

    class Program
    {
        //stiamo definendo un tipo di puntatore a funzione
        delegate bool CondizioneRicerca(Student s);

        public static void AzioneSuElemento(Student s)
        {
            Console.WriteLine(s);
        }

        //metodo statico
        public static bool VerificaCondizione(Student s)
        {
            return s.Age >= 18 && s.Age <= 25;
        }

        static void Main(string[] args)
        {

            Student[] studentArray1 = {
                new Student() { StudentID = 1, StudentName = "John", Age = 18, MediaVoti=6.5} ,
                new Student() { StudentID = 2, StudentName = "Steve", Age = 21, MediaVoti=8} ,
                new Student() { StudentID = 3, StudentName = "Bill", Age = 25, MediaVoti= 7.4 } ,
                new Student() { StudentID = 4, StudentName = "Ram" , Age = 20, MediaVoti = 10 } ,
                new Student() { StudentID = 5, StudentName = "Ron" , Age = 31, MediaVoti = 9 } ,
                new Student() { StudentID = 6, StudentName = "Chris", Age = 17, MediaVoti = 8.4 } ,
                new Student() { StudentID = 7, StudentName = "Rob",Age = 19 , MediaVoti=7.7} ,
                new Student() { StudentID = 8, StudentName = "Robert",Age = 22, MediaVoti=8.1 } ,
                new Student() { StudentID = 9, StudentName = "Alexander",Age = 18, MediaVoti=4 } ,
                new Student() { StudentID = 10, StudentName = "John", Age = 18 , MediaVoti=6} ,
                new Student() { StudentID = 11, StudentName = "John", Age = 21 , MediaVoti=8.5} ,
            };
        }
    }
}

```

```

new Student() { StudentID = 12, StudentName = "Bill", Age = 25, MediaVoti= 7 } ,
new Student() { StudentID = 13, StudentName = "Ram" , Age = 20, MediaVoti = 9 } ,
new Student() { StudentID = 14, StudentName = "Ron" , Age = 31, MediaVoti = 9.5 } ,
new Student() { StudentID = 15, StudentName = "Chris", Age = 17, MediaVoti = 8 } ,
new Student() { StudentID = 16, StudentName = "Rob2",Age = 19 , MediaVoti=7} ,
new Student() { StudentID = 17, StudentName = "Robert2",Age = 22, MediaVoti=8 } ,
new Student() { StudentID = 18, StudentName = "Alexander2",Age = 18, MediaVoti=9 } ,
};

List<Student> studentLinst1 = studentArray1.ToList();

//Studiamo la clausola Where

//definiamo delle condizioni di ricerca
//primo modo: uso di Func con lambda
Func<Student, bool> condizioneDiRicerca = s => s.Age >= 18 && s.Age <= 25;
//secondo modo: uso di un delegato implementato attraverso lambda
CondizioneRicerca condizioneDiRicerca2 = s => s.Age >= 18 && s.Age <= 25;
//terzo modo: uso di un delegato che punta a un metodo precedentemente definito
CondizioneRicerca condizioneDiRicerca3 = VerificaCondizione;
//quarto modo: usiamo direttamente la lambda - il più comodo

Student[] studentResultArray;
List<Student> studentResultList;

//utilizzo dei LINQ extension methods

//trovare tutti gli studenti che hanno età compresa tra 18 e 25 anni, caso dell'array
studentResultArray = studentArray1.Where(s => s.Age>=18 && s.Age<=25).ToArray();
//trovare tutti gli studenti che hanno età compresa tra 18 e 25 anni, caso della lista
studentResultList = studentLinst1.Where(s => s.Age >= 18 && s.Age <= 25).ToList();
//uso di delegati o di metodi
//https://stackoverflow.com/questions/1906787/cast-delegate-to-func-in-c-sharp

studentResultArray = studentArray1.Where(new Func<Student,bool>(condizioneDiRicerca2)).ToArray();
studentResultList = studentLinst1.Where(condizioneDiRicerca).ToList();
//oppure
studentResultArray = studentArray1.Where(new Func<Student,bool>(condizioneDiRicerca3)).ToArray();
studentResultList = studentLinst1.Where(VerificaCondizione).ToList();

Console.WriteLine("stampa su Array con Action");
Array.ForEach(studentResultArray, AzioneSuElemento);

Console.WriteLine("stampa su Lista con Action");
studentResultList.
    ForEach(s => Console.WriteLine(s.StudentName + " age = " + s.Age));

//USO DELLA VERSIONE CON INDICE DELLA WHERE

//il metodo Where ha anche una versione con l'indice della collection

```

```

//in questo esempio prendiamo solo quelli che verificano la condizione sull'età e hanno indice pari
Console.WriteLine("selezioniamo solo quelli che verificano la condizione e hanno indice pari");

studentResultArray = studentArray1.Where(
    (s, i) => (s.Age >= 18 && s.Age <= 25) && i % 2==0 ).ToArray();
Console.WriteLine("stampa su array");
Array.ForEach(studentResultArray, s => Console.WriteLine(s.StudentName + " age = " + s.Age));

studentResultList=studentLinst1.Where(
    (s, i) => (s.Age >= 18 && s.Age <= 25) && i % 2 == 0).ToList();
Console.WriteLine("stampa su list");
studentResultList.
    ForEach(s => Console.WriteLine(s.StudentName + " age = " + s.Age));

//E' possibile anche far applicare più volte la where per ottenere filtri multipli
Console.WriteLine("doppia where: quelli che verificano la condizione e che hanno ID>3");
studentResultList = studentLinst1.
    Where(s => s.Age >= 18 && s.Age <= 25).
    Where(s => s.StudentID > 3).ToList();
studentResultList.
    ForEach(s => Console.WriteLine(s.StudentName + " age = " + s.Age+" ID = "+s.StudentID));

//utilizzo di LINQ query

studentResultArray = (from s in studentArray1
                      where s.Age >= 18 && s.Age <= 25
                      select s).ToArray();

studentResultList = (from s in studentResultList
                      where s.Age >= 18 && s.Age <= 25
                      select s).ToList();
Console.WriteLine("Stampa del risultato con LINQ query");
Array.ForEach(studentResultArray, s => Console.WriteLine(s.StudentName + " age = " + s.Age));
studentResultList.
    ForEach(s => Console.WriteLine(s.StudentName + " age = " + s.Age));

//foreach(var elem in studentResult)
//{
//    Console.WriteLine(elem);
//}
//Console.WriteLine(studentLinst1);

//Studiamo la clausola OfType

//nel caso di collection di tipo diverso è possibile trarre vantaggio dal metodo OfType
IList mixedList = new ArrayList();
mixedList.Add(5);
mixedList.Add("numero uno");
mixedList.Add(true);

```

```

mixedList.Add("numero due");
mixedList.Add(new Student() { StudentID = 10, Age = 30, StudentName = "Roberto" });
IList mixedListResult = mixedList.OfType<string>().ToList();
IList mixedListResult2 =
    (from s in mixedList.OfType<Student>()
     where s.Age > 20
     select s).
    ToList();
IList mixedListResult2 =
    mixedList.OfType<Student>().Where(s => s.Age > 20).ToList();

Console.WriteLine("\nStampa del risultato con OfType method");
(mixedListResult as List<string>).ForEach(s => Console.WriteLine(s));
(mixedListResult2 as List<Student>).ForEach(s => Console.WriteLine(s));

//Studiamo la clausola OrderBy

//ordiniamo una lista di elementi
Console.WriteLine("\nOrdiniamo gli elementi di una lista con la clausola OrderBy");
Console.WriteLine("\nOrdiniamo in base all'età - LINQ method");
//per invertire l'ordine esiste anche OrderByDescending
studentResultArray = studentArray1.OrderBy(s => s.Age).ToArray();
Console.WriteLine("stampa su array");
Array.ForEach(studentResultArray, s => Console.WriteLine(s.StudentName + " age = " + s.Age));

studentResultList = studentLinst1.OrderBy(s => s.Age ).ToList();
Console.WriteLine("\nstampa su list");
studentResultList.
    ForEach(s => Console.WriteLine(s.StudentName + " age = " + s.Age));

Console.WriteLine("\nOrdiniamo in base all'età - LINQ query");
studentResultArray = (from s in studentArray1
                      orderby s.Age
                      select s).ToArray();
Console.WriteLine("stampa su array");
Array.ForEach(studentResultArray, s => Console.WriteLine(s.StudentName + " age = " + s.Age));

studentResultList = (from s in studentLinst1
                      orderby s.Age
                      select s).ToList();
Console.WriteLine("\nstampa su list");
studentResultList.
    ForEach(s => Console.WriteLine(s.StudentName + " age = " + s.Age));

//ordinamenti multipli
Console.WriteLine("\nOrdinamenti multipli - LINQ method");
studentResultArray = studentArray1.OrderBy(s => s.Age).ThenBy(s => s.StudentName).ToArray();
Console.WriteLine("stampa su array");
Array.ForEach(studentResultArray, s => Console.WriteLine(s.StudentName + " age = " + s.Age));

```

```

studentResultList = studentLinst1.OrderBy(s => s.Age).ThenBy(s => s.StudentName).ToList();
Console.WriteLine("\nstampa su list");
studentResultList.
    ForEach(s => Console.WriteLine(s.StudentName + " age = " + s.Age));

Console.WriteLine("\nOrdinamenti multipli - LINQ query");
studentResultArray = (from s in studentArray1
                      orderby s.Age, s.StudentName descending
                      select s).ToArray();
Console.WriteLine("stampa su array");
Array.ForEach(studentResultArray, s => Console.WriteLine(s.StudentName + " age = " + s.Age));

studentResultList = (from s in studentLinst1
                      orderby s.Age, s.StudentName descending
                      select s).ToList();
Console.WriteLine("\nstampa su list");
studentResultList.
    ForEach(s => Console.WriteLine(s.StudentName + " age = " + s.Age));

//Studiamo la clausola select - proiettiamo gli elementi della sequenza in una nuova forma
Console.WriteLine("\nClausola select - LINQ method");
Console.WriteLine("\nUso di tipi anonimi");
//uso di tipi anonimi
Array.ForEach(
    studentArray1.
        Select(s => new {Nome = s.StudentName, Eta = s.Age }).
        ToArray(),
    Console.WriteLine);
studentLinst1.
    Select(s => new { Nome = s.StudentName, Eta = s.Age }).
    ToList().
    ForEach(p => Console.WriteLine(p.Nome));

//uso di tipi diversi - il risultato di una elaborazione LINQ diventa una lista di tipo diverso
Console.WriteLine("\nUso di tipi diversi");
Persona[] personaResultArray = studentArray1.
    Select(s => new Persona() { Nome = s.StudentName, Eta = s.Age }).ToArray();
Array.ForEach(personaResultArray, s => Console.WriteLine(s.Nome + " " + s.Eta));

List<Persona> personaResultList =
    studentLinst1.
        Select(s => new Persona() { Nome = s.StudentName, Eta = s.Age }).
        ToList();
personaResultList.ForEach(p => Console.WriteLine(p));

Console.WriteLine("\nClausola select - LINQ query");
var studentNames = (from s in studentArray1
                      select new { Nome = s.StudentName }).ToArray();
Array.ForEach(studentNames, s => Console.WriteLine(s));
var personNamesList = (from s in studentLinst1

```

```

        select new Persona() { Nome = s.StudentName, Eta = s.Age }).ToList();
personNamesList.ForEach(p => Console.WriteLine(p.Nome + " " + p.Eta));

//Group By
//IEnumerable<IGrouping< int, Student>>
var groupedResult = studentLinst1.GroupBy(s => s.Age);

foreach(var group in groupedResult)
{
    Console.WriteLine("Group key(Age) = {0}", group.Key);
    foreach(var student in group)
    {
        Console.WriteLine("Student = {0}", student);
    }
    //calcoliamo una funzione di gruppo: min, max, avg, count
    // funzione count: quanti studenti con la stessa età
    Console.WriteLine("Numero studenti con la stessa età nel gruppo = {0}", group.Count());
    Console.WriteLine("Valore medio dei voti = {0}", group.Average(s => s.MediaVoti));
    Console.WriteLine("Voto massimo nel gruppo = {0}", group.Max(s => s.MediaVoti));
    Console.WriteLine("Voto minimo nel gruppo = {0}", group.Min(s => s.MediaVoti));
    //C# implementa anche il metodo ToLookup che fa la stessa cosa di GroupBy ma la
    //differenza sta nel fatto che con grandi basi di dati ToLookup carica tutto il risultato in
memoria

    //GroupBy carica il risultato associato a una chiave quando serve
    //https://stackoverflow.com/questions/10215428/why-are-tolookup-and-groupby-different
    //https://stackoverflow.com/a/10215531

}

Console.WriteLine("STAMPA RAGGRUPPAMENTO PER NOME");
var groupedResult2 = studentLinst1.GroupBy(s => s.StudentName);

foreach (var group in groupedResult2)
{
    Console.WriteLine("Chiave di raggruppamento (Nome) = " + group.Key);
    foreach(var student in group)
    {
        Console.WriteLine(student);
    }
    Console.WriteLine("Numero studenti omonimi: " + group.Count());
    Console.WriteLine("Voto medio degli omonimi: " + group.Average(s => s.MediaVoti));

}

//intersezione tra due collection - Join
//creiamo un elenco di assenze di studenti
List<Assenza> assenzeList1 = new List<Assenza>
{
    new Assenza(){ID = 1, Giorno = DateTime.Today, StudentID = 1 },
    new Assenza(){ID = 2, Giorno = DateTime.Today.AddDays(-1) ,StudentID = 1 },
    new Assenza(){ID = 3, Giorno = DateTime.Today.AddDays(-3), StudentID = 1 },
}

```

```
        new Assenza(){ID = 4, Giorno = new DateTime(2020,11,30), StudentID = 2 },
        new Assenza(){ID = 5, Giorno = new DateTime(2020,11,8), StudentID = 3 }

    };

    //vogliamo riportare il nome dello studente e le date delle sue assenze
    //facciamo una join tra la lista degli studenti e la lista delle assenze degli studenti e poi facciamo la proiezione del risultato su un nuovo oggetto

    var innerJoinStudentiAssenze = studentList1.Join(assenzeList1,
        s => s.StudentID,
        a => a.StudentID,
        (s, a) => new { ID= s.StudentID, Nome = s.StudentName, GiornoAssenza = a.Giorno});

    foreach(var obj in innerJoinStudentiAssenze)
    {
        Console.WriteLine($"ID = {obj.ID}, Nome = {obj.Nome}, GiornoAssenza = {obj.GiornoAssenza.ToStringDateString()}");
    }

    Console.ReadLine();

}

}

}
```

#### **4.2.7.5. Esercitazione – LINQ al Museo**

Creare i seguenti POCO:

**Artista (Id, Nome, Cognome, Nazionalità)**

**Opera (Id, Titolo, Quotazione, FkArtistaId)**

## Personaggio (Id, Nome, FkOperaid)

Creare per ciascun POCO una collection (lista) di tali oggetti:

**artisti** è una lista di oggetti di tipo **Artista**;

**opere** è una lista di oggetti di tipo **Opera**;

**personaggi** è una lista di oggetti di tipo **Personaggio**;

Nota: le property che iniziano con Fk indicano una foreign key ossia una property che “punta” ad una property di un altro oggetto. Ad esempio, i valori di FkOperId devono corrispondere a valori dell’Id nella collection delle opere. In altri termini una foreign key è una property i cui valori devono corrispondere a valori già presenti nella collection a cui puntano.

Effettuare le seguenti query:

- 1) Stampare le opere di un dato autore (ad esempio Picasso)
  - 2) Riportare per ogni nazionalità (raggruppare per nazionalità) gli artisti
  - 3) Contare quanti sono gli artisti per ogni nazionalità (raggruppare per nazionalità e contare)
  - 4) Trovare la quotazione media, minima e massima delle opere di Picasso

- 5) Trovare la quotazione media, minima e massima di ogni artista
- 6) Raggruppare le opere in base alla nazionalità e in base al cognome dell'artista  
(Raggruppamento in base a più proprietà)
- 7) Trovare gli artisti di cui sono presenti almeno 2 opere
- 8) Trovare le opere che hanno personaggi
- 9) Trovare le opere che non hanno personaggi
- 10) Trovare l'opera con il maggior numero di personaggi

#### 4.2.7.5.1. Svolgimento

//File: Artista.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LinqAlMuseo
{
    public class Artista
    {
        public int Id { get; set; }
        public string Nome { get; set; } = null!;
        public string Cognome { get; set; } = null!;
        public string? Nazionalita { get; set; }

        public override string ToString()
        {
            return string.Format($"[ID = {Id}, Nome = {Nome}, Cognome = {Cognome},
Nazionalità = {Nazionalita}]");
        }
    }
}
```

//File: Opera.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LinqAlMuseo
{
    public class Opera
    {
        public int Id { get; set; }
        public string Titolo { get; set; } = null!;

        public decimal Quotazione { get; set; }

        public int FkArtista { get; set; }
        public override string ToString()
        {
            return String.Format($"[ID = {Id}, Titolo = {Titolo}, Quotazione =
{Quotazione}, FkArtista = {FkArtista}]");
        }
    }
}
```

```

        }
    }

//File: Personaggio.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LinqAlMuseo
{
    public class Personaggio
    {
        public int Id { get; set; }
        public string Nome { get; set; } = null!;
        public int FkOperaId { get; set; }
        public override string ToString()
        {
            return string.Format($"[ID = {Id}, Nome = {Nome}, FkOperaId = {FkOperaId}]");
        }
    }
}

```

//File: Program.cs

```

//Esercizio: LINQ al museo

using LinqAlMuseo;
using System.Globalization;
using System.Text;

//creazione delle collection
//si parte da quelle che non puntano a nulla, ossia quelle che non hanno chiavi esterne

IList<Artista> artisti = new List<Artista>()
{
    new Artista(){Id=1, Cognome="Picasso", Nome="Pablo", Nazionalita="Spagna"},
    new Artista(){Id=2, Cognome="Dali", Nome="Salvator", Nazionalita="Spagna"},
    new Artista(){Id=3, Cognome="De Chirico", Nome="Giorgio", Nazionalita="Italia"},
    new Artista(){Id=4, Cognome="Guttuso", Nome="Renato", Nazionalita="Italia"}
};

//poi le collection che hanno Fk
IList<Opera> opere = new List<Opera>() {
    new Opera(){Id=1, Titolo="Guernica", Quotazione=50000000.00m, FkArtista=1}, //opera di Picasso
    new Opera(){Id=2, Titolo="I tre musici", Quotazione=15000000.00m, FkArtista=1}, //opera di Picasso
    new Opera(){Id=3, Titolo="Les demoiselles d'Avignon", Quotazione=12000000.00m, FkArtista=1}, //opera di Picasso
}

```

```

        new Opera(){Id=4, Titolo="La persistenza della memoria",
Quotazione=16000000.00m, FkArtista=2},//opera di Dalì
        new Opera(){Id=5, Titolo="Metamorfosi di Narciso", Quotazione=8000000.00m,
FkArtista=2},//opera di Dalì
        new Opera(){Id=6, Titolo="Le Muse inquietanti",
Quotazione=22000000.00m, FkArtista=3},//opera di De Chirico
    };

IList<Personaggio> personaggi = new List<Personaggio>() {
    new Personaggio(){Id=1, Nome="Uomo morente", FkOperaId=1},//un personaggio di Guernica
    new Personaggio(){Id=2, Nome="Un musicante", FkOperaId=2},
    new Personaggio(){Id=3, Nome="una ragazza di Avignone", FkOperaId=3},
    new Personaggio(){Id=4, Nome="una seconda ragazza di Avignone", FkOperaId=3},
    new Personaggio(){Id=5, Nome="Narciso", FkOperaId=5},
    new Personaggio(){Id=6, Nome="Una musa metafisica", FkOperaId=6},
};

//impostiamo la console in modo che stampi correttamente il carattere dell'euro e che utilizzi le
impostazioni di cultura italiana
Console.OutputEncoding = Encoding.UTF8;
Thread.CurrentThread.CurrentCulture = new CultureInfo("it-IT");

//Le query da sviluppare sono:
//Effettuare le seguenti query:
//1) Stampare le opere di un dato autore (ad esempio Picasso)
//2) Riportare per ogni nazionalità (raggruppare per nazionalità) gli artisti
//3) Contare quanti sono gli artisti per ogni nazionalità (raggruppare per nazionalità e contare)
//4) Trovare la quotazione media, minima e massima delle opere di Picasso
//5) Trovare la quotazione media, minima e massima di ogni artista
//6) Raggruppare le opere in base alla nazionalità e in base al cognome dell'artista
(Raggruppamento in base a più proprietà)
//7) Trovare gli artisti di cui sono presenti almeno 2 opere
//8) Trovare le opere che hanno personaggi
//9) Trovare le opere che non hanno personaggi
//10) Trovare l'opera con il maggior numero di personaggi

//svolgimento delle query richieste
//1) Stampare le opere di un dato autore (ad esempio Picasso)
Console.WriteLine("**** 1) Stampare le opere di un dato autore (ad esempio Picasso)\n");
//facciamo prima il filtraggio con la Where e poi la join
var opereDiArtista = artisti.Where(a => a.Cognome == "Picasso").Join(opere,
    a => a.Id,
    o => o.FkArtista,
    (a, o) => o.Titolo);
opereDiArtista.ToList().ForEach(t => Console.WriteLine(t));
//altro metodo: facciamo prima la Join e poi il filtraggio con la Where sull'autore
Console.WriteLine();

```

```

var opereDiArtista2 = artisti.Join(opere,
    a => a.Id,
    o => o.FkArtista,
    (a, o) => new { a, o }).Where(t => t.a.Cognome == "Picasso");
opereDiArtista2.ToList().ForEach(t => Console.WriteLine(t.o.Titolo));
Console.WriteLine();
//altro modo:
//step n.1: calcoliamo l'id di Picasso
//step. n.2: calcoliamo le opere di quell'autore
var autore = artisti.Where(a => a.Cognome == "Picasso").FirstOrDefault();
if(autore != null)
{
    var opereDiArtista3 = opere.Where(o => o.FkArtista == autore.Id);
    opereDiArtista3.ToList().ForEach(t => Console.WriteLine(t.Titolo));
}

//2) Riportare per ogni nazionalità (raggruppare per nazionalità) gli artisti
Console.WriteLine("\n**** 2) Riportare per ogni nazionalità (raggruppare per nazionalità) gli
artisti\n");

//raggruppare gli artisti per nazionalità
var artistiPerNazionalità = artisti.GroupBy(a => a.Nazionalita);
foreach (var group in artistiPerNazionalità)
{
    Console.WriteLine($"Nazionalità: {group.Key}");
    foreach (var artista in group)
    {
        Console.WriteLine($"{artista.Nome} {artista.Cognome}");
    }
}

//3) Contare quanti sono gli artisti per ogni nazionalità (raggruppare per nazionalità e contare)
Console.WriteLine("\n**** 3) Contare quanti sono gli artisti per ogni nazionalità (raggruppare per
nazionalità e contare)\n");

foreach (var group in artistiPerNazionalità)
{
    Console.WriteLine($"Nazionalità: {group.Key} Numero artisti: {group.Count()}");
}

//4)Trovare la quotazione media, minima e massima delle opere di Picasso
Console.WriteLine("\n**** 4) Trovare la quotazione media, minima e massima delle opere di Picasso\n");

//troviamo le opere di Picasso
var opereDiPicasso = artisti.Where(a => a.Cognome == "Picasso")
    .Join(opere,
        a => a.Id,
        o => o.FkArtista,

```

```

(a, o) => o).ToList();
//troviamo le quotazioni
var quotazioneMinima= opereDiPicasso.Min(o => o.Quotazione);
var quotazioneMedia = opereDiPicasso.Average(o => o.Quotazione);
var quotazioneMassima = opereDiPicasso.Max(o => o.Quotazione);
//stampiamo il risultato
Console.WriteLine($"Quotazione minima = {quotazioneMinima}, " +
    $"quotazione media = {quotazioneMedia:F2}, quotazione massima = {quotazioneMassima}");

//5)Trovare la quotazione media, minima e massima di ogni artista
Console.WriteLine("\n**** 5) Trovare la quotazione media, minima e massima di ogni artista\n");

//raggruppiamo per artista (usando FkArtista) e poi su ogni gruppo di opere calcoliamo le funzioni di
gruppo
var operePerArtista = opere.GroupBy(o => o.FkArtista);

foreach (var group in operePerArtista)
{
    Console.Write($"Id artista = {group.Key} ");
    //voglio conoscere i dettagli dell'artista di cui so l'id
    var artista = artisti.Where(a => a.Id== group.Key).FirstOrDefault();
    if (artista != null)
    {
        Console.WriteLine($"{artista.Nome} {artista.Cognome} ");
    }
    Console.WriteLine($"Quotazione minima = {group.Min(o => o.Quotazione):C2}; " +
        $" media = {group.Average(o => o.Quotazione):C2}; " +
        $" massima = {group.Max(o => o.Quotazione):C2}");
}

//stessa query - versione con inner join
//effettuiamo prima la join tra opere e artisti e poi il raggruppamento
var opereDiArtistaGroupBy = artisti.Join(opere,
    a => a.Id,
    o => o.FkArtista,
    (a, o) => new { a, o }).GroupBy(t => t.a.Id);

foreach (var group in opereDiArtistaGroupBy)
{
    Console.Write($"Id artista = {group.Key} |");
    var artistaOpera = group.FirstOrDefault();
    if (artistaOpera!=null)
    {
        Console.WriteLine($"Cognome = {artistaOpera.a.Cognome}");
    }
    Console.WriteLine($" | Quotazione media ={group.Average(t => t.o.Quotazione):C2} " +
        $" | Quotazione minima = {group.Min(t => t.o.Quotazione):C2} " +
        $" | Quotazione massima = {group.Max(t => t.o.Quotazione):C2} ");
}

```

```

//6)    Raggruppare le opere in base alla nazionalità e in base al cognome dell'artista
(Raggruppamento in base a più proprietà)
Console.WriteLine("\n**** 6) Raggruppare le opere in base alla nazionalità e in base al cognome
dell'artista (Raggruppamento in base a più proprietà)\n");
var opereDiArtistaGroupByMultiplo = artisti.Join(opere,
    a => a.Id,
    o => o.FkArtista,
    (a, o) => new { a, o }).GroupBy(t => new { t.a.Nazionalita, t.a.Cognome });

foreach (var group in opereDiArtistaGroupByMultiplo)
{
    //Console.WriteLine($"Chiave di raggruppamento = {group.Key}");
    //foreach (var item in group)
    //{
        //    Console.WriteLine($"Elemento = {item}");
    //}
}

Console.WriteLine($"{group.Key.Nazionalita} {group.Key.Cognome} ");
foreach (var item in group)
{
    Console.WriteLine($"\\tOpera = {item.o.Titolo}");
}
}

//7) Trovare gli artisti di cui sono presenti almeno 2 opere
Console.WriteLine("\n**** 7) Trovare gli artisti di cui sono presenti almeno 2 opere\n");
//intanto calcolo gli artisti di cui ho almeno un'opera
var artistiConAlmeno10opera = artisti.Join(opere,
    a => a.Id,
    o => o.FkArtista,
    (a, o) => a);
//per calcolare gli artisti di cui sono presenti almeno due opere procedo così:
//raggruppo gli artisti per FkArtista e successivamente filtro in base al conteggio degli
//elementi in ogni gruppo
Console.WriteLine("Artisti con almeno due opere");
var artistiConAlmeno20pere = opere.GroupBy(o => o.FkArtista)
    .Where(g => g.Count() >= 2).Join(artisti,
    g => g.Key,
    a => a.Id,
    (g, a) => a);
foreach (var artista in artistiConAlmeno20pere)
{
    Console.WriteLine(artista);
}

//altra variante - riporta gli artisti con il relativo numero di opere
opere.GroupBy(o => o.FkArtista).
Select(group => new { group.Key, NumeroOpere = group.Count() }).
Where(g => g.NumeroOpere >= 2).

```

```

Join(artisti,
a2 => a2.Key,
a => a.Id,
(a2, a) => new { ID = a.Id, NomeArtista = a.Cognome, a2.NumeroOpere }
).ToList().ForEach(s => Console.WriteLine(s));

//foreach (var group in artistiConAlmeno20opera)
//{
//    Console.WriteLine($"{group.Key}");
//}

//8)Trovare le opere che hanno personaggi
Console.WriteLine("\n**** 8) Trovare le opere che hanno personaggi\n");
//le opere con personaggi (è una semplice join)
var opereConPersonaggi = opere.Join(personaggi,
    o => o.Id,
    p => p.FkOperaId,
    (o, p) => o);
Console.WriteLine("Opere con personaggi");
foreach (var opera in opereConPersonaggi)
{
    Console.WriteLine(opera);
}

//9)Trovare le opere che non hanno personaggi
Console.WriteLine("\n**** 9) Trovare le opere che non hanno personaggi\n");
//opere senza personaggi: dall'insieme delle opere prendo solo quelle che non sono contenute in
opereConPersonaggi
var opereSenzaPersonaggi = opere
    .Where(o => !opereConPersonaggi.Contains(o));

Console.WriteLine("Opere senza personaggi");
foreach (var opera in opereSenzaPersonaggi)
{
    Console.WriteLine(opera);
}

//10)Trovare le opere con il maggior numero di personaggi
Console.WriteLine("\n**** 10) Trovare le opere con il maggior numero di personaggi\n");
//primo step: calcolo il numero di personaggi per opera
var personaggiPerOpera = personaggi.
    GroupBy(p => p.FkOperaId).
    Select(group => new { IdOpera = group.Key, NumeroPersonaggi = group.Count() });
//secondo step: dobbiamo filtrare gli oggetti in modo da prendere solo quelli con il numero massimo di
personaggi
var numeroMassimoPersonaggi = personaggiPerOpera.Max(t => t.NumeroPersonaggi);
//terzo step: filtro i dati in modo da prendere solo gli oggetti con il numero massimo di personaggi
var opereConMaxNumeroPersonaggi = personaggiPerOpera
    .Where(t => t.NumeroPersonaggi == numeroMassimoPersonaggi)
    .Join(opere,

```

```

t => t.IdOpera,
o => o.Id,
(t, o) => new { id = o.Id, Titolo=o.Titolo, Personaggi=t.NumeroPersonaggi });
foreach (var item in opereConMaxNumeroPersonaggi)
{
    Console.WriteLine(item);
}

```

#### 4.2.7.6. Points to Remember

1. As name suggest, **Method Syntax** is like calling extension method.
2. LINQ **Method Syntax** aka Fluent syntax because it allows series of extension methods call.
3. Implicitly typed variable -var can be used to hold the result of the LINQ query.

## 5. SQLite

SQLite è un SQL database engine.

<https://www.sqlite.org/index.html>

<https://www.sqlite.org/datatype3.html>

<https://www.sqlitetutorial.net/>

### 5.1. SQLite and C#

Alcuni esempi di Applicazioni C# che utilizzano SQLite

[https://www.codeguru.com/csharp/.net/net\\_data/using-sqlite-in-a-c-application.html](https://www.codeguru.com/csharp/.net/net_data/using-sqlite-in-a-c-application.html)

<https://www.c-sharpcorner.com/UploadFile/ranjancse/net-co-introduction-to-microsoft-data-sqlite/>

<https://docs.microsoft.com/en-us/dotnet/standard/data/sqlite/?tabs=netcore-cli>

<https://docs.microsoft.com/en-us/dotnet/standard/data/sqlite/connection-strings>

<https://www.c-sharpcorner.com/article/how-to-use-sqlite-with-c-sharp/>

<https://sqlitebrowser.org/about/>

## 6. EF Core model

Link utili:

<https://docs.microsoft.com/en-us/ef/#pivot=efcore>

<https://www.learnentityframeworkcore.com/>

<https://www.learnentityframeworkcore5.com/>

<https://ef.readthedocs.io/en/staging/intro.html>

<https://docs.microsoft.com/en-us/ef/core/modeling/>

Working with Nullable Reference Types in EF Core: <https://learn.microsoft.com/en-us/ef/core/miscellaneous/nullable-reference-types>

<https://www.meziantou.net/entity-framework-core-specifying-data-type-length-and-precision.htm>

<https://ef.readthedocs.io/en/staging/platforms/index.html> (getting started)

L'utilizzo di SQLite presuppone la conoscenza del codice SQL che sarà trattato in dettaglio l'anno prossimo; tuttavia, è possibile utilizzare il database engine SQLite e in generale un modello dei dati ad oggetti quasi ignorando del tutto l'SQL standard. È, infatti, possibile utilizzare un ORM (Object Relational Mapper) che semplifica molto l'interfacciamento verso il database engine.

<https://learn.microsoft.com/en-us/ef/core/>

Entity Framework (EF) Core is a lightweight, extensible, [open source](#) and cross-platform version of the popular Entity Framework data access technology.

EF Core can serve as an object-relational mapper (O/RM), which:

- Enables .NET developers to work with a database using .NET objects.
- Eliminates the need for most of the data-access code that typically needs to be written.

EF Core supports many database engines, see [Database Providers](#) for details.

With EF Core, data access is performed using a model. A model is made up of entity classes and a context object that represents a session with the database. The context object allows querying and saving data. For more information, see [Creating a Model](#).

EF supports the following model development approaches:

- Generate a model from an existing database.
- Hand code a model to match the database.
- Once a model is created, use [EF Migrations](#) to create a database from the model. Migrations allow evolving the database as the model changes.

Un esempio di Modello:

```
using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;

namespace Intro;

public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        optionsBuilder.UseSqlServer(
            @"Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
```

```

    public int Rating { get; set; }
    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}

```

Una volta associato il modello al database è possibile effettuare delle interrogazioni sui dati (query) usando il linguaggio LINQ:

Instances of your entity classes are retrieved from the database using [Language Integrated Query \(LINQ\)](#). For more information, see [Querying Data](#).

```

using (var db = new BloggingContext())
{
    var blogs = db.Blogs
        .Where(b => b.Rating > 3)
        .OrderBy(b => b.Url)
        .ToList();
}

```

Il salvataggio, la cancellazione o la modifica dei dati nel database sono ottenuti mediante operazioni su oggetti di entità (classi di oggetti che si mappano sulle righe delle tabelle del database).

Data is created, deleted, and modified in the database using instances of your entity classes. See [Saving Data](#) to learn more.

```

using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Add(blog);
    db.SaveChanges();
}

```

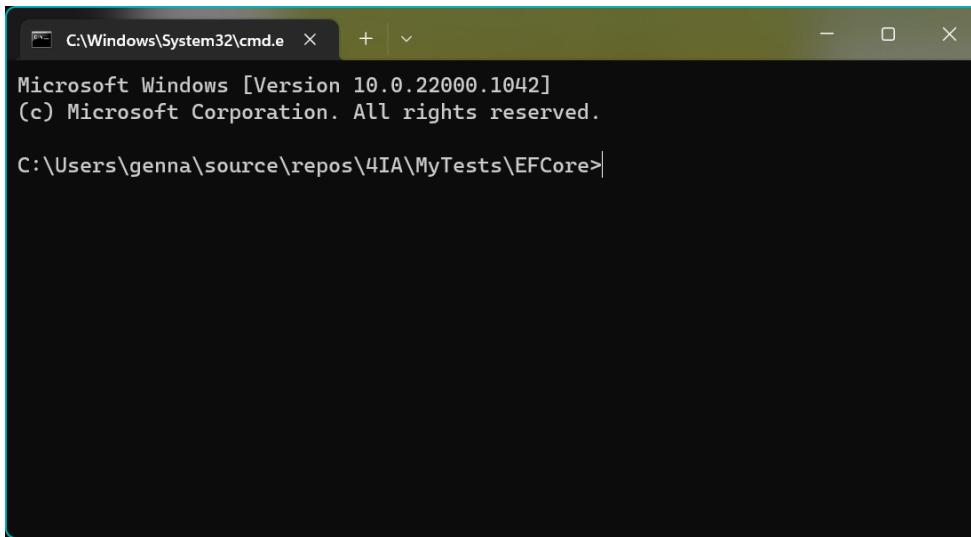
## 6.1. EF Core Model Tutorial - Console App (utilizzo comandi .NET SDK CLI)

<https://learn.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli> (esempio di EF Core con un'app console .Net Core)

In questo esempio verranno illustrati i passaggi per la creazione di un progetto di un'applicazione .NET Core Console, utilizzando i comandi di .NET SDK CLI (Command Line Interface). Nello specifico verranno richiamati i comandi di base di .NET CLI e i passaggi necessari per la configurazione del framework EF Core.

La documentazione per .NET CLI si trova qui: <https://learn.microsoft.com/en-us/dotnet/core/tools/>

Apriamo una CMD di Windows nella cartella nella quale vogliamo creare il nostro progetto. Supponendo di voler creare il nostro progetto all'interno della cartella EFCore:



```
C:\Windows\System32\cmd.e + | - □ X
Microsoft Windows [Version 10.0.22000.1042]
(c) Microsoft Corporation. All rights reserved.

C:\Users\genna\source\repos\4IA\MyTests\EFCore>
```

Usiamo il comando dotnet per creare il progetto.

Digitare:

`dotnet -h`

per vedere l'help in linea

digitare:

`dotnet --list-sdks`

per vedere le versioni delle SDK installate

digitare:

`dotnet --list-runtimes`

Per vedere l'elenco dei runtime disponibili sul proprio PC

Per creare un nuovo progetto il comando è:

`dotnet new`

Per verificare tutte le opzioni, digitare l'help specifico per il comando `dotnet new`

`dotnet new -h`

Dall'help notiamo che con il comando

`dotnet new -l`

è possibile vedere l'elenco dei template disponibili (in base ai pacchetti installati il numero cambia).

In particolare, per un progetto di tipo console il comando è:

`dotnet new console`

e con l'opzione `-o` definiamo la cartella dove creare il progetto

Per creare un nuovo progetto console nella cartella EFGetStarted il comando completo è dunque:

`dotnet new console -o EFGetStarted`

Questo comando crea un nuovo progetto nella cartella EFGetStarted:

```
C:\Windows\System32\cmd.e >dotnet new console -o EFGetStarted
The template "Console App" was created successfully.

Processing post-creation actions ...
Running 'dotnet restore' on C:\Users\genna\source\repos\4IA\MyTests\EFCore\EFGetStarted\EFGetStarted.csproj ...
  Determining projects to restore ...
    Restored C:\Users\genna\source\repos\4IA\MyTests\EFCore\EFGetStarted\EFGetStarted.csproj (in 77 ms).
Restore succeeded.
```

Andiamo nella cartella del progetto appena creato, digitando:

```
cd EFGetStarted
```

Con il comando `dir` vediamo il contenuto della cartella di progetto:

```
C:\Users\genna\source\repos\4IA\MyTests\EFCore\EFGetStarted>dir
 Volume in drive C is Windows-SSD
 Volume Serial Number is F278-752F

 Directory of C:\Users\genna\source\repos\4IA\MyTests\EFCore\EFGetStarted

09/28/2022  07:19 PM    <DIR>          .
09/28/2022  07:19 PM    <DIR>          ..
09/28/2022  07:19 PM           249 EFGetStarted.csproj
09/28/2022  07:19 PM    <DIR>          obj
09/28/2022  07:19 PM           105 Program.cs
                           2 File(s)        354 bytes
                           3 Dir(s)   460,797,272,064 bytes free

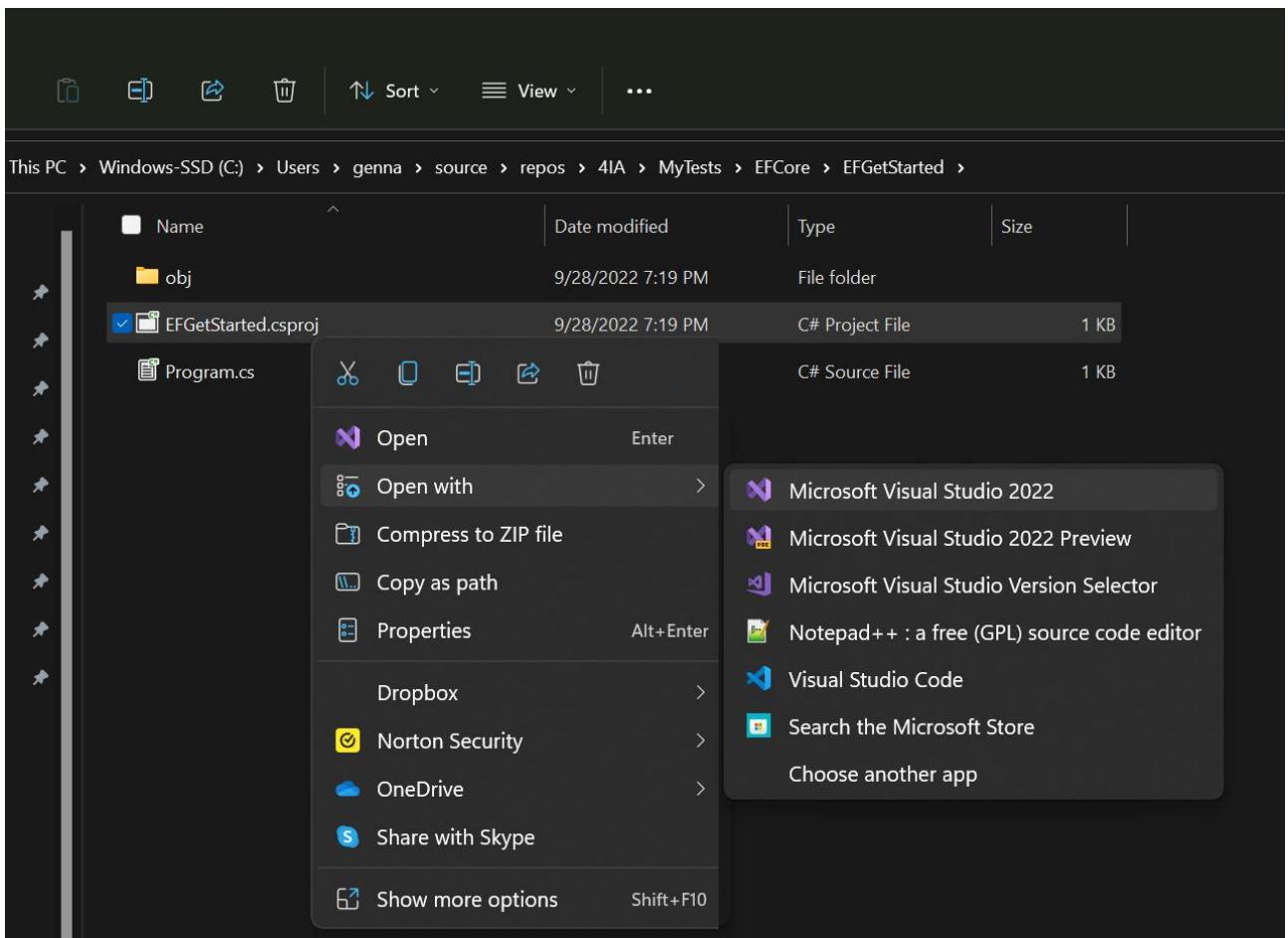
C:\Users\genna\source\repos\4IA\MyTests\EFCore\EFGetStarted>
```

A questo punto il progetto è stato creato e possiamo aprirlo con Visual Studio:

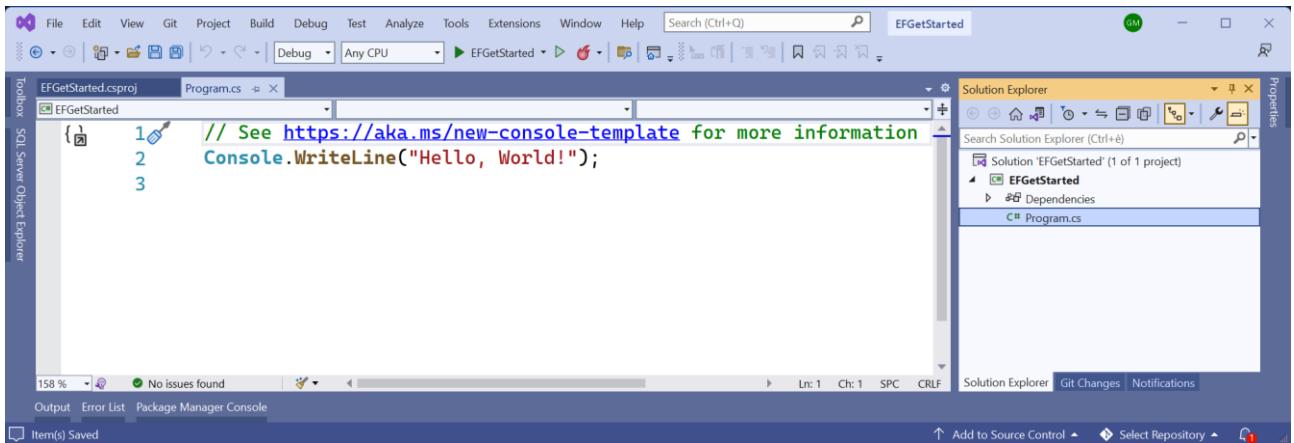
Dalla console digitiamo:

```
explorer .
```

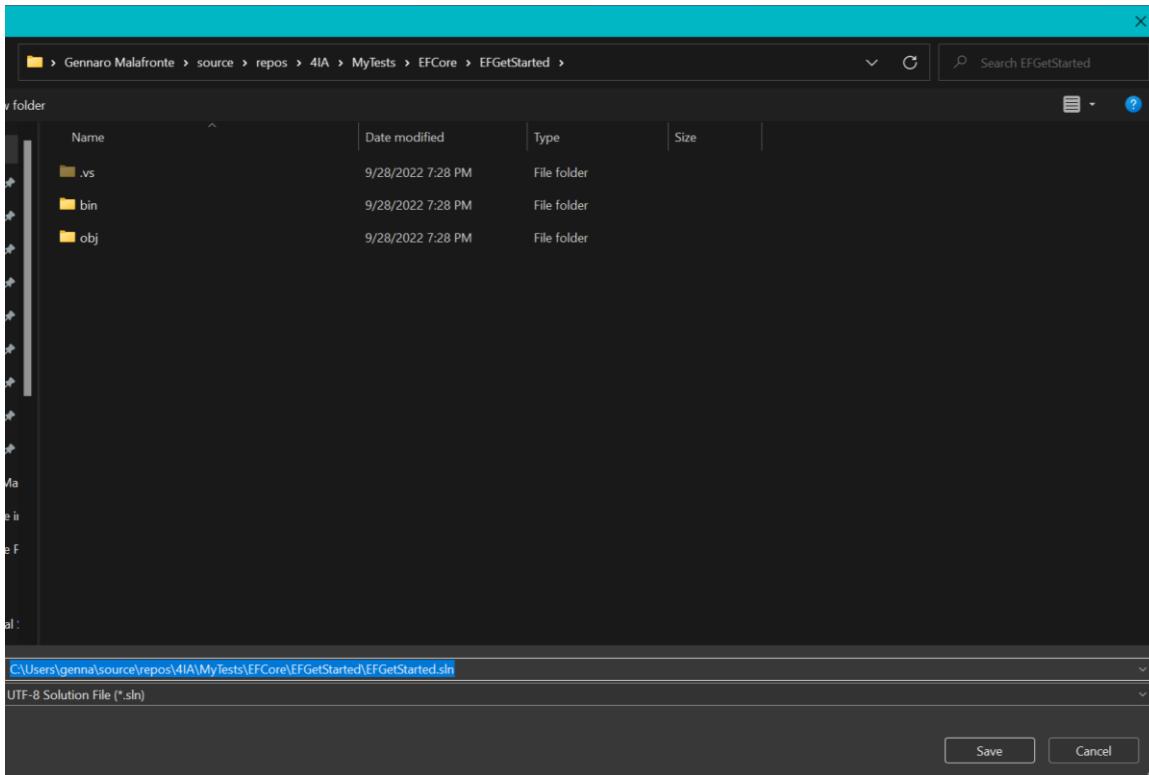
per far partire l'Esplora Risorse direttamente sulla cartella corrente. Facendo click destro sul file con estensione .csproj scegliamo di aprirlo con Visual Studio.



Da Visual Studio vediamo il progetto console esattamente come se fosse stato creato a partire dall'ambiente grafico di Visual Studio.



Effettuiamo un salvataggio di tutti i file, poiché Visual Studio aggiunge un suo file .sln (file per gestire la soluzione). Scegliamo di salvare il file di soluzione nella directory corrente.



Il progetto è pronto e potremmo mandarlo in esecuzione per vedere la classica scritta “Hello, World!” a console.

Per installare Entity Framework Core nel progetto digitiamo il comando seguente nella cmd che abbiamo aperto sulla cartella di progetto:

**dotnet add package Microsoft.EntityFrameworkCore.Sqlite**

**Nota importante:** se si esegue l’installazione dei pacchetti da scuola occorre impostare il proxy come variabile d’ambiente nella finestra cmd corrente, per fare in modo che il gestore dei pacchetti possa connettersi con il server NuGet remoto.

Per impostare le variabili d’ambiente del proxy nella finestra cmd corrente digitare le seguenti istruzioni:

```
set http_proxy=proxy:3128  
set https_proxy=proxy:3128
```

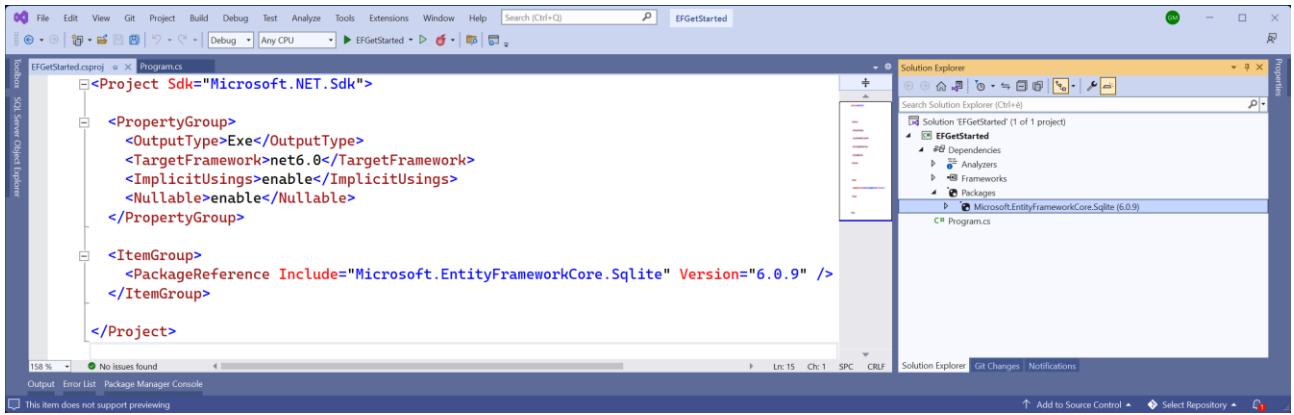
Opzionalmente si può verificare di aver impostato correttamente le variabili d’ambiente digitando nella console corrente i comandi:

```
echo %http_proxy%  
echo %https_proxy%
```

I comandi echo suddetti dovrebbero stampare la configurazione del proxy corrente.

Si osservi che le impostazioni delle variabili d’ambiente sono valide per la finestra cmd corrente. Quando questa viene chiusa, oppure si apre un’altra finestra, le variabili non sono più impostate.

L’installazione del pacchetto di Sqlite può essere osservata anche dai file di progetto di Visual Studio:



A questo punto creiamo il file Model.cs con al suo interno il seguente codice d'esempio, preso dal tutorial Microsoft, opportunamente adattato alle impostazioni di C# 10:

```
//file: Model.cs

using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EFGetStarted;

public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; } = null!;
    public DbSet<Post> Posts { get; set; } = null!;

    public string DbPath { get; }

    public BloggingContext()
    {
        var folder = Environment.SpecialFolder.LocalApplicationData;
        var path = Environment.GetFolderPath(folder);
        DbPath = System.IO.Path.Join(path, "blogging.db");
    }

    // The following configures EF to create a Sqlite database file in the
    // special "local" folder for your platform.
    protected override void OnConfiguring(DbContextOptionsBuilder options)
        => options.UseSqlite($"Data Source={DbPath}");
}

public class Blog
{
    public int BlogId { get; set; }
    public string? Url { get; set; }

    public List<Post> Posts { get; } = new();
}

public class Post
{
    public int PostId { get; set; }
    public string? Title { get; set; }
    public string? Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; } = null!;
}
```

```
}
```

Nel file Program.cs aggiungiamo il codice:

```
using EFGetStarted;  
  
using var db = new BloggingContext();
```

A questo punto possiamo procedere con la creazione del database per mezzo della migrazione. L'operazione di migrazione (migration) è l'operazione di creazione del database a partire da uno script.

Per poter effettuare la migrazione verifichiamo prima di tutto che sia installato il tool di dotnet che permette di gestire il framework EF Core.

Dalla console digitiamo il comando

```
dotnet tool install --global dotnet-ef
```

Se il tool è già installato viene mostrato a console un messaggio che avverte che il tool è già installato. L'installazione del tool dotnet-ef va fatta solo la prima volta.

Nel caso in cui venga mostrato a console un messaggio che avverte che il tool è in una versione obsoleta, si può installare l'aggiornamento con il comando

```
dotnet tool update --global dotnet-ef
```

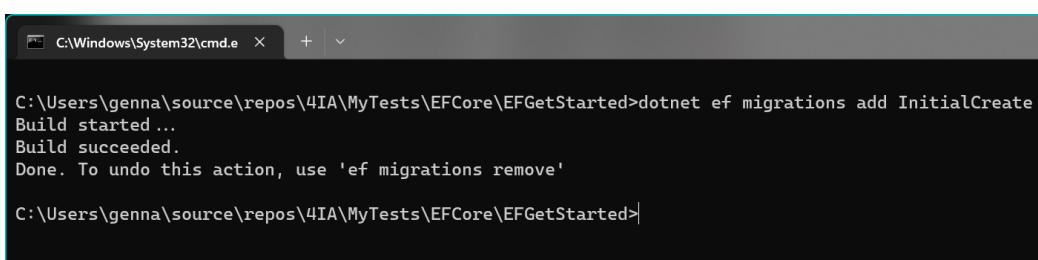
Per la creazione del database occorre installare il pacchetto Microsoft.EntityFrameworkCore.Design con l'istruzione:

```
dotnet add package Microsoft.EntityFrameworkCore.Design
```

Creiamo la prima migrazione con il comando:

```
dotnet ef migrations add InitialCreate
```

Si ottiene la risposta:



```
C:\Windows\System32\cmd.exe > + <  
C:\Users\genna\source\repos\4IA\MyTests\EFCore\EFGetStarted>dotnet ef migrations add InitialCreate  
Build started ...  
Build succeeded.  
Done. To undo this action, use 'ef migrations remove'  
C:\Users\genna\source\repos\4IA\MyTests\EFCore\EFGetStarted>
```

**Osservazione importante:** nel file Program.cs deve esserci la definizione dell'oggetto BloggingContext altrimenti la migrazione va in errore perché il tool non trova il DBContext:

//File: Program.cs

```
using EFGetStarted;  
  
using var db = new BloggingContext();
```

Si osservi ora il contenuto della cartella di progetto: si nota che è stata creata la cartella Migrations con all'interno due file che serviranno per la creazione fisica del database.

Per creare fisicamente il database eseguiamo il comando:

```
dotnet ef database update
```

Si ottiene la risposta:

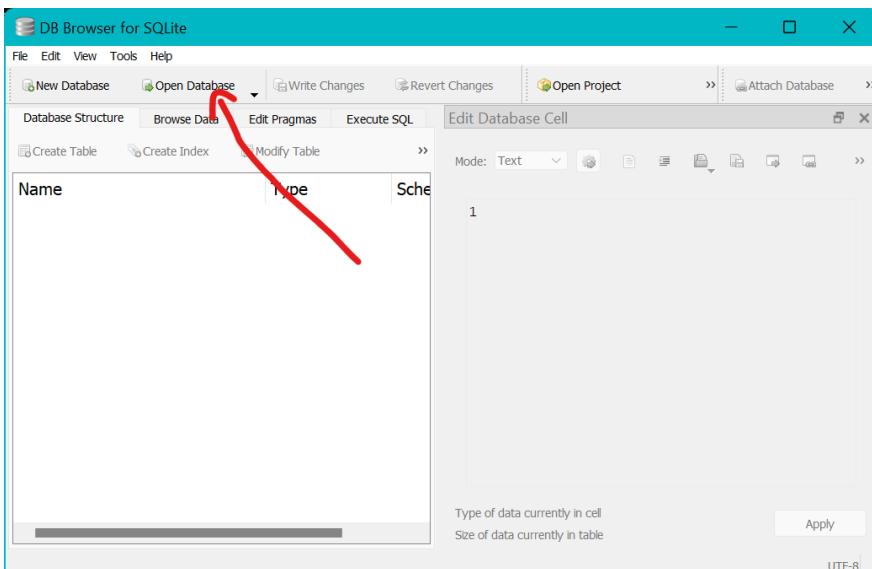
```
C:\Users\genna\source\repos\4IA\MyTests\EFCore\EFGetStarted>dotnet ef database update
Build started ...
Build succeeded.
Applying migration '20220929173426_InitialCreate'.
Done.

C:\Users\genna\source\repos\4IA\MyTests\EFCore\EFGetStarted>
```

Il codice inserito nel file Model.cs crea il database nel file:

C:\Users\NomeUtente\AppData\Local\blogging.db

Proviamo ad aprire il file del database di SQLite, utilizzando il programma DB Browser, cliccando sul pulsante “Open Database”:



Selezioniamo il file del database di SQLite e osserviamo le tabelle create:

Name	Type	Schema
Tables (4)		
Blogs		CREATE TABLE "Blogs" ( "BlogId" INTEGER NOT NULL CONSTRAINT "PK_Blogs" PRIMARY KEY AUTOINCREMENT, "Url" TEXT NULL ) "BlogId" INTEGER NOT NULL "Url" TEXT
Posts		CREATE TABLE "Posts" ( "PostId" INTEGER NOT NULL CONSTRAINT "PK_Posts" PRIMARY KEY AUTOINCREMENT, "Title" TEXT NULL, "Content" TEXT NULL, "BlogId" INTEGER NOT NULL "PostId" INTEGER NOT NULL "Title" TEXT "Content" TEXT "BlogId" INTEGER NOT NULL
__EFMigrationsHistory		CREATE TABLE "__EFMigrationsHistory" ( "MigrationId" TEXT NOT NULL CONSTRAINT "PK__EFMigrationsHistory" PRIMARY KEY, "ProductVersion" TEXT NOT NULL ) CREATE TABLE sqlite_sequence(name,seq)
Indices (1)		CREATE INDEX "IX_Posts_BlogId" ON "Posts" ("BlogId")

Cliccando sul pulsante “Browse Data” vediamo che non ci sono ancora dati nel database.

Per inserire dati nel database modifichiamo il file Program.cs con il seguente codice:

```
using EFGetStarted;

using var db = new BloggingContext();
// Note: This sample requires the database to be created before running.
Console.WriteLine($"Database path: {db.DbPath}.");

// Create
```

```

Console.WriteLine("Inserting a new blog");
db.Add(new Blog { Url = "http://blogs.msdn.com/adonet" });
db.SaveChanges();

// Read
Console.WriteLine("Querying for a blog");
var blog = db.Blogs
    .OrderBy(b => b.BlogId)
    .First();
Console.WriteLine($"blog: {blog.Url}");

```

Osserviamo il contenuto della tabella Blogs dal programma DB Browser:

The screenshot shows the DB Browser for SQLite interface. The title bar reads "DB Browser for SQLite - C:\Users\genna\AppData\Local\blogging.db". The main window displays a table named "Blogs". The table has two columns: "BlogId" and "Url". There is one row with values: BlogId is 1 and Url is "http://blogs.msdn.com/adonet". The bottom status bar indicates "1 - 1 of 1".

BlogId	Url
1	http://blogs.msdn.com/adonet

Proviamo a modificare il contenuto del blog con BlogId=1 e aggiungiamo un post associato a questo blog. Modifichiamo il codice del file Program.cs come segue:

```

using EFGetStarted;

using var db = new BloggingContext();
// Note: This sample requires the database to be created before running.
Console.WriteLine($"Database path: {db.DbPath}.");

// Create
//Console.WriteLine("Inserting a new blog");
//db.Add(new Blog { Url = "http://blogs.msdn.com/adonet" });
//db.SaveChanges();

//// Read
Console.WriteLine("Querying for a blog");
var blog = db.Blogs
    .OrderBy(b => b.BlogId)
    .First();
Console.WriteLine($"blog: {blog.Url}");

// Update
Console.WriteLine("Updating the blog and adding a post");
blog.Url = "https://devblogs.microsoft.com/dotnet";
blog.Posts.Add(
    new Post { Title = "Hello World", Content = "I wrote an app using EF Core!" });
db.SaveChanges();

```

Proviamo a cancellare un blog e tutti i post ad esso associati. Modifichiamo il codice del file Program.cs come segue:

```

using EFGetStarted;

using var db = new BloggingContext();
// Note: This sample requires the database to be created before running.
Console.WriteLine($"Database path: {db.DbPath}.");

// Create
//Console.WriteLine("Inserting a new blog");
//db.Add(new Blog { Url = "http://blogs.msdn.com/adonet" });
//db.SaveChanges();

//// Read
Console.WriteLine("Querying for a blog");
var blog = db.Blogs
    .OrderBy(b => b.BlogId)
    .First();
Console.WriteLine($"blog: {blog.Url}");

// Update
//Console.WriteLine("Updating the blog and adding a post");
//blog.Url = "https://devblogs.microsoft.com/dotnet";
//blog.Posts.Add(
//    new Post { Title = "Hello World", Content = "I wrote an app using EF Core!" });
//db.SaveChanges();

// Delete
Console.WriteLine("Delete the blog");
db.Remove(blog);
db.SaveChanges();

```

Possiamo osservare che il database è ritornato ad essere vuoto.

Nota: se provando a creare il database si ottiene un messaggio d'errore “SQLite Error 1: 'table "Blogs" already exists.'” significa che il database esiste già. Se si vuole ricrearlo da capo bisogna cancellare il vecchio file e lanciare nuovamente il comando:

`dotnet ef database update`

## 6.2.

### 6.3. EF Core Model Tutorial 2 - Console App (utilizzo comandi .NET SDK CLI)

1. Come nell'esempio precedente, creare in progetto console .NET Core con il seguente comando:

`dotnet new console -o GestioneFattureClienti`

2. Posizionarsi nella cartella del progetto:

`cd GestioneFattureClienti`

3. Aggiungere i pacchetti necessari per l'utilizzo di SQLite con EF Core:

`dotnet add package Microsoft.EntityFrameworkCore.Sqlite`

`dotnet add package Microsoft.EntityFrameworkCore.Design`

4. Aprire il progetto con Visual Studio 2022: l'apertura del progetto con Visual Studio 2022 può essere fatta in diversi modi:

- 1) Dalla CMD, posizionata sulla cartella di progetto, lanciando il comando `explorer .`

per aprire l’Esplora Risorse direttamente sulla cartella di progetto e poi selezionando il file con estensione .csproj e con il tasto destro del mouse scegliendo la voce “Apri con Visual Studio 2022”, come già mostrato in dettaglio nell’esempio precedente.

- 2) In alternativa (scorciatoia più veloce) dalla CMD, posizionata sulla cartella di progetto, lanciando il comando `start devenv fileprogetto.csproj`

Ad esempio, per lanciare Visual Studio 2022 in modo che apra direttamente il progetto GestioneFattureClienti:

```
start devenv GestioneFattureClienti.csproj
```

```
C:\Users\genna\source\repos\4IA\Info4IA2223LINQDatabase\EFCore>cd ..  
C:\Users\genna\source\repos\4IA\Info4IA2223LINQDatabase\EFCore>dir  
Volume in drive C is Windows-SSD  
Volume Serial Number is F278-752F  
  
Directory of C:\Users\genna\source\repos\4IA\Info4IA2223LINQDatabase\EFCore  
09/30/2022 12:20 PM <DIR> .  
09/28/2022 08:56 AM <DIR> ..  
09/30/2022 12:16 PM <DIR> EFGetStarted  
09/30/2022 05:34 PM <DIR> GestioneFattureClienti  
 0 File(s) 0 bytes  
 4 Dir(s) 465,020,260,352 bytes free  
  
C:\Users\genna\source\repos\4IA\Info4IA2223LINQDatabase\EFCore>cd GestioneFattureClienti  
C:\Users\genna\source\repos\4IA\Info4IA2223LINQDatabase\EFCore\GestioneFattureClienti>start devenv GestioneFattureClienti.csproj
```

5. Cliccare sul pulsante “Save All” di Visual Studio per salvare il file con estensione .sln (solution) aggiunto automaticamente da Visual Studio.
6. In Visual Studio creare la cartella Model e al suo interno aggiungere la classe `Cliente` nel file `Cliente.cs` con il seguente codice:

```
namespace GestioneFattureClienti.Model;  
public class Cliente  
{  
    public int ClienteId { get; set; }  
    public string RagioneSociale { get; set; } = null!;  
    public string PartitaIVA { get; set; } = null!;  
    public string? Citta { get; set; }  
    public string? Via { get; set; }  
    public string? Civico { get; set; }  
    public string? CAP { get; set; }  
    public List<Fattura> Fatture { get; } = new List<Fattura>();  
  
    public override string ToString()  
    {  
        return string.Format($"[{nameof(ClienteId)}= {ClienteId},  
{nameof(RagioneSociale)}= {RagioneSociale}, " +  
            $"{nameof(PartitaIVA)}= {PartitaIVA}, {nameof(Citta)}= {Citta},  
{nameof(Via)}= {Via}, {nameof(Civico)}= {Civico}, {nameof(CAP)}= {CAP} ]");  
    }  
}
```

7. Aggiungere la classe `Fattura` nel file `Fattura.cs` all’interno della cartella Model, con dentro il seguente codice:

```
namespace GestioneFattureClienti.Model;  
public class Fattura
```

```

{
    public int FatturaId { get; set; }
    public DateTime Data { get; set; }
    public decimal Importo { get; set; }
    public int ClienteId { get; set; }
    public Cliente Cliente { get; set; } = null!;

    public override string ToString()
    {
        return string.Format($"{nameof(FatturaId)} = {FatturaId}, {nameof(Data)} =
{Data.ToShortDateString()}, {nameof(Importo)} = {Importo}, {nameof(ClienteId)} =
{ClienteId}");
    }
}

```

8. Creare una cartella Data e al suo interno aggiungere la classe **FattureClientiContext** nel file FattureClientiContext.cs con il codice:

```

using GestioneFattureClienti.Model;
using Microsoft.EntityFrameworkCore;

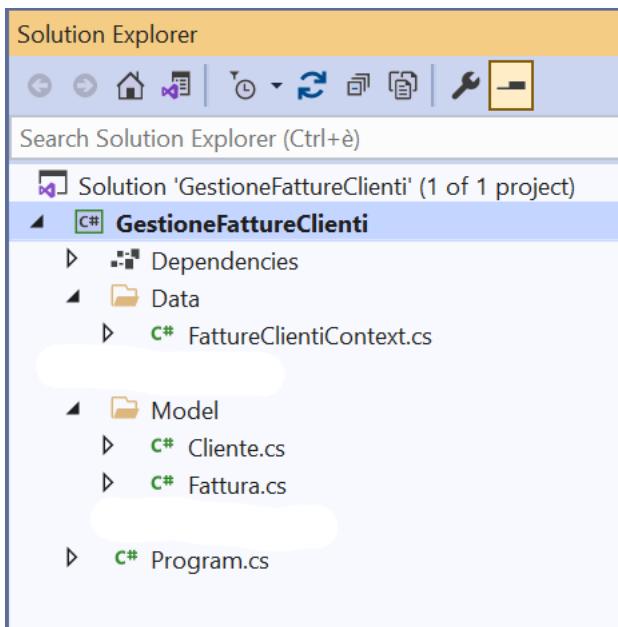
namespace GestioneFattureClienti.Data;

public class FattureClientiContext : DbContext
{
    public DbSet<Fattura> Fatture { get; set; } = null!;
    public DbSet<Cliente> Clienti { get; set; } = null!;
    public string DbPath { get; }

    public FattureClientiContext()
    {
        //https://www.hanselman.com/blog/how-do-i-find-which-directory-my-net-core-
        //console-application-was-started-in-or-is-running-from
        var folder = ApplicationContext.BaseDirectory;
        //La BaseDirectory restituisce la cartella dove si trova l'assembly (.dll e
        .exe del programma compilato)
        //il database, per comodità, è inserito nella cartella di progetto, dove si
        trova anche il file Program.cs
        var path = Path.Combine(folder, "../../fattureclienti.db");
        DbPath = path;
    }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        => optionsBuilder.UseSqlite($"Data Source={DbPath}");
}

```

A questo punto nel progetto dovrebbe essere presente una struttura di file e cartelle come quella riportata nella figura seguente:



9. Salvare tutti i file da Visual Studio; dalla CMD del progetto lanciare il comando per la creazione della migrazione:

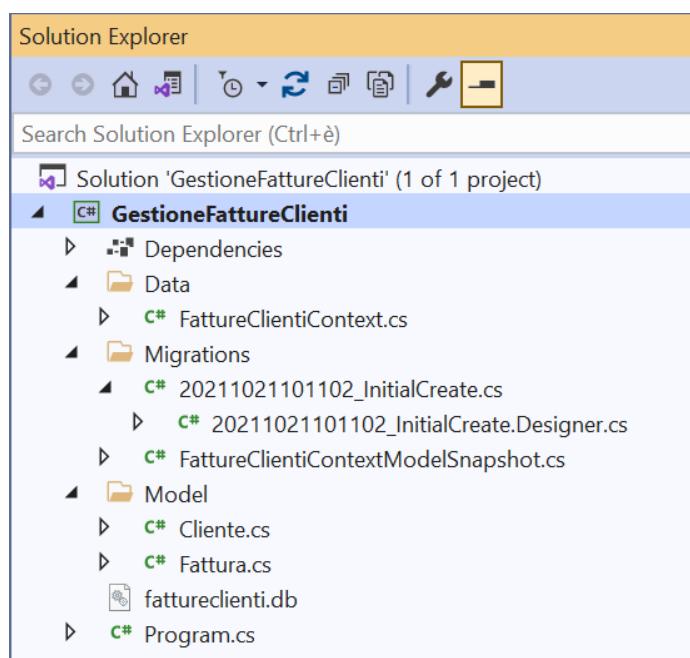
```
dotnet ef migrations add InitialCreate
```

10. Eseguire il comando per la creazione fisica del database:

```
dotnet ef database update
```

Una volta eseguita la migration e aggiornato il database, dovrebbe essere presente un file di database di SQLite fattureclienti.db con le tabelle corrispondenti agli oggetti Fatture e Clienti (di tipo DBSet<Fattura> e DBSet<Cliente> rispettivamente).

Dopo aver eseguito la migration e l'aggiornamento del database, la struttura dei file e delle cartelle dovrebbe essere come quella della figura seguente:



Scriviamo il codice nella classe Program per testare le funzionalità di EF Core con SQLite:

//File Program.cs

```
using GestioneFattureClienti.Data;
using GestioneFattureClienti;
using System.Text;
using GestioneFattureClienti.Model;

bool creazione = false;
bool lettura = true;
bool modifica = false;
bool cancellazione = false;
List<Cliente>? listaClienti;
List<Fattura>? listaFatture;
Console.OutputEncoding = Encoding.UTF8;
Thread.CurrentThread.CurrentCulture = System.Globalization.CultureInfo.GetCultureInfo("it-IT");

if (creazione)
{
    //Creazione di Clienti - gli id vengono generati automaticamente come campi autoincremento quando si effettua l'inserimento, tuttavia
    //è bene inserire esplicitamente l'id degli oggetti quando si procede all'inserimento massivo gli elementi mediante un foreach perché
    //EF core potrebbe inserire nel database gli oggetti in un ordine diverso rispetto a quello del foreach
    // https://stackoverflow.com/a/54692592
    // https://stackoverflow.com/questions/11521057/insertion-order-of-multiple-records-in-entity-framework/
    listaClienti = new List<Cliente>()
    {
        new Cliente(){ClienteId=1, RagioneSociale= "Cliente 1", PartitaIVA= "111111111111",
        Citta = "Napoli", Via="Via dei Mille", Civico= "23", CAP="80100"}, 
        new Cliente(){ClienteId=2, RagioneSociale= "Cliente 2", PartitaIVA= "111111111112",
        Citta = "Roma", Via="Via dei Fori Imperiali", Civico= "1", CAP="00100"}, 
        new Cliente(){ClienteId=3, RagioneSociale= "Cliente 3", PartitaIVA= "111111111113",
        Citta = "Firenze", Via="Via Raffaello", Civico= "10", CAP="50100"};
    };

    //Creazione delle Fatture -
    listaFatture = new List<Fattura>()
    {
        new Fattura(){FatturaId=1, Data= DateTime.Now.Date, Importo = 1200.45m, ClienteId = 1},
        new Fattura(){FatturaId=2, Data= DateTime.Now.AddDays(-5).Date, Importo = 3200.65m, ClienteId = 1},
        new Fattura(){FatturaId=3, Data= new DateTime(2019,10,20).Date, Importo = 5200.45m, ClienteId = 1},
        new Fattura(){FatturaId=4, Data= DateTime.Now.Date, Importo = 5200.45m, ClienteId = 2},
        new Fattura(){FatturaId=5, Data= new DateTime(2019,08,20).Date, Importo = 7200.45m, ClienteId = 2};
    };

    //inseriamo i dati nel database
    using var db = new FattureClientiContext();
    Console.WriteLine("Inseriamo i clienti nel database");
    listaClienti?.ForEach(c => db.Add(c));
    db.SaveChanges();
    Console.WriteLine("Inseriamo le fatture nel database");
    listaFatture?.ForEach(f => db.Add(f));
    db.SaveChanges();
}

if (lettura)
{
    //recuperiamo i dati dal database
    using var db = new FattureClientiContext();

    Console.WriteLine("Recuperiamo i dati dal database - senza alcuna elaborazione");
    listaClienti = db.Clienti.ToList();
    listaFatture = db.Fatture.ToList();
    Console.WriteLine("Stampa dei clienti");
    listaClienti.ForEach(c => Console.WriteLine(c));
    Console.WriteLine("Stampa delle fatture");
    listaFatture.ForEach(f => Console.WriteLine(f));
}
```

```

Console.WriteLine("Recuperiamo i dati dal database - uso di WHERE");
Console.WriteLine("Recuperiamo i dati dal database - trovare le fatture fatte da almeno tre
giorni");
    db.Fatture.Where(f => f.Data < DateTime.Now.AddDays(-2)).ToList().ForEach(f =>
Console.WriteLine(f));
    Console.WriteLine("Recuperiamo i dati dal database - trovare l'importo complessivo delle fatture
fatte da almeno tre giorni ");
    Console.WriteLine($"Importo complessivo: {db.Fatture.Where(f => f.Data < DateTime.Now.AddDays(-
2)).Sum(f => (double)f.Importo):C2}");
    Console.WriteLine("Recuperiamo i dati dal database - trovare l'importo medio delle fatture fatte
da almeno tre giorni ");
    Console.WriteLine($"Importo medio: {db.Fatture.Where(f => f.Data < DateTime.Now.AddDays(-
2)).Average(f => (double)f.Importo):C2}");
    Console.WriteLine("Recuperiamo i dati dal database - trovare l'importo massimo delle fatture fatte
da almeno tre giorni ");
    Console.WriteLine($"Importo massimo: {db.Fatture.Where(f => f.Data < DateTime.Now.AddDays(-
2)).Max(f => (double)f.Importo):C2}");
    Console.WriteLine("Recuperiamo i dati dal database - trovare l'importo minimo delle fatture fatte
da almeno tre giorni ");
    Console.WriteLine($"Importo minimo: {db.Fatture.Where(f => f.Data < DateTime.Now.AddDays(-
2)).Min(f => (double)f.Importo):C2}");
    Console.WriteLine("Recuperiamo i dati dal database - trovare il numero delle fatture fatte da
almeno tre giorni ");
    Console.WriteLine("Numero fatture: " + db.Fatture.Where(f => f.Data < DateTime.Now.AddDays(-
2)).Count());
}

Console.WriteLine("Recuperiamo i dati dal database - uso di WHERE e JOIN");
Console.WriteLine("trovare il nome e l'indirizzo dei clienti che hanno speso più di 5000 EUR");
var clientiConSpesa5000Plus = db.Fatture.Where(f => f.Importo > 5000).Join(db.Clienti,
    f => f.ClienteId,
    c => c.ClienteId,
    (f, c) => new { NumeroFattura = f.FatturaId, DataFattura = f.Data, NomeCliente =
c.RagioneSociale, Indirizzo = c.Via + " N." + c.Civico + " " + c.CAP + " " + c.Citta });
clientiConSpesa5000Plus.ToList().ForEach(c => Console.WriteLine(c));

//altro modo - uso delle navigation properties
Console.WriteLine("Recuperiamo i dati dal database - Uso di Navigation Property per ottenere i
dati dei clienti a partire dalle fatture");
var clientiConSpesa5000Plus2 = db.Fatture.Where(f => f.Importo > 5000).
    Select(f => new
    {
        NumeroFattura = f.FatturaId,
        DataFattura = f.Data,
        NomeCliente = f.Cliente.RagioneSociale,
        Indirizzo = f.Cliente.Via + " N." + f.Cliente.Civico + " " + f.Cliente.CAP + " " +
f.Cliente.Citta
    });
clientiConSpesa5000Plus2.ToList().ForEach(c => Console.WriteLine(c));
}

if (modifica)
{
    using var db = new FattureClientiContext();
    Console.WriteLine("\nModifichiamo i dati nel database");
    Console.WriteLine("Modifichiamo l'importo della prima fattura");

    Console.WriteLine("\nle fatture prima della modifica sono:\n");

    //stampo la lista delle fatture per vedere il risultato

    listaFatture = db.Fatture.ToList();
    Console.WriteLine("Stampa delle fatture");
    listaFatture.ForEach(f => Console.WriteLine(f));

    //accedo all'elemento da modificare
    var fattura = db.Fatture.Find(1); //trova l'entity con il valore della chiave (FatturaId)
    specificato
    if (fattura != null) //se ho trovato la fattura con FatturaId specificato
    {
        //modifico la fattura
        fattura.Importo *= 1.2m; //incremento del 20% l'importo
        db.SaveChanges(); //aggiorno il database

        //stampo la lista delle fatture per vedere il risultato
        listaFatture = db.Fatture.ToList();
        Console.WriteLine("Stampa delle fatture dopo la modifica");
    }
}

```

```

        listaFatture.ForEach(f => Console.WriteLine(f));
    }
    Console.WriteLine("\nmodifichiamo anche la seconda fattura");
    //non è possibile usare ElementAt - https://stackoverflow.com/questions/5147767/why-is-the-query-
operator-elementat-is-not-supported-in-linq-to-sql
    var secondaFattura = db.Fatture.Skip(1).First();
    secondaFattura.Importo *= 1.3m;
    db.SaveChanges();
    //stampo la lista delle fatture per vedere il risultato
    listaFatture = db.Fatture.ToList();
    Console.WriteLine("Stampa delle fatture");
    listaFatture.ForEach(f => Console.WriteLine(f));

    Console.WriteLine("\nmodifichiamo anche l'ultima");
    //non è possibile usare ElementAt - https://stackoverflow.com/questions/5147767/why-is-the-query-
operator-elementat-is-not-supported-in-linq-to-sql
    var ultimaFattura = db.Fatture.OrderBy(f => f.FatturaId).Last();
    ultimaFattura.Importo *= 1.5m;
    db.SaveChanges();
    //stampo la lista delle fatture per vedere il risultato
    listaFatture = db.Fatture.ToList();
    Console.WriteLine("Stampa delle fatture");
    listaFatture.ForEach(f => Console.WriteLine(f));
}
if (cancellazione)
{
    using var db = new FattureClientiContext();
    Console.WriteLine("\nEliminiamo un dato dal database");
    Console.WriteLine("\nPrima della cancellazione le fatture sono:");
    listaFatture = db.Fatture.ToList();
    Console.WriteLine("Stampa delle fatture");
    listaFatture.ForEach(f => Console.WriteLine(f));
    Console.WriteLine("\nEliminiamo la terza fattura");
    //accedo alla terza fattura - posso accedere o mediante chiave oppure in vase ad un elenco prindo
la terza
    //decidiamo di eliminare la fattura con FatturaId=3
    var fatturaDaEliminare = db.Fatture.Find(3);
    if (fatturaDaEliminare != null)//se ho trovato la fattura con id =3
    {
        Console.WriteLine($"eliminiamo la fattura con id {fatturaDaEliminare.FatturaId}");
        db.Remove(fatturaDaEliminare);
        db.SaveChanges();
        //stampiamo nuovamente l'elenco delle fatture per verificare che la fattura con id specifico è
stata eliminata
        listaFatture = db.Fatture.ToList();
        Console.WriteLine("Stampa delle fatture");
        listaFatture.ForEach(f => Console.WriteLine(f));
    }
}

```

## 6.4. EF Core: DbContext

<https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext>

A `DbContext` instance represents a session with the database and can be used to query and save instances of your entities. `DbContext` is a combination of the Unit Of Work and Repository patterns.

Entity Framework Core does not support multiple parallel operations being run on the same `DbContext` instance. This includes both parallel execution of async queries and any explicit concurrent use from multiple threads. Therefore, always await async calls immediately, or use separate `DbContext` instances for operations that execute in parallel. See [Avoiding DbContext threading issues](#) for more information.

### 6.4.1. Design Patterns – Cenni

[Design patterns](#) are a solution to some of the recurring problems that occur in applications, and the *Repository pattern* is one of the most popular design patterns among them.

#### **6.4.1.1. Repository pattern**

<https://www.codeguru.com/csharp/repository-pattern-c-sharp/>

The Repository pattern is used for abstracting how data is persisted or retrieved from a database. The idea behind the Repository pattern is to decouple the data access layer from the business access layer of the application so that the operations (such as adding, updating, deleting, and selecting items from the collection) is done through straightforward methods without dealing with database concerns such as connections, commands, and so forth.

Before diving into the implementation of the Repository design pattern, let us first learn what a repository is. In brief, a *repository* can be defined as the collection of domain objects present in memory. According to the [MSDN](#), “*a repository is responsible for separating the logic of retrieving data and mapping it to the entity model from the business logic that acts on the model.*” In simple terms, data comprising a data source layer – such as a database or a web service – should be agnostic from the business layer.

Repository pattern is one of the preferred patterns to apply in an application because it allows programmers to integrate all of the fundamental data operations related to an entity in one main class. Without this pattern, developers would need to create multiple classes for each entity with the same logic.

#### **6.4.1.2. Unity of Work Pattern**

<https://dev.to/moe23/step-by-step-repository-pattern-and-unit-of-work-with-asp-net-core-5-3l92>

If the Repository pattern is our abstraction over the idea of persistent storage, the Unit of Work (UoW) pattern is our abstraction over the idea of atomic operations. It will allow us to finally and fully decouple our service layer from the data layer.

The unit of work pattern now manages the database states. Once all updates of the entities in a scope are completed, the tracked changes are played onto the database in a transaction so that the database reflects the desired changes.

Thus, the unit of work pattern tracks a business transaction and translates it into a database transaction, wherein steps are collectively run as a single unit. To ensure that data integrity is not compromised, the transaction commits or is rolled back discretely, thus preventing indeterminate state.

#### **6.4.2. DbContext Lifetime, Configuration, and Initialization**

The lifetime of a DbContext begins when the instance is created and ends when the instance is [disposed](#). A DbContext instance is designed to be used for a *single unit-of-work*. This means that the lifetime of a DbContext instance is usually very short.

A typical unit-of-work when using Entity Framework Core (EF Core) involves:

- Creation of a DbContext instance
- Tracking of entity instances by the context. Entities become tracked by
  - Being [returned from a query](#)
  - Being [added or attached to the context](#)
- Changes are made to the tracked entities as needed to implement the business rule
- [SaveChanges](#) or [SaveChangesAsync](#) is called. EF Core detects the changes made and writes them to the database.
- The DbContext instance is disposed

- It is very important to dispose the **DbContext** after use. This ensures both that any unmanaged resources are freed, and that any events or other hooks are unregistered so as to prevent memory leaks in case the instance remains referenced.
- **DbContext is not thread-safe.** Do not share contexts between threads. Make sure to **await** all async calls before continuing to use the context instance.
- An **InvalidOperationException** thrown by EF Core code can put the context into an unrecoverable state. Such exceptions indicate a program error and are not designed to be recovered from.

Quando si crea un DbContext specifico per un determinato dominio applicativo, si crea una classe che discende dalla classe **DbContext**. In questa classe, sono molto importanti i metodi:

[OnConfiguring\(DbContextOptionsBuilder\)](#) Override this method to configure the database (and other options) to be used for this context. This method is called for each instance of the context that is created. The base implementation does nothing.

Questo metodo, di solito, contiene informazioni importanti che servono al DbContext per sapere dove si trova il Data Source, quali sono i parametri di connessione, etc.

[OnModelCreating\(ModelBuilder\)](#)

Override this method to further configure the model that was discovered by convention from the entity types exposed in **DbSet< TEntity >** properties on your derived context. The resulting model may be cached and re-used for subsequent instances of your derived context.

Questo metodo viene usato per completare la definizione del modello dei dati, aggiungendo vincoli e parametri che non sono ricavati automaticamente da EF Core mediante l'ispezione delle classi del modello dei dati.

## 6.5. Creazione di modelli di dati con EF Core

<https://www.learnentityframeworkcore.com/>

<https://docs.microsoft.com/en-us/ef/core/modeling/>

La creazione di un modello dei dati è un'operazione molto importante perché definisce la struttura dei dati all'interno del database che andremo ad utilizzare. EF Core è un ORM, ossia un Object Relational Mapper: è un framework che permette di mappare oggetti su tabelle di un database relazionale. I Relational DBMS (RDBMS) saranno oggetto di studio dettagliato l'anno prossimo, ma per il momento si può immaginare un database relazionale come un insieme di file che memorizzano tabelle collegate tra di loro, mediante il meccanismo delle chiavi esterne (come, ad esempio, le fatture sono collegate ai clienti mediante il ClienteId all'interno della fattura).

Per semplicità si potrà assumere che ogni istanza di una classe sarà mappata in una riga di una tabella che corrisponde alla classe

Classe → Tabella

Oggetto → Riga di una tabella.

Nell'esempio precedente di GestioneFattureClienti il modello creato con EF Core ha prodotto un database con le tabelle Fatture e Clienti. Aprendo il file di database fattureclienti.db con un

programma come **SQLite browser** possiamo vedere la struttura del database:

## Tables (4)

Name	Type	Schema
<b>Clienti</b>		CREATE TABLE "Clienti" ( "ClienteId" INTEGER NOT NULL CONSTRAINT "PK_Clienti" PRIMARY KEY AUTOINCREMENT, "RagioneSociale" TEXT NOT NULL, "PartitaIVA" TEXT NOT NULL, "Citta" TEXT NULL, "Via" TEXT NULL, "Civico" TEXT NULL, "CAP" TEXT NULL )
ClienteId	INTEGER	"ClienteId" INTEGER NOT NULL
RagioneSociale	TEXT	"RagioneSociale" TEXT NOT NULL
PartitaIVA	TEXT	"PartitaIVA" TEXT NOT NULL
Citta	TEXT	"Citta" TEXT
Via	TEXT	"Via" TEXT
Civico	TEXT	"Civico" TEXT
CAP	TEXT	"CAP" TEXT
<b>Fatture</b>		CREATE TABLE "Fatture" ( "FatturaId" INTEGER NOT NULL CONSTRAINT "PK_Fatture" PRIMARY KEY AUTOINCREMENT, "Data" TEXT NOT NULL, "Importo" TEXT NOT NULL, "ClienteId" INTEGER NOT NULL, CONSTRAINT "FK_Fatture_Clienti_ClienteId" FOREIGN KEY ("ClienteId") REFERENCES "Clienti" ("ClienteId") ON DELETE CASCADE )
FatturaId	INTEGER	"FatturaId" INTEGER NOT NULL
Data	TEXT	"Data" TEXT NOT NULL
Importo	TEXT	"Importo" TEXT NOT NULL
ClienteId	INTEGER	"ClienteId" INTEGER NOT NULL
<b>__EFMigrationsHistory</b>		CREATE TABLE "__EFMigrationsHistory" ( "MigrationId" TEXT NOT NULL CONSTRAINT "PK__EFMigrationsHistory" PRIMARY KEY, "ProductVersion" TEXT NOT NULL )
MigrationId	TEXT	"MigrationId" TEXT NOT NULL
ProductVersion	TEXT	"ProductVersion" TEXT NOT NULL
<b>sqlite_sequence</b>		CREATE TABLE sqlite_sequence(name,seq)
name		"name"
seq		"seq"

## Indices (1)

Name	Type	Schema
<b>IX_Fatture_ClienteId</b>		CREATE INDEX "IX_Fatture_ClienteId" ON "Fatture" ("ClienteId")
ClienteId		"ClienteId"

## Views (0)

Name	Type	Schema

## Triggers (0)

Name	Type	Schema

Struttura database Naviga nei dati Modifica Pramas Esegui SQL

Tabella: Clienti

ClienteId	RagioneSociale	PartitaIVA	Citta	Via	Civico	CAP
Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro
1 1	Cliente 1	1111111111	Napoli	Via dei Mille	23	80100
2 2	Cliente 2	1111111112	Roma	Via dei Fori I...	1	00100
3 3	Cliente 3	1111111113	Firenze	Via Raffaello	10	50100

	FatturaId	Data	Importo	ClienteId
	Filtro	Filtro	Filtro	Filtro
1	1	2019-10-27 0...	1440.54	1
2	2	2019-10-22 0...	4160.845	1
3	4	2019-10-27 0...	5200.45	2
4	5	2019-08-20 0...	10800.675	2

Per creare una tabella nel database si parte dalla definizione del modello come classe. Ad esempio, nel caso dei clienti siamo partiti dalla definizione della classe:

```
public class Cliente
{
    public int ClienteId { get; set; }
    public string RagioneSociale { get; set; } = null!;
    public string PartitaIVA { get; set; }=null!;
    public string? Citta { get; set; }
    public string? Via { get; set; }
    public string? Civico { get; set; }
    public string? CAP { get; set; }
    public List<Fattura> Fatture { get; } = new List<Fattura>();

    public override string ToString()
    {
        return string.Format($"[{nameof(ClienteId)}= {ClienteId}, {nameof(RagioneSociale)}= {RagioneSociale}, " +
            $"{nameof(PartitaIVA)}= {PartitaIVA}, {nameof(Citta)}= {Citta}, {nameof(Via)}= {Via}, {nameof(Civico)}= {Civico}, {nameof(CAP)}= {CAP}]");
    }
}
```

Si noti che EF Core utilizza alcune convenzioni e impostazioni di default (che possono essere modificate a patto di conoscere le caratteristiche di EF Core). Ad esempio, per creare una chiave primaria (ossia un campo che permette di identificare univocamente un'istanza di un oggetto all'interno di una tabella (ad esempio una riga all'interno della tabella dei clienti) bisogna utilizzare la convenzione **NomeClasseId**. Ad esempio, il campo ClienteId è chiave primaria (ad auto incremento) nella tabella che memorizzerà gli oggetti di tipo Cliente.

Quando bisogna creare i collegamenti tra le tabelle del database occorre indicare dei legami tra gli oggetti delle classi che rappresentano le righe delle tabelle. Ad esempio, per esprimere il fatto che un cliente può avere molte fatture (una associazione 1 a molti) si è inserito all'interno della classe Cliente la property:

```
public List<Fattura> Fatture { get; } = new List<Fattura>();
```

Questa property verrà usata da EF Core per capire il tipo di collegamento da fare tra la tabella dei Clienti e la tabella delle Fatture.

Analogamente la classe che descrive le fatture è del tipo:

```
public class Fattura
{
    public int FatturaId { get; set; }
    public DateTime Data { get; set; }
```

```

public decimal Importo { get; set; }
public int ClienteId { get; set; }
public Cliente Cliente { get; set; } = null!;

public override string ToString()
{
    return string.Format($"{nameof(FatturaId)} = {FatturaId}, {nameof(Data)} =
{Data.ToShortDateString()}, {nameof(Importo)} = {Importo}, {nameof(ClienteId)} =
{ClienteId}");
}
}

```

Nella classe Fattura notiamo alcuni elementi importantissimi:

**La chiave primaria FatturaId:** è un campo di tipo intero che permette di individuare univocamente una fattura. Possiamo impostare da codice C# il valore da inserire nel database, ma solitamente è più comodo affidarsi alla proprietà di auto incremento che il database attribuisce alla chiave primaria, come si può vedere dallo schema della tabella che è stata creata da SQLite quando è stata fatta la migrazione.

**La chiave esterna (foreign key) ClienteId:** è un campo che non è chiave primaria nella tabella in cui compare ma lo è nella tabella riferita. In questo caso il campo ClienteId all'interno della classe Fattura permette di individuare univocamente il cliente intestatario della fattura. EF Core ha bisogno anche del campo Cliente di tipo Cliente nella classe Fattura per creare la chiave esterna nella tabella delle Fatture che punta alla chiave primaria della tabella dei Clienti. Il campo Cliente non sarà usato dal nostro codice, ma da EF Core per definire il tipo di mapping tra la tabella delle Fatture e la tabella dei Clienti.

### 6.5.1. Creating and configuring a model

<https://learn.microsoft.com/en-us/ef/core/modeling/>

Entity Framework Core uses a set of conventions to build a model based on the shape of your entity classes. You can specify additional configuration to supplement and/or override what was discovered by convention.

You can override the `OnModelCreating` method in your derived context and use the `ModelBuilder API` to configure your model. This is the most powerful method of configuration and allows configuration to be specified without modifying your entity classes. `Fluent API` configuration has the highest precedence and will override conventions and data annotations.

```

using Microsoft.EntityFrameworkCore;

namespace EFModeling.EntityProperties.FluentAPI.Required;

internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    #region Required
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Url)
            .IsRequired();
    }
    #endregion
}

```

```

}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}

```

Si possono anche creare delle classi che implementano [IEntityTypeConfiguration< TEntity >](#) per rendere più snello il codice del metodo OnModelCreating. Per i dettagli si veda il link:

<https://learn.microsoft.com/en-us/ef/core/modeling/#grouping-configuration>

You can also apply attributes (known as Data Annotations) to your classes and properties. Data annotations will override conventions but will be overridden by Fluent API configuration.

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Microsoft.EntityFrameworkCore;

namespace EFModeling.EntityProperties.DataAnnotations.Annotations;

internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
}

[Table("Blogs")]
public class Blog
{
    public int BlogId { get; set; }

    [Required]
    public string Url { get; set; }
}

```

Nell'esempio sopra riportato l'annotazione racchiusa tra parentesi quadre specifica che l'entità Blog deve essere mappata su una tabella che si chiama Blogs.

## 6.5.2. Entities

<https://learn.microsoft.com/en-us/ef/core/modeling/entity-types>

Including a DbSet of a type on your context means that it is included in EF Core's model; we usually refer to such a type as an *entity*. EF Core can read and write entity instances from/to the database, and if you're using a relational database, EF Core can create tables for your entities via migrations.

By convention, types that are exposed in DbSet properties on your context are included in the model as entities. Entity types that are specified in the OnModelCreating method are also included, as are any types that are found by recursively exploring the navigation properties of other discovered entity types.

Per i dettagli si veda il link: <https://learn.microsoft.com/en-us/ef/core/modeling/entity-types>

### 6.5.2.1. Entities Properties

<https://learn.microsoft.com/en-us/ef/core/modeling/entity-properties>

Each entity type in your model has a set of properties, which EF Core will read and write from the database. If you're using a relational database, entity properties map to table columns.

By convention, when using a relational database, entity properties are mapped to table columns having the same name as the property.

If you prefer to configure your columns with different names, you can do so as following code snippet:

Uso di Data Annotation

```
public class Blog
{
    [Column("blog_id")]
    public int BlogId { get; set; }

    public string Url { get; set; }
}
```

Uso di Fluent API

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .Property(b => b.BlogId)
        .HasColumnName("blog_id");
}
```

#### 6.5.2.1.1. Required and optional properties

A property is considered optional if it is valid for it to contain null. If null is not a valid value to be assigned to a property then it is considered to be a required property. When mapping to a relational database schema, required properties are created as non-nullable columns, and optional properties are created as nullable columns.

By convention, a property whose .NET type can contain null will be configured as optional, whereas properties whose .NET type cannot contain null will be configured as required. For example, all properties with .NET value types (int, decimal, bool, etc.) are configured as required, and all properties with nullable .NET value types (int?, decimal?, bool?, etc.) are configured as optional.

C# 8 introduced a new feature called [nullable reference types \(NRT\)](#), which allows reference types to be annotated, indicating whether it is valid for them to contain null or not. This feature is enabled by default in new project templates, but remains disabled in existing projects unless explicitly opted into. [Nullable reference types affect EF Core's behavior in the following way:](#)

- If nullable reference types are disabled, all properties with .NET reference types are configured as optional by convention (for example, string).
- If nullable reference types are enabled, properties will be configured based on the C# nullability of their .NET type: string? will be configured as optional, but string will be configured as required.

The following example shows an entity type with required and optional properties, with the nullable reference feature disabled and enabled:

Nel caso in cui Nullable Reference Types è abilitato (default nei progetti da .NET 6 in avanti), si ha, ad

esempio:

```
public class Customer
{
    public int Id { get; set; }
    public string FirstName { get; set; } // Required by convention
    public string LastName { get; set; } // Required by convention
    public string? MiddleName { get; set; } // Optional by convention

    // Note the following use of constructor binding, which avoids compiled
    // warnings
    // for uninitialized non-nullable properties.
    public Customer(string firstName, string lastName, string? middleName =
null)
    {
        FirstName = firstName;
        LastName = lastName;
        MiddleName = middleName;
    }
}
```

Altre impostazioni sulle proprietà delle entità si trovano elencate qui:

<https://learn.microsoft.com/en-us/ef/core/modeling/entity-properties>

### 6.5.3. Chiavi – Keys

<https://learn.microsoft.com/en-us/ef/core/modeling/keys>

A key serves as a unique identifier for each entity instance. Most entities in EF have a single key, which maps to the concept of a *primary key* in relational databases (for entities without keys, see [Keyless entities](#)). Entities can have additional keys beyond the primary key (see [Alternate Keys](#) for more information).

#### 6.5.3.1. Chiave primaria

By convention, a property named Id or <type name>Id will be configured as the primary key of an entity.

Ad esempio:

```
internal class Car
{
    public string Id { get; set; }

    public string Make { get; set; }
    public string Model { get; set; }
}

internal class Truck
{
    public string TruckId { get; set; }

    public string Make { get; set; }
```

```
    public string Model { get; set; }  
}
```

You can configure a single property to be the primary key of an entity as follows:

Uso di Data Annotation:

```
internal class Car  
{  
    [Key]  
    public string LicensePlate { get; set; }  
  
    public string Make { get; set; }  
    public string Model { get; set; }  
}
```

Uso di FluentApi

```
protected override void OnModelCreating(ModelBuilder modelBuilder)  
{  
    modelBuilder.Entity<Car>()  
        .HasKey(c => c.LicensePlate);  
}
```

You can also configure multiple properties to be the key of an entity - this is known as a composite key. Composite keys can only be configured using the Fluent API; conventions will never set up a composite key, and you can not use Data Annotations to configure one.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)  
{  
    modelBuilder.Entity<Car>()  
        .HasKey(c => new { c.State, c.LicensePlate });  
}
```

## Value generation

For non-composite numeric and GUID primary keys, EF Core sets up value generation for you by convention. For example, a numeric primary key in SQL Server is automatically set up to be an IDENTITY column. For more information, see [the documentation on value generation](#).

While EF Core supports using properties of any primitive type as the primary key, including string, Guid, byte[] and others, not all databases support all types as keys. In some cases the key values can be converted to a supported type automatically, otherwise the conversion should be [specified manually](#).

### 6.5.3.2. Chiavi alternative

<https://learn.microsoft.com/en-us/ef/core/modeling/keys?source=recommendations&tabs=fluent-api#alternate-keys>

An alternate key serves as an alternate unique identifier for each entity instance in addition to the primary key; it can be used as the target of a relationship. When using a relational database this maps to the concept of a unique index/constraint on the alternate key column(s) and one or more foreign key constraints that reference the column(s).

Alternate keys are typically introduced for you when needed and you do not need to manually

configure them. By convention, an alternate key is introduced for you when you identify a property which isn't the primary key as the target of a relationship.

Esempio di chiave alternativa: in questo caso la property Url all'interno di Blog diventa una chiave alternativa perché è referenziata dalla property BlogUrl di Post. In questo esempio l'istruzione HasPrincipalKey(b => b.Url) poteva anche non essere inserita, poiché sarebbe stata creata comunque una chiave alternativa per la property Url di Blog.

```
internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts)
            .HasForeignKey(p => p.BlogUrl)
            .HasPrincipalKey(b => b.Url);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public string BlogUrl { get; set; }
    public Blog Blog { get; set; }
}
```

You can also configure a single property to be an alternate key:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Car>()
        .HasAlternateKey(c => c.LicensePlate);
}
```

You can also configure multiple properties to be an alternate key (known as a composite alternate key):

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
```

```

{
    modelBuilder.Entity<Car>()
        .HasAlternateKey(c => new { c.State, c.LicensePlate });
}

```

If you just want to enforce uniqueness on a column, define a unique index rather than an alternate key (see [Indexes](#))

#### 6.5.4. Indici

<https://learn.microsoft.com/en-us/ef/core/modeling/indexes>

Indexes are a common concept across many data stores. While their implementation in the data store may vary, they are used to make lookups based on a column (or set of columns) more efficient. See the [indexes section](#) in the performance documentation for more information on good index usage.

You can specify an index over a column as follows:

Uso di Data Annotation

```

[Index(nameof(Url))]
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}

```

Uso di Fluent API

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .HasIndex(b => b.Url);
}

```

By convention, an index is created in each property (or set of properties) that are used as a foreign key. EF Core only supports one index per distinct set of properties.

An index can also span more than one column:

```

[Index(nameof.FirstName), nameof.LastName)]
public class Person
{
    public int PersonId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}

```

Oppure:

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Person>()
        .HasIndex(p => new { p.FirstName, p.LastName });
}

```

By default, indexes aren't unique: multiple rows are allowed to have the same value(s) for the index's column set. You can make an index unique as follows:

```
[Index(nameof(Url), IsUnique = true)]  
public class Blog  
{  
    public int BlogId { get; set; }  
    public string Url { get; set; }  
}
```

Oppure:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)  
{  
    modelBuilder.Entity<Blog>()  
        .HasIndex(b => b.Url)  
        .IsUnique();  
}
```

The index sort order is ascending by default

In EF Core 7.0 it is possible to specify the order of the index:

<https://learn.microsoft.com/en-us/ef/core/modeling/indexes?tabs=fluent-api#index-sort-order>

## 6.5.5. Associazioni – Relationships

<https://learn.microsoft.com/en-us/ef/core/modeling/relationships>

A relationship defines how two entities relate to each other. In a relational database, this is represented by a foreign key constraint.

### 6.5.5.1. Definition of Terms

There are several terms used to describe relationships

- **Dependent entity:** This is the entity that contains the foreign key properties. Sometimes referred to as the 'child' of the relationship.
- **Principal entity:** This is the entity that contains the primary/alternate key properties. Sometimes referred to as the 'parent' of the relationship.
- **Principal key:** The properties that uniquely identify the principal entity. This may be the primary key or an alternate key.
- **Foreign key:** The properties in the dependent entity that are used to store the principal key values for the related entity.
- **Navigation property:** A property defined on the principal and/or dependent entity that references the related entity.
- **Collection navigation property:** A navigation property that contains references to many related entities.
- **Reference navigation property:** A navigation property that holds a reference to a single related entity.
- **Inverse navigation property:** When discussing a particular navigation property, this term refers to the navigation property on the other end of the relationship.

- **Self-referencing relationship:** A relationship in which the dependent and the principal entity types are the same.

### 6.5.5.2. Associazione uno a molti - 1:N

The following code shows a one-to-many relationship between Blog and Post

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

- Post is the dependent entity
- Blog is the principal entity
- Blog.BlogId is the principal key (in this case it is a primary key rather than an alternate key)
- Post.BlogId is the foreign key
- Post.Blog is a reference navigation property
- Blog.Posts is a collection navigation property
- Post.Blog is the inverse navigation property of Blog.Posts (and vice versa)

#### 6.5.5.2.1. Conventions

By convention, a relationship will be created when there is a navigation property discovered on a type. A property is considered a navigation property if the type it points to cannot be mapped as a scalar type by the current database provider.

##### Note

Relationships that are discovered by convention will always target the primary key of the principal entity. To target an alternate key, additional configuration must be performed using the Fluent API.

##### 6.5.5.2.1.1. Fully Defined Relationships

The most common pattern for relationships is to have navigation properties defined on both ends of the relationship and a foreign key property defined in the dependent entity class.

- If a pair of navigation properties is found between two types, then they will be configured as inverse navigation properties of the same relationship.
- If the dependent entity contains a property with a name matching one of these patterns then it will be configured as the foreign key:

```


- <navigation property name><principal key property name>
- <navigation property name>Id
- <principal entity name><principal key property name>
- <principal entity name>Id



public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}

```

In this example the highlighted properties will be used to configure the relationship.

If the property is the primary key or is of a type not compatible with the principal key then it won't be configured as the foreign key.

#### **6.5.5.2.1.2. No Foreign Key Property – Shadow Foreign Key**

While it is recommended to have a foreign key property defined in the dependent entity class, it is not required. If no foreign key property is found, a [shadow foreign key property](#) will be introduced with the name <navigation property name><principal key property name> or <principal entity name><principal key property name> if no navigation is present on the dependent type.

```

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}

```

In this example, the shadow foreign key is BlogId because prepending the navigation name would be redundant.

#### 6.5.5.2.1.3. Single Navigation Property

Including just one navigation property (no inverse navigation, and no foreign key property) is enough to have a relationship defined by convention. You can also have a single navigation property and a foreign key property.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
}
```

#### 6.5.5.2.1.4. Limitations

When there are multiple navigation properties defined between two types (that is, more than just one pair of navigations that point to each other) the relationships represented by the navigation properties are ambiguous. You will need to manually configure them to resolve the ambiguity.

#### 6.5.5.2.1.5. Cascade Delete

By convention, cascade delete will be set to *Cascade* for required relationships and *ClientSetNull* for optional relationships. *Cascade* means dependent entities are also deleted. *ClientSetNull* means that dependent entities that are not loaded into memory will remain unchanged and must be manually deleted, or updated to point to a valid principal entity. For entities that are loaded into memory, EF Core will attempt to set the foreign key properties to null.

See the [Required and Optional Relationships](#) section for the difference between required and optional relationships.

See [Cascade Delete](#) for more details about the different delete behaviors and the defaults used by convention.

#### 6.5.5.2.2. Manual Configuration

La configurazione manuale delle associazioni si rende necessaria quando le entità coinvolte non ricadono nei termini delle convenzioni sui nomi delle chiavi oppure delle Navigation Properties, oppure nel caso di particolari associazioni (come nel caso della “molti a molti” che verrà illustrato nei prossimi paragrafi).

La configurazione manuale delle associazioni può essere sempre eseguita utilizzando i metodi Fluent API, oppure, in alternativa, in alcuni casi, utilizzando il meccanismo delle annotazioni.

#### 6.5.5.2.3. Fluent Api – manual configuration

To configure a relationship in the Fluent API, you start by identifying the navigation properties that make up the relationship. HasOne or HasMany identifies the navigation property on the entity type

you are beginning the configuration on. You then chain a call to WithOne or WithMany to identify the inverse navigation. HasOne/WithOne are used for reference navigation properties and HasMany/WithMany are used for collection navigation properties.

```
internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}
```

#### 6.5.5.2.3.1. Single navigation property – manual configuration

If you only have one navigation property then there are parameterless overloads of WithOne and WithMany. This indicates that there is conceptually a reference or collection on the other end of the relationship, but there is no navigation property included in the entity class.

```
internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .HasMany(b => b.Posts)
            .WithOne();
    }
}
```

```

}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
}

```

#### **6.5.5.2.3.2. Foreign Key – manual configuration**

You can use the Fluent API to configure which property should be used as the foreign key property for a given relationship:

```

internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts)
            .HasForeignKey(p => p.BlogForeignKey);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogForeignKey { get; set; }
}

```

```

    public Blog Blog { get; set; }
}

```

Nell'esempio appena mostrato la foreign key (chiave esterna) non rispetta la convenzione sui nomi e va configurata manualmente.

Vediamo un altro esempio con chiave primaria e chiave esterna composita:

```

internal class MyContext : DbContext
{
    public DbSet<Car> Cars { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Car>()
            .HasKey(c => new { c.State, c.LicensePlate });

        modelBuilder.Entity<RecordOfSale>()
            .HasOne(s => s.Car)
            .WithMany(c => c.SaleHistory)
            .HasForeignKey(s => new { s.CarState, s.CarLicensePlate });
    }
}

public class Car
{
    public string State { get; set; }
    public string LicensePlate { get; set; }
    public string Make { get; set; }
    public string Model { get; set; }

    public List<RecordOfSale> SaleHistory { get; set; }
}

public class RecordOfSale
{
    public int RecordOfSaleId { get; set; }
    public DateTime DateSold { get; set; }
    public decimal Price { get; set; }

    public string CarState { get; set; }
    public string CarLicensePlate { get; set; }
    public Car Car { get; set; }
}

```

Nell'esempio appena mostrato l'entità Car ha una chiave primaria composita ed è obbligatorio ricorrere alle Fluent API per impostare la chiave primaria. Allo stesso modo l'entità RecordOfSale ha una chiave esterna composita e anche in questo caso è obbligatorio ricorrere alle Fluent API per impostare le chiavi esterne.

#### **6.5.5.2.3.3. *Shadow Foreign Key – manual configuration***

You can use the string overload of HasForeignKey(...) to configure a shadow property as a foreign key (see [Shadow Properties](#) for more information). We recommend explicitly adding the shadow

property to the model before using it as a foreign key (as shown below).

```
internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        // Add the shadow property to the model
        modelBuilder.Entity<Post>()
            .Property<int>("BlogForeignKey");

        // Use the shadow property as a foreign key
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts)
            .HasForeignKey("BlogForeignKey");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}
```

Nell'esempio appena mostrato non c'è la foreign key nell'entità Post, ma solo la Navigation Property. Le Fluent API sono utilizzate per configurare manualmente la shadow foreign key a cui viene dato il nome di BlogForeignKey. Si noti che la shadow foreign key sarebbe stata configurata anche senza ricorrere all'uso delle Fluent API, come descritto nel paragrafo 6.5.5.2.1.2, in tal caso il nome sarebbe stato solo BlogId.

#### 6.5.5.2.3.4. ***Without navigation property – manual configuration***

You don't necessarily need to provide a navigation property. You can simply provide a foreign key on one side of the relationship.

```
internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
```

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Post>()
        .HasOne<Blog>()
        .WithMany()
        .HasForeignKey(p => p.BlogId);
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
}

```

Si noti che in questo caso non c'è nessuna Navigation Property e quindi EF Core non creerebbe nessuna associazione tra le due entità in automatico. Pertanto, l'uso delle Fluent API non è ridondante in questo caso.

#### **6.5.5.2.3.5. Principal Key of a Relationship – manual configuration**

If you want the foreign key to reference a property other than the primary key, you can use the Fluent API to configure the principal key property for the relationship. The property that you configure as the principal key will automatically be set up as an [alternate key](#).

Warning: The order in which you specify principal key properties must match the order in which they are specified for the foreign key.

Esempio di chiave semplice:

```

internal class MyContext : DbContext
{
    public DbSet<Car> Cars { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<RecordOfSale>()
            .HasOne(s => s.Car)
            .WithMany(c => c.SaleHistory)
            .HasForeignKey(s => s.CarLicensePlate)
            .HasPrincipalKey(c => c.LicensePlate);
    }
}

```

```

public class Car
{
    public int CarId { get; set; }
    public string LicensePlate { get; set; }
    public string Make { get; set; }
    public string Model { get; set; }

    public List<RecordOfSale> SaleHistory { get; set; }
}

public class RecordOfSale
{
    public int RecordOfSaleId { get; set; }
    public DateTime DateSold { get; set; }
    public decimal Price { get; set; }

    public string CarLicensePlate { get; set; }
    public Car Car { get; set; }
}

```

In questo esempio la chiave esterna nell'entità RecordOfSale non rispetta la convenzione sui nomi, inoltre non punta alla chiave primaria dell'entità Car (che è CarId), ma punta alla chiave alternativa LicensePlate. In questo caso il codice Fluent API inserito nel metodo OnModelCreating è necessario.

Esempio di chiave composta:

```

internal class MyContext : DbContext
{
    public DbSet<Car> Cars { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<RecordOfSale>()
            .HasOne(s => s.Car)
            .WithMany(c => c.SaleHistory)
            .HasForeignKey(s => new { s.CarState, s.CarLicensePlate })
            .HasPrincipalKey(c => new { c.State, c.LicensePlate });
    }
}

public class Car
{
    public int CarId { get; set; }
    public string State { get; set; }
    public string LicensePlate { get; set; }
    public string Make { get; set; }
    public string Model { get; set; }

    public List<RecordOfSale> SaleHistory { get; set; }
}

```

```

public class RecordOfSale
{
    public int RecordOfSaleId { get; set; }
    public DateTime DateSold { get; set; }
    public decimal Price { get; set; }

    public string CarState { get; set; }
    public string CarLicensePlate { get; set; }
    public Car Car { get; set; }
}

```

In questo esempio la chiave esterna in RecordOfSale è composta da due proprietà (CarState e CarLicensePlate) e punta ad una chiave alternativa composta nell'entità Car (costituita da State e LicensePlate).

#### **6.5.5.2.3.6. *Required and optional relationships – Manual configuration***

You can use the Fluent API to configure whether the relationship is required or optional. Ultimately this controls whether the foreign key property is required or optional. This is most useful when you are using a shadow state foreign key. If you have a foreign key property in your entity class then the requiredness of the relationship is determined based on whether the foreign key property is required or optional (see [Required and Optional properties](#) for more information).

The foreign key properties are located on the dependent entity type, so if they are configured as required it means that every dependent entity is required to have a corresponding principal entity.

Una foreign key che non può assumere il valore nullo è automaticamente required. Se si utilizzano i Nullable Reference Types, basta dichiarare il campo non nullable.

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Post>()
        .HasOne(p => p.Blog)
        .WithMany(b => b.Posts)
        .IsRequired();
}

```

#### **6.5.5.2.3.7. *Cascade delete – manual configuration***

You can use the Fluent API to configure the cascade delete behavior for a given relationship explicitly.

See [Cascade Delete](#) for a detailed discussion of each option.

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Post>()
        .HasOne(p => p.Blog)
        .WithMany(b => b.Posts)
        .OnDelete(DeleteBehavior.Cascade);
}

```

Si noti che questa impostazione è da usarsi solo nel caso in cui non si voglia utilizzare il comportamento di default relativo al Cascade Delete come riportato nel paragrafo 6.5.5.2.1.5.

#### 6.5.5.2.4. Data Annotations – manual configuration

There are two data annotations that can be used to configure relationships, [ForeignKey] and [InverseProperty]. These are available in the System.ComponentModel.DataAnnotations.Schema namespace.

##### 6.5.5.2.4.1. Data Annotations [ForeignKey] – manual configuration

You can use the Data Annotations to configure which property should be used as the foreign key property for a given relationship. This is typically done when the foreign key property is not discovered by convention.

```
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;

namespace EFModeling.DataAnnotations.Relationships.ForeignKey
{
    class MyContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
    }

    #region Entities
    public class Blog
    {
        public int BlogId { get; set; }
        public string Url { get; set; }

        public List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogForeignKey { get; set; }

        [ForeignKey("BlogForeignKey")]
        public Blog Blog { get; set; }
    }
    #endregion
}
```

The [ForeignKey] annotation can be placed on either navigation property in the relationship. It does not need to go on the navigation property in the dependent entity class.

#### **6.5.5.2.4.2. Data Annotations [*InverseProperty*] – manual configuration**

You can use the Data Annotations to configure how navigation properties on the dependent and principal entities pair up. This is typically done when there is more than one pair of navigation properties between two entity types.

```
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;

namespace EFModeling.DataAnnotations.Relationships.InverseProperty
{
    class MyContext : DbContext
    {
        public DbSet<Post> Posts { get; set; }
        public DbSet<User> Users { get; set; }
    }

    #region Entities
    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int AuthorUserId { get; set; }
        public User Author { get; set; }

        public int ContributorUserId { get; set; }
        public User Contributor { get; set; }
    }

    public class User
    {
        public string UserId { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }

        [InverseProperty("Author")]
        public List<Post> AuthoredPosts { get; set; }

        [InverseProperty("Contributor")]
        public List<Post> ContributedToPosts { get; set; }
    }
    #endregion
}
```

#### **6.5.5.2.4.3. Data Annotations [*Primary Key*] – manual configuration**

You can use Data Annotations to configure a single property to be the key of an entity.

```
using Microsoft.EntityFrameworkCore;
```

```

using System.ComponentModel.DataAnnotations;

namespace EFModeling.DataAnnotations.KeySingle
{
    class MyContext : DbContext
    {
        public DbSet<Car> Cars { get; set; }
    }

    class Car
    {
        [Key]
        public string LicensePlate { get; set; }

        public string Make { get; set; }
        public string Model { get; set; }
    }
}

```

In questo esempio la chiave primaria non rispetta la convenzione sul nome e pertanto la definizione della chiave primaria deve essere fatta manualmente. Ciò può essere ottenuto mediante Fluent API, oppure mediante Data Annotation.

### 6.5.6. Caso di associazione molti a molti – N:M

<https://learn.microsoft.com/en-us/ef/core/modeling/relationships#many-to-many>

Many-to-many relationships require a collection navigation property on both sides. They will be discovered by convention like other types of relationships.

```

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public ICollection<Tag> Tags { get; set; }
}

public class Tag
{
    public string TagId { get; set; }

    public ICollection<Post> Posts { get; set; }
}

```

The way this relationship is implemented in the database is by a join table that contains foreign keys to both Post and Tag.

Internally, EF creates an entity type to represent the join table that will be referred to as the join entity type. Dictionary<string, object> is currently used for it to handle any combination of foreign key properties, see [property bag entity types](#) for more information. More than one many-to-many relationships can exist in the model, therefore the join entity type must be given a unique name, in this case PostTag. The feature that allows this is called shared-type entity type.

La creazione dell'associazione molti a molti deve essere fatta utilizzando le Fluent API, all'interno del metodo OnModelCreating. Ad esempio:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
```

```

{
    modelBuilder
        .Entity<Post>()
        .HasMany(p => p.Tags)
        .WithMany(p => p.Posts)
        .UsingEntity(j => j.ToTable("PostTags"));
}

```

In questo caso viene creata da EF Core un'associazione molti a molti (e la corrispondente tabella nel database relazionale) chiamata PostTags.

EF uses two one-to-many relationships on the join entity type to represent the many-to-many relationship. You can configure these relationships in the UsingEntity arguments.

La stessa configurazione della tabella di collegamento, con più opzioni può essere fatta come segue:

`protected override void OnModelCreating(ModelBuilder modelBuilder)`

```

{
    modelBuilder.Entity<Post>()
        .HasMany(p => p.Tags)
        .WithMany(p => p.Posts)
        .UsingEntity<Dictionary<string, object>>(
            "PostTag",
            j => j
                .HasOne<Tag>()
                .WithMany()
                .HasForeignKey("TagId")
                .HasConstraintName("FK_PostTag_Tags_TagId")
                .OnDelete(DeleteBehavior.Cascade),
            j => j
                .HasOne<Post>()
                .WithMany()
                .HasForeignKey("PostId")
                .HasConstraintName("FK_PostTag_Posts_PostId")
                .OnDelete(DeleteBehavior.ClientCascade));
}

```

}

Nella modellazione delle associazioni molti a molti, anche se non obbligatorio, è comunque meglio definire esplicitamente un'entità di collegamento che verrà utilizzata da EF Core per il mapping su una tabella di collegamento. Ad esempio:

```

internal class MyContext : DbContext
{
    public MyContext(DbContextOptions<MyContext> options)
        : base(options)
    {
    }

    public DbSet<Post> Posts { get; set; }
    public DbSet<Tag> Tags { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()

```

```

        .HasMany(p => p.Tags)
        .WithMany(p => p.Posts)
        .UsingEntity<PostTag>(
            j => j
                .HasOne(pt => pt.Tag)
                .WithMany(t => t.PostTags)
                .HasForeignKey(pt => pt.TagId),
            j => j
                .HasOne(pt => pt.Post)
                .WithMany(p => p.PostTags)
                .HasForeignKey(pt => pt.PostId),
            j =>
            {
                j.Property(pt =>
pt.PublicationDate).HasDefaultValueSql("CURRENT_TIMESTAMP");
                j.HasKey(t => new { t.PostId, t.TagId });
            });
        }
    }

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public ICollection<Tag> Tags { get; set; }
    public List<PostTag> PostTags { get; set; }
}

public class Tag
{
    public string TagId { get; set; }

    public ICollection<Post> Posts { get; set; }
    public List<PostTag> PostTags { get; set; }
}

public class PostTag
{
    public DateTime PublicationDate { get; set; }

    public int PostId { get; set; }
    public Post Post { get; set; }

    public string TagId { get; set; }
    public Tag Tag { get; set; }
}

```

In alternativa all'uso del metodo UsingEntity si può definire l'associazione molti a molti, ricorrendo alla definizione di due associazioni uno a molti, come descritto nel seguente esempio:

```
public class MyContext : DbContext
{
    public MyContext(DbContextOptions<MyContext> options)
        : base(options)
    {
    }

    public DbSet<Post> Posts { get; set; }
    public DbSet<Tag> Tags { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<PostTag>()
            .HasKey(t => new { t.PostId, t.TagId });

        modelBuilder.Entity<PostTag>()
            .HasOne(pt => pt.Post)
            .WithMany(p => p.PostTags)
            .HasForeignKey(pt => pt.PostId);

        modelBuilder.Entity<PostTag>()
            .HasOne(pt => pt.Tag)
            .WithMany(t => t.PostTags)
            .HasForeignKey(pt => pt.TagId);
    }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public List<PostTag> PostTags { get; set; }
}

public class Tag
{
    public string TagId { get; set; }

    public List<PostTag> PostTags { get; set; }
}

public class PostTag
{
    public DateTime PublicationDate { get; set; }
```

```

    public int PostId { get; set; }
    public Post Post { get; set; }

    public string TagId { get; set; }
    public Tag Tag { get; set; }
}

```

### 6.5.7. Caso di associazione 1 a 1 – 1:1

<https://learn.microsoft.com/en-us/ef/core/modeling/relationships#one-to-one>

One to one relationships have a reference navigation property on both sides. They follow the same conventions as one-to-many relationships, but a unique index is introduced on the foreign key property to ensure only one dependent is related to each principal.

```

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public BlogImage BlogImage { get; set; }
}

public class BlogImage
{
    public int BlogImageId { get; set; }
    public byte[] Image { get; set; }
    public string Caption { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}

```

Nota: EF will choose one of the entities to be the dependent based on its ability to detect a foreign key property. If the wrong entity is chosen as the dependent, you can use the Fluent API to correct this.

When configuring the relationship with the Fluent API, you use the HasOne and WithOne methods.

Vediamo un esempio in cui viene specificato mediante le Fluent API quale deve essere l'entità dipendente (quella in cui figura la chiave esterna):

```

internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<BlogImage> BlogImages { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .HasOne(b => b.BlogImage)
            .WithOne(i => i.Blog)
            .HasForeignKey<BlogImage>(b => b.BlogForeignKey);
    }
}

```

```

    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public BlogImage BlogImage { get; set; }
}

public class BlogImage
{
    public int BlogImageId { get; set; }
    public byte[] Image { get; set; }
    public string Caption { get; set; }

    public int BlogForeignKey { get; set; }
    public Blog Blog { get; set; }
}

```

The dependent side is considered optional by default, but can be configured as required.

## Table Splitting

<https://www.learnentityframeworkcore.com/configuration/one-to-one-relationship-configuration>

Table splitting is a technique that enables you to use a single table to represent both entities in a one-to-one relationship. Using this feature, both entities in the one-to-one relationship illustrated above will be stored in a database table together.

Prendiamo il caso:

```

public class Author
{
    public int AuthorId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public AuthorBiography Biography { get; set; }
}

public class AuthorBiography
{
    public int AuthorBiographyId { get; set; }
    public string Biography { get; set; }
    public DateTime DateOfBirth { get; set; }
    public string PlaceOfBirth { get; set; }
    public string Nationality { get; set; }
    public int AuthorRef { get; set; }
    public Author Author { get; set; }
}

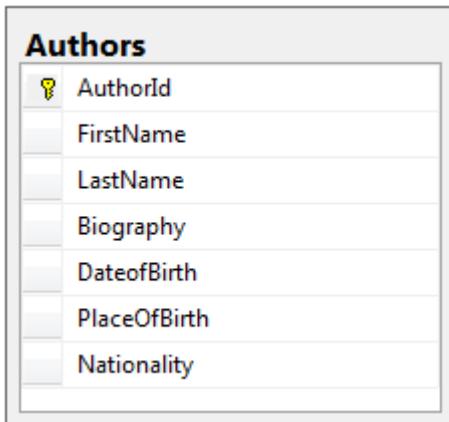
```

Il table splitting può essere configurato, impostando la stessa tabella di destinazione per entrambe le entità.

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Author>()
        .HasOne(a => a.AuthorBiography).WithOne(b => b.Author)
        .HasForeignKey<AuthorBiography>(e => e.AuthorId);
    modelBuilder.Entity<Author>().ToTable("Authors");
    modelBuilder.Entity<AuthorBiography>().ToTable("Authors");
}

```



Both entities must share the same primary key, which is used as a foreign key in the dependent end of the relationship.

## 6.5.8. Generazione del database - Migrations

<https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations>

Dagli esempi visti precedentemente si è visto come sia possibile generare un database a partire dal modello dei dati. Ad esempio:

Definite le classi per Fattura e Cliente e definito il DbContext

```

public class FattureClientiContext : DbContext
{
    public DbSet<Fattura> Fatture { get; set; }
    public DbSet<Cliente> Clienti { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder options)
        => options.UseSqlite("Data Source=fattureclienti.db");
}

```

il mapping delle classi (Entity) sulle corrispondenti tabelle del database relazionale (Relational DBMS) viene eseguito per mezzo dell'operazione di migration:

**dotnet ef migrations add InitialCreate**

Una volta fatta la migration per creare il database fisico si può impartire il comando:

**dotnet ef database update**

oppure da codice (ad esempio dal main) scrivere una sequenza di istruzioni come la seguente (vedremo in seguito alcuni esempi...):

```

using (var db = new BloggingContext())
{

```

```
        db.Database.Migrate();
    }
```

In generale il meccanismo di migration è più sofisticato di quello che può apparire dagli esempi riportati precedentemente. Infatti, leggendo la documentazione Microsoft si evince che il meccanismo delle migrations è stato concepito per poter seguire l'evoluzione dello schema (ossia il modello) dei dati:

In real world projects, data models change as features get implemented: new entities or properties are added and removed, and database schemas need to be changed accordingly to be kept in sync with the application. The migrations feature in EF Core provides a way to incrementally update the database schema to keep it in sync with the application's data model while preserving existing data in the database.

At a high level, migrations function in the following way:

- When a data model change is introduced, the developer uses EF Core tools to add a corresponding migration describing the updates necessary to keep the database schema in sync. EF Core compares the current model against a snapshot of the old model to determine the differences, and generates migration source files; the files can be tracked in your project's source control like any other source file.
- Once a new migration has been generated, it can be applied to a database in various ways. EF Core records all applied migrations in a special history table, allowing it to know which migrations have been applied and which haven't.

The rest of this page is a step-by-step beginner's guide for using migrations. Consult the other pages in this section for more in-depth information.

Di seguito si riporta un esempio di generazione di database con più migrations:

Let's assume you've just completed your first EF Core application, which contains the following simple model:

```
//File Blog.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MigrationsTest.Model
{
    public class Blog
    {
        public int Id { get; set; }
        public string? Name { get; set; }
    }
}
```

```
//File BloggingContext.cs

using Microsoft.EntityFrameworkCore;
using MigrationsTest.Model;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MigrationsTest.Data
{
```

```

public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; } = null!;
    public string DbPath { get; }
    public BloggingContext()
    {
        //https://www.hanselman.com/blog/how-do-i-find-which-directory-my-net-core-console-application-was-started-in-or-is-running-from
        var folder = ApplicationContext.BaseDirectory;
        //La BaseDirectory restituisce la cartella dove si trova l'assembly (.dll e .exe del programma compilato)
        //il database, per comodità, è inserito nella cartella di progetto, dove si trova anche il file Program.cs
        var path = Path.Combine(folder, "../../../../../blogs.db");
        DbPath = path;
    }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite($"Data Source = {DbPath}");
    }
}

```

Aggiungiamo i pacchetti per EF Core con SQLite:

```

dotnet add package Microsoft.EntityFrameworkCore.Sqlite
dotnet add package Microsoft.EntityFrameworkCore.Design

```

You're now ready to add your first migration! Instruct EF Core to create a migration named **InitialCreate**:

```
dotnet ef migrations add InitialCreate
```

EF Core will create a directory called **Migrations** in your project and generate some files. It's a good idea to inspect what exactly EF Core generated - and possibly amend it - but we'll skip over that for now.

At this point you can have EF create your database and create your schema from the migration. This can be done via the following:

```
dotnet ef database update
```

That's all there is to it - your application is ready to run on your new database, and you didn't need to write a single line of SQL. Note that this way of applying migrations is ideal for local development, but is less suitable for production environments - see the [Applying Migrations page](#) for more info.

Supponiamo ora di dover modificare il modello dei dati....

A few days have passed, and you're asked to add a creation timestamp to your blogs. You've done the necessary changes to your application, and your model now looks like this:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MigrationsTest.Model
{
    public class Blog
    {
        public int Id { get; set; }

```

```

        public string? Name { get; set; }
        public DateTime CreatedTimestamp { get; set; }
    }
}

```

Your model and your production database are now out of sync - we must add a new column to your database schema. Let's create a new migration for this:

**dotnet ef migrations add AddBlogCreatedTimestamp**

Note that we give migrations a descriptive name, to make it easier to understand the project history later.

Since this isn't the project's first migration, EF Core now compares your updated model against a snapshot of the old model, before the column was added; the model snapshot is one of the files generated by EF Core when you add a migration, and is checked into source control. Based on that comparison, EF Core detects that a column has been added, and adds the appropriate migration.

You can now apply your migration as before:

**dotnet ef database update**

Note that this time, EF detects that the database already exists. In addition, when our first migration was applied above, this fact was recorded in a special migrations history table in your database; this allows EF to automatically apply only the new migration.

E' anche possibile escludere dei tipi (Entity) da una migrazione; per vedere come sia possibile si veda:

<https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli#excluding-parts-of-your-model>

## 6.6. Tipi di dato

<https://learn.microsoft.com/en-us/ef/core/modeling/entity-properties?tabs=data-annotations%2Cwithout-nrt#column-data-types>

When using a relational database, the database provider selects a data type based on the .NET type of the property. It also takes into account other metadata, such as the configured maximum length, whether the property is part of a primary key, etc.

For example, SQL Server maps DateTime properties to datetime2(7) columns, and string properties to nvarchar(max) columns (or to nvarchar(450) for properties that are used as a key).

You can also configure your columns to specify an exact data type for a column. For example, the following code configures Url as a non-unicode string with maximum length of 200 and Rating as decimal with precision of 5 and scale of 2:

I tipi di dato dipendono dal database utilizzato.

### 6.6.1. Data Annotations

You can use Data Annotations to specify an exact data type for a column.

For example the following code configures Url as a non-unicode string with maximum length of 200 and Rating as decimal with precision of 5 and scale of 2.

```

public class Blog
{
    public int BlogId { get; set; }
    [Column(TypeName = "varchar(200)")]

```

```

    public string Url { get; set; }
    [Column(TypeName = "decimal(5, 2)")]
    public decimal Rating { get; set; }
}

```

## 6.6.2. Fluent API

You can also use the Fluent API to specify the same data types for the columns.

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>(eb =>
        {
            eb.Property(b => b.Url).HasColumnType("varchar(200)");
            eb.Property(b => b.Rating).HasColumnType("decimal(5, 2)");
        });
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    public decimal Rating { get; set; }
}

```

## 6.7. Conversioni di valore

<https://learn.microsoft.com/en-us/ef/core/modeling/value-conversions?tabs=data-annotations>

Value converters allow property values to be converted when reading from or writing to the database. This conversion can be from one value to another of the same type (for example, encrypting strings) or from a value of one type to a value of another type (for example, converting enum values to and from strings in the database.)

Value conversions are defined on properties in the `OnModelCreating` of your `DbContext`. For example, consider an enum and entity type defined as:

```

public class Rider
{
    public int Id { get; set; }
    public EquineBeast Mount { get; set; }
}

public enum EquineBeast
{
    Donkey,
    Mule,
}

```

```

Horse,
Unicorn
}

```

Then conversions can be defined in OnModelCreating to store the enum values as strings (for example, "Donkey", "Mule", ...) in the database:

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder
        .Entity<Rider>()
        .Property(e => e.Mount)
        .HasConversion(
            v => v.ToString(),
            v => (EquineBeast)Enum.Parse(typeof(EquineBeast), v));
}

```

Note: A null value will never be passed to a value converter. This makes the implementation of conversions easier and allows them to be shared amongst nullable and non-nullable properties.

### 6.7.1. Built-in converters

<https://learn.microsoft.com/en-us/ef/core/modeling/value-conversions?tabs=data-annotations#built-in-converters>

EF Core ships with a set of pre-defined `ValueConverter<TModel,TProvider>` classes, found in the `Microsoft.EntityFrameworkCore.Storage.ValueConversion` namespace. In many cases EF will choose the appropriate built-in converter based on the type of the property in the model and the type requested in the database, as shown above for enums. For example, using `.HasConversion<int>()` on a bool property will cause EF Core to convert bool values to numerical zero and one values:

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder
        .Entity<User>()
        .Property(e => e.IsActive)
        .HasConversion<int>();
}

```

EF Core ships with a set of pre-defined ValueConverter classes. The full list of built-in converters is:

- Converting bool properties:
  - [BoolToStringConverter](#) - Bool to strings such as "N" and "Y"
  - [BoolToTwoValuesConverter<TProvider>](#) - Bool to any two values
  - [BoolToZeroOneConverter<TProvider>](#) - Bool to zero and one
- Converting byte array properties:
  - [BytesToStringConverter](#) - Byte array to Base64-encoded string
- Any conversion that requires only a type-cast
  - [CastingConverter<TModel,TProvider>](#) - Conversions that require only a type cast

- Converting char properties:
  - [CharToStringConverter](#) - Char to single character string
- Converting [DateTimeOffset](#) properties:
  - [DateTimeOffsetToBinaryConverter](#) - [DateTimeOffset](#) to binary-encoded 64-bit value
  - [DateTimeOffsetToBytesConverter](#) - [DateTimeOffset](#) to byte array
  - [DateTimeOffsetToStringConverter](#) - [DateTimeOffset](#) to string
- Converting [DateTime](#) properties:
  - [DateTimeToBinaryConverter](#) - [DateTime](#) to 64-bit value including [DateTimeKind](#)
  - [DateTimeToStringConverter](#) - [DateTime](#) to string
  - [DateTimeToTicksConverter](#) - [DateTime](#) to ticks
- Converting enum properties:
  - [EnumToNumberConverter<TEnum,TNumber>](#) - Enum to underlying number
  - [EnumToStringConverter<TEnum>](#) - Enum to string
- Converting [Guid](#) properties:
  - [GuidToBytesConverter](#) - [Guid](#) to byte array
  - [GuidToStringConverter](#) - [Guid](#) to string
- Converting [IPAddress](#) properties:
  - [IPAddressToBytesConverter](#) - [IPAddress](#) to byte array
  - [IPAddressToStringConverter](#) - [IPAddress](#) to string
- Converting numeric (int, double, decimal, etc.) properties:
  - [NumberToBytesConverter<TNumber>](#) - Any numerical value to byte array
  - [NumberToStringConverter<TNumber>](#) - Any numerical value to string
- Converting [PhysicalAddress](#) properties:
  - [PhysicalAddressToBytesConverter](#) - [PhysicalAddress](#) to byte array
  - [PhysicalAddressToStringConverter](#) - [PhysicalAddress](#) to string
- Converting string properties:
  - [StringToBoolConverter](#) - Strings such as "N" and "Y" to bool
  - [StringToBytesConverter](#) - String to UTF8 bytes
  - [StringToCharConverter](#) - String to character
  - [StringToDateConverter](#) - String to [DateTime](#)
  - [StringToDateOffsetConverter](#) - String to [DateTimeOffset](#)
  - [StringToEnumConverter<TEnum>](#) - String to enum
  - [StringToGuidConverter](#) - String to [Guid](#)
  - [StringToNumberConverter<TNumber>](#) - String to numeric type
  - [StringToTimeSpanConverter](#) - String to [TimeSpan](#)
  - [StringToUriConverter](#) - String to [Uri](#)
- Converting [TimeSpan](#) properties:
  - [TimeSpanToStringConverter](#) - [TimeSpan](#) to string

- [TimeSpanToTicksConverter](#) - [TimeSpan](#) to ticks
- Converting [Uri](#) properties:
  - [UriToStringConverter](#) - [Uri](#) to string

Notice that [EnumToStringConverter](#) is included in this list. This means that there is no need to specify the conversion explicitly, as shown above. Instead, just use the built-in converter:

```
var converter = new EnumToStringConverter<EquineBeast>();
```

```
modelBuilder
    .Entity<Rider>()
    .Property(e => e.Mount)
    .HasConversion(converter);
```

Note that all the built-in converters are stateless and so a single instance can be safely shared by multiple properties.

### 6.7.2. Pre-defined conversions

For common conversions for which a built-in converter exists there is no need to specify the converter explicitly. Instead, just configure which provider type should be used and EF will automatically use the appropriate built-in converter. [Enum to string conversions](#) are used as an example above, but EF will actually do this automatically if the provider type is configured:

```
modelBuilder
    .Entity<Rider>()
    .Property(e => e.Mount)
    .HasConversion<string>();
```

The same thing can be achieved by explicitly specifying the column type. For example, if the entity type is defined like so:

```
public class Rider
{
    public int Id { get; set; }

    [Column(TypeName = "varchar(24)")]
    public EquineBeast Mount { get; set; }
}
```

Then the enum values will be saved as strings in the database without any further configuration in [OnModelCreating](#).

## 6.8. Data seeding

<https://learn.microsoft.com/en-us/ef/core/modeling/data-seeding>

Data seeding is the process of populating a database with an initial set of data.

There are several ways this can be accomplished in EF Core:

- Model seed data
- Manual migration customization
- Custom initialization logic

### 6.8.1. Model seed data

In EF Core, seeding data can be associated with an entity type as part of the model configuration.

Then EF Core [migrations](#) can automatically compute what insert, update or delete operations need to be applied when upgrading the database to a new version of the model.

Important: any changes to the data performed outside of migrations might be lost or cause an error.

As an example, this will configure seed data for a Blog in OnModelCreating:

```
modelBuilder.Entity<Blog>().HasData(new Blog { BlogId = 1, Url = "http://sample.com" });
```

To add entities that have a relationship the foreign key values need to be specified:

```
modelBuilder.Entity<Post>().HasData(
    new Post { BlogId = 1, PostId = 1, Title = "First post", Content = "Test 1" });
```

Pro TIP: If you need to apply migrations as part of an automated deployment you can [create a SQL script](#) that can be previewed before execution. (questo aspetto verrà ripreso nella seconda parte del corso)

Alternatively, you can use context.Database.EnsureCreated() to create a new database containing the seed data, for example for a test database or when using the in-memory provider or any non-relational database. Note that if the database already exists, EnsureCreated() will neither update the schema nor seed data in the database. For relational databases you shouldn't call EnsureCreated() if you plan to use Migrations.

Durante il testing può essere utile un metodo che ripulisce il database e lo ricrea da capo con i valori inseriti nel modello tramite il modelBuilder:

<https://github.com/dotnet/EntityFramework.Docs/blob/main/samples/core/Modeling/ValueConversions/Program.cs>

```
protected static void CleanDatabase(DbContext context)
{
    ConsoleWriteLines("Deleting and re-creating database...");
    context.Database.EnsureDeleted();
    context.Database.EnsureCreated();
    ConsoleWriteLines("Done. Database is clean and fresh.");
}
```

Per approfondimenti si veda anche: <https://learn.microsoft.com/en-us/ef/core/modeling/data-seeding#limitations-of-model-seed-data>

## 6.9. Tipi di dato usati da SQLite

<https://www.sqlite.org/datatype3.html>

Most SQL database engines (every SQL database engine other than SQLite, as far as we know) uses static, rigid typing. With static typing, the datatype of a value is determined by its container - the particular column in which the value is stored.

SQLite uses a more general dynamic type system. In SQLite, the datatype of a value is associated with the value itself, not with its container. The dynamic type system of SQLite is backwards compatible with the more common static type systems of other database engines in the sense that SQL statements that work on statically typed databases should work the same way in SQLite. However, the dynamic typing in SQLite allows it to do things which are not possible in traditional rigidly typed databases.

### 6.9.1. Storage Classes and Datatypes

Each value stored in an SQLite database (or manipulated by the database engine) has one of the following storage classes:

**NULL.** The value is a NULL value.

**INTEGER.** The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.

**REAL.** The value is a floating point value, stored as an 8-byte IEEE floating point number.

**TEXT.** The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).

**BLOB.** The value is a blob of data, stored exactly as it was input.

A storage class is more general than a datatype. The INTEGER storage class, for example, includes 6 different integer datatypes of different lengths. This makes a difference on disk. But as soon as INTEGER values are read off of disk and into memory for processing, they are converted to the most general datatype (8-byte signed integer). And so for the most part, "storage class" is indistinguishable from "datatype" and the two terms can be used interchangeably.

Any column in an SQLite version 3 database, except an INTEGER PRIMARY KEY column, may be used to store a value of any storage class.

All values in SQL statements, whether they are literals embedded in SQL statement text or parameters bound to precompiled SQL statements have an implicit storage class. Under circumstances described below, the database engine may convert values between numeric storage classes (INTEGER and REAL) and TEXT during query execution.

## 6.10. Esercitazione con il database dei Romanzi

### prima parte

- Creare un progetto per un'applicazione Console .Net Core, chiamata Romanzi che utilizzi il Framework EF Core per implementare il database (**romanzi.db**) con le seguenti classi:

Autore(AutoreId, Nome, Cognome, Nazionalità)

Romanzo(RomanzoId, Titolo, AutoreId\*, AnnoPubblicazione)

Personaggio(PersonaggioId, Nome, RomanzoId\*, Sesso, Ruolo)

Dove:

L'attributo sottolineato rappresenta la Primary Key, mentre l'attributo con l'asterisco rappresenta una Foreign Key.

Assumere che:

Nome, Cognome, Nazionalità dentro la classe Autore siano di tipo string

Titolo dentro Romanzo sia di tipo string; AnnoPubblicazione dentro Romanzo sia di tipo int

Nome, Sesso, Ruolo dentro Personaggio siano di tipo string

Nota N1: creare il progetto dalla console con il comando:

```
dotnet new console -o Romanzi
cd Romanzi
```

Nota N2: installare le librerie per l'utilizzo di EF Core con SQLite:

```
dotnet add package Microsoft.EntityFrameworkCore.Sqlite
dotnet add package Microsoft.EntityFrameworkCore.Design
```

Nel caso di collegamento ad Internet mediante il proxy della scuola, impostare le variabili per il proxy prima di installare i pacchetti:

```
set http_proxy=proxy:3128
```

```
set https_proxy=proxy:3128
```

b) Aprire il progetto da Visual Studio:

```
start devenv Romanzi.csproj
```

- c) Creare la cartella Model con dentro le classi del modello dei dati
- d) Creare la cartella Data con al suo interno la classe RomanziContext

Nota N3: la classe **RomanziContext** discende da **DbContext** ed espone una property per ciascuna tabella del database, come esemplificato di seguito:

```
public DbSet<NomeClasse> NomeTabella { get; set; }
```

Nota N4: la classe **RomanziContext** potrebbe, se richiesto, implementare i metodi:

```
protected override void OnConfiguring(DbContextOptionsBuilder options)
```

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
```

e) Effettuare la migration del database e la creazione fisica del file di database di SQLite:

```
dotnet ef migrations add InitialMigration
```

```
dotnet ef database update
```

Nota N5: nel caso non fosse installato il tool ef per dotnet, installarlo mediante l'istruzione:

```
dotnet tool install --global dotnet-ef
```

f) Creare un metodo **public static void PopulateDB()** nella classe Program che inserisce:

Almeno 5 autori di nazionalità compresa tra "Americana", "Belga", "Inglese".

Almeno 10 romanzi degli autori precedentemente inseriti

Almeno 5 personaggi presenti nei romanzi precedentemente inseriti.

## **seconda parte**

Q1: creare un metodo che prende in input la nazionalità e stampa gli autori che hanno la nazionalità specificata

Q2: creare un metodo che prende in input il nome e il cognome di un autore e stampa tutti i romanzi di quell'autore

Q3: creare un metodo che prende in input la nazionalità e stampa quanti romanzi di quella nazionalità sono presenti nel database

Q4: creare un metodo che per ogni nazionalità stampa quanti romanzi di autori di quella nazionalità sono presenti nel database

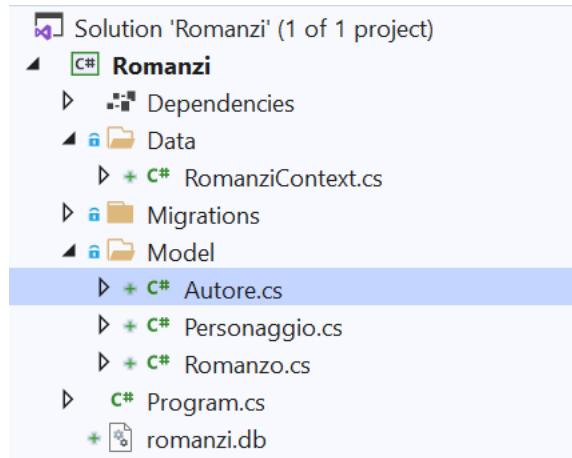
Q5: creare un metodo che stampa il nome dei personaggi presenti in romanzi di autori di una data nazionalità

## 6.10.1. Svolgimento

Seguendo le indicazioni del § 6.2 procediamo con la reazione di un'App Console (.Net Core) con il seguente Model:

### 6.10.1.1. Model

Creare la cartella Model con dentro i file relativi alle classi che rappresentano le relazioni del database.



```
//file: Autore.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Romanzi.Model
{
    public class Autore
    {
        public int AutoreId { get; set; }
        public string Nome { get; set; } = null!;
        public string Cognome { get; set; } = null!;
        public string? Nazionalità { get; set; }
        public List<Romanzo> Romanzi { get; set; } = null!;
    }
}

//file: Personaggio.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```

using System.Threading.Tasks;

namespace Romanzi.Model
{
    public class Personaggio
    {
        public int PersonaggioId { get; set; }
        public string Nome { get; set; } = null!;
        public int RomanzoId { get; set; }
        public Romanzo Romanzo { get; set; }=null!;
        public string? Sesso { get; set; }
        public string? Ruolo { get; set; }
    }
}

//file: Romanzo.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Romanzi.Model
{
    public class Romanzo
    {
        public int RomanzoId { get; set; }
        public string Titolo { get; set; } = null!;
        public int AutoreId { get; set; }
        public Autore Autore { get; set; }=null!;
        public int? AnnoPubblicazione { get; set; }
        public List<Personaggio> Personaggi { get; set; }=null!;
    }
}

```

### 6.10.1.2. Data

Creare la cartella Data con dentro i file RomanziContext.cs relativo alla definizione della classe RomanziContext.

```

using Microsoft.EntityFrameworkCore;
using Romanzi.Model;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Romanzi.Data
{
    public class RomanziContext:DbContext
    {
        public DbSet<Personaggio> Personaggi { get; set; } = null!;
        public DbSet<Autore> Autori { get; set; }=null!;
        public DbSet<Romanzo> Romanzi { get; set; } = null!;
        public string DbPath { get; }

        public RomanziContext()
        {

```

```

        var appDir = ApplicationContext.BaseDirectory;
        var path = Path.Combine(appDir, "../../romanzi.db");
        DbPath = path;
    }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite($"Data Source = {DbPath}");
    }
}

```

### 6.10.1.3. Migration e creazione del database

Dalla console aperta per creare il progetto, oppure dal terminal di Visual Studio (menu: View->Terminal) eseguiamo i seguenti comandi:

```
dotnet ef migrations add InitialMigration
```

```
dotnet ef database update
```

### 6.10.1.4. Applicazione

```
//File Program.cs

using Microsoft.EntityFrameworkCore;
using Romanzi.Data;
using Romanzi.Model;

//PopulateDB();
//Q1("Americana");
//Q2("Ernest", "Hemingway");
//Q3("Americana");
//Q4();
Q5("Inglese");

//Q1: creare un metodo che prende in input la nazionalità e
//stampa gli autori che hanno la nazionalità specificata
static void Q1(string nazionalità)
{
    using var db = new RomanziContext();
    Console.WriteLine($"Artisti di nazionalità {nazionalità}");
    db.Autori
        .Where(a => a.Nazionalità != null && a.Nazionalità.Contains(nazionalità))
        .ToList()
        .ForEach(a => Console.WriteLine($"{a.Nome} {a.Cognome}"));
}

//Q2: creare un metodo che prende in input il nome e il cognome di un autore
//e stampa tutti i romanzi di quell'autore
static void Q2(string nome, string cognome)
{
    using var db = new RomanziContext();
    //primo modo - uso di inner join
    Console.WriteLine("primo modo - uso di Join");
    db.Autori.
```

```

Where(a => a.Nome.Contains(nome) && a.Cognome.Contains(cognome)).//filtro per nome e cognome
Join(db.Romanzi,//Join con romanzi
a => a.AutoreId,
r => r.AutoreId,
(a, r) => new { a.Nome, a.Cognome, r.Titolo, r.AnnoPubblicazione }).//prendo le colonne che
interessano
ToList()//restituisco al client il risultato
ForEach(t => Console.WriteLine(t));//processo il risultato

//secondo metodo
Console.WriteLine("secondo modo - uso di navigation property a partire da Romanzi");
db.Romanzi.Where(r => r.Autore.Nome.Contains(nome) && r.Autore.Cognome.Contains(cognome))
    .ToList()//restituisco al client il risultato
    .ForEach(t => Console.WriteLine($"\\t{t.Titolo} {t.AnnoPubblicazione}")); //processo il
risultato

//terzo modo - uso di navigation property
Console.WriteLine("terzo modo - uso di navigation property");
db.Autori.
    Where(a => a.Nome.Contains(nome) && a.Cognome.Contains(cognome)).//filtro per nome e cognome
    Select(a => new { a, RomanziDiAutore = a.Romanzi } ).//uso la navigation property a.Romanzi
    ToList()//restituisco al client il risultato
    ForEach(t =>
    {
        //processo il risultato
        Console.WriteLine($"Autore = {t.a.Nome} {t.a.Cognome}");
        if (t.RomanziDiAutore != null)
        {
            foreach (var libro in t.RomanziDiAutore)
            {
                Console.WriteLine($"\\t{libro.Titolo} {libro.AnnoPubblicazione}");
            }
        }
    });
});

//quarto metodo
//attenzione al fatto che le collection property e le navigation property non sono
//automaticamente caricate nell'app, quando si esegue una query.

//In questo esempio se togliamo l'include e commentiamo i due modi precedenti di eseguire la query
//vediamo che il risultato non include i romanzi
Console.WriteLine("quarto modo - uso di navigation property e di Include");
db.Autori.
    Where(a => a.Nome.Contains(nome) && a.Cognome.Contains(cognome))//filtro per nome e cognome
    .Include(a =>a.Romanzi)//chiedo al provider di caricare i romanzi tramite la navigation
property
    .ToList()//restituisco al client il risultato
    ForEach(a =>
    {
        //processo il risultato
        Console.WriteLine($"Autore = {a.Nome} {a.Cognome}");
        if (a.Romanzi != null)
    });

```

```

        {
            foreach (var libro in a.Romanzi)
            {
                Console.WriteLine($"\\t{libro.Titolo} {libro.AnnoPubblicazione}");
            }
        }
    });

//Q3: creare un metodo che prende in input la nazionalità e stampa quanti romanzi di quella
nazionalità
//sono presenti nel database
static void Q3(string nazionalità)
{
    using var db = new RomanziContext();

    //primo modo - uso di join
    var numeroRomanzi = db.Autori.
        Where(a => a.Nazionalità != null && a.Nazionalità.Contains(nazionalità)).//filtro per nome e
cognome
        Join(db.Romanzi,//Join con romanzi
            a => a.AutoreId,
            r => r.AutoreId,
            (a, r) => new { r.RomanzoId }).Count();
    Console.WriteLine($"I romanzi di nazionalità {nazionalità} sono: {numeroRomanzi}");

    //secondo modo - uso di navigation property
    var numeroRomanzi2 = db.Romanzi.Where(r => r.Autore.Nazionalità != null &&
r.Autore.Nazionalità.Contains(nazionalità)).Count();
    Console.WriteLine("Secondo metodo");
    Console.WriteLine($"I romanzi di nazionalità {nazionalità} sono: {numeroRomanzi2}");
}

//Q4: creare un metodo che per ogni nazionalità stampa quanti romanzi di autori di quella
//nazionalità sono presenti nel database
static void Q4()
{
    //facciamo una join seguita da un raggruppamento
    //In questo esempio è molto importante effettuare la Select dopo la GroupBy, altrimenti si ottiene
un errore:
    //la Select serve a completare l'esecuzione della query all'interno del database con il calcolo di
eventuale
    //funzione di gruppo. Solo successivamente, con il ToList si restituisce il risultato della
GroupBy all'applicazione
    Console.WriteLine("Primo metodo - uso di Join e di Group By");
    using var db = new RomanziContext();
    db.Autori.Join(db.Romanzi,
        a => a.AutoreId,
        r => r.AutoreId,
        (a, r) => new { Nazionalita = a.Nazionalità, r.RomanzoId })
        .GroupBy(r => r.Nazionalita)//effettuo il raggruppamento
}

```

```

    .Select(g => new { Nazionalità = g.Key, NumeroRomanzi = g.Count() })/calcolo la funzione di
gruppo - Count in questo caso
    .ToList() //restituisco il risultato all'app
    .ForEach(t => Console.WriteLine($"Nazionalità = {t.Nazionalità}; numero romanzi =
{t.NumeroRomanzi}"));//processo il risultato

    //uso di navigation properties
    Console.WriteLine("\nSecondo metodo - uso di GroupBy con navigation properties");
    db.Romanzi
        .GroupBy(r => r.Autore.Nazionalità)//effettuo il raggruppamento
        .Select(g => new {Nazionalità = g.Key, NumeroRomanzi=g.Count()}//calcolo la funzione di
gruppo - Count in questo caso
        .ToList()//restituisco il risultato all'app
        .ForEach(t => Console.WriteLine($"Nazionalità = {t.Nazionalità}; numero romanzi =
{t.NumeroRomanzi}"));//processo il risultato
    }

//Q5: creare un metodo che stampa il nome dei personaggi presenti in romanzi di autori
//di una data nazionalità
static void Q5(string nazionalità)
{
    using var db = new RomanziContext();
    //primo modo - uso di join di join
    Console.WriteLine("Primo modo - uso di join");
    db.Autori.Where(a => a.Nazionalità != null && a.Nazionalità.Contains(nazionalità)).
        Join(db.Romanzi,
            a => a.AutoreId,
            r => r.AutoreId,
            (a, r) => new { r.RomanzoId }).
        Join(db.Personaggi,
            r => r.RomanzoId,
            p => p.RomanzoId,
            (r, p) => p).
        ToList().
        ForEach(p => Console.WriteLine($"{p.Nome} {p.Ruolo} {p.Sesso}"));

    //uso di navigation property
    Console.WriteLine("\nSecondo modo - uso di navigation property");
    db.Personaggi.Where(p => p.Romanzo.Autore.Nazionalità!=null &&
p.Romanzo.Autore.Nazionalità.Contains(nazionalità))
        .ToList()
        .ForEach(p => Console.WriteLine($"{p.Nome} {p.Ruolo} {p.Sesso}"));
}

static void PopulateDB()
{
    //Almeno 5 autori di nazionalità compresa tra "Americana", "Belga", "Inglese".
    List<Autore> autori = new List<Autore>()
    {
        new Autore(){AutoreId=1, Nome="Ernest", Cognome="Hemingway",

```

```

        Nazionalità="Americana"}, //AutoreId=1
        new Autore(){AutoreId=2, Nome="Philip", Cognome="Roth",
        Nazionalità="Americana"}, //AutoreId=2
        new Autore(){AutoreId=3, Nome="Thomas", Cognome="Owen",
        Nazionalità="Belga"}, //AutoreId=3
        new Autore(){AutoreId=4, Nome="William", Cognome="Shakespeare",
        Nazionalità="Inglese"}, //AutoreId=4
        new Autore(){AutoreId=5, Nome="Charles", Cognome="Dickens",
        Nazionalità="Inglese"}, //AutoreId=5
    };

using var db = new RomanziContext();
autori.ForEach(a => db.Add(a));
db.SaveChanges();
//Almeno 10 romanzi degli autori precedentemente inseriti
List<Romanzo> romanzi = new List<Romanzo>()
{
    new Romanzo(){RomanzoId=1, Titolo="For Whom the Bell Tolls",
    AnnoPubblicazione=1940, AutoreId=1}, //RomanzoId=1
    new Romanzo(){RomanzoId=2, Titolo="The Old Man and the Sea",
    AnnoPubblicazione=1952, AutoreId=1},
    new Romanzo(){RomanzoId=3, Titolo="A Farewell to Arms",
    AnnoPubblicazione=1929, AutoreId=1},
    new Romanzo(){RomanzoId=4, Titolo="Letting Go", AnnoPubblicazione=1962,
    AutoreId=2},
    new Romanzo(){RomanzoId=5, Titolo="When She Was Good",
    AnnoPubblicazione=1967, AutoreId=2},
    new Romanzo(){RomanzoId=6, Titolo="Destination Inconnue",
    AnnoPubblicazione=1942, AutoreId=3},
    new Romanzo(){RomanzoId=7, Titolo="Les Fruits de l'orage",
    AnnoPubblicazione=1984, AutoreId=3},
    new Romanzo(){RomanzoId=8, Titolo="Giulio Cesare",
    AnnoPubblicazione=1599, AutoreId=4},
    new Romanzo(){RomanzoId=9, Titolo="Otello", AnnoPubblicazione=1604,
    AutoreId=4},
    new Romanzo(){RomanzoId=10, Titolo="David Copperfield",
    AnnoPubblicazione=1849, AutoreId=5},
};
romanzi.ForEach(r => db.Add(r));
db.SaveChanges();
//Almeno 5 personaggi presenti nei romanzi precedentemente inseriti
List<Personaggio> personaggi = new List<Personaggio>()
{
    new Personaggio(){PersonaggioId=1, Nome="Desdemona",
    Ruolo="Protagonista", Sesso="Femmina", RomanzoId=9}, //PersonaggioId=1
    new Personaggio(){PersonaggioId=2, Nome="Jago", Ruolo="Protagonista",
    Sesso="Maschio", RomanzoId=9},
    new Personaggio(){PersonaggioId=3, Nome="Robert", Ruolo="Protagonista",
    Sesso="Maschio", RomanzoId=1},
    new Personaggio(){PersonaggioId=4, Nome="Cesare", Ruolo="Protagonista",
    Sesso="Maschio", RomanzoId=1},
}

```

```

        Sesso="Maschio", RomanzoId=8},
        new Personaggio(){PersonaggioId=5, Nome="David", Ruolo="Protagonista",
        Sesso="Maschio", RomanzoId=10}
    };
    personaggi.ForEach(p => db.Add(p));
    db.SaveChanges();
}

```

## 6.11. Esercitazione con il database dell'Università

Creare un'applicazione .Net Core per la gestione dei corsi di laurea universitari.

Usando il framework EF Core creare il database dei corsi di laurea e degli studenti, sapendo che nel model sono previste le seguenti classi:

**Studente** (Matricola, Nome, Cognome, CorsoLaureaId\*, AnnoNascita)

**CorsoLaurea** (CorsoLaureaId, TipoLaurea, Facoltà)

**Frequenta** (Matricola\*, CodCorso\*)

**Corso** (CodiceCorso, Nome, CodDocente\*)

**Docente**(CodDocente, Nome, Cognome, Dipartimento)

Dove:

**Facoltà** è un enumerativo che contiene i valori: Medicina, Ingegneria, MatematicaFisicaScienze, Economia

**Dipartimento** è un enumerativo che contiene i valori: Matematica, Fisica, Economia, IngegneriaCivile, IngegneriaInformatica, Medicina.

**TipoLaurea** è un enumerativo che contiene i valori: Triennale, Magistrale

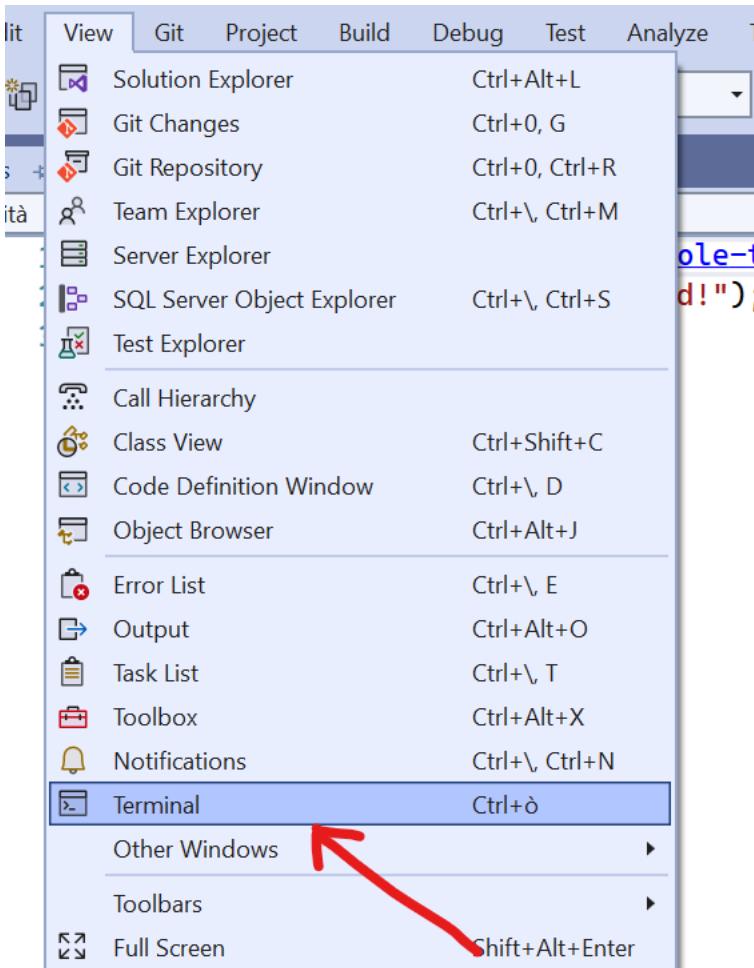
Nello schema precedente si è usata la convenzione di indicare la chiave primaria con attributi sottolineati, mentre la chiave esterna è indicata con un asterisco.

### 6.11.1. Svolgimento

Seguendo le indicazioni del § 6.2 procediamo con la reazione di un'App Console (.Net Core).

Per lo svolgimento di questo esercizio verrà mostrato un procedimento leggermente diverso rispetto a quello già visto negli esempi di partenza. Procediamo con la creazione di un progetto .NET Core Console direttamente da Visual Studio.

Dopo aver creato il progetto, apriamo la finestra Terminale (Terminal) di Visual Studio:



Viene mostrata la Developer Power Shell:

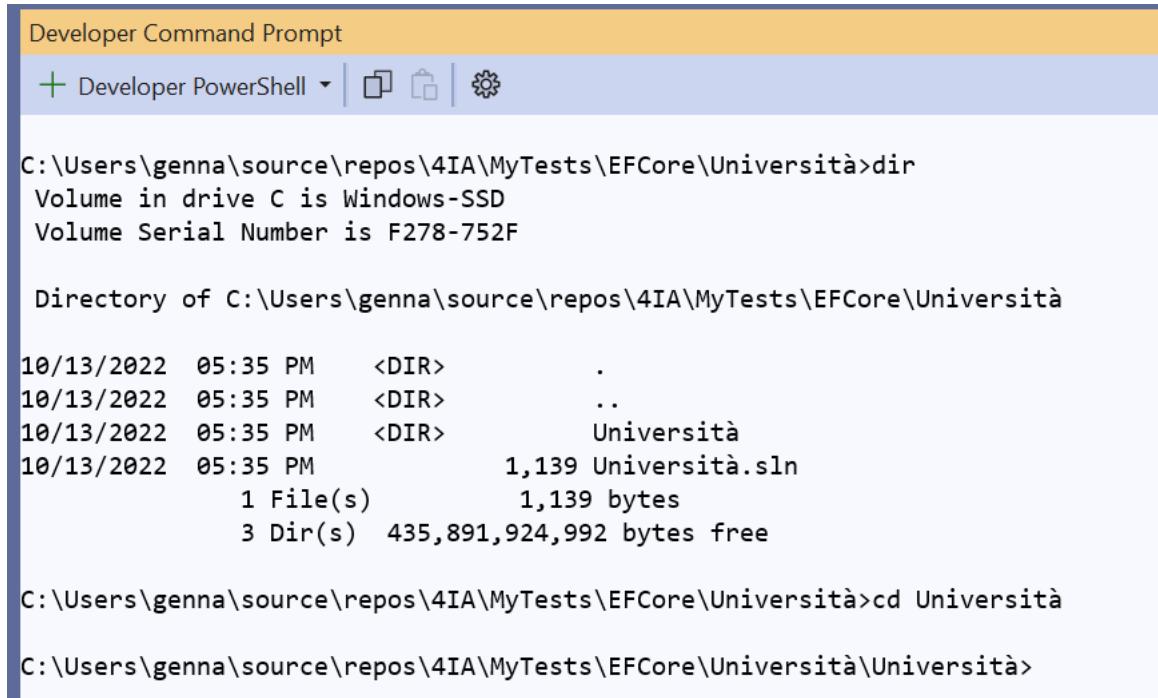
```
Developer PowerShell
+ Developer PowerShell | ⌂ ⌂ | ⚙
*****
** Visual Studio 2022 Developer PowerShell v17.3.4
** Copyright (c) 2022 Microsoft Corporation
*****
PS C:\Users\genna\source\repos\4IA\MyTests\EFCore\Università>
```

Passiamo dalla Developer Power Shell alla Developer Command Prompt:

The screenshot shows the 'Developer PowerShell' window with a dropdown menu open. The 'Developer Command Prompt' option is highlighted with a blue selection bar. A red arrow points from the 'Developer Command Prompt' entry to the 'Ubuntu' entry, which is also highlighted with a blue selection bar. Other options in the list include Developer PowerShell, Developer Powershell 7, Ubuntu, docker-desktop, and docker-desktop-data.

Ci spostiamo dalla cartella della soluzione a quella del progetto:

```
cd Università
```



```
Developer Command Prompt
+ Developer PowerShell | □ | ☰

C:\Users\genna\source\repos\4IA\MyTests\EFCore\Università>dir
 Volume in drive C is Windows-SSD
 Volume Serial Number is F278-752F

Directory of C:\Users\genna\source\repos\4IA\MyTests\EFCore\Università

10/13/2022  05:35 PM    <DIR>        .
10/13/2022  05:35 PM    <DIR>        ..
10/13/2022  05:35 PM    <DIR>        Università
10/13/2022  05:35 PM            1,139 Università.sln
                           1 File(s)      1,139 bytes
                           3 Dir(s)  435,891,924,992 bytes free

C:\Users\genna\source\repos\4IA\MyTests\EFCore\Università>cd Università

C:\Users\genna\source\repos\4IA\MyTests\EFCore\Università\Università>
```

A questo punto disponiamo di un terminale, posizionato sulla cartella di progetto, come quello visto negli esempi precedenti, ma integrato in Visual Studio.

Installiamo i pacchetti necessari per il funzionamento di EF Core:

```
dotnet add package Microsoft.EntityFrameworkCore.Sqlite
dotnet add package Microsoft.EntityFrameworkCore.Design
```

Nota: nel caso di accesso alla rete mediante il proxy bisogna impostare il proxy con le solite variabili di sessione:

```
set http_proxy=proxy:3128
set https_proxy=proxy:3128
```

Verifichiamo che nel file di configurazione del progetto, siano stati creati i riferimenti alle librerie installate:

```
<Project Sdk="Microsoft.NET.Sdk">

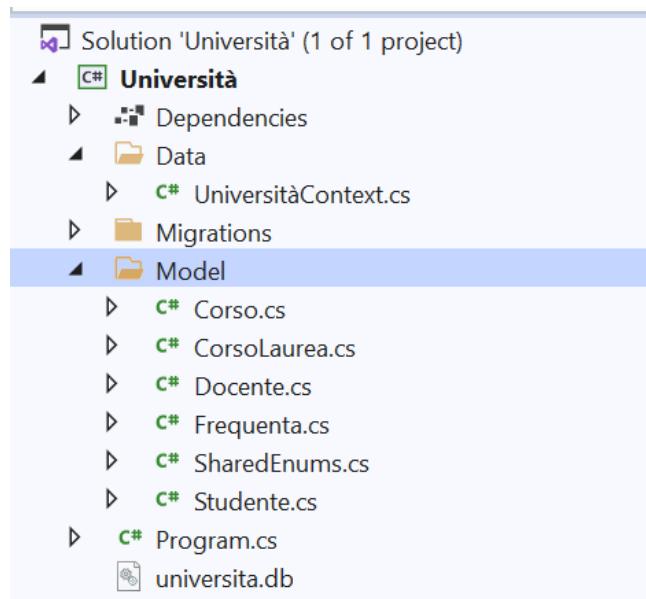
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="6.0.10">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="6.0.10" />
  </ItemGroup>

</Project>
```

### 6.11.1.1. Model

Creare la cartella Model con dentro i file relativi alle classi che rappresentano le relazioni del database.



```
//File SharedEnums.cs
```

```
namespace Università.Model
{
    public enum Facoltà
    {
        Medicina,
        Ingegneria,
        MatematicaFisicaScienze,
        Economia
    }
    public enum Dipartimento
    {
        Matematica,
        Fisica,
        Economia,
        IngegneriaCivile,
        IngegneriaInformatica,
        Medicina
    }
    public enum TipoLaurea
    {
        Triennale,
        Magistrale
    }
}
```

```
//File Studente.cs
```

```
using System.ComponentModel.DataAnnotations;

namespace Università.Model
{
    public class Studente
```

```

{
    //qui l'annotation è resa necessaria perché non si segue la convenzione sul
    nome della chiave
    [Key]
    public int Matricola { get; set; }
    public string Nome { get; set; } = null!;
    public string Cognome { get; set; } = null!;
    public int AnnoNascita { get; set; }

    //Chiave esterna
    public int CorsoLaureaId { get; set; }
    //NAVIGATION PROPERTY PER CHIAVE ESTERNA
    public CorsoLaurea CorsoLaurea { get; set; } = null!;

    //NAVIGATION PROPERTY PER MOLTI A MOLTI STUDENTI – CORSI
    public List<Frequenta> Frequenta { get; set; } =null!;
    public ICollection<Corso> Corsi { get; set; } = null!;

}
}

//File Docente.cs

using System.ComponentModel.DataAnnotations;

namespace Università.Model
{
    public class Docente
    {
        //qui l'annotation è resa necessaria perché non si segue la convenzione sul
        nome della chiave
        [Key]
        public int CodDocente { get; set; }
        public string Nome { get; set; } = null!;
        public string Cognome { get; set; } = null!;
        public Dipartimento Dipartimento { get; set; }
        //NAVIGATION PROPERTY
        public List<Corso> Corsi { get; set; }=null!;
    }
}

//File Corso.cs

using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;

namespace Università.Model
{
    public class Corso
    {
        //qui l'annotation è resa necessaria perché non si segue la convenzione sul
        nome della chiave
        [Key]
        public int CodiceCorso { get; set; }
        public string Nome { get; set; } = null!;
        //qui l'annotation è resa necessaria perché non si segue la convenzione sul
        nome della chiave
        public int? CodDocente { get; set; }
    }
}

```

```

//NAVIGATION PROPERTY
[ForeignKey("CodDocente")]
public Docente? Docente { get; set; }

//NAVIGATION PROPERTY PER MOLTI A MOLTI STUDENTI - CORSI
public List<Frequenta> Frequenta { get; set; } = null!;
public ICollection<Studente> Studenti { get; set; } = null!;

}

}

//File CorsoLaurea.cs

namespace Università.Model
{
    public class CorsoLaurea
    {
        //Chiave primaria
        public int CorsoLaureaId { get; set; }

        public TipoLaurea TipoLaurea { get; set; }

        public Facoltà Facoltà { get; set; }

        //NAVIGATION PROPERTY
        public List<Studente> Studenti { get; set; } = null!;
    }
}

//File Frequenta.cs

namespace Università.Model
{
    public class Frequenta
    {
        //la chiave primaria è composta da Matricola e CodCorso tramite Fluent API
        //le chiavi esterne vanno configurate mediante Fluent API
        public int Matricola { get; set; }

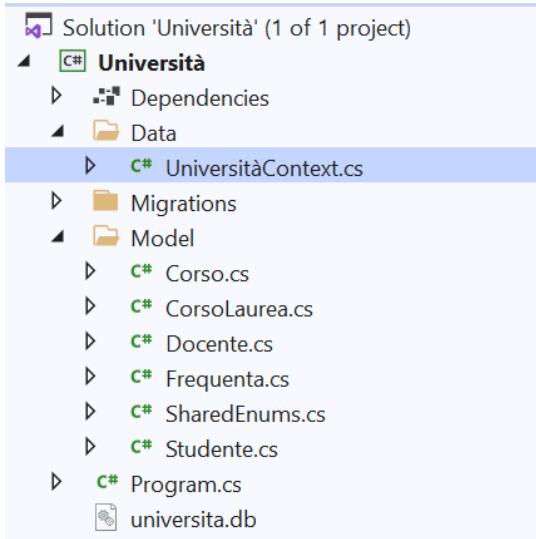
        //NAVIGATION PROPERTY
        public Studente Studente { get; set; } = null!;
        public int CodCorso { get; set; }

        //NAVIGATION PROPERTY
        public Corso Corso { get; set; } = null!;
    }
}

```

### 6.11.1.2. Data

Creare la cartella Data con dentro i file UniversitàContext.cs relativo alla definizione della classe UniversitàContext.



```
//File UniversitàContext.cs

using Microsoft.EntityFrameworkCore.Storage.ValueConversion;
using Microsoft.EntityFrameworkCore;

using Università.Model;

namespace Università.Data
{
    public class UniversitàContext : DbContext
    {
        //creazione delle tabelle
        public DbSet<Studente> Studenti { get; set; } = null!;
        public DbSet<CorsoLaurea> CorsiLaurea { get; set; } = null!;
        public DbSet<Frequenta> Frequenta { get; set; } = null!;
        public DbSet<Corso> Corsi { get; set; } = null!;
        public DbSet<Docente> Docenti { get; set; } = null!;
        public string DbPath { get; }

        public UniversitàContext()
        {
            var dir = ApplicationContext.BaseDirectory;
            var path = Path.Combine(dir, "../../universita.db");
            DbPath = path;
        }

        protected override void OnConfiguring(DbContextOptionsBuilder options)
            => options.UseSqlite($"Data Source={DbPath}");

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            //per gestire la conversione da Enumerativo a string:
https://learn.microsoft.com/en-us/ef/core/modeling/value-conversions
            var converterTipoLaurea = new EnumToStringConverter<TipoLaurea>();
            var converterFacoltà = new EnumToStringConverter<Facoltà>();
            var converterDipartimento = new EnumToStringConverter<Dipartimento>();
            modelBuilder
                .Entity<CorsoLaurea>()
                .Property(cl => cl.TipoLaurea)
                .HasConversion(converterTipoLaurea);
            modelBuilder
                .Entity<CorsoLaurea>()
                .Property(cl => cl.Facoltà)
                .HasConversion(converterFacoltà);
            modelBuilder
                .Entity<Docente>()
        }
    }
}
```

```

        .Property(d => d.Dipartimento)
        .HasConversion(converterDipartimento);

    //una tabella di collegamento molti a molti deve essere configurata
    tramite Fluent API

    ////la chiave primaria
    //modelBuilder.Entity<Frequenta>()
    //    .HasKey(f => new { f.Matricola, f.CodCorso });
    ////chiave esterna su Studente
    //modelBuilder.Entity<Frequenta>()
    //    .HasOne(fr => fr.Studente)
    //    .WithMany(s => s.Frequenta)
    //    .HasForeignKey(fr => fr.Matricola);
    ////chiave esterna su Corso
    //modelBuilder.Entity<Frequenta>()
    //    .HasOne(fr => fr.Corso)
    //    .WithMany(c => c.Frequenta)
    //    .HasForeignKey(fr => fr.CodCorso);

    modelBuilder.Entity<Corso>()
        .HasMany(c => c.Studenti)
        .WithMany(s => s.Corsi)
        .UsingEntity<Frequenta>
        (
            //chiave esterna su Studente
            j => j
            .HasOne(fr => fr.Studente)
            .WithMany(s => s.Frequenta)
            .HasForeignKey(fr => fr.Matricola),
            //chiave esterna su Corso
            j => j
            .HasOne(fr => fr.Corso)
            .WithMany(c => c.Frequenta)
            .HasForeignKey(fr => fr.CodCorso),
            //definizione di chiave primaria
            j=> j
            .HasKey(fr => new { fr.Matricola, fr.CodCorso})
        );
    }

}

}

```

### 6.11.1.3. Migration e creazione del database

Effettuare la migrazione e la creazione fisica del database con i comandi

**dotnet ef migrations add InitialMigrarte**

**dotnet ef database update**

```
C:\Users\genna\source\repos\4IA\MyTests\EFCore\Università\Università>dotnet ef migrations add InitialMigrarte
Build started...
Build succeeded.
Done. To undo this action, use 'ef migrations remove'

C:\Users\genna\source\repos\4IA\MyTests\EFCore\Università\Università>dotnet ef database update
Build started...
Build succeeded.
Applying migration '20221013173317_InitialMigrarte'.
Done.

C:\Users\genna\source\repos\4IA\MyTests\EFCore\Università\Università>
```

#### 6.11.1.4. Applicazione

In questo caso conviene scrivere la classe Program e il metodo Main, poiché ci sono molti metodi che vengono richiamati.

```
//File Program.cs

using Università.Data;
using Università.Model;

namespace Università;

class Program
{
    /// <summary>
    /// Popola il database
    /// </summary>
    public static void PopulateDB()
    {
        //1) inserisco istanze nelle tabelle che non hanno chiavi esterne -->CorsoDiLaurea, Docente
        //creo una lista di CorsoDiLaurea e di Docente
        List<Docente> docenti = new List<Docente>()
        {
            new Docente(){CodDocente=1, Cognome="Malafronte",
Nome="Gennaro",Dipartimento=Dipartimento.IngegneriaInformatica },
            new Docente(){CodDocente=2, Cognome="Rossi", Nome="Mario",
Dipartimento=Dipartimento.Matematica},
            new Docente(){CodDocente=3, Cognome="Verdi", Nome="Giuseppe",
Dipartimento=Dipartimento.Fisica},
            new Docente(){CodDocente=4, Cognome= "Smith", Nome="Albert",
Dipartimento=Dipartimento.Economia}
        };
        List<CorsoLaurea> corsiDiLaurea = new List<CorsoLaurea>()
        {
            new CorsoLaurea(){CorsoLaureaId = 1,TipoLaurea=TipoLaurea.Magistrale,
Facoltà=Facoltà.Ingegneria},
            new CorsoLaurea(){CorsoLaureaId = 2,TipoLaurea=TipoLaurea.Triennale,
Facoltà=Facoltà.MatematicaFisicaScienze},
            new CorsoLaurea(){CorsoLaureaId = 3,TipoLaurea=TipoLaurea.Magistrale,
Facoltà=Facoltà.Economia},
        };
        using (var db = new UniversitàContext())
        {
            docenti.ForEach(d => db.Add(d));
            corsiDiLaurea.ForEach(cl => db.Add(cl));
            db.SaveChanges();
        }
        //2) inserisco altre istanze: Inserisco istanze di Corso e di Studente
        List<Corso> corsi = new List<Corso>()
        {
            new Corso(){CodiceCorso=1,Nome="Fondamenti di Informatica 1", CodDocente=1},
            new Corso(){CodiceCorso=2,Nome="Analisi Matematica 1", CodDocente=2},
            new Corso(){CodiceCorso=3,Nome="Programmazione Orientata agli Oggetti", CodDocente=3},
            new Corso(){CodiceCorso=4,Nome="Database", CodDocente=4},
            new Corso(){CodiceCorso=5,Nome="Progettazione di Sistemi Distribuiti", CodDocente=1},
            new Corso(){CodiceCorso=6,Nome="Introduzione alla Ricerca Operativa", CodDocente=2},
            new Corso(){CodiceCorso=7,Nome="Analisi dei Dati", CodDocente=3},
            new Corso(){CodiceCorso=8,Nome="Introduzione alla Bioinformatica", CodDocente=4}
        };
        foreach (var corso in corsi)
        {
            db.Add(corso);
        }
        db.SaveChanges();
    }
}
```

```

        new Corso(){CodiceCorso=2, Nome="Analisi Matematica 1", CodDocente=2},
        new Corso(){CodiceCorso=3, Nome="Fisica 1", CodDocente=3},
        new Corso(){CodiceCorso=4, Nome="Microeconomia 1", CodDocente=4}
    };
    List<Studente> studenti = new List<Studente>()
    {
        new Studente(){Matricola=1, Nome="Giovanni", Cognome="Casiraghi", CorsoLaureaId=1,
        AnnoNascita=2000},
        new Studente(){Matricola=2, Nome="Alberto", Cognome="Angela", CorsoLaureaId=2,
        AnnoNascita=1999},
        new Studente(){Matricola=3, Nome="Piero", Cognome="Gallo", CorsoLaureaId=3,
        AnnoNascita=2000}
    };
    using (var db = new UniversitàContext())
    {
        corsi.ForEach(c => db.Add(c));
        studenti.ForEach(s => db.Add(s));
        db.SaveChanges();
    }
    //4) inserisco Frequenta - è la tabella molti a molti
    List<Frequenta> frequenta = new List<Frequenta>()
    {
        new Frequenta(){Matricola=1, CodCorso=1}, // Giovanni Casiraghi frequenta il corso di
        Fondamenti di Informatica 1
        new Frequenta(){Matricola=1, CodCorso=2}, // Giovanni Casiraghi frequenta il corso di
        Analisi Matematica 1
        new Frequenta(){Matricola=2, CodCorso=2},
        new Frequenta(){Matricola=2, CodCorso=3},
        new Frequenta(){Matricola=3, CodCorso=4}
    };
    using (var db = new UniversitàContext())
    {
        frequenta.ForEach(f => db.Add(f));
        db.SaveChanges();
    }
}

/// <summary>
/// Stampa a Console l'elenco degli studenti
/// </summary>
public static void PrintStudents()
{
    using var db = new UniversitàContext();
    //leggo gli studenti
    List<Studente> studenti = db.Studenti.ToList();
    //stampo gli studenti
    studenti.ForEach(s => Console.WriteLine($"Matricola = {s.Matricola}, Nome = {s.Nome}, Cognome
    = {s.Cognome}"));
}

```

```

}

/// <summary>
/// Stampa a console l'elenco dei corsi
/// </summary>
public static void PrintCourses()
{
    //leggo gli studenti
    using var db = new UniversitàContext();
    List<Corso> corsi = db.Corsi.ToList();
    corsi.ForEach(s => Console.WriteLine($"CodCorso = {s.CodiceCorso}, Nome = {s.Nome}, CodDocente
= {s.CodDocente}"));

}

/// <summary>
/// Un metodo che stampa i corsi seguiti da uno studente di cui si conosce nome e cognome
/// </summary>
/// <param name="nomeStudente">Nome dello studente</param>
/// <param name="cognomeStudente">Nome dello studente</param>
public static void PrintCorsidiStudente(string nomeStudente, string cognomeStudente)
{
    //trovare i corsi seguiti da uno studente - doppio join
    using var db = new UniversitàContext();
    //attenzione - potrebbero esserci casi di omonimia!
    var corsiFrequentatiDaStudente = db.Studenti.
        Where(s => s.Nome.ToUpper().Contains(nomeStudente.ToUpper()) &&
s.Cognome.ToUpper().Contains(cognomeStudente.ToUpper()).
        Join(db.Frequenta,
            s => s.Matricola,
            f => f.Matricola,
            (s, f) => new { f.CodCorso });
    //Console.WriteLine($"nQuery eseguita sul database per {nameof(corsiFrequentatiDaStudente)} =
{corsiFrequentatiDaStudente.ToString()}\n");
    var dettaglioCorsiFrequentati = corsiFrequentatiDaStudente.Join(db.Corsi,
        cf => cf.CodCorso,
        c => c.CodiceCorso,
        (cf, c) => c);
    //Console.WriteLine($"nQuery eseguita sul database per {nameof(dettaglioCorsiFrequentati)} =
{dettaglioCorsiFrequentati.ToString()}\n");
    dettaglioCorsiFrequentati.ToList().ForEach(c => Console.WriteLine($"Nome Corso = {c.Nome}"));

    Console.WriteLine("Altro metodo - uso di navigation property");
    //altro metodo - navigation property
    db.Studenti
        .Where(s => s.Nome.ToUpper().Contains(nomeStudente.ToUpper()) &&
s.Cognome.ToUpper().Contains(cognomeStudente.ToUpper()))
        .SelectMany(x => x.Corsi) //how to flatten a list of list in C#:
https://stackoverflow.com/questions/1145558/linq-list-of-lists-to-single-list
        .ToList()
}

```

```

        .ForEach(c => Console.WriteLine($"Nome Corso = {c.Nome}"));

    }

    /// <summary>
    /// Stampa a console il numero di corsi seguiti da uno studente
    /// </summary>
    /// <param name="codStudente">Codice dello studente di cui si vuole contare i corsi</param>
    public static void PrintNumeroCorsiDiStudente(int codStudente)
    {
        using var db = new UniversitàContext();
        //uso della tabella di collegamento molti a molti - Frequentia in questo caso
        Console.WriteLine("Uso della tabella di collegamento molti a molti - Frequentia");
        var numeroCorsiFrequentatiDaStudente = db.Frequentia.Where(f => f.Matricola ==
codStudente).Count();
        Console.WriteLine($"Numero corsi frequentati dallo studente con Matricola = {codStudente} ->
numero corsi: {numeroCorsiFrequentatiDaStudente}");

        //Uso di collection property
        Console.WriteLine("Uso di collection property");
        db.Studenti.
            Where(s => s.Matricola == codStudente)
            .Select(s => new {s.Matricola, s.Nome, s.Cognome, NumeroCorsi=s.Corsi.Count})
            .ToList()
            .ForEach(s => Console.WriteLine($"Numero corsi frequentati dallo studente matricola
{s.Matricola} {s.Nome} {s.Cognome} -> numero corsi: {s.NumeroCorsi}"));

        //Uso di Entry per recuperare dati di una collection collegata a un oggetto già caricato in
memoria
        Console.WriteLine("Uso di Entry per recuperare dati di una collection collegata a un oggetto
già caricato in memoria");
        var studente = db.Studenti.Where(s => s.Matricola == codStudente).FirstOrDefault();
        if(studente != null)
        { //come effettuare una query su una collection property:
https://www.entityframeworktutorial.net/EntityFramework4.3/explicit-loading-with-dbcontext.aspx
        var numeroCorsiFrequentatiDaStudente2 = db.Entry(studente).Collection(s =>
s.Corsi).Query().Count();
        Console.WriteLine($"Numero corsi frequentati dallo studente con Matricola = {codStudente} -
> numero corsi: {numeroCorsiFrequentatiDaStudente2}");
    }

}

    /// <summary>
    /// Stampa a console il numero di corsi seguiti da uno studente
    /// </summary>
    /// <param name="nomeStudente"></param>
    /// <param name="cognomeStudente"></param>

```

```

public static void PrintNumeroCorsiDiStudente(string nomeStudente, string cognomeStudente)
{
    using var db = new UniversitàContext();
    //https://docs.microsoft.com/en-us/ef/core/miscellaneous/collations-and-case-sensitivity
    //https://medium.com/@bridgesquared/efcore-sqlite-case-insensitive-order-21371b256e5
    //https://github.com/dotnet/efcore/issues/8033
    //per SQLite la collation predefinita è case sensitive

    //versione con join e group by
    db.Studenti.
        Where(s => s.Nome.ToUpper().Contains(nomeStudente.ToUpper()) &&
s.Cognome.ToUpper().Contains(cognomeStudente.ToUpper())).
        Join(db.Frequenta,
        s => s.Matricola,
        f => f.Matricola,
        (s, f) => new { s.Matricola, s.Nome, s.Cognome, f.CodCorso }).
        GroupBy(t => t.Matricola).
        Select(g => new { Matricola=g.Key, Nome = g.First().Nome, Cognome = g.First().Cognome,
NumeroCorsi=g.Count()}).
        ToList().
        .ForEach(s => Console.WriteLine($"Numero corsi frequentati dallo studente matricola
{s.Matricola} {s.Nome} {s.Cognome} -> numero corsi: {s.NumeroCorsi}"));

    //Uso di collection property
    //poiché vengono dati in input nome e cognome, ci potrebbero essere più studenti con lo stesso
    nome e cognome
    //quindi il risultato dovrebbe riportare, per ciascuno il numero di corsi seguiti

    Console.WriteLine("Uso di collection property");
    db.Studenti.
        Where(s => s.Nome.ToUpper().Contains(nomeStudente.ToUpper()) &&
s.Cognome.ToUpper().Contains(cognomeStudente.ToUpper())).
        Select(s => new {s.Matricola, s.Nome, s.Cognome, NumeroCorsi = s.Corsi.Count }).
        ToList().
        .ForEach(s => Console.WriteLine($"Numero corsi frequentati dallo studente matricola
{s.Matricola} {s.Nome} {s.Cognome} -> numero corsi: {s.NumeroCorsi}"));

}

```

/// <summary>  
 /// Modifica il docente di un corso di cui è noto l'id  
 /// </summary>  
 /// <param name="codCorso">codice del corso</param>  
 /// <param name="nuovoCodDocente">nuovo codice del docente</param>

```

public static void ModificaDocenteCorso(int codCorso, int nuovoCodDocente)
{
    using var db = new UniversitàContext();
    //accedo al corso
    var corso = db.Corsi.Find(codCorso);
    //accedo al docente per verificare che esiste quel docente
}

```

```

var docente = db.Docenti.Find(nuovoCodDocente);
//se esiste il docente con il nuovoCodDocente aggiorno il corso
if (docente != null && corso != null)
{
    corso.CodDocente = nuovoCodDocente;
    db.SaveChanges();
    Console.WriteLine("Aggiornamento effettuato");
}
else if (docente == null)
{
    Console.WriteLine("Impossibile aggiornare, il docente con il codice specificato non esiste");
}
else if (corso == null)
{
    Console.WriteLine("Impossibile aggiornare, il corso il codice specificato non esiste");
}

/// <summary>
/// Per ogni studente stampa il numero di corsi frequentati
/// </summary>
public static void PrintNumeroCorsiFrequentatiPerStudente()
{
    //raggruppo su Frequenta per Matricola e poi faccio la join con Studente per avere i dati di ciascuno studente
    using var db = new UniversitàContext();

    //MOLTO IMPORTANTE: https://docs.microsoft.com/en-us/ef/core/what-is-new/ef-core-3.x/breaking-changes
    //https://docs.microsoft.com/en-us/ef/core/what-is-new/ef-core-3.x/breaking-changes#linq-queries-are-no-longer-evaluated-on-the-client
    //https://stackoverflow.com/questions/58138556/client-side-groupby-is-not-supported
    //https://stackoverflow.com/a/60778664
    //The LINQ GroupBy is much different from the SQL GROUP BY statement:
    //LINQ just divides the underlying collection into chunks depending on a key,
    //while SQL additionally applies an aggregation function to condense each of these chunks down into a single value.
    var grouped = db.Frequenta.GroupBy(f => f.Matricola).Select(e => new { e.Key, Count = e.Count() });
    //vedere anche
    //https://www.thinktecture.com/en/entity-framework-core/hidden-group-by-capabilities-in-3-0-part-1/
    //https://www.thinktecture.com/en/entity-framework-core/hidden-group-by-capabilities-in-3-0-part-2/

    //https://stackoverflow.com/questions/37527783/get-sql-code-from-an-entity-framework-core-iqueryable
    //https://stackoverflow.com/a/51583047
    //richiede using Microsoft.EntityFrameworkCore;

```

```

//Console.WriteLine($"\\nQuery eseguita sul database per {nameof(grouped)} =
{grouped.ToQueryString()}\\n");

var result = grouped.Join(db.Studenti,
    group => group.Key,
    s => s.Matricola,
    (group, s) => new { s.Matricola, s.Nome, s.Cognome, NumeroCorsiFrequentati = group.Count
});
//Console.WriteLine($"\\nQuery eseguita sul database per {nameof(result)} =
{result.ToQueryString()}\\n");
//stampa risultato
result.ToList().ForEach(r => Console.WriteLine(r));

////IL SEGUENTE APPROCCIO NON FUNZIONA DA EC 3.X IN AVANTI
////il problema si ha quando si tenta di passare dalla GroupBy al ToList
//var grouped2 = db.Frequenta.GroupBy(f => f.Matricola);
//var result2 = grouped2.Join(db.Studenti,
//    group => group.Key,
//    s => s.Matricola,
//    (group, s) => new { s.Matricola, s.Nome, s.Cognome, NumeroCorsiFrequentati =
group.Count() });
//stampa risultato
////qui ho un errore perché si tenta di trasformare la group by dell'SQL (aggregazione) in un
collection di collection
//result2.ToList().ForEach(r => Console.WriteLine(r));
}

static void Main(string[] args)
{
    //1) inserisco istanze nelle tabelle che non hanno chiavi esterne -->CorsoDiLaurea, Docente
    //2) inserisco altre istanze: Inserisco istanze di Corso
    //3) inserisco Studente
    //4) inserisco Frequenta
    PopulateDB(); //va eseguito solo la prima volta
    PrintStudents();

    //PrintCourses();
    //ModificaDocenteCorso(1, 1);
    //PrintCourses();

    //PrintNumeroCorsiDiStudente(1);
    //PrintNumeroCorsiDiStudente("Giovanni", "Casiraghi");

    //PrintNumeroCorsiDiStudente(3);
    //PrintNumeroCorsiDiStudente("Piero", "Gallo");

    //PrintNumeroCorsiFrequentatiPerStudente();
}

```

```

//PrintCorsiDiStudente("Piero", "Gallo");
Console.WriteLine("Finito!");

}
}

```

## 6.12. EF Core 3.x e successive versioni

<https://docs.microsoft.com/en-us/ef/core/what-is-new/ef-core-3.0/>

<https://stackoverflow.com/questions/58074844/ef-linq-error-after-change-from-dotnet-core-2-2-6-to-3-0-0>

<https://docs.microsoft.com/en-us/ef/core/querying/client-eval>

Prior to version 3.0, Entity Framework Core supported client evaluation anywhere in the query. For more information, see the [previous versions section](#).

<https://docs.microsoft.com/en-us/ef/core/querying/how-query-works>

<https://stackoverflow.com/questions/17968469/whats-the-differences-between-tolist-asenumerable-asqueryable>

<https://docs.microsoft.com/en-us/ef/core/what-is-new/ef-core-3.0/breaking-changes#linq-queries-are-no-longer-evaluated-on-the-client>

### Old behavior

Before 3.0, when EF Core couldn't convert an expression that was part of a query to either SQL or a parameter, it automatically evaluated the expression on the client. By default, client evaluation of potentially expensive expressions only triggered a warning.

### New behavior

Starting with 3.0, EF Core only allows expressions in the top-level projection (the last `Select()` call in the query) to be evaluated on the client. When expressions in any other part of the query can't be converted to either SQL or a parameter, an exception is thrown.

Con EF Core da versione 3.x in avanti le query LINQ vengono mappate direttamente su query SQL nel database sottostante, ciò vuol dire che le query LINQ vengono eseguite direttamente sul server (database) e non sul client (applicazione C#). Alcune operazioni che normalmente sarebbero possibili sulle collection in memoria non lo sono quando vengono tradotte in un'equivalente query SQL. Questo non è dovuto ad una limitazione di EF Core, ma al fatto che le operazioni possibili sui database in SQL sono in alcuni casi più limitate. In alcuni casi, quando si vuole forzare la manipolazione dei dati sul client (all'interno dell'applicazione C# invece che sul database) è possibile utilizzare

### 6.12.1. Corso su EF Core 3

<https://channel9.msdn.com/Series/Entity-Framework-Core-101/Getting-Started-with-Entity-Framework-Core>

### 6.12.2. EF Core con altri database (MySQL, MariaDB, etc.)

Database Providers:

<https://docs.microsoft.com/en-us/ef/core/what-is-new/nuget-packages#database-providers>

Documentazione ufficiale di MySQL:

<https://dev.mysql.com/doc/connector-net/en/connector-net-entityframework-core.html>

Pomelo provider per MySQL e MariaDB:

<https://github.com/PomeloFoundation/Pomelo.EntityFrameworkCore.MySql>

<https://github.com/PomeloFoundation/Pomelo.EntityFrameworkCore.MySql/wiki/Configuration-Options#optional-settings>

<https://www.tektutorialshub.com/entity-framework-core-tutorial/>

<https://www.tektutorialshub.com/entity-framework-core/using-mysql-mariadb-in-entity-framework-core/>

<https://riptutorial.com/ef-core-providers/learn/1000016/pomelo-mysql>

<https://stackoverflow.com/questions/45893732/how-do-you-show-underlying-sql-query-in-ef-core>

## 7. Multithreading

<https://docs.microsoft.com/it-it/dotnet/standard/threading/>

<https://kudchikarsk.com/c-multithreading/>

<http://www.albahari.com/threading/>

### 7.1. Thread e threading

<https://docs.microsoft.com/en-us/dotnet/standard/threading/threads-and-threading>

Il multithreading consente di aumentare la velocità di risposta dell'applicazione e, se l'applicazione viene eseguita in un sistema multiprocessore o multicore, consente di aumentare la velocità effettiva.

**Un processo è un programma in esecuzione.** Un sistema operativo usa i processi per separare le applicazioni che vengono eseguite. **Un thread è l'unità di base in cui un sistema operativo alloca il tempo del processore.** Ogni thread ha una priorità di pianificazione e un insieme di strutture usate dal sistema per salvare il contesto del thread quando viene sospesa l'esecuzione del thread. Nel contesto del thread sono presenti tutte le informazioni necessarie per riprendere senza problemi l'esecuzione, incluso il set di registri della CPU del thread e lo stack. Nel contesto di un processo possono essere eseguiti più thread. Tutti i thread di un processo ne condividono lo spazio degli indirizzi virtuali. Un thread può eseguire qualsiasi parte del codice del programma, comprese le parti attualmente eseguite da un altro thread.

Per impostazione predefinita, viene avviato un programma .NET con un thread singolo, spesso chiamato thread primario. Tuttavia, è possibile creare thread aggiuntivi per eseguire il codice in parallelo o contemporaneamente al thread primario. Questi thread sono spesso chiamati thread di lavoro.

L'uso di più thread consente di aumentare la velocità di risposta dell'applicazione e di usare un sistema multiprocessore o multicore per aumentare la velocità effettiva dell'applicazione.

Si consideri un'applicazione desktop in cui il thread primario è responsabile degli elementi dell'interfaccia utente e risponde alle azioni dell'utente. Usare i thread di lavoro per eseguire operazioni che richiedono molto tempo e che altrimenti occuperebbero il thread primario e impedirebbero all'interfaccia utente di rispondere. È anche possibile usare un thread dedicato per fare in modo che la comunicazione di rete o del dispositivo sia maggiormente reattiva ai messaggi in ingresso o agli eventi.

Se il programma esegue operazioni che possono essere eseguite in parallelo, il tempo di esecuzione totale può essere ridotto eseguendo queste operazioni in thread separati ed eseguendo il programma in un sistema multiprocessore o multicore. In un sistema di questo tipo, l'uso del multithreading potrebbe aumentare la velocità effettiva e la velocità di risposta.

#### 7.1.1. What Are Threads in C#?

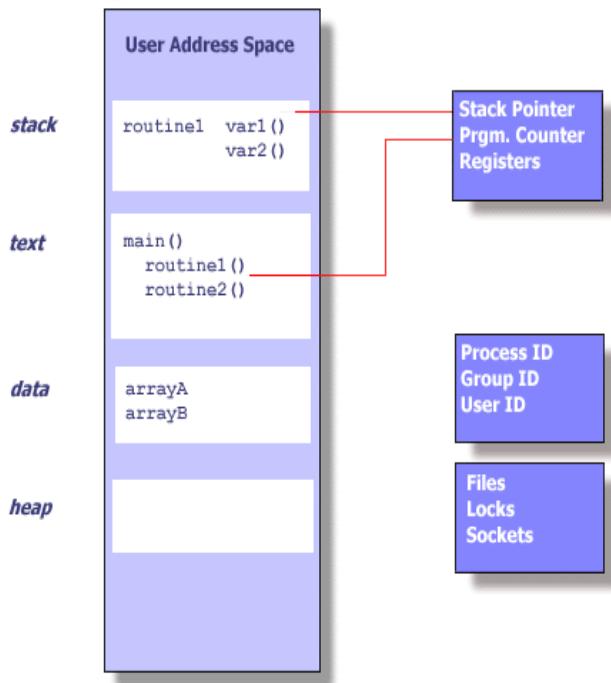
In CLR or Windows environment each program you run creates a virtual address space and known as process. Contents of a process is not addressable directly to another process.

Each process has its own thread(s) and this thread has access to all data in that process.

So, a process in its execution lifetime has this main thread that runs execution starting from the main method and in during this execution it may create one or more thread. This thread can execute code of the same executable or code defined in other dll in the same process

In Windows world, if a process does not have a thread it gets terminate.

## 7.1.2. Processes vs Threads



**A process is created by the operating system, and requires a fair amount of "overhead".** Processes contain information about program resources and program execution state.



Processes are described via [PCB](#) (Process Control Block)

*Process control information* is used by the OS to manage the process itself. This includes:

- **The process scheduling state:** The state of the process in terms of "ready", "suspended", etc., and other scheduling information as well, like priority value, the amount of time elapsed since the process gained control of the CPU or since it was suspended. Also, in case of a suspended process, event identification data must be recorded for the event the process is waiting for.
- **Process structuring information:** process's children id's, or the id's of other processes related to the current one in some functional way, which may be represented as a queue, a ring or other data structures.
- **Interprocess communication information:** various flags, signals and messages associated with the communication among independent processes may be stored in the PCB.
- **Process Privileges** in terms of allowed/disallowed access to system resources.
- **Process State:** State may enter into new, ready, running, waiting, dead depending on CPU scheduling.
- **Process Number (PID):** A unique identification number for each process in the operating system (also known as [Process ID](#)).
- **Program Counter (PC):** A pointer to the address of the next instruction to be executed for this process.
- **CPU Registers:** Indicates various register set of CPU where process need to be stored for execution for running state.
- **CPU Scheduling Information:** indicates the information of a process with which it uses the CPU time through scheduling.
- **Memory Management Information:** includes the information of page table, memory limits,

Segment table depending on memory used by the operating system.

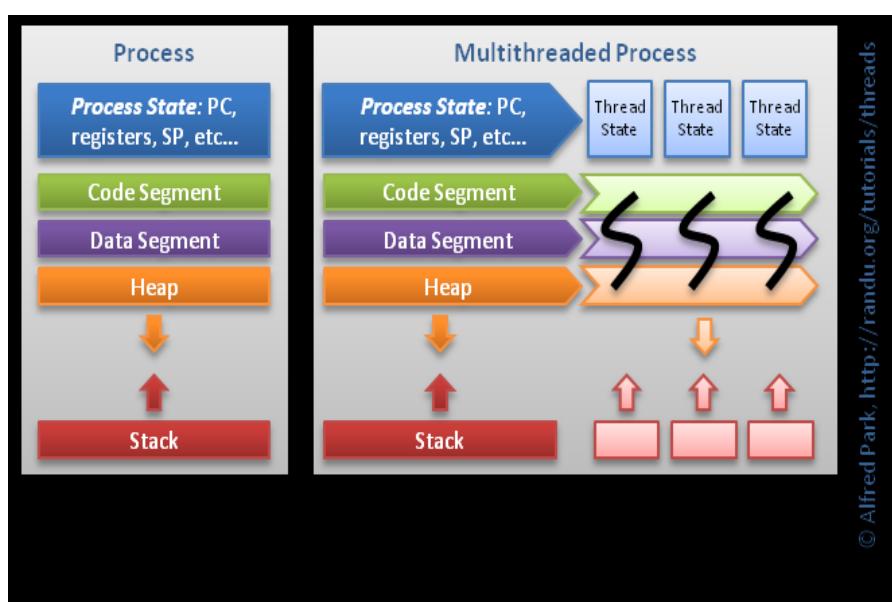
- **Accounting Information:** Includes the amount of CPU used for process execution, time limits, execution ID etc.
- **I/O Status Information:** Includes a list of I/O devices allocated to the process.

<https://docs.microsoft.com/en-us/windows/win32/procthread/about-processes-and-threads>

Each *process* provides the resources needed to execute a program. A process has a virtual address space, executable code, open handles to system objects, a security context, a unique process identifier, environment variables, a priority class, minimum and maximum working set sizes, and at least one thread of execution. Each process is started with a single thread, often called the *primary thread*, but can create additional threads from any of its threads.

A *thread* is the entity within a process that can be scheduled for execution. All threads of a process share its virtual address space and system resources. In addition, each thread maintains exception handlers, a scheduling priority, thread local storage, a unique thread identifier, and a set of structures the system will use to save the thread context until it is scheduled. The *thread context* includes the thread's set of machine registers, the kernel stack, a thread environment block, and a user stack in the address space of the thread's process. Threads can also have their own security context, which can be used for impersonating clients.

Microsoft Windows supports *preemptive multitasking*, which creates the effect of simultaneous execution of multiple threads from multiple processes. On a multiprocessor computer, the system can simultaneously execute as many threads as there are processors on the computer.



Technically, a thread is defined as an independent stream of instructions that can be scheduled to run as such by the operating system. But what does this mean?

To the software developer, the concept of a "method" that runs independently from its main program may best describe a thread.

To go one step further, imagine a main program that contains a number of method. Then imagine all of these methods being able to be scheduled to run simultaneously and/or independently by the operating system. That would describe a "multi-threaded" program.

A thread is described by its TCB (Thread Control Block):

An example of information contained within a TCB is:

- Thread Identifier: Unique id (tid) is assigned to every new thread
- Stack pointer: Points to thread's stack in the process

- Program counter: Points to the current program instruction of the thread
- State of the thread (running, ready, waiting, start, done)
- Thread's register values
- Pointer to the Process control block (PCB) of the process that the thread lives on

### 7.1.3. Come usare il multithreading in .NET

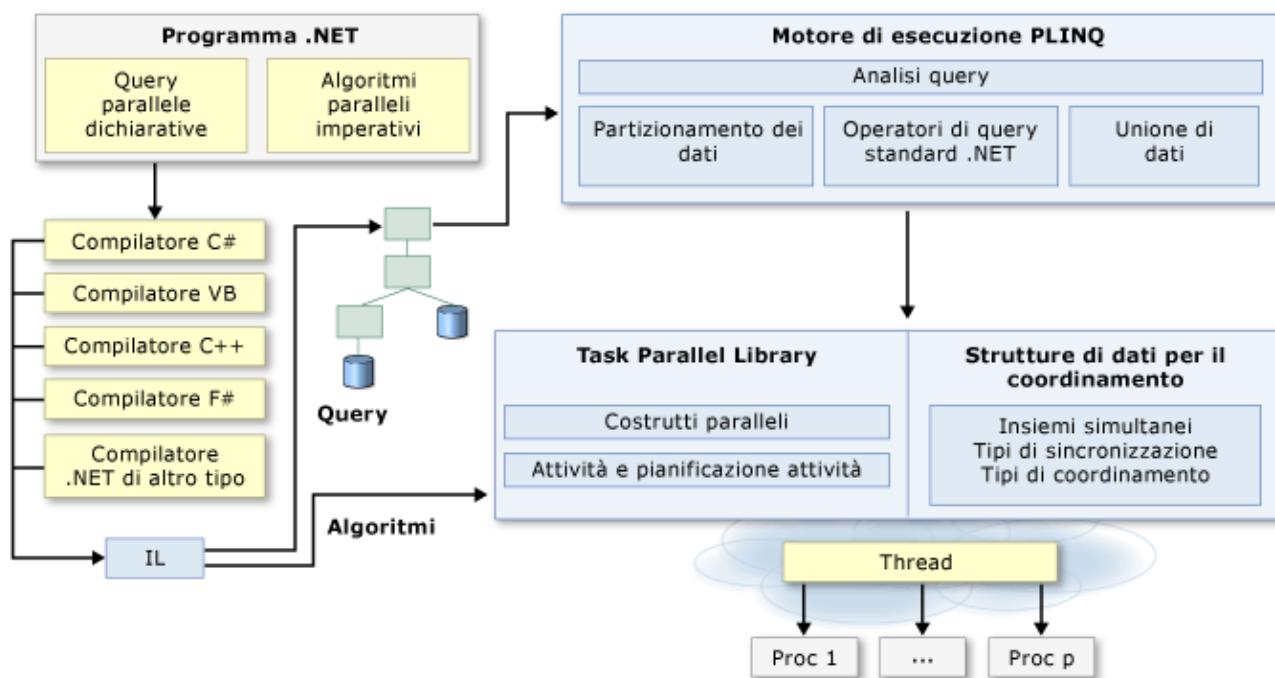
<https://docs.microsoft.com/it-it/dotnet/standard/threading/threads-and-threading#how-to-use-multithreading-in-net>

In queste note ci riferiremo al threading gestito dal .NET CLR

- <https://docs.microsoft.com/en-us/dotnet/standard/threading/>

e alle API per la programmazione parallela di .NET -

<https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/index>



Con .NET è possibile scrivere applicazioni in grado di eseguire più operazioni contemporaneamente. Le operazioni potenzialmente in grado di compromettere altre operazioni possono essere eseguite su thread separati, un processo noto con il nome di multithreading o threading Free.

Le applicazioni che usano il multithreading sono più reattive all'input dell'utente poiché l'interfaccia utente rimane attiva mentre le attività che richiedono un uso intensivo del processore vengono eseguite su thread separati. Il multithreading è utile anche durante la creazione di applicazioni scalabili poiché consente di aggiungere thread man mano che aumenta il carico di lavoro.

<https://docs.microsoft.com/it-it/dotnet/standard/threading/using-threads-and-threading>

Nel caso sia necessario un maggiore controllo sul comportamento dei thread dell'applicazione, è possibile gestirli personalmente. Tuttavia, a partire da .NET Framework 4, la programmazione multithreading è notevolmente semplificata con le classi System.Threading.Tasks.Parallel e System.Threading.Tasks.Task, Parallel LINQ (PLINQ), le nuove classi di raccolta simultanee

nello spazio dei nomi [System.Collections.Concurrent](#) e un nuovo modello di programmazione che si basa sul concetto di attività anziché di thread. Per altre informazioni, vedere [Programmazione parallela](#) e [Task Parallel Library \(TPL\)](#).

### 7.1.4. When To Use Thread In C#?

You can use threads in the following cases:

- **Scalability** (Be parallel) - If you have long running CPU bound operations, like to compute if 80 digit number is prime or not, you can scale this operation by paralleling this operation to multiple threads
- **Responsive** - You can keep client application responsive by keeping off lengthy operations from main thread (like CPU bound operation) and thus can also leverage the benefit of canceling the task
- **Leverage asynchronous technique** - If you have IO bound operation such reading a web content it may require some time in order of minutes, so you can leverage another thread to wait for this operation while you perform other task and thus even keep UI responsive. However, C# provides **async await** syntax for this kind of asynchronous technique.

Also, being asynchronous is not parallel it just keeps the application responsive. Asynchronous means not waiting for an operation to finish, but registering a listener instead.

In general, use parallel threads (using [Thread class](#) and [Task class](#) in C#) or asynchronous technique (using [C# async await keyword](#)) depending upon whether the problem is CPU bound or IO bound respectively.

Thumb rule is to use threads for CPU bound operation and async for IO bound operation for a client application, and always use async for a server application.

### 7.1.5. C# Thread

<https://docs.microsoft.com/it-it/dotnet/standard/threading/creating-threads-and-passing-data-at-start-time>

Threads in C# are modelled by Thread Class. When a process starts (you run a program) you get a single thread (also known as the main thread) to run your application code. To explicitly start another thread (other than your application main thread) you have to create an instance of thread class and call its Start method to run the thread using C#. Let's see an example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ThreadingDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            //initialize a thread class object
            //And pass your custom method name to the constructor parameter
            Thread t = new Thread(SomeMethod);

            //start running your thread
            t.Start();
        }
    }
}
```

```

        //while thread is running in parallel
        //you can carry out other operations here

        Console.WriteLine("Press Enter to terminate!");
        Console.ReadLine();
    }

    private static void SomeMethod()
    {
        //your code here that you want to run parallel
        //most of the time it will be a CPU bound operation

        Console.WriteLine("Hello World!");
    }
}

```

When you run this program you may see *Press Enter to terminate!* message first and then *Hello World!* as they both run in parallel, so it is not guaranteed which execute first.

So,

We can use Thread's Join() method to halt our main thread until reference thread (that is "t" variable in our case) is truly shutdown.

```

t.Join();
Console.WriteLine("Press Enter to terminate!");
Console.ReadLine();

```

Another method to do this would be by using boolean IsAlive property of thread which gives instantaneous snapshot of thread's state whether it is running or not.

Now, Thread doesn't start running until you call thread.Start() method, So before calling this Start method you can set some properties of a thread like its name and priority. Setting name of the thread will only help you in debugging, by setting name you can easily point out your thread in Visual Studio Thread window, Let's see an example

```

Thread t = new Thread(SomeMethod);

t.Name = "My Parallel Thread";

t.Priority = ThreadPriority.BelowNormal;

```

I parametri di un thread possono essere definiti anche direttamente nell'inizializzatore del thread:

```

Thread t = new Thread(SomeMethod)
{
    Name = "My Parallel Thread",

    Priority = ThreadPriority.BelowNormal
};

```

### 7.1.5.1. Difference Between Foreground And Background Thread In C#

There is also this another thread property IsBackground. If set to true your thread will be a background thread otherwise it will be a foreground thread, by default its false so it will always be a foreground thread, Let's see an example.

```
//set thread object as a background thread
```

```
t.IsBackground = true;
```

Suppose if a foreground thread is the only thread (your main thread is done with execution and terminated) in your process, so your process is about to exit. However, it won't, your process will wait for foreground thread to complete its execution. Thus, It will prevent application to exit until the foreground thread is done with the execution. However, background thread will exit as soon as your process exits even though background thread is not completely done with the execution.

Learn more about how foreground & background threads work in C# .NET [here](#).

### 7.1.5.2. C# Start Thread With Parameters

<https://docs.microsoft.com/it-it/dotnet/standard/threading/creating-threads-and-passing-data-at-start-time>

As you saw in example before that we pass method name to thread constructor parameter like this,

```
Thread t = new Thread(SomeMethod);
```

We able to do this because this thread contructor takes delegate as parameter. Its supports two type of delegates, Here is the definition of first delegate

```
public delegate void ThreadStart()
```

this we already saw in the above example, other is

```
public delegate void ParameterizedThreadStart(object obj)
```

If your custom method takes argument you can pass a ParameterizedThreadStart delegate to constructor, Let's see an example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace StartWithParameters
{
    class Program
    {
        static void Main(string[] args)
        {

            //initialize a thread class object
            //And pass your custom method name to the constructor parameter
            Thread t = new Thread(Speak!);

            //start running your thread
            //dont forget to pass your parameter for the Speak method
            //in Thread's Start method below
            t.Start("Hello World!");

            //wait until Thread "t" is done with its execution.
            t.Join();

            Console.WriteLine("Press Enter to terminate!");
            Console.ReadLine();
        }

        private static void Speak(object s)
        {
            //your code here that you want to run parallel
            //most of the time it will be a CPU bound operation
        }
    }
}
```

```

        string? say = s as string;
        Console.WriteLine(say);

    }
}
}

```

Did you notice now we need to pass the Speak method argument to Start method.

So far we have used only static method. However, you can also use instance methods as a thread constructor parameter, Let's see an example

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace StartWithParameters
{
    class Program
    {
        static void Main(string[] args)
        {

            Person person = new Person();

            //initialize a thread class object
            //And pass your custom method name to the constructor parameter
            Thread t = new Thread(person.Speak!);

            //start running your thread
            //dont forget to pass your parameter for
            //the Speak method in Thread's Start method below
            t.Start("Hello World!");

            //wait until Thread "t" is done with its execution.
            t.Join();

            Console.WriteLine("Press Enter to terminate!");
            Console.ReadLine();
        }

        private static void Speak(object s)
        {
            //your code here that you want to run parallel
            //most of the time it will be a CPU bound operation

            string? say = s as string;
            Console.WriteLine(say);
        }
    }

    public class Person
    {
        public void Speak(object s)
        {
            //your code here that you want to run parallel
            //most of the time it will be a CPU bound operation

            string? say = s as string;
            Console.WriteLine(say);
        }
    }
}

```

```
    }
```

```
}
```

}  
In the above example, we used ParameterizedThreadStart delegate however same applies to ThreadStart delegate, both of them can be used with an instance method.

### 7.1.5.3. Thread Life Cycle In C#

A thread in C# at any point of time exists in any one of the following states. A thread lies only in one of the shown states at any instant:

<https://learn.microsoft.com/en-us/dotnet/api/system.threading.threadstate?view=net-6.0>

Aborted	256	The thread state includes <u>AbortRequested</u> and the thread is now dead, but its state has not yet changed to <u>Stopped</u> .
AbortRequested	128	The <u>Abort(Object)</u> method has been invoked on the thread, but the thread has not yet received the pending <u>ThreadAbortException</u> that will attempt to terminate it.
Background	4	The thread is being executed as a background thread, as opposed to a foreground thread. This state is controlled by setting the <u>IsBackground</u> property.
Running	0	The thread has been started and not yet stopped.
Stopped	16	The thread has stopped.
StopRequested	1	The thread is being requested to stop. This is for internal use only.
Suspended	64	The thread has been suspended.
SuspendRequested	2	The thread is being requested to suspend.
Unstarted	8	The <u>Start()</u> method has not been invoked on the thread.

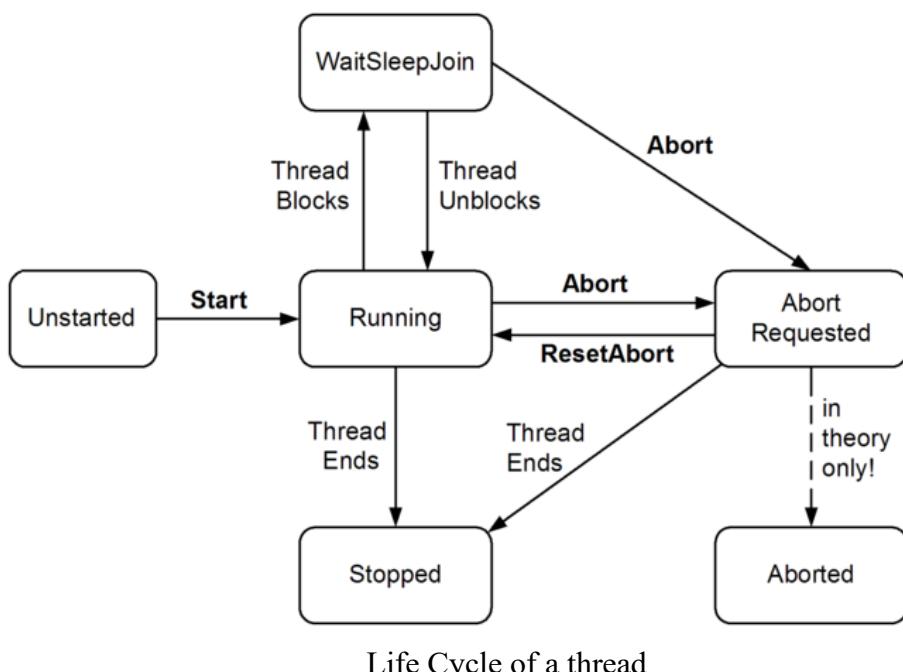
WaitSleepJoin	32	The thread is blocked. This could be the result of calling <code>Sleep(Int32)</code> or <code>Join()</code> , of requesting a lock - for example, by calling <code>Enter(Object)</code> or <code>Wait(Object, Int32, Boolean)</code> - or of waiting on a thread synchronization object such as <code>ManualResetEvent</code> .
---------------	----	---

The `ThreadState` enumeration defines a set of all possible execution states for threads. It's of interest only in a few debugging scenarios. Your code should never use the thread state to synchronize the activities of threads.

Once a thread is created, it's in at least one of the states until it terminates. Threads created within the common language runtime are initially in the `Unstarted` state, while external, or unmanaged, threads that come into the runtime are already in the `Running` state. A thread is transitioned from the `Unstarted` state into the `Running` state by calling `Thread.Start`. Once a thread leaves the `Unstarted` state as the result of a call to `Start`, it can never return to the `Unstarted` state.

A thread can be in more than one state at a given time.

**Thread States:** <http://www.albahari.com/threading/part2.aspx>



In C#, to get the current state of the thread, use `ThreadState` or `IsAlive` property provided by the `Thread` class.

Syntax:

```
public ThreadState ThreadState{ get; }
```

OR

```
public bool IsAlive { get; }
```

`Thread` class provides different types of methods to implement the states of the threads.

`Thread.Sleep()` method is a static used to temporarily suspend the current execution of the thread

for specified milliseconds, so that other threads can get the chance to start the execution, or may get the CPU for execution.

**Join()** method is used to make all the calling thread to wait until the main thread, i.e. joined thread complete its work.

**Start()** method is used to send a thread into runnable State.

So now we know how thread class models a thread. This thread, however, doesn't stay for infinity and has lifespan which is up to the return of the thread delegate method, Let's see an example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace StartWithParameters
{
    class Program
    {
        public static void Main()
        {
            //initialize a thread class object
            //And pass your custom method name to the constructor parameter

            Thread t = new Thread(Speak!);

            //start running your thread
            //dont forget to pass your parameter for the Speak method in
            //Thread's Start method below
            t.Start("Hello World!");

            //wait until Thread "t" is done with its execution.
            t.Join();

            Console.WriteLine("Press Enter to terminate!");
            Console.ReadLine();
        }

        private static void Speak(object s)
        {
            //your code here that you want to run parallel
            //most of the time it will be a CPU bound operation

            string? say = s as string;
            Console.WriteLine(say);

        } // <-- this line is where thread exit and shutdown
    }
}
```

Some other reasons of thread shutdown are as follows:

### Synchronous exception

Thread also gets exit if it runs into an unhandled exception. This exception is considered as synchronous exception which occurs in normal sequential program like IndexOutOfRangeException.

## **Asynchronous exception**

This exception is an explicit exception raised by calling thread's Abort or Interrupt method in the running thread by some other thread which has reference to the running thread. This exception also exits thread execution. However, this is not a recommended method to shutdown a thread as it leaves the program to some improper state.

### **7.1.6. Sospensione e interruzione di thread**

<https://docs.microsoft.com/it-it/dotnet/standard/threading/pausing-and-resuming-threads>

Le tecniche più comuni per sincronizzare le attività dei thread consistono nel blocco e nel rilascio dei thread oppure nel blocco di oggetti o aree di codice. Per altre informazioni su questi meccanismi di blocco, vedere [Panoramica delle primitive di sincronizzazione](#).

È anche possibile fare in modo che i thread vengano sospesi automaticamente. Quando i thread sono bloccati o sospesi, è possibile usare un'eccezione [ThreadInterruptedException](#) per interromperne lo stato di attesa.

#### **7.1.6.1. Metodi Thread.Sleep, Thread.Interrupt**

Se si chiama il metodo [Thread.Sleep](#), il thread corrente viene bloccato immediatamente per il numero di millisecondi o l'intervallo di tempo passato al metodo, cedendo il resto della porzione di tempo a un altro thread. Una volta trascorso tale intervallo, il thread inattivo riprende l'esecuzione.

Un thread non può chiamare [Thread.Sleep](#) su un altro thread. [Thread.Sleep](#) è un metodo statico che determina sempre il thread corrente da sospendere.

Se si chiama [Thread.Sleep](#) con un valore di [Timeout.Infinite](#), un thread rimarrà sospeso finché non verrà interrotto da un altro thread tramite una chiamata al metodo [ThreadInterrupt](#) nel thread sospeso o finché non verrà terminato da una chiamata al relativo metodo [ThreadAbort](#). L'esempio seguente illustra entrambi i metodi per interrompere un thread inattivo.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace SleepingDemo01
{
    class Program
    {
        static void Main(string[] args)
        {
            // Interrupt a sleeping thread.
            var sleepingThread = new Thread(Program.SleepIndefinitely);
            sleepingThread.Name = "Sleeping";
            sleepingThread.Start();
            Thread.Sleep(2000);
            sleepingThread.Interrupt();

            //Abort è deprecato nelle ultime versioni di .NET

        }
        private static void SleepIndefinitely()
        {
            Console.WriteLine("Thread '{0}' about to sleep indefinitely.",
                Thread.CurrentThread.Name);
            try
            {
```

```

        Thread.Sleep(Timeout.Infinite);
    }
    catch (ThreadInterruptedException)
    {
        Console.WriteLine("Thread '{0}' awoken.",
                           Thread.CurrentThread.Name);
    }

    finally
    {
        Console.WriteLine("Thread '{0}' executing finally block.",
                           Thread.CurrentThread.Name);
    }
    Console.WriteLine("Thread '{0}' finishing normal execution.",
                      Thread.CurrentThread.Name);
    Console.WriteLine();
}
}

// The example displays the following output:
//      Thread 'Sleeping' about to sleep indefinitely.
//      Thread 'Sleeping' awoken.
//      Thread 'Sleeping' executing finally block.
//      Thread 'Sleeping' finishing normal execution.

```

È possibile interrompere un thread in attesa chiamando il metodo [ThreadInterrupt](#) sul thread bloccato per generare un'eccezione [ThreadInterruptedException](#), che fa uscire il thread dalla chiamata che lo blocca. Il thread dovrebbe intercettare l'eccezione [ThreadInterruptedException](#) ed eseguire le operazioni appropriate per continuare a funzionare. Se il thread ignora l'eccezione, l'ambiente di esecuzione la intercetta e interrompe il thread.

**È consigliato non ricorrere al metodo [Interrupt](#) per interrompere un thread perché potrebbe lasciarlo in uno stato inconsistente.**

<http://www.albahari.com/threading/part3.aspx# Interrupt and Abort>

### Interrupt

Interrupting a thread does not cause the thread to end, unless the [ThreadInterruptedException](#) is unhandled.

If [Interrupt](#) is called on a thread that's not blocked, the thread continues executing until it next blocks, at which point a [ThreadInterruptedException](#) is thrown.

Interrupting a thread arbitrarily is dangerous, however, because any framework or third-party methods in the calling stack could unexpectedly receive the interrupt rather than your intended code. All it would take is for the thread to block briefly on a simple [lock](#) or synchronization resource, and any pending interruption would kick in. If the method isn't designed to be interrupted (with appropriate cleanup code in [finally](#) blocks), objects could be left in an unusable state or resources incompletely released.

Moreover, [Interrupt](#) is unnecessary: if you are writing the code that blocks, you can achieve the same result more safely with a signaling construct — or [cancellation tokens](#).

Si può definire comunque un **approccio cooperativo** nel quale il thread che esegue un certo task controlla se ha il permesso di eseguire una certa operazione come nell'esempio seguente:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace StopThread
{
    class Program
    {
        //set to volatile as its liable to change so we JIT to don't cache the value
        private static volatile bool _cancel = false;

        public static void Main()
        {
            //initialize a thread class object
            //And pass your custom method name to the constructor parameter

            Thread t = new Thread(Speak!);

            //start running your thread
            //dont forget to pass your parameter for the
            //Speak method (ParameterizedThreadStart delegate) in Start method
            t.Start("Hello World!");

            //wait for 5 secs while Speak method print Hello World! for multiple times
            Thread.Sleep(5000);

            //signal thread to terminate
            _cancel = true;

            //wait until CLR confirms that thread is shutdown
            t.Join();
            Console.WriteLine("\nSono il main thread, ho aspettato per 5 secondi che l'altro
thread si divertisse a scrivere \"Hello World\", ma ora esco!");
            Console.ReadLine();
        }

        private static void Speak(object s)
        {
            while (!_cancel)
            {
                string? say = s as string;
                Console.WriteLine(say);
            }
        }
    }
}

```

Here we used a boolean field to signal another thread Speak method to stop running when `_cancel` is set to true.

Did you notice how we need to set the `_cancel` field as volatile. JIT usually cache this kind of fields as it doesn't seem to change within Speak method in the loop. By setting it to volatile we are signaling JIT not to cache this field because it is liable to change.

You can use your own communication mechanism to tell the ThreadStart method to finish, which is recommended method. Alternatively the Thread class has in-built support for instructing the thread to stop. The two principle methods are `Thread.Interrupt()` and `Thread.Abort()`, which is not

recommended.

### 7.1.7. Threadpool

As we learned in previous section thread shutdown after its work is done which is a great thing, CLR clears the resource after thread shutdown and thus free up space for smooth program execution without you to write any code for thread management and garbage collection. However, creation of thread is something that costs time and resource and thus will be difficult to manage when dealing with a large number of threads. Thread pool is used in this kind of scenario.

When you work with thread pool from .NET you queue your work item in thread pool from where it gets processed by an available thread in the thread pool. But, after work is done this thread doesn't get shutdown. Instead of shutting down this thread get back to thread pool where it waits for another work item. The creation and deletion of this threads are managed by thread pool depending upon the work item queued in the thread pool. If no work is there in the thread pool it may decide to kill those threads so they no longer consume the resources.

#### 7.1.7.1. C# Thread Pool Queue

ThreadPool.QueueUserWorkItem is a static method that is used to queue the user work item in the thread pool. Just like you pass a delegate to a thread constructor to create a thread you have to pass a delegate to this method to queue your work.

Here is an example,

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ThreadPoolDemo
{
    class Program
    {
        public static void Main()
        {
            // call QueueUserWorkItem to queue your work item
            ThreadPool.QueueUserWorkItem(Speak);

            Console.WriteLine("Press Enter to terminate!");
            Console.ReadLine();
        }

        //your custom method you want to run in another thread
        public static void Speak(object? stateInfo)
        {
            // No state object was passed to QueueUserWorkItem, so stateInfo is null.
            Console.WriteLine("Hello World!");
        }
    }
}
```

As you can see we can directly pass this Speak method name to the QueueUserWorkItem method as it takes WaitCallback delegate as a parameter.

Here is the definition of this delegate,

```
public delegate void WaitCallback(object state);
```

See how it share the same signature like our Speak method with void as return type and take object as parameter.

QueueUserWorkItem also has overload for parameterised method like this,

`QueueUserWorkItem(WaitCallback, Object)`

Here the first parameter is your method name and the second parameter is the object that you want to pass to your method.

Here is an example,

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ThreadPoolDemo
{
    class Program
    {
        public static void Main()
        {
            // call QueueUserWorkItem to queue your work item
            ThreadPool.QueueUserWorkItem(Speak, "Hello World!");

            Console.WriteLine("Press Enter to terminate!");
            Console.ReadLine();
        }

        //your custom method you want to run in another thread
        public static void Speak(object? s)
        {
            string? say = s as string;
            Console.WriteLine(say);
        }
    }
}
```

### 7.1.7.2. Limitations To Thread Pool Queue

ThreadPool.QueueUserWorkItem is really easy way to schedule your work into thread pool however it has its limitation, like **you cannot tell whether a particular work operation is finished and also it does not return a value**. However, a Task is something that you can use in place of ThreadPool.QueueUserWorkItem. It tells whether an operation is completed and also returns a value after the task is completed. We will learn more about Tasks later. But, before that we'll learn what is a Race Condition in a multithreaded program and how much it is critical to synchronize a multithreaded program having Shared Resources.

## 7.1.8. Race Condition

A race condition occurs when two or more threads are able to access shared data and they try to change it at the same time. To fully understand a race condition we will first talk about shared resources and than discuss about what is a race condition in threading.

### 7.1.8.1. Shared Resources

Not all resources are meant to be used concurrently. Resources like integers and collection must be handled carefully when accessed through multiple threads, resources that are accessed and updated within multiple threads are known as Shared Resources. Let's see an example,

```
using System;
```

```

using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace SharedResources01
{
    class Program
    {
        private static int sum;

        static void Main(string[] args)
        {
            //create thread t1 using anonymous method
            Thread t1 = new Thread(() => {
                for (int i = 0; i < 10000000; i++)
                {
                    //increment sum value
                    sum++;
                }
            });

            //create thread t2 using anonymous method
            Thread t2 = new Thread(() => {
                for (int i = 0; i < 10000000; i++)
                {
                    //increment sum value
                    sum++;
                }
            });

            //start thread t1 and t2
            t1.Start();
            t2.Start();

            //wait for thread t1 and t2 to finish their execution
            t1.Join();
            t2.Join();

            //write final sum on screen
            Console.WriteLine("sum: " + sum);

            Console.WriteLine("Press enter to terminate!");
            Console.ReadLine();
        }
    }
}

```

However, there is really some problem with the code above because every time we run it we see different output.

To truly understand the problem we must first understand what is a Race condition.

### 7.1.8.2. What Is Race Condition?

Race Condition is a scenario where the outcome of the program is affected because of timing.

A race condition occurs when two or more threads can access shared data and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e. both threads are “racing”

to access/change the data.

In our case, the line which is causing race condition is `sum++`, though this line seems to single line code and must not affect with concurrency but this single line of code gets transformed into multiline processor level instructions by JIT at the time of execution, below is the example

```
mov eax, dword ptr [sum]  
inc eax  
mov dword ptr [sum], eax
```

So what happens when our multiple threads execute this part of the code. Let's assume there is this thread X and thread Y. Suppose thread X reads the value of some variable and store in register X.eax for increment but after doing increment from value 0 to 1, X thread got suspended by Thread scheduler and Y thread start executing this part of the code where Y thread also reads the value of variable sum in register Y.eax and does the increment from value 0 to 1 and now after doing this increment both thread will update sum variable to 1 thus its value will be 1 even though both the threads incremented the value.

So in simple words, it's just the race between threads X and Y to read and update the value of variable sum and thus cause the race condition.

But we can overcome this kind of problems using some of the thread synchronization techniques that are:

Atomic Update

Data Partitioning

Wait-Based Technique

We will learn more about these thread synchronization techniques in the next section

### 7.1.9. Sezione critica

[https://en.wikipedia.org/wiki/Critical\\_section](https://en.wikipedia.org/wiki/Critical_section)

In concurrent programming, concurrent accesses to shared resources can lead to unexpected or erroneous behavior, so parts of the program where the shared resource is accessed need to be protected in ways that avoid the concurrent access. This protected section is the **critical section** or **critical region**. It cannot be executed by more than one process/thread at a time. Typically, the critical section accesses a shared resource, such as a data structure, a peripheral device, or a network connection, that would not operate correctly in the context of multiple concurrent accesses.

Critical section is a piece of a program that requires mutual exclusion of access.

In the case of mutual exclusion (Mutex/Monitor), one thread blocks a critical section by using locking techniques when it needs to access the shared resource and other threads have to wait to get their turn to enter into the section. This prevents conflicts when two or more threads share the same memory space and want to access a common resource.

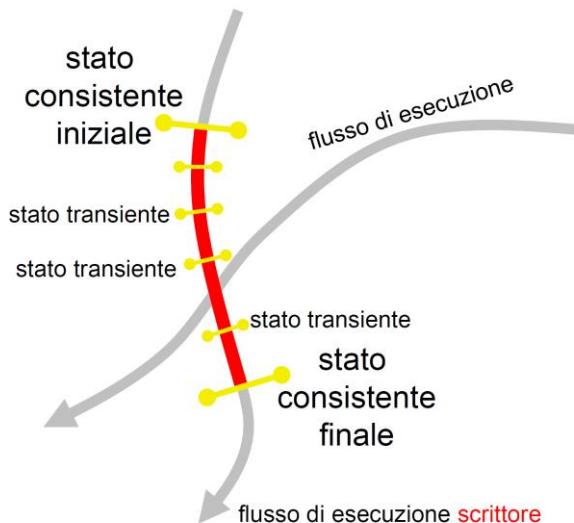
### 7.1.10. Stati transienti ed interferenze

<http://homes.di.unimi.it/ceselli/SO/slides/04a-thread.pdf>

# Stati transienti ed interferenze

- Le strutture dati accedute da un programma multithread sono oggetto di aggiornamenti da parte di più thread
- Gli **aggiornamenti non avvengono atomicamente**, ma sono decomponibili in varie operazioni di modifica intermedie e di una certa durata
- Durante il transitorio la struttura dati “perde significato” (inconsistente), e passa per una serie di **stati transienti**
- Un tale stato **non dovrebbe essere visibile** a thread diversi dal thread che esegue l’aggiornamento, altrimenti si generano **interferenze**

## Origine dei fenomeni di interferenza



## Interferenza

- Si ha **interferenza** in presenza di
  - due o più flussi di esecuzione
  - almeno un flusso di esecuzione esegue scritture (aggiorna la struttura dati!)
- **Perché**
  - un flusso esegue un cambio di stato dell’area di memoria in maniera non atomica
  - gli stati *transienti* che intercorrono tra quello iniziale a quello finale sono visibili a flussi di esecuzione diversi da quello che li sta producendo

## Esempio di Interferenza (1)

- La disponibilità di un volo di una compagnia aerea è memorizzata in **POSTI=1**. Due signori nel medesimo istante ma da due postazioni distinte, chiedono rispettivamente di prenotare l'ultimo posto e di disdire la prenotazione già effettuata.
- Le due richieste vengono tradotte in queste sequenze di **istruzioni elementari indivisibili**:

```
procedure Prenota
begin
    Ra ← POSTI - 1;
    POSTI ← Ra;
end
```

```
procedure Disdici
begin
    Rb ← POSTI + 1;
    POSTI ← Rb;
end
```

## Esempio di Interferenza (2)

- Inizialmente POSTI=1
- L'esecuzione concorrente da luogo ad una qualsiasi delle possibili **sequenze di interleaving**
- Consideriamo un campione di tre sequenze:

$R_a \leftarrow \text{POSTI} - 1;$ $R_b \leftarrow \text{POSTI} + 1;$ $\text{POSTI} \leftarrow R_b;$ $\text{POSTI} \leftarrow R_a;$	$R_a \leftarrow \text{POSTI} - 1;$ $\text{POSTI} \leftarrow R_a;$ $R_b \leftarrow \text{POSTI} + 1;$ $\text{POSTI} \leftarrow R_b;$	$R_b \leftarrow \text{POSTI} + 1;$ $R_a \leftarrow \text{POSTI} - 1;$ $\text{POSTI} \leftarrow R_a;$ $\text{POSTI} \leftarrow R_b;$
(POSTI=0)	(POSTI=1)	(POSTI=2)
ERRORE	OK	ERRORE

### 7.1.10.1. Thread Safeness

## Thread Safeness

- *Def.: Programma thread-safe*: Un programma si dice **thread safe** se garantisce che **nessun thread possa accedere a dati in uno stato inconsistente**
- Un **programma thread safe protegge l'accesso alle strutture** in stato inconsistente da parte di altri thread **per evitare interferenze**
  - costringendoli in attesa (passiva) del suo ritorno in uno stato consistente
- Il termine *thread safeness* si applica anche a librerie ed a strutture dati ad indicare la loro predisposizione ad essere inseriti in programmi multithread

### 7.1.10.2. Dominio e Rango

## Dominio e Rango

- Indichiamo con A, B, ... X, Y, ... un'area di memoria
- **Una istruzione  $i$** 
  - dipende da una o più aree di memoria che denotiamo  $\text{domain}(i)$ , ovvero dominio di  $i$
  - altera il contenuto di una o più aree di memoria che denotiamo  $\text{range}(i)$  di  $i$ , ovvero rango di  $i$
- Ad es. per la procedura **P**

**procedure P**

```
begin                                 $\text{domain(P)} = \{A, B, X\}$ 
 $X \leftarrow A + X;$                    $\text{range(P)} = \{X, Y\}$ 
 $Y \leftarrow A * B;$ 
end
```

### 7.1.10.3. Condizioni di Bernstein

## Condizioni di Bernstein

Quando è lecito eseguire  
concorrentemente due istruzioni  $i_a$  e  $i_b$ ?

- se valgono le seguenti condizioni, dette **Condizioni di Bernstein**:
  1.  $\text{range}(i_a) \cap \text{range}(i_b) = \emptyset$
  2.  $\text{range}(i_a) \cap \text{domain}(i_b) = \emptyset$
  3.  $\text{domain}(i_a) \cap \text{range}(i_b) = \emptyset$

## Condizioni di Bernstein (2)

- Si osservi che non si impone alcuna condizione su  $\text{domain}(i_a) \cap \text{domain}(i_b)$
- Sono banalmente estendibili al caso di tre o più istruzioni
- Esempi di violazione per le due istruzioni:
  - $X \leftarrow Y + 1; \quad X \leftarrow Y - 1; \quad (\text{violano la 1.})$
  - $X \leftarrow Y + 1; \quad Y \leftarrow X - 1; \quad (\text{violano la 2. e la 3.})$
  - $\text{scrivi } X; \quad X \leftarrow X + Y; \quad (\text{violano la 3.})$

## Effetti delle violazioni

- Quando un insieme di istruzioni soddisfa le condizioni di Bernstein, il loro esito complessivo sarà sempre lo stesso
  - **indipendentemente dall'ordine e dalle velocità relative** con cui vengono eseguite
  - ovvero, sarà sempre equivalente ad una loro esecuzione seriale
- Al contrario, **in caso di violazione, gli errori dipendono dall'ordine e dalle velocità relative** generando il fenomeno dell'*interferenze*

## Il Programmatore e gli errori dipendenti dal tempo

- Un programma che (implicitamente od esplicitamente) basa la propria correttezza su ipotesi circa
  - **la velocità relativa dei vari processori virtuali o**
  - **sulla sequenza di interleaving eseguita,****è scorretto**
- Esiste una sola assunzione che possono fare i programmati sulla velocità dei processori virtuali...

## Assunzione di Progresso Finito

*Tutti i processori virtuali hanno  
una velocità finita non nulla*

- Questa assunzione è **l'unica** che si può fare sui processori virtuali e sulle loro velocità relative

## Thread Safeness & Condizioni di Bernstein

- **Dato un programma multithread, quali strutture dati bisogna proteggere per garantire la thread safeness?**  
**Tutte le strutture dati oggetto di accessi concorrenti che violano le condizioni di Bernstein**

in altre parole,

*le strutture dati oggetto di scrittura  
concorrenti da parte di due o più thread*

## 7.1.11. C# Thread synchronization techniques

<https://docs.microsoft.com/it-it/dotnet/standard/threading/synchronizing-data-for-multithreading>

Thread synchronization refers to the act of shielding against multithreading issues such as data races, deadlocks and starvation.

Se più thread sono in grado di effettuare chiamate alle proprietà e ai metodi di un singolo oggetto, è essenziale che tali chiamate siano sincronizzate. In caso contrario un thread potrebbe interrompere le operazioni eseguite da un altro thread e l'oggetto potrebbe rimanere in uno stato non valido. Una classe i cui membri sono protetti da tali interruzioni è detta thread-safe.

.NET offre diverse strategie per sincronizzare l'accesso a membri statici e di istanza:

- Aree di codice sincronizzate. È possibile usare la classe [Monitor](#) o il supporto del compilatore per questa classe per sincronizzare solo il codice di blocco necessario, migliorando le prestazioni.
- Sincronizzazione manuale. È possibile usare gli oggetti di sincronizzazione della libreria di classi .NET. Vedere la [panoramica sulle primitive di sincronizzazione](#), che include una descrizione della classe [Monitor](#).

### 7.1.11.1. C# Interlocked

So as we saw in previous chapter processor increments the variable, written in a single line of C# code in three steps (three line of code) in processor-specific language, that is read, increment and update the variable. One way to tackle this problem is to carry out all this three operation in one single atomic operation. This can be done only on data that is word-sized. Here, by atomic I mean uninterruptable and word-sized means value that can fit in a register for the update, which is a single integer in our case. However, today's processors already provide lock feature to carry out an atomic update on word-sized data. However, we can't use this processor specific instruction directly in C# code but there is Interlocked Class in DotNet Framework that is a wrapper around this processor-level instruction that can be used to carry out atomic operations like increment and decrement on a word-sized data.

Let's see an updated version of the buggy code we saw in the previous chapter:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace AtomicUpdate01
{
    class Program
    {
        private static int sum;
        static void Main(string[] args)
        {

            //create thread t1 using anonymous method
            Thread t1 = new Thread(() =>
            {
                for (int i = 0; i < 10000000; i++)
                {
                    //use threading Interlocked class for atomic update
                    Interlocked.Increment(ref sum);
                }
            });
            t1.Start();
            Thread.Sleep(1000);
            Console.WriteLine("Final Value: " + sum);
        }
    }
}
```

```
        }

    });

    //create thread t2 using anonymous method
    Thread t2 = new Thread(() =>
    {
        for (int i = 0; i < 100000000; i++)
        {
            //use threading Interlocked class for atomic update
            Interlocked.Increment(ref sum);
        }
    });

    //start thread t1 and t2
    t1.Start();
    t2.Start();

    //wait for thread t1 and t2 to finish their execution
    t1.Join();
    t2.Join();

    //write final sum on screen
    Console.WriteLine("sum: " + sum);

    Console.WriteLine("Press enter to terminate!");
    Console.ReadLine();
}

}
```

### **7.1.11.2. Data Partitioning**

Data Partitioning is actually kind of strategy where you decide to process data by partitioning it for multiple threads. Its kind of “you do that and I will do that” strategy.

To use data partitioning you must have some domain-specific knowledge of data (such as an array or multiple files manipulation), where you decide that one thread will process just one slice of data while other thread will work on another slice. Let's see an example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace DataPartition01
{
    class Program
    {
        private static int[] array;
        private static int sum1;
        private static int sum2;

        static void Main(string[] args)
        {
            //set length for the array size
            int length = 1000000;
            //create new array of size lenght
            array = new int[length];

            //initialize array element with value of their respective index
            for (int i = 0; i < length; i++)
            {
                array[i] = i;
            }

            //partition array into two halves
            int mid = length / 2;
            sum1 = 0;
            sum2 = 0;
            for (int i = 0; i < mid; i++)
            {
                sum1 += array[i];
            }
            for (int i = mid; i < length; i++)
            {
                sum2 += array[i];
            }

            //print the sum of both halves
            Console.WriteLine("Sum of first half: " + sum1);
            Console.WriteLine("Sum of second half: " + sum2);

            //check if the sums are equal
            if (sum1 == sum2)
            {
                Console.WriteLine("The array is partitioned into two halves with equal sum.");
            }
            else
            {
                Console.WriteLine("The array is not partitioned into two halves with equal sum.");
            }
        }
    }
}
```

```
{  
    array[i] = i;  
}  
  
//index to split on  
int dataSplitAt = length / 2;  
  
//create thread t1 using anonymous method  
Thread t1 = new Thread(() =>  
{  
    //calculate sum1  
    for (int i = 0; i < dataSplitAt; i++)  
    {  
        sum1 = sum1 + array[i];  
    }  
});  
  
//create thread t2 using anonymous method  
Thread t2 = new Thread(() =>  
{  
    //calculate sum2  
    for (int i = dataSplitAt; i < length; i++)  
    {  
        sum2 = sum2 + array[i];  
    }  
});  
  
//start thread t1 and t2  
t1.Start();  
t2.Start();  
  
//wait for thread t1 and t2 to finish their execution  
t1.Join();  
t2.Join();  
  
//calculate final sum  
int sum = sum1 + sum2;  
  
//write final sum on screen  
Console.WriteLine("Sum:" + sum);  
  
Console.WriteLine("Press enter to terminate!");  
Console.ReadLine();  
}  
}
```

However, this technique can't be adapted for every scenario there may be a situation where one slice of data depends on the output of the previous slice of data; one example of this scenario is Fibonacci series where,  $\text{data}[n] = \text{data}[n-1] + \text{data}[n-2]$  in such a situation data partitioning can't be adopted.

### **7.1.11.3. Wait Based Synchronization**

Now, the third technique is a Wait-Based technique which is a very sophisticated way to handle the race condition, used in a situation where above two methods can't be adopted that easily. **In this technique, a thread is blocked until someone decides its safe for them to proceed.**

Suppose there are two threads namely X and Y and both want to access some resource R. Now to protect this resource we choose some lock primitive or synchronization primitive as LR (primitive

here is some primitive type like int or array). Now when thread X want to access resource R it will first acquire the lock ownership of LR, once this thread got ownership of LR it can access the resource R safely. As long as thread X has this ownership no other thread can access the LR ownership

While X has ownership if Y request to acquire the ownership of lock LR it requests will get block until thread X releases its ownership.

### 7.1.11.3.1. Panoramica delle primitive di sincronizzazione in .NET

<https://docs.microsoft.com/it-it/dotnet/standard/threading/overview-of-synchronization-primitives>

.Net has following Wait Based Primitives that you can use to apply Wait-Based technique. They all share the same basic usage:

- **Access the lock ownership**
- **Manipulate the protected resource**
- **Release the lock ownership**

.NET offre una gamma di tipi che è possibile usare per sincronizzare l'accesso a una risorsa condivisa o coordinare l'interazione tra thread.

- **Classe WaitHandle e tipi di sincronizzazione leggeri**

Più primitive di sincronizzazione .NET derivano dalla classe [System.Threading.WaitHandle](#), che incapsula un handle di sincronizzazione del sistema operativo nativo e usa un meccanismo di segnalazione per l'interazione tra thread. Tali classi includono:

- [System.Threading.Mutex](#), che concede l'accesso esclusivo a una risorsa condivisa. Lo stato di un mutex viene segnalato se nessun thread lo possiede.
- [System.Threading.Semaphore](#), che limita il numero di thread che possono accedere simultaneamente a una risorsa condivisa o a un pool di risorse. Lo stato di un semaforo denominato è impostato come segnalato quando il relativo conteggio è maggiore di zero e come non segnalato quando il relativo conteggio è zero.
- [System.Threading.EventWaitHandle](#), che rappresenta un evento di sincronizzazione di thread e può trovarsi in uno stato segnalato o non segnalato.
- [System.Threading.AutoResetEvent](#), che deriva da [EventWaitHandle](#) e, quando segnalata si reimposta automaticamente in uno stato non segnalato dopo il rilascio di un singolo thread in attesa.
- [System.Threading.ManualResetEvent](#), che deriva da [EventWaitHandle](#) e, quando segnalata, rimane in uno stato segnalato finché non viene chiamato il metodo [Reset](#).

In .NET Framework, poiché [WaitHandle](#) deriva da [System.MarshalByRefObject](#), questi tipi possono essere usati per sincronizzare le attività dei thread tra limiti dei domini delle applicazioni.

In .NET Framework e .NET Core, alcuni di questi tipi possono rappresentare handle di sincronizzazione di sistema denominati, che sono visibili in tutto il sistema operativo e possono essere usati per la sincronizzazione interprocesso:

- [Mutex](#) (.NET Framework e .NET Core).
- [Semaphore](#) (.NET Framework e .NET Core in Windows).
- [EventWaitHandle](#) (.NET Framework e .NET Core in Windows).

Per altre informazioni, vedere le informazioni di riferimento sull'API [WaitHandle](#).

I tipi di sincronizzazione leggeri non si basano su handle del sistema operativo sottostanti e in genere offrono prestazioni migliori. Tuttavia, non possono essere usati per la sincronizzazione interprocesso.

Usare tali tipi per la sincronizzazione dei thread all'interno di un'applicazione.

Alcuni di questi tipi rappresentano alternative ai tipi derivati da [WaitHandle](#). I tipi di sincronizzazione leggera che verranno illustrati in questa guida sono:

- [SemaphoreSlim](#) è un'alternativa leggera a [Semaphore](#).
- La classe [System.Threading.Monitor](#) concede l'accesso con esclusione reciproca a una risorsa condivisa tramite l'acquisizione o il rilascio di un blocco sull'oggetto che identifica la risorsa. Mentre è attivo un blocco, il thread che contiene il blocco può ancora acquisire e rilasciare il blocco. Gli altri thread non possono acquisire il blocco e il metodo [Monitor.Enter](#) attende il rilascio del blocco. Il metodo [Enter](#) acquisisce un blocco rilasciato. È anche possibile usare il metodo [Monitor.TryEnter](#) per specificare l'intervallo di tempo durante il quale un thread cerca di acquisire un blocco. Poiché la classe [Monitor](#) presenta affinità di thread, il thread che ha acquisito un blocco deve rilasciare il blocco chiamando il metodo [Monitor.Exit](#). È possibile coordinare l'interazione tra i thread che acquisiscono un blocco sullo stesso oggetto usando i metodi [Monitor.Wait](#), [Monitor.Pulse](#) e [Monitor.PulseAll](#).

<http://www.albahari.com/threading/part2.aspx>

#### Comparison of Locking Constructs

Construct	Purpose	Cross-process?	Overhead*
<a href="#">lock</a> ( <a href="#">Monitor.Enter</a> / <a href="#">Monitor.Exit</a> )	Ensures just one thread can access a resource, or section of code at a time	-	20ns
<a href="#">Mutex</a>		Yes	1000ns
<a href="#">SemaphoreSlim</a> (introduced in Framework 4.0)	Ensures not more than a specified number of concurrent threads can access a resource, or section of code	-	200ns
<a href="#">Semaphore</a>		Yes	1000ns

\*Time taken to lock and unlock the construct once on the same thread (assuming no blocking), as measured on an Intel Core i7 860.

#### 7.1.11.3.2. Monitor Class Usage In C# (approfondimento)

Monitor class is one of the wait based synchronization primitive that provides gated access to the resource. It gates or throttles the access to the shared resource. So, Monitor assure that thread access the shared resource one thread at a time. Here is the code to use Monitor for shared resources to avoid

the race condition

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace Monitor01
{
    class Program
    {
        private static int sum;
        //https://cezarypiatek.github.io/post/avoid-multithreading-traps-p1/
        private static readonly object _lock = new object();

        static void Main(string[] args)
        {

            //create thread t1 using anonymous method
            Thread t1 = new Thread(() => {
                for (int i = 0; i < 10000000; i++)
                {
                    //acquire lock ownership
                    Monitor.Enter(_lock);

                    //increment sum value
                    sum++;

                    //release lock ownership
                    Monitor.Exit(_lock);
                }
            });

            //create thread t2 using anonymous method
            Thread t2 = new Thread(() => {
                for (int i = 0; i < 10000000; i++)
                {
                    //acquire lock ownership
                    Monitor.Enter(_lock);

                    //increment sum value
                    sum++;

                    //release lock ownership
                    Monitor.Exit(_lock);
                }
            });

            //start thread t1 and t2
            t1.Start();
            t2.Start();

            //wait for thread t1 and t2 to finish their execution
            t1.Join();
            t2.Join();

            //write final sum on screen
            Console.WriteLine("sum: " + sum);

            Console.WriteLine("Press enter to terminate!");
            Console.ReadLine();
        }
    }
}
```

```
}
```

Here, sum++ is considered as the critical section, as this operation should be done in a thread-safe manner we use the monitor to carry out this operation as one thread at a time.

As you saw Monitor use Enter and Exit method which accepts an object to associate with lock primitive. Basically, you can use any type of object to associate with a lock. However, the recommended method is to use private readonly objects and always avoid string as lock objects as the issue they cause due to their implementation method in CLR. See:

<https://cezarypiatek.github.io/post/avoid-multithreading-traps-p1/#best-practice-for-locking>

So Monitor has the same wait based technique usage. When you call the Monitor.Enter method you get the ownership of the lock, then you perform your thread-safe operation and then release the lock using Monitor.Exit.

#### 7.1.11.3.2.1. Exception Aware Monitor Usage (approfondimento)

Now, consider the same example code above and there are two threads trying to acquire the lock X and Y now thread X got the ownership and Y got blocked until X releases the ownership. However, before releasing the lock thread X threw some runtime exception error hence it will exit the code before releasing the lock, as a result, thread Y will get blocked forever. We want to throw the exception but also want to release the lock.

So, to overcome this problem we have to use proper try-finally construct to manage the exception (not handle it). Let's see the code how to do it,

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ExceptionAwareMonitor
{
    class Program
    {
        private static int sum;
        private static readonly object _lock = new object();

        static void Main(string[] args)
        {

            //create thread t1 using anonymous method
            Thread t1 = new Thread(() => {
                for (int i = 0; i < 10000000; i++)
                {
                    //https://docs.microsoft.com/en-us/dotnet/api/system.threading.monitor.enter?view=netframework-4.8#System_Threading_Monitor_Enter_System_Object_System_Boolean_
                    //acquire lock ownership
                    bool acquiredLock = false;
                    try
                    {
                        Monitor.Enter(_lock, ref acquiredLock);
                        // Code that accesses resources that are protected by the lock.
                        sum++;
                    }
                    finally
                    {
                        if (acquiredLock)
                    }
                }
            });
            t1.Start();
        }
    }
}
```

```
        {
            Monitor.Exit(_lock);
        }
    });
}

//create thread t2 using anonymous method
Thread t2 = new Thread(() => {
    for (int i = 0; i < 100000000; i++)
    {
        //https://docs.microsoft.com/en-us/dotnet/api/system.threading.monitor.enter?view=netframework-4.8#System_Threading_Monitor_Enter_System_Object_System_Boolean__
        //acquire lock ownership
        bool acquiredLock = false;
        try
        {
            Monitor.Enter(_lock, ref acquiredLock);
            // Code that accesses resources that are protected by the lock.
            sum++;
        }
        finally
        {
            if (acquiredLock)
            {
                Monitor.Exit(_lock);
            }
        }
    }
});

//start thread t1 and t2
t1.Start();
t2.Start();

//wait for thread t1 and t2 to finish their execution
t1.Join();
t2.Join();

//write final sum on screen
Console.WriteLine("sum: " + sum);

Console.WriteLine("Press enter to terminate!");
Console.ReadLine();
}
}
```

### **7.1.11.3.2.2. C# Lock Keyword (il costrutto da usare per lock tra thread dello stesso processo)**

Some high-level languages have syntactic sugar which reduces the amount of code that must be written in some common situation like above. C# has this lock syntax for the same code we wrote above. Here is the code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
```

```

namespace LockKeyword
{
    class Program
    {
        private static int sum;
        private static readonly object _lock = new object();

        static void Main(string[] args)
        {

            //create thread t1 using anonymous method
            Thread t1 = new Thread(() => {
                for (int i = 0; i < 10000000; i++)
                {
                    lock (_lock)
                    {
                        //increment sum value
                        sum++;
                    }
                }
            });

            //create thread t2 using anonymous method
            Thread t2 = new Thread(() => {
                for (int i = 0; i < 10000000; i++)
                {
                    lock (_lock)
                    {
                        //increment sum value
                        sum++;
                    }
                }
            });

            //start thread t1 and t2
            t1.Start();
            t2.Start();

            //wait for thread t1 and t2 to finish their execution
            t1.Join();
            t2.Join();

            //write final sum on screen
            Console.WriteLine("sum: " + sum);

            Console.WriteLine("Press enter to terminate!");
            Console.ReadLine();
        }
    }
}

```

**So we simply use the lock keyword syntax and write critical section code in its body and compiler will generate the Exception Aware Monitor code for us. Sweet!**

### 7.1.12. Semafori

<https://docs.microsoft.com/en-us/dotnet/api/system.threading.semaphoreslim?view=net-5.0>

Semaphores are of two types: local semaphores and named system semaphores. Local semaphores are local to an application, system semaphores are visible throughout the operating system and are suitable for inter-process synchronization. The [SemaphoreSlim](#) is a lightweight alternative to

the [Semaphore](#) class that doesn't use Windows kernel semaphores. Unlike the [Semaphore](#) class, the [SemaphoreSlim](#) class doesn't support named system semaphores. You can use it as a local semaphore only. The [SemaphoreSlim](#) class is the recommended semaphore for synchronization within a single app.

A lightweight semaphore controls access to a pool of resources that is local to your application. When you instantiate a semaphore, you can specify the maximum number of threads that can enter the semaphore concurrently. You also specify the initial number of threads that can enter the semaphore concurrently. This defines the semaphore's count.

The count is decremented each time a thread enters the semaphore, and incremented each time a thread releases the semaphore. To enter the semaphore, a thread calls one of the [Wait](#) or [WaitAsync](#) overloads. To release the semaphore, it calls one of the [Release](#) overloads. When the count reaches zero, subsequent calls to one of the Wait methods block until other threads release the semaphore. If multiple threads are blocked, there is no guaranteed order, such as FIFO or LIFO, that controls when threads enter the semaphore.

The basic structure for code that uses a semaphore to protect resources is:

```
' Enter semaphore by calling one of the Wait or WaitAsync methods.  
SemaphoreSlim.Wait()  
'  
' Execute code protected by the semaphore.  
'  
SemaphoreSlim.Release()
```

When all threads have released the semaphore, the count is at the maximum value specified when the semaphore was created. The semaphore's count is available from the [CurrentCount](#) property.

## CONSTRUCTORS

<a href="#">SemaphoreSlim(Int32)</a>	Initializes a new instance of the <a href="#">SemaphoreSlim</a> class, specifying the initial number of requests that can be granted concurrently.
<a href="#">SemaphoreSlim(Int32, Int32)</a>	Initializes a new instance of the <a href="#">SemaphoreSlim</a> class, specifying the initial and maximum number of requests that can be granted concurrently.

A thread can enter the semaphore multiple times by calling the [System.Threading.Semaphore](#) object's [WaitOne](#) method or the [SemaphoreSlim](#) object's [Wait](#) method repeatedly. To release the semaphore, the thread can either call the [Semaphore.Release\(\)](#) or [SemaphoreSlim.Release\(\)](#) method overload the same number of times, or call the [Semaphore.Release\(Int32\)](#) or [SemaphoreSlim.Release\(Int32\)](#) method overload and specify the number of entries to be released.

Esempio con i thread:

```
using System;  
using System.Threading;  
using System.Threading.Tasks;  
  
namespace ThreadGym  
{  
    class Program  
    {  
        private static SemaphoreSlim semaphore;  
        // A padding interval to make the output more orderly.  
        private static int padding;
```

```

public static void Main()
{
    // Create the semaphore.
    semaphore = new SemaphoreSlim(0, 3);
    Console.WriteLine("{0} Thread can enter the semaphore.",
                      semaphore.CurrentCount);
    Thread[] threads = new Thread[5];

    // Create and start five numbered tasks.
    for (int i = 0; i <= 4; i++)
    {
        threads[i] = new Thread(() => {
            // Each task begins by requesting the semaphore.
            Console.WriteLine("Thread {0} begins and waits for the semaphore.",
                              Thread.CurrentThread.ManagedThreadId);

            //blocks the thread/task current in waiting to enter the semaphore
            semaphore.Wait();
            //https://docs.microsoft.com/en-
us/dotnet/api/system.threading.interlocked?view=netframework-4.8
            //Adds two integers and replaces the first integer with the sum, as an
atomic operation.
            Interlocked.Add(ref padding, 100);

            Console.WriteLine("Thread {0} enters the semaphore.",
                             Thread.CurrentThread.ManagedThreadId);

            // The task just sleeps for 1+ seconds.
            Thread.Sleep(1000 + padding);
            //releases the semaphore and increments the counter
            Console.WriteLine("Thread {0} releases the semaphore; previous count:
{1}.",
                             Thread.CurrentThread.ManagedThreadId,
semaphore.Release());
                });
    }
    foreach (var thread in threads)
    {
        thread.Start();
    }
    // Wait for half a second, to allow all the tasks to start and block.
    Thread.Sleep(500);

    // Restore the semaphore count to its maximum value.
    Console.Write("Main thread calls Release(3) --> ");
    //releases the semaphore with the number specified of values
    semaphore.Release(3);
    Console.WriteLine("{0} Threads can enter the semaphore.",
                      semaphore.CurrentCount);
    // Main thread waits for the tasks to complete.
    foreach (var thread in threads)
    {
        thread.Join();
    }

    Console.WriteLine("Main thread exits.");
    Console.ReadLine();
}
}

```

Esempio con in task (i task saranno introdotti nei prossimi paragrafi):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace SemaforoDemo01
{
    class Program
    {
        private static SemaphoreSlim semaphore;
        // A padding interval to make the output more orderly.
        private static int padding;

        public static void Main()
        {
            // Create the semaphore.
            semaphore = new SemaphoreSlim(0, 3);
            Console.WriteLine("{0} tasks can enter the semaphore.", semaphore.CurrentCount);
            Task[] tasks = new Task[5];

            // Create and start five numbered tasks.
            for (int i = 0; i <= 4; i++)
            {
                tasks[i] = Task.Run(() => {
                    // Each task begins by requesting the semaphore.
                    Console.WriteLine("Task {0} begins and waits for the semaphore.", Task.CurrentId);

                    //blocks the thread/task current in waiting for the semaphore
                    semaphore.Wait();
                    //https://docs.microsoft.com/en-us/dotnet/api/system.threading.interlocked?view=netframework-4.8
                    //Adds two integers and replaces the first integer with the sum, as an atomic operation.
                    Interlocked.Add(ref padding, 100);

                    Console.WriteLine("Task {0} enters the semaphore.", Task.CurrentId);

                    // The task just sleeps for 1+ seconds.
                    Thread.Sleep(1000 + padding);

                    Console.WriteLine("Task {0} releases the semaphore; previous count: {1}.",
                        Task.CurrentId, semaphore.Release()); //releases the semaphore and increments the counter
                });
            }

            // Wait for half a second, to allow all the tasks to start and block.
            Thread.Sleep(500);

            // Restore the semaphore count to its maximum value.
            Console.Write("Main thread calls Release(3) --> ");
            //releases the semaphore with the number specified in values
            semaphore.Release(3);
            Console.WriteLine("{0} tasks can enter the semaphore.", semaphore.CurrentCount);
            // Main thread waits for the tasks to complete.
            Task.WaitAll(tasks);

            Console.WriteLine("Main thread exits.");
            Console.ReadLine();
        }
    }
}
```

<https://docs.microsoft.com/en-us/dotnet/standard/threading/semaphore-and-semaphoreslim>

### 7.1.12.1. Prodotto consumatore

Prima versione: buffer a gestione LIFO (Stack)

```
namespace ProducerConsumerLIFO
{
    class Program
    {
        static readonly object _lock = new ();
        static int BufferSize = 10;
        static SemaphoreSlim emptyPosCount = new (BufferSize,BufferSize);
        static SemaphoreSlim fillPosCount = new (0,BufferSize);
        static int firstEmptyPos = 0;
        static int[] buffer = new int[BufferSize];
        static void Main(string[] args)
        {
            Thread producer = new (ProducerWork) { Name = "Producer" };
            Thread consumer = new (ConsumerWork) { Name = "Consumer" };
            producer.Start();
            consumer.Start();

            //attendo la fine dei thread producer e consumer
            producer.Join();
            consumer.Join();

            Console.WriteLine("Fine");
        }

        private static void ConsumerWork()
        {
            while (true)
            {
                fillPosCount.Wait(); //passo il semaforo - ci sono celle piene
                lock (_lock)
                {
                    buffer[--firstEmptyPos] = 0; //simulo la lettura del dato
                    Console.WriteLine("Consumato prodotto alla posizione {0} da thread
id = {1}, thread name = {2}",
                        firstEmptyPos,
                        Environment.CurrentManagedThreadId,
                        Thread.CurrentThread.Name);
                    PrintArray(buffer);
                }
                emptyPosCount.Release();
                Thread.Sleep(2500);
            }
        }

        private static void ProducerWork()
        {
            while (true)
            {
                emptyPosCount.Wait(); //passo il semaforo - ci sono celle vuote
                lock (_lock) //sezione critica
                {
                    buffer[firstEmptyPos] = 1;
                    Console.WriteLine("Aggiunto prodotto alla posizione {0} da thread
id = {1}, thread name = {2}",
                        firstEmptyPos,
                        Environment.CurrentManagedThreadId,
                        Thread.CurrentThread.Name);
                    PrintArray(buffer);
                }
            }
        }
    }
}
```

```

        firstEmptyPos++;
    }
    fillPosCount.Release();
    Thread.Sleep(1000);
}
}

static void PrintArray(int[] array)
{
    foreach (var item in array)
    {
        Console.Write(item + " ");
    }
    Console.WriteLine();
}
}
}

```

Seconda versione: buffer gestito con tecnica FIFO, con coda circolare

```

namespace ProducerConsumerFIFO
{
    class Program
    {
        static readonly object _lock = new ();
        static int bufferSize = 10;
        static int[] buffer = new int[bufferSize];
        static SemaphoreSlim emptyPosCount = new (bufferSize, bufferSize);
        static SemaphoreSlim fillPosCount = new (0, bufferSize);
        static int writePos = 0;
        static int readPos = 0;

        static void Main(string[] args)
        {
            Thread producer = new (ProducerWork) { Name = "Producer" };
            Thread consumer = new (ConsumerWork) { Name = "Consumer" };
            producer.Start();
            consumer.Start();

            //attendo la fine dei thread producer e consumer
            producer.Join();
            consumer.Join();

            Console.WriteLine("Fine");
        }

        private static void ConsumerWork()
        {
            while (true)
            {
                fillPosCount.Wait(); //passo il semaforo - ci sono celle piene
                lock (_lock)
                {
                    buffer[readPos] = 0;
                    Console.WriteLine($"Consumato prodotto alla posizione {0} da thread
id = {1}, thread name = {2}",
                        readPos,
                        Environment.CurrentManagedThreadId,
                        Thread.CurrentThread.Name);
                    Console.WriteLine($"Numero celle libere {0}",
                        emptyPosCount.CurrentCount + 1);
                    PrintArray(buffer);
                }
            }
        }

        static void ProducerWork()
        {
            lock (_lock)
            {
                if (emptyPosCount.CurrentCount < bufferSize)
                {
                    int value = readPos + 1;
                    if (value == bufferSize)
                        value = 0;
                    buffer[value] = value;
                    emptyPosCount.Release();
                }
            }
        }
    }
}

```

```

        //incremento modulo BufferSize - coda circolare
        readPos = (readPos + 1) % bufferSize;

    }
    emptyPosCount.Release();
    Thread.Sleep(2500);
}

private static void ProducerWork()
{
    while (true)
    {
        emptyPosCount.Wait(); //passo il semaforo - ci sono celle vuote
        lock (_lock) //sezione critica
        {
            buffer[writePos] = 1;
            Console.WriteLine("Aggiunto prodotto alla posizione {0} da thread
id = {1}, thread name = {2}",
                writePos,
                Environment.CurrentManagedThreadId,
                Thread.CurrentThread.Name);
            Console.WriteLine("Numero celle occupate {0}",
                fillPosCount.CurrentCount + 1);
            PrintArray(buffer);
            //incremento modulo BufferSize - coda circolare
            writePos = (writePos+1)%bufferSize;
        }
        fillPosCount.Release();
        Thread.Sleep(1000);
    }
}

static void PrintArray(int[] array)
{
    foreach (var item in array)
    {
        Console.Write(item + " ");
    }
    Console.WriteLine();
}
}
}

```

### 7.1.12.2. Algoritmo del barbiere

Concetti ed esempi fondamentali:

[https://en.wikipedia.org/wiki/Sleeping\\_barber\\_problem](https://en.wikipedia.org/wiki/Sleeping_barber_problem)

Imagine a hypothetical barbershop with one barber, one barber chair, and a waiting room with  $n$  chairs ( $n$  may be 0) for waiting customers. The following rules apply:<sup>[4]</sup>

- If there are no customers, the barber falls asleep in the chair
- A customer must wake the barber if he is asleep
- If a customer arrives while the barber is working, the customer leaves if all chairs are occupied and sits in an empty chair if it's available
- When the barber finishes a haircut, he inspects the waiting room to see if there are any waiting customers and falls asleep if there are none<sup>[3][5]</sup>

There are two main complications. First, there is a risk that a [race condition](#), where the barber sleeps while a customer waits for the barber to get them for a haircut, arises because all of the actions—checking the waiting room, entering the shop, taking a waiting room chair—take a certain amount of time. Specifically, a customer may arrive to find the barber cutting hair so they return to the waiting room to take a seat but while walking back to the waiting room the barber finishes the haircut and goes to the waiting room, which he finds empty (because the customer walks slowly or went to the restroom) and thus goes to sleep in the barber chair. Second, another problem may occur when two customers arrive at the same time when there is only one empty seat in the waiting room and both try to sit in the single chair; only the first person to get to the chair will be able to sit.

### Prima versione – per la gestione della sezione critica si utilizza un semaforo

```
using System;
using System.Threading;

namespace BarberShopV1
{
    class Program
    {
        //per la gestione della sezione critica si utilizza un semaforo
        const int numberofSeats = 10;
        static int freeSeats = numberofSeats;
        static SemaphoreSlim seatAccess = new SemaphoreSlim(1, 1);
        static SemaphoreSlim barberReady = new SemaphoreSlim(1, 1);
        static SemaphoreSlim clientReady = new SemaphoreSlim(0, numberofSeats);
        static void Main(string[] args)
        {
            //Attivazione del thread del barbiere
            //il barbiere dorme sulla sua sedia da lavoro
            //quando arriva un cliente, il barbiere si sveglia e gli taglia i capelli
            //quando ha finito di tagliare i capelli, se c'è un altro cliente seduto
            //su una delle sedie del negozio, lo serve immediatamente, altrimenti ritorna a dormire
            //il barbiere si sveglia appena arriva un nuovo cliente
            Thread barber = new Thread(Barber);
            barber.Start();

            //Attivazione dei thread dei clienti
            //un cliente arriva in un momento qualsiasi, se trova un posto libero nella sala d'attesa si siede e aspetta fino a che non viene servito; se arriva e trova
            //tutte le sedie occupate, se ne va via
            int numberofClients = 30;
            for (int i = 0; i < numberofClients; i++)
            {
                new Thread(Client).Start();
                //i clienti arrivano con una certa frequenza al negozio
                Thread.Sleep(500);
            }
            barber.Join();
        }

        private static void Client(object obj)
        {
            //il cliente deve verificare se c'è posto
            //per farlo deve controllare il numero di posti liberi e quindi deve accedere alla
            //sezione critica
            seatAccess.Wait();
            if (freeSeats > 0)
```

```

    {
        //il cliente si siede ed occupa un posto
        freeSeats--;
        Console.WriteLine($"Il cliente con Thread Id =
{Thread.CurrentThread.ManagedThreadId} trova posto e attende di essere servito; posti
disponibili = {freeSeats}");
        seatAccess.Release(); //esce dalla sezione critica
        clientReady.Release(); //aumenta il numero di clienti disponibili
        barberReady.Wait(); //attende che il barbiere sia disponibile
        //il cliente viene servito
        Console.WriteLine($"Sono il cliente con ThreadId =
{Thread.CurrentThread.ManagedThreadId} " +
$". Il Barbiere mi sta tagliando i capelli. Posti disponibili =
{freeSeats}");
    }
    else
    {
        //libero la sezione critica
        seatAccess.Release();
        //il cliente se ne va
        Console.WriteLine($"Il cliente con ThreadId =
{Thread.CurrentThread.ManagedThreadId} non ha trovato posto e se ne va");
    }
}

private static void Barber(object obj)
{
    while (true)
    {
        clientReady.Wait(); //attendo che ci sia un cliente
        //il cliente si accomoda sulla sedia del barbiere e libera un posto
        seatAccess.Wait(); //il barbiere deve accedere alla sezione critica
        freeSeats++; //il barbiere libera un posto
        Console.WriteLine("Il barbiere libera un posto");
        seatAccess.Release(); //il barbiere esce dalla sezione critica
        Console.WriteLine($"Il Barbiere sta tagliando i capelli");
        Thread.Sleep(2000); //il barbiere sta tagliando i capelli; questo è il tempo
impiegato per tagliare i capelli
        barberReady.Release(); //il barbiere è nuovamente disponibile
    }
}
}
}

```

Seconda versione - per la gestione della sezione critica si utilizza un Monitor (lock)

```

using System;
using System.Threading;

namespace BarberShopV2
{
    class Program
    {
        //per la gestione della sezione critica si utilizza un Monitor (lock)
        const int numberOfRowsSeats = 10;
        static int freeSeats = numberOfRowsSeats;
        private static readonly object _lock = new object();
        static SemaphoreSlim barberReady = new SemaphoreSlim(1, 1);
        static SemaphoreSlim clientReady = new SemaphoreSlim(0, numberOfRowsSeats);

        static void Main(string[] args)
    }
}

```

```

{
    //Attivazione del thread del barbiere
    //il barbiere dorme sulla sua sedia da lavoro
    //quando arriva un cliente, il barbiere si sveglia e gli taglia i capelli
    //quando ha finito di tagliare i capelli, se c'è un altro cliente seduto
    //su una delle sedie del negozio, lo serve immediatamente, altrimenti ritorna a
dormire
    //il barbiere si sveglia appena arriva un nuovo cliente
    Thread barber = new Thread(Barber);
    barber.Start();

    //Attivazione dei thread dei clienti
    //un cliente arriva in un momento qualsiasi, se trova un posto libero nella
    //sala d'attesa si siede e aspetta fino a che non viene servito; se arriva e
trova
    //tutte le sedie occupate, se ne va via
    int numberOfClients = 30;
    for (int i = 0; i < numberOfClients; i++)
    {
        new Thread(Client).Start();
        //i clienti arrivano con una certa frequenza al negozio
        Thread.Sleep(500);
    }
    barber.Join();
}

private static void Client(object obj)
{
    //il cliente deve verificare se c'è posto
    //per farlo deve controllare il numero di posti liberi e quindi deve accedere
alla
    bool clienteSiSiede = false;
    //sezione critica
    //le operazioni in sezione critica devono durare il minimo possibile
    lock (_lock)
    {
        if (freeSeats > 0)
        {
            //il cliente si siede ed occupa un posto
            freeSeats--;
            Console.WriteLine($"Il cliente con Thread Id =
{Thread.CurrentThread.ManagedThreadId} trova posto e attende di essere servito;" +
                $" posti disponibili = {freeSeats}");
            clienteSiSiede = true;
        }
    }
    if (clienteSiSiede)
    {
        clientReady.Release(); //aumenta il numero di clienti disponibili
        barberReady.Wait(); //attende che il barbiere sia disponibile
        //il cliente viene servito
        Console.WriteLine($"Sono il cliente con ThreadId =
{Thread.CurrentThread.ManagedThreadId} " +
            $" Il Barbiere mi sta tagliando i capelli. Posti disponibili =
{freeSeats}");
    }
    else
    {
        //il cliente se ne va
        Console.WriteLine($"Il cliente con ThreadId =
{Thread.CurrentThread.ManagedThreadId} non ha trovato posto e se ne va");
    }
}

```

```

private static void Barber(object obj)
{
    while (true)
    {
        clientReady.Wait(); //attendo che ci sia un cliente
        //il cliente si accomoda sulla sedia del barbiere e libera un posto
        lock (_lock)
        {
            freeSeats++; //il barbiere libera un posto
            Console.WriteLine("Il barbiere libera un posto");
            }//il barbiere deve accedere alla sezione critica
            Console.WriteLine($"Il Barbiere sta tagliando i capelli");
            Thread.Sleep(2000); //il barbiere sta tagliando i capelli; questo è il tempo
            impiegato per tagliare i capelli
            barberReady.Release(); //il barbiere è nuovamente disponibile
        }
    }
}

```

Qualche spunto:

<https://github.com/meysam81/Sleeping-Barber-Problem>

<https://github.com/meysam81/Networked-Sleeping-Barber-Problem>

### 7.1.13. Deadlock

<http://www.diag.uniroma1.it/~salza/SO/SO-III-2p.pdf>

# Deadlock (stallo)

---

- Su di un tavolo ci sono un piatto ed una forchetta
- **A e B** sono seduti al tavolo, per mangiare ciascuno dei due ha bisogno sia del piatto che della forchetta
- Sciagurata sequenza di eventi:
  1. A prende la forchetta
  2. B prende il piatto
  3. Essendo il piatto occupato, **A si pone in attesa**
  4. Essendo la forchetta occupata, **B si pone in attesa**
  5. **A e B attendono per sempre, e muoiono di fame**
- Si è verificata una situazione di *deadlock*

## Risorse

---

- Il deadlock si verifica quando i Thread possono acquisire e detenere in maniera esclusiva le risorse
- Sono risorse, la CPU la memoria (centrale e di massa), e tutti i dispositivi
- Risorse di due tipi:
  - **Rilasciabili (preemptable)**: possono essere sottratte al processo senza pregiudicarne l'esecuzione
  - **Non Rilasciabili (nonpreemptable)**: se sottratte causano il fallimento dell'esecuzione del processo

# Uso delle risorse

---

- Sequenza di azioni nell'uso di una risorsa:
  1. *Richiesta della risorsa*
  2. *Uso della risorsa*
  3. *Rilascio della risorsa*
- Se al passo 1. la risorsa non è disponibile, il processo può essere messo in attesa, o la sua esecuzione può essere terminata (a seconda dei casi)
- Si suppone ciascun processo detenga la risorsa per un tempo *finito*

## Definizione

---

*Un insieme di processi si dice in deadlock se ciascuno dei processi è in attesa di un evento che solo qualcuno degli altri può causare*

- Tipicamente l'evento è il rilascio di una risorsa
- La situazione è di stallo: nessuno dei processi può:
  - *Continuare l'esecuzione*
  - *Rilasciare una risorsa*
  - *Essere risvegliato*

# Condizioni necessarie

---

- È stato dimostrato che per il deadlock sussistono le seguenti condizioni necessarie:
  1. Mutual exclusion: alcune risorse vengono assegnate in uso esclusivo
  2. Hold and wait: processi che detengono risorse possono richiederne altre
  3. No preemption: non è possibile sottrarre risorse già assegnate
  4. Circular wait: si deve essere creata una situazione di attesa circolare
- Se c'è un deadlock tutte queste condizioni sono sempre soddisfatte

## 7.1.13.1. Un esempio di deadlock: il barbiere sbagliato!

La seguente versione dell'Algoritmo del Barbiere va in deadlock. Si è introdotto appositamente un errore:

```
using System;
using System.Threading;

namespace BarberShopDeadLock
{
    class Program
    {
        //per la gestione della sezione critica si utilizza un Monitor (lock)
        const int numberofSeats = 10;
        static int freeSeats = numberofSeats;
        private static readonly object _lock = new object();
        static SemaphoreSlim barberReady = new SemaphoreSlim(1, 1);
        static SemaphoreSlim clientReady = new SemaphoreSlim(0, numberofSeats);

        static void Main(string[] args)
        {
            //Attivazione del thread del barbiere
            //il barbiere dorme sulla sua sedia da lavoro
            //quando arriva un cliente, il barbiere si sveglia e gli taglia i capelli
            //quando ha finito di tagliare i capelli, se c'è un altro cliente seduto
            //su una delle sedie del negozio, lo serve immediatamente, altrimenti ritorna a
dormire
            //il barbiere si sveglia appena arriva un nuovo cliente
            Thread barber = new Thread(Barber);
            barber.Start();

            //Attivazione dei thread dei clienti
            //un cliente arriva in un momento qualsiasi, se trova un posto libero nella
```

```

        //sala d'attesa si siede e aspetta fino a che non viene servito; se arriva e
trova
        //tutte le sedie occupate, se ne va via
        int numberOfClients = 30;
        for (int i = 0; i < numberOfClients; i++)
        {
            new Thread(Client).Start();
            //i clienti arrivano con una certa frequenza al negozio
            Thread.Sleep(500);
        }
        barber.Join();
    }

    private static void Client(object obj)
    {
        //il cliente deve verificare se c'è posto
        //per farlo deve controllare il numero di posti liberi e quindi deve accedere
alla

        //sezione critica
        //QUESTA VERSIONE VA IN DEADLOCK - QUESTO E' VOLUTO PER MOSTRARE IL PROBLEMA
DELL'ATTESA CIRCOLARE
        //IL DEADLOCK E' DOVUTO AL FATTO CHE QUANDO IL CLIENT PRENDE IL LOCK RIMANE IN
ATTESA CHE IL BARBIERE SIA DISPONIBILE
        //MA IL BARBIERE PER TAGLIARE I CAPELLI DEVE PRENDERE IL LOCK CHE NON ARRIVERA'
MAI
        lock (_lock)
    {
        if (freeSeats > 0)
        {
            //il cliente si siede ed occupa un posto
            freeSeats--;
            Console.WriteLine($"Il cliente con Thread Id =
{Thread.CurrentThread.ManagedThreadId} trova posto e attende di essere servito;" +
                $" posti disponibili = {freeSeats}");
            clientReady.Release(); //aumenta il numero di clienti disponibili
            barberReady.Wait(); //attende che il barbiere sia disponibile
            //il cliente viene servito
            Console.WriteLine($"Sono il cliente con ThreadId =
{Thread.CurrentThread.ManagedThreadId} " +
                $. Il Barbiere mi sta tagliando i capelli. Posti disponibili =
{freeSeats}");
        }
        else
        {
            //il cliente se ne va
            Console.WriteLine($"Il cliente con ThreadId =
{Thread.CurrentThread.ManagedThreadId} non ha trovato posto e se ne va");
        }
    }
}

private static void Barber(object obj)
{
    while (true)
    {
        clientReady.Wait(); //attendo che ci sia un cliente
        //il cliente si accomoda sulla sedia del barbiere e libera un posto
        Console.WriteLine("Barbiere: Sono in attesa di entrare in sezione critica");
        lock (_lock)
    }
}

```

```
        freeSeats++; //il barbiere libera un posto
        Console.WriteLine("Il barbiere libera un posto");
    }//il barbiere deve accedere alla sezione critica
    Console.WriteLine($"Il Barbiere sta tagliando i capelli");
    Thread.Sleep(2000); //il barbiere sta tagliando i capelli; questo è il tempo
impiegato per tagliare i capelli
    barberReady.Release(); //il barbiere è nuovamente disponibile
}
}
}
}
```

## 7.2. Costrutti per la programmazione concorrente – uso di thread

Gli esempi in questa sezione sono tratti da: "ITT "Giacomo Fauser" – Novara Tecnologia e progettazione di sistemi informatici e di telecomunicazioni prof. R. Fuligni"

### 7.2.1. Costrutto fork/join

Il costrutto fork permette a un processo/thread di creare un porcesso/thread figlio a cui far eseguire un'attività.

Il costrutto join permette a un processo/thread di attendere la fine di un processo/thread figlio.

### **7.2.1.1. Costrutto fork/join: esempio 1**

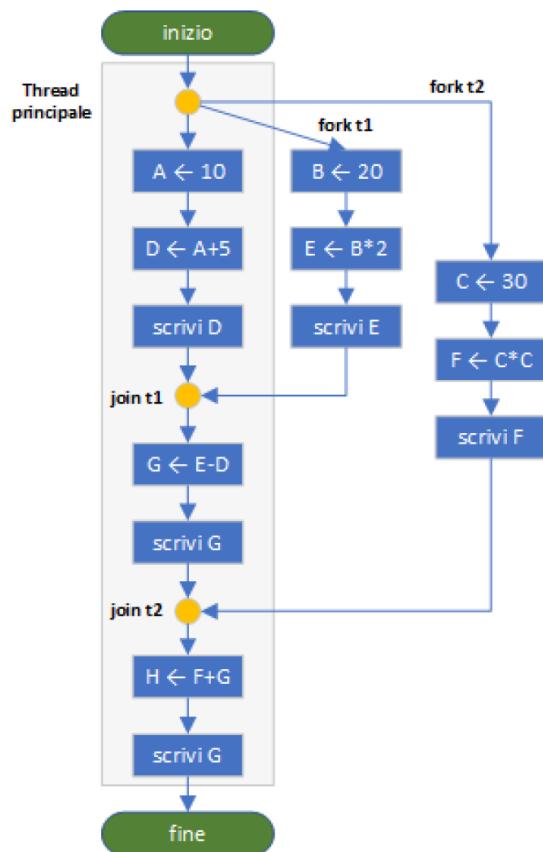
Scrivere un programma concorrente che, utilizzando il costrutto *fork/join*, esegua le operazioni indicate di seguito esprimendo il massimo grado di parallelismo.

```

inizio
    A ← 10
    B ← 20
    C ← 30
    D ← A + 5
    scrivi D
    E ← B * 2
    scrivi E
    F ← C * C
    scrivi F
    G ← E - D
    scrivi G
    H ← F + G
    scrivi H
fine

```

### Scomposizione dell'algoritmo mediante fork/join



## Pesudocodifica

<pre> /* main */ inizio     t1 ← fork(proc1)     t2 ← fork(proc2)     A ← 10     D ← A + 5     scrivi D      join t1      G ← E - D     scrivi G      join t2      H ← F + G     scrivi H fine </pre>	<pre> procedura proc1 inizio     B ← 20     E ← B * 2     scrivi E fine </pre>	<pre> procedura proc2 inizio     C ← 30     F ← C * C     scrivi F fine </pre>
---	--	--

## Codifica con l'uso dei Thread

```

using System;
using System.Threading;

namespace CostruttiProgrammazioneConcorrente
{
    class Program
    {
        static int A, B, C, D, E, F, G, H;
        static void Main(string[] args)
        {
            Thread t1 = new Thread(Proc1);
            Thread t2 = new Thread(Proc2);
            t1.Start();
            t2.Start();
            A = 10;
            D = A + 5;
            Console.WriteLine("D: {0}", D);
            t1.Join(); // Il main thread attende prima la fine di t1...
            G = E - D;
            Console.WriteLine("G: {0}", G);
            t2.Join(); // ... e poi quella di t2.
            H = F + G;
            Console.WriteLine("H: {0}", H);
        }

        static void Proc1()
        {
            B = 20;
            E = B * 2;
            Console.WriteLine("E: {0}", E);
        }
        static void Proc2()
        {
            C = 30;
            F = C * C;
            Console.WriteLine("F: {0}", F);
        }
    }
}

```

## 7.2.2. Costrutto join(count)

Riferimenti: J. Albahari, Threading in C# - Part 2: Basic Synchronization – CountdownEvent

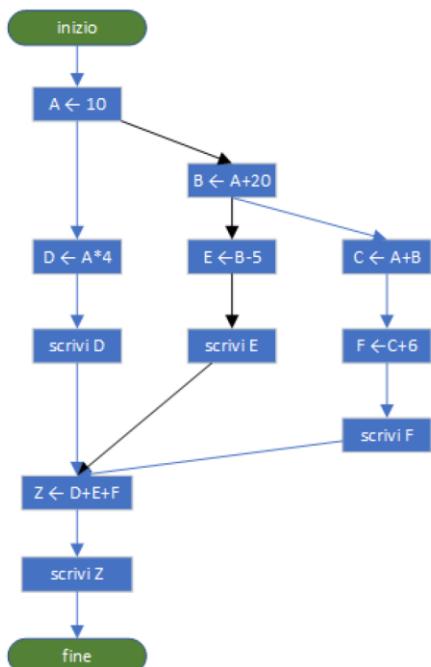
Il costrutto *join(count)* è un particolare tipo di join che permette di attendere un numero prefissato di processi/thread mediante l'utilizzo di un contatore speciale.

In C# il costrutto *join(count)* è realizzabile utilizzando uno speciale contatore costituito da un'istanza della classe *CountdownEvent*: esso è inizialmente impostato a un valore pari al numero di specificato da *count* ed è decrementato di un'unità ogni volta che uno dei thread da sincronizzare termina il proprio compito. Per decrementare il contatore si usa il metodo *Signal*.

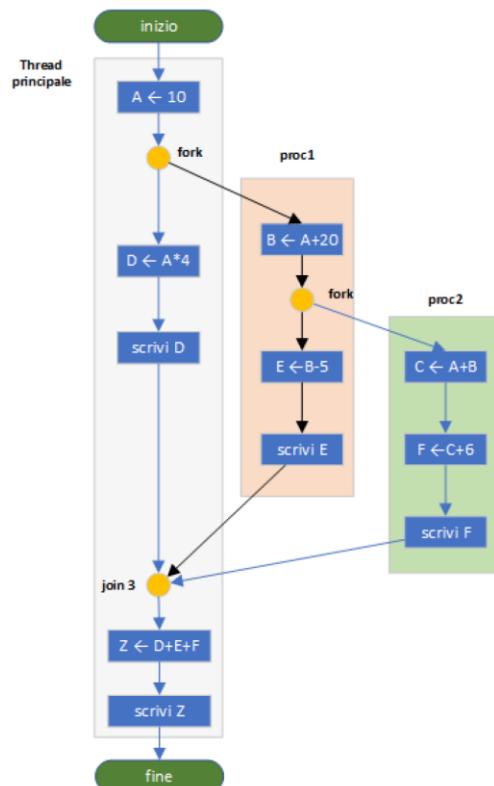
### 7.2.2.1. Costrutto join(count): esempio 1

Scrivere, utilizzando il costrutto *join(count)*, un algoritmo concorrente per il seguente diagramma delle precedenze.

Diagramma iniziale



join(count)



Pseudocodifica

```

Conta ← 3
inizio /*main*/
  A ← 10
  fork(proc1)
  D ← A * 4
  scrivi D
  L: join Conta
  Z ← D + E + F
  scrivi Z
fine
  
```

```

procedura proc1
inizio
  B ← A + 20
  fork(proc2)
  E ← B - 5
  scrivi E
  goto L
fine
  
```

```

procedura proc2
inizio
  C ← A + B
  F ← C + 6
  scrivi F
  goto L
fine
  
```

Codifica in C# con i thread

```

using System;
using System.Threading;

namespace JoinCountDemo
{
    class Program
    {
        static int A, B, C, D, E, F, Z;
        static CountdownEvent L = new CountdownEvent(3);
        static void Main(string[] args)
        {
            A = 10;
            Thread t = new Thread(Proc1);
            t.Start();
            D = A * 4;
            Console.WriteLine("D: {0}", D);
            L.Signal(); // Decrementa il contatore L
            L.Wait(); // Attende che il contatore L sia uguale a zero
            Z = D + E + F;
            Console.WriteLine("Z: {0}", Z);
        }
        static void Proc1()
        {
            Console.WriteLine("Procedura n. 1");
            B = A + 20;
            Thread t = new Thread(Proc2);
            t.Start();
            E = B - 5;
            Console.WriteLine("E: {0}", E);
            L.Signal(); // Decrementa il contatore L
        }
        static void Proc2()
        {
            Console.WriteLine("Procedura n. 2");
            C = A + B;
            F = C + 6;
            Console.WriteLine("F: {0}", F);
            L.Signal(); // Decrementa il contatore L
        }
    }
}

```

### 7.2.3. Costrutto cobegin/coend

Il costrutto cobegin si ha quando si fanno partire in parallelo più flussi di esecuzione mediante thread/processi.

Il costrutto coend si ha quando si attende la fine di più flussi di esecuzione paralleli

In C# il costrutto cobegin/coend si può realizzare per mezzo della classe Parallel.

#### 7.2.3.1. Costrutto cobegin/coend: esempio 1

Scrivere un programma concorrente a partire dal seguente diagramma, usando il costrutto *cobegin/coend*

Diagramma iniziale

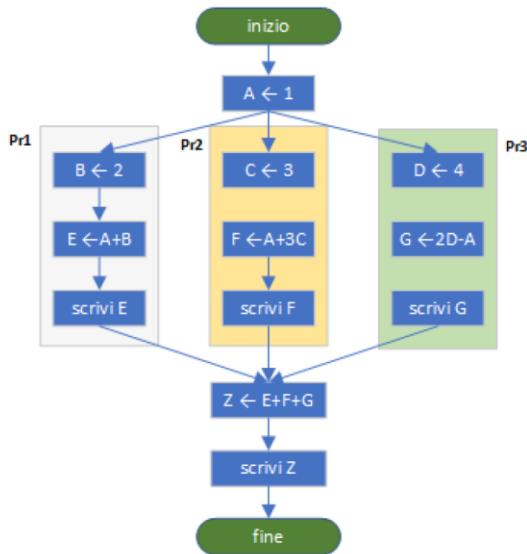
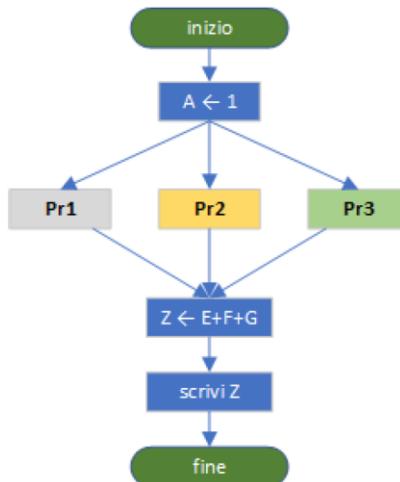


Diagramma trasformato



Pseudocodifica

inizio A ← 1 cobegin proc1 proc2 proc3 coend Z ← E + F + G scrivi Z fine	procedura proc1 inizio B ← 2 E ← A + B scrivi E fine	procedura proc2 inizio C ← 3 F ← A + 3*C scrivi F fine	procedura proc3 inizio D ← 4 G ← 2*D - A scrivi G fine
---	---	---	---

Codifica in C#

```

using System;
using System.Threading.Tasks;

namespace ParallelDemo
{
    class Program
    {
        static int A, B, C, D, E, F, G, Z;
        static void Main(string[] args)
        {
            A = 1;
            // Esecuzione parallela delle tre procedure, equivalente a:
            // cobegin
            // Proc1();
            // Proc2();
            // Proc3();
            // coend
            // Il metodo Invoke esegue le tre procedure, normalmente in parallelo.
            // Il main thread attende il completamento di tutte le procedure indicate.
            Parallel.Invoke(Proc1, Proc2, Proc3);
            Z = E + F + G;
            Console.WriteLine("Z: {0}", Z);
        }

        static void Proc1()
        {
            B = 2;
            E = A + B;
            Console.WriteLine("E: {0}", E);
        }
    }
}
  
```

```

        }
        static void Proc2()
        {
            C = 3;
            F = A + 3 * C;
            Console.WriteLine("F: {0}", F);
        }
        static void Proc3()
        {
            D = 4;
            G = 2 * D - A;
            Console.WriteLine("G: {0}", G);
        }
    }
}

```

## 7.3. Task Parallel Library (TPL)

<https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-parallel-library-tpl>

La libreria Task Parallel Library (TPL) è un set di tipi e API pubblici negli spazi dei nomi `System.Threading` e `System.Threading.Tasks`. Lo scopo di TPL è di rendere gli sviluppatori più produttivi mediante la semplificazione del processo di aggiunta di parallelismo e concorrenza alle applicazioni. La libreria TPL ridimensiona il grado di concorrenza dinamicamente per utilizzare in modo efficace tutti i processori disponibili. La libreria TPL gestisce inoltre il partizionamento del lavoro, la pianificazione dei thread in `ThreadPool`, il supporto per l'annullamento, la gestione dello stato e altri dettagli di basso livello. Quando si utilizza TPL, è possibile ottimizzare le prestazioni del codice concentrandosi sulle operazioni per cui il programma è stato progettato.

A partire da .NET Framework 4, TPL è la soluzione più adatta per scrivere codice multithreading e parallelo. Tuttavia, non tutto il codice è adatto per la parallelizzazione; ad esempio, se un ciclo esegue solo una piccola parte di lavoro in ciascuna iterazione o non viene eseguito per molte iterazioni, il sovraccarico della parallelizzazione potrebbe provocare un rallentamento dell'esecuzione del codice. Inoltre, la parallelizzazione come qualsiasi codice multithreading, aggiunge complessità all'esecuzione del programma. Benché la libreria TPL semplifichi gli scenari multithreading, è consigliabile avere una conoscenza di base dei concetti relativi all'utilizzo dei thread, ad esempio blocchi, deadlock e race condition. Infatti, ciò consentirà di utilizzare la libreria TPL in modo più efficace.

### 7.3.1. C# Task

Task is one of the central elements of the task-based asynchronous pattern in the .NET. Task object typically executes asynchronously on a thread pool thread rather than synchronously on the main application thread. As we learned in the previous chapter we can offload the work to thread pool using the queue user work item method. However, this method has its weaknesses as we can't tell whether the operation has finished or what a return value is. This is where a Task can be helpful. The Task can tell you if the work is completed and if the operation returns a result. A Task is an object that represents some work that should be done.

Tasks provide a sophisticated way to handle async or parallel operation by providing various options like:

- Ability to cancel an ongoing operation
- Return resulting value from operation (like a method functions)
- Easy Exception Handling

- High-Level constructs like a parallel loop
- Task continuation

Tasks can be used to make your application more responsive. If the thread that manages the user interface offloads work to another thread from the thread pool, it can keep processing user events and ensure that the application can still be used.

You can also parallelize your CPU bound operation on to multiple processors using task.

### 7.3.1.1. C# Task: Programmazione asincrona e/o parallela basata su attività

<https://docs.microsoft.com/it-it/dotnet/standard/parallel-programming/task-based-asynchronous-programming>

La libreria Task Parallel Library (TPL) si basa sul concetto di *attività*, che rappresenta un'operazione asincrona. Per certi versi, un'attività è analoga a un thread o a un elemento di lavoro [ThreadPool](#), ma a un livello più generale di astrazione. L'espressione *parallelismo delle attività* fa riferimento a una o più attività indipendenti eseguite contemporaneamente. Le attività forniscono due vantaggi principali:

- di sistema.

Ai fini del funzionamento, le attività vengono accodate in un [ThreadPool](#), che è stato migliorato con algoritmi che determinano e modificano il numero di thread e che ottimizzano la velocità effettiva grazie al bilanciamento del carico. In questo modo le attività diventano relativamente leggere ed è possibile crearne molte in modo da ottenere un parallelismo accurato.

- Maggior controllo a livello di codice rispetto a quello ottenibile usando un thread o un elemento di lavoro.

Le attività e il framework in cui esse sono inserite forniscono un ampio set di API che supportano varie funzionalità tra cui attesa, annullamento, continuazioni, gestione affidabile delle eccezioni, stato dettagliato e pianificazione personalizzata.

Per entrambi questi motivi, in .NET TPL rappresenta l'API preferita per la scrittura di codice multithreading, asincrono e parallelo.

Il metodo [Parallel.Invoke](#) rappresenta un modo pratico per eseguire simultaneamente un numero qualsiasi di istruzioni arbitrarie. Basta passare un delegato [Action](#) per ogni elemento di lavoro. Il modo più semplice per creare questi delegati è usare le espressioni lambda. L'espressione lambda può chiamare un metodo denominato o fornire il codice inline. Nell'esempio seguente viene mostrata una chiamata a [Invoke](#) di base che crea e avvia due attività in esecuzione simultanea. La prima attività è rappresentata da un'espressione lambda che chiama un metodo denominato DoSomeWork e la seconda attività è rappresentata da un'espressione lambda che chiama un metodo denominato DoSomeOtherWork.

```
Parallel.Invoke(() => DoSomeWork(), () => DoSomeOtherWork());
```

Un'attività che non restituisce un valore è rappresentata dalla classe.

[System.Threading.Tasks.Task](#).

Un'attività che restituisce un valore viene rappresentata dalla classe.

[System.Threading.Tasks.Task<TResult>](#) che eredita da [Task](#).

L'oggetto attività gestisce i dettagli dell'infrastruttura e fornisce metodi e proprietà accessibili dal thread chiamante per tutta la durata dell'attività. Ad esempio, è possibile accedere in qualsiasi momento alla proprietà [Status](#) di un'attività per determinare se è stata avviata, eseguita fino al completamento o annullata oppure se ha generato un'eccezione. Lo stato è rappresentato da un'enumerazione [TaskStatus](#).

Quando si crea un'attività si assegna ad essa un delegato dell'utente che incapsula il codice che verrà eseguito dall'attività. Il delegato può essere espresso come un delegato denominato, un metodo anonimo o un'espressione lambda. Le espressioni lambda possono contenere una chiamata a un metodo denominato, come mostrato nell'esempio seguente. Si noti che l'esempio include una chiamata al metodo [Task.Wait](#) per garantire che l'attività venga completata prima che termini l'applicazione in modalità console.

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace TaskIntro
{
    class Program
    {
        static void Main(string[] args)
        {
            Thread.CurrentThread.Name = "Main";

            // Create a task and supply a user delegate by using a lambda expression.
            Task taskA = new Task(() => Console.WriteLine("Hello from taskA."));
            // Start the task.
            taskA.Start();

            // Output a message from the calling thread.
            Console.WriteLine("Hello from thread '{0}'.",
                Thread.CurrentThread.Name);
            taskA.Wait();
        }
    }
}

// The example displays output like the following:
//      Hello from thread 'Main'.
//      Hello from taskA.
```

È anche possibile usare i metodi [Task.Run](#) per creare e avviare un'attività in un'unica operazione. Per gestire l'attività, i metodi [Run](#) utilizzano l'utilità di pianificazione delle attività predefinita, indipendentemente dall'utilità di pianificazione associata al thread corrente. I metodi [Run](#) rappresentano il modo preferito per creare e avviare le attività nei casi in cui non occorre un maggiore controllo sulla creazione e la pianificazione di attività.

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace TaskRunDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            Thread.CurrentThread.Name = "Main";

            // Define and run the task.
            Task taskA = Task.Run(() => Console.WriteLine("Hello from taskA."));
        }
    }
}
```

```

        // Output a message from the calling thread.
        Console.WriteLine("Hello from thread '{0}'.",
                           Thread.CurrentThread.Name);
        taskA.Wait();
    }
}
// The example displays output like the following:
//      Hello from thread 'Main'.
//      Hello from taskA.

```

È anche possibile usare il metodo [TaskFactory.StartNew](#) per creare e avviare un'attività in un'unica operazione. Usare questo metodo quando non occorre separare la creazione e la pianificazione ed è necessario avvalersi di opzioni aggiuntive di creazione delle attività, o usare un'utilità di pianificazione specifica, oppure quando si deve passare uno stato aggiuntivo all'attività che è possibile recuperare tramite la proprietà [Task.AsyncState](#), come illustrato nell'esempio seguente.

```

using System;
using System.Threading;
using System.Threading.Tasks;

namespace TaskWithCustomData
{
    class CustomData
    {
        public long CreationTime;
        public int Name;
        public int ThreadNum;
    }

    class Program
    {
        public static void Main()
        {
            Task[] taskArray = new Task[10];
            for (int i = 0; i < taskArray.Length; i++)
            {
                taskArray[i] = Task.Factory.StartNew((object obj) =>
                {
                    CustomData data = obj as CustomData;
                    if (data == null)
                        return;

                    data.ThreadNum = Thread.CurrentThread.ManagedThreadId;
                }, new CustomData() { Name = i, CreationTime = DateTime.Now.Ticks });
            }
            Task.WaitAll(taskArray);
            foreach (var task in taskArray)
            {
                //AsyncState restituisce l'oggetto che è stato passato quando il Task è stato creato, oppure
                //null se non è stato passato nulla
                var data = task.AsyncState as CustomData;
                if (data != null)
                    Console.WriteLine($"Task Name = {data.Name}, Task Id = {task.Id}, Task status = {task.Status}, created at {data.CreationTime}, ran on Thread Id = {data.ThreadNum}.");
            }
        }
    }
}
// The example displays output like the following:
Task Name = 0, Task Id = 1, Task status = RanToCompletion, created at 637737183135068576, ran on Thread Id = 5.
Task Name = 1, Task Id = 2, Task status = RanToCompletion, created at 637737183135317927, ran on Thread Id = 6.
Task Name = 2, Task Id = 3, Task status = RanToCompletion, created at 637737183135345356, ran on Thread Id = 4.
Task Name = 3, Task Id = 4, Task status = RanToCompletion, created at 637737183135369204, ran on Thread Id = 7.
Task Name = 4, Task Id = 5, Task status = RanToCompletion, created at 637737183135391197, ran on Thread Id = 11.
Task Name = 5, Task Id = 6, Task status = RanToCompletion, created at 637737183135415846, ran on Thread Id = 13.

```

```

Task Name = 6, Task Id = 7, Task status = RanToCompletion, created at 637737183135794745, ran on Thread Id = 8.
Task Name = 7, Task Id = 8, Task status = RanToCompletion, created at 637737183135879399, ran on Thread Id = 10.
Task Name = 8, Task Id = 9, Task status = RanToCompletion, created at 637737183135879724, ran on Thread Id = 12.
Task Name = 9, Task Id = 10, Task status = RanToCompletion, created at 637737183135879799, ran on Thread Id = 9.

```

Ogni attività riceve un ID di tipo Integer con cui viene identificata in modo univoco in un dominio applicazione e a cui è possibile accedere tramite la proprietà [Task.Id](#).

Quando si usa un'espressione lambda per creare un delegato, si avrà accesso a tutte le variabili visibili in quel punto del codice sorgente. Tuttavia, in alcuni casi, in particolare all'interno dei cicli, un'espressione lambda non acquisisce la variabile come previsto. Acquisisce solo il valore finale, non il valore mentre viene modificato dopo ogni iterazione. Nell'esempio che segue viene illustrato il problema. Si passa un contatore di cicli a un'espressione lambda che crea un'istanza di un oggetto CustomData e usa il contatore di cicli come identificatore dell'oggetto. Come indica l'output dell'esempio, ogni oggetto CustomData dispone di un identificatore identico.

```

using System;
using System.Threading;
using System.Threading.Tasks;

namespace TaskWithCustomDataSbagliato
{
    class CustomData
    {
        public long CreationTime;
        public int Name;
        public int ThreadNum;
    }

    class Program
    {
        public static void Main()
        {
            Task[] taskArray = new Task[10];
            for (int i = 0; i < taskArray.Length; i++)
            {
                taskArray[i] = Task.Factory.StartNew((object obj) => {
                    var data = new CustomData() { Name = i, CreationTime = DateTime.Now.Ticks };
                    data.ThreadNum = Thread.CurrentThread.ManagedThreadId;
                    Console.WriteLine($"Task Name = {data.Name}, created at {data.CreationTime}, ran on
Thread Id = {data.ThreadNum}.");
                }, i);
            }
            Task.WaitAll(taskArray);
            foreach (var task in taskArray)
            {
                //AsyncState restituisce l'oggetto che è stato passato quando il Task è stato creato, oppure
                null se non è stato passato nulla
                var data = task.AsyncState as int?;
                if (data != null)
                    Console.WriteLine($"Supplied object= {data}, Task Id = {task.Id}, Task status =
{task.Status}.");
            }
        }
    }
}

// The example displays output like the following:
Task Name = 10, created at 637738089310514965, ran on Thread Id = 5.
Task Name = 10, created at 637738089310514608, ran on Thread Id = 11.
Task Name = 10, created at 637738089310514702, ran on Thread Id = 4.
Task Name = 10, created at 637738089310514798, ran on Thread Id = 6.
Task Name = 10, created at 637738089310514412, ran on Thread Id = 7.
Task Name = 10, created at 637738089310514896, ran on Thread Id = 8.
Task Name = 10, created at 637738089310514197, ran on Thread Id = 9.
Task Name = 10, created at 637738089310514323, ran on Thread Id = 10.
Task Name = 10, created at 637738089310514505, ran on Thread Id = 12.
Task Name = 10, created at 637738089310514224, ran on Thread Id = 13.
Supplied object = 0, Task Id = 1, Task status = RanToCompletion.

```

```

Supplied object = 1, Task Id = 2, Task status = RanToCompletion.
Supplied object = 2, Task Id = 3, Task status = RanToCompletion.
Supplied object = 3, Task Id = 4, Task status = RanToCompletion.
Supplied object = 4, Task Id = 5, Task status = RanToCompletion.
Supplied object = 5, Task Id = 6, Task status = RanToCompletion.
Supplied object = 6, Task Id = 7, Task status = RanToCompletion.
Supplied object = 7, Task Id = 8, Task status = RanToCompletion.
Supplied object = 8, Task Id = 9, Task status = RanToCompletion.
Supplied object = 9, Task Id = 10, Task status = RanToCompletion.

```

Come si vede dall'esempio sbagliato, nonostante il valore del *supplied object* passato al task sia quello corretto, la lambda espression accede al valore della variabile di conteggio *i* quando il suo valore è già arrivato al massimo.

La soluzione è in questo caso quella di passare l'oggetto CustomData e non l'intero *i* da cui viene creato CustomData.

### 7.3.1.1.1. Execution Model Of A Task

Let's discuss the basic execution model of a task. In the example above, we created a task and provide a basic operation to be performed by the task. So this task operation is executed in a separate thread. **Each task we create is executed within its own thread**. In a single-core machine, this won't be efficient as there will be a lot of context switching between these threads and they will end up using the same single-core processor. However, in a multi-core processor, which is the case nowadays, this will be very efficient as these threads will get distributed within the multi-core machine (each thread executing in the separate processor) and thus will perform better than a sequential program.

Please note that:

```
//wait for task t to complete its execution  
t.Wait();
```

**Calling Wait method of the task is equivalent to calling the Join method on a thread.** When the Wait method is called within the Main method the main thread pauses its execution until the task *t* completes its execution.

### 7.3.1.1.2. C# Task That Returns A Result

The .NET also has the generic version of task Task<T> class that you can use if a Task should return a value. Here T is the data type you want to return as a result. Below code shows how this works.

```

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace TaskReturnsValue01  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            //qui int è il tipo restituito dal Task  
            //si noti che in questo caso si è usato il costrutto Task.Run(lambda) e non new  
            Task(lambda)  
            {  
                Task<int> t = Task.Run(() =>  
                {  
                    return 32;  
                });  
  
                //qui leggiamo il valore restituito dal Task  
                //è come fare una Join sul thread del Task oppure come invocare la Wait sul Task
            }
        }
    }
}
```

```

        Console.WriteLine(t.Result);

        Console.WriteLine();
        Console.WriteLine("Press Enter to terminate!");
        Console.ReadLine();
    }
}
}

```

**Attempting to read the Result property on a Task will force the thread that's trying to read the result to wait until the task is finished**, which is equivalent to calling Join method on a thread and calling Wait method in the task as mentioned before. As long as the Task has not finished, it is impossible to give the result. If the task is not finished, this call will block the current thread.

### 7.3.1.1.3. Adding continuation

Nella programmazione asincrona è comune che, dopo il completamento, un'operazione asincrona chiami una seconda operazione e passi i dati a tale operazione. Tradizionalmente le continuazioni venivano eseguite tramite metodi di callback. In Task Parallel Library la stessa funzionalità viene resa possibile dalle *attività di continuazione*. Un'attivazione di continuazione, chiamata anche semplicemente continuazione, è un'attività asincrona richiamata da un'altra attività, denominata *attività precedente*, al termine di quest'ultima.

Le continuazioni sono relativamente facili da usare, ma sono comunque potenti e flessibili. Ad esempio, è possibile eseguire queste operazioni:

- Passare dati dall'attività precedente alla continuazione.
- Specificare le condizioni esatte in cui la continuazione verrà richiamata o non richiamata.
- Annullare una continuazione prima del suo avvio o cooperativamente durante la sua esecuzione.
- Fornire suggerimenti sul modo in cui pianificare la continuazione.
- Richiamare più continuazioni dalla stessa attività precedente.
- Richiamare una continuazione al termine di tutte o una delle attività precedenti.
- Concatenare continuazioni una dopo l'altra a qualsiasi lunghezza arbitraria.
- Usare una continuazione per gestire le eccezioni generate dall'attività precedente.

Per creare una continuazione che viene eseguita una volta completata l'attività precedente, chiamare il metodo [Task.ContinueWith](#). L'esempio seguente mostra il modello di base. Per chiarezza, viene omessa la gestione delle eccezioni. Viene eseguita un'attività precedente, taskA, che restituisce un oggetto [DayOfWeek](#) che indica il nome del giorno corrente della settimana. Al termine dell'attività precedente, all'attività di continuazione continuation viene passata l'attività precedente e viene visualizzata una stringa che ne include i risultati.

```

using System;
using System.Threading.Tasks;

namespace ContinuationDemo01
{
    class Program
    {
        static void Main(string[] args)
        {
            // Execute the antecedent.
            Task<DayOfWeek> taskA = Task.Run(() => DateTime.Today.DayOfWeek);

            // Execute the continuation when the antecedent finishes.
            taskA.ContinueWith(antecedent => Console.WriteLine("Today is {0}.",
antecedent.Result));
            Console.ReadLine();
        }
    }
}

```

```
}
```

This method is similar to calling the callback method when a certain operation is finished. Below is an example of creating such a continuation.

```
using System;
using System.Threading.Tasks;

namespace TaskContinuation01
{
    class Program
    {
        static void Main(string[] args)
        {
            Task<int> t = Task.Run(() =>
            {
                return 32;
            }).ContinueWith((i) =>
            {
                return i.Result * 2;
            });

            t.ContinueWith((i) =>
            {
                Console.WriteLine(i.Result);
            });

            Console.WriteLine("Press Enter to terminate!");
            Console.ReadLine();
        }
    }
}
```

È anche possibile creare una continuazione che verrà eseguita al termine di una o tutte le attività di un gruppo di attività. Per eseguire una continuazione al termine di tutte le attività precedenti, chiamare il metodo statico [Task.WhenAll](#) o il metodo [TaskFactory.ContinueWhenAll](#) dell'istanza. Per eseguire una continuazione al termine di almeno una delle attività precedenti, chiamare il metodo statico [Task.WhenAny](#) o il metodo [TaskFactory.ContinueWhenAny](#) dell'istanza.

Nell'esempio seguente viene chiamato il metodo [Task.WhenAll\(IEnumerable<Task>\)](#) per creare un'attività di continuazione che riflette il risultato delle 10 attività precedenti. Ogni attività precedente eleva al quadrato un valore di indice compreso tra uno e 10. Se le attività precedenti vengono completate correttamente (ovvero la relativa proprietà [Task.Status](#) è [TaskStatus.RanToCompletion](#)), la proprietà [Task<TResult>.Result](#) della continuazione è un vettore dei valori [Task<TResult>.Result](#) restituiti da ogni attività precedente. L'esempio somma tali valori per calcolare la somma dei quadrati per tutti i numeri compresi tra uno e 10.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace ContinuationAllDemo01
{
    class Program
    {
        static void Main(string[] args)
        {
            List<Task<int>> tasks = new List<Task<int>>();
            for (int ctr = 1; ctr <= 10; ctr++)
            {
                int baseValue = ctr;
                tasks.Add(Task.Factory.StartNew((b) => {
                    int i = (int)b;
                    return i * i;
                }));
            }

            Task<int> continuation = Task.WhenAll(tasks).ContinueWith((i) => {
                int sum = 0;
                foreach (var task in tasks)
                {
                    sum += task.Result;
                }
                return sum;
            });
            continuation.Wait();
        }
    }
}
```

```
        }, baseValue));
    }
    //continuation è di tipo Task<int[]>
    var continuation = Task.WhenAll(tasks);

    long sum = 0;
    for (int ctr = 0; ctr <= continuation.Result.Length - 1; ctr++)
    {
        Console.Write($"{continuation.Result[ctr]}{(ctr == continuation.Result.Length - 1 ? " = " : " +
        ")}");
        sum += continuation.Result[ctr];
    }
    Console.WriteLine(sum);
}
}
```

#### 7.3.1.1.4. Scheduling Different Continuation Tasks

The `ContinueWith` method has a couple of overloads that you can use to configure when the continuation will run. This way you can add different continuation methods that will run when an exception happens, the Task is canceled, or the Task completes successfully. Below code shows how to do this.

```
using System;
using System.Threading.Tasks;

namespace ContinuationWithExceptionHandling01
{
    class Program
    {
        static void Main(string[] args)
        {
            var t = Task.Run(() => {
                DateTime dat = DateTime.Now;
                if (dat == DateTime.MinValue)
                    throw new ArgumentException("The clock is not working.");

                if (dat.Hour > 17)
                    return "evening";
                else if (dat.Hour > 12)
                    return "afternoon";
                else
                    return "morning";
            });
            var c = t.ContinueWith((antecedent) => {
                if (antecedent.Status == TaskStatus.RanToCompletion)
                {
                    Console.WriteLine("Good {0}!",
                        antecedent.Result);
                    Console.WriteLine("And how are you this fine {0}?",
                        antecedent.Result);
                }
                else if (antecedent.Status == TaskStatus.Faulted)
                {
                    Console.WriteLine(t.Exception.GetBaseException().Message);
                }
            });
            c.Wait();
        }
    }
}
```

Altro esempio: continuazione condizionale.

```
using System;
using System.Collections.Generic;
```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DifferentContinuationTasks
{
    class Program
    {
        static void Main(string[] args)
        {
            Task<int> t = Task.Run(() =>
            {
                //se il numero è messo a zero viene sollevata un'eccezione non gestita
                int numero = 1;
                if (numero==0) {
                    throw new Exception();
                }
                return 32;
            });
            t.ContinueWith((i) =>
            {
                Console.WriteLine("Faulted");
            }, TaskContinuationOptions.OnlyOnFaulted);
            t.ContinueWith((i) =>
            {
                Console.WriteLine("Canceled");
            }, TaskContinuationOptions.OnlyOnCanceled);

            var completedTask = t.ContinueWith((i) =>
            {
                Console.WriteLine(i.Result);
                Console.WriteLine("Completed");
            }, TaskContinuationOptions.OnlyOnRanToCompletion);

            Console.WriteLine("Press Enter to terminate!");
            Console.ReadLine();
        }
    }
}

```

Per l'esempio precedente, nel caso in cui viene sollevata un'eccezione Visual Studio blocca il programma nel punto in cui c'è l'eccezione non gestita. Se si fa partire l'applicazione direttamente dall'eseguibile il programma non si blocca e va direttamente nel continuation task che ha come impostazione TaskContinuationOptions.OnlyOnFaulted. Si può eventualmente andare nelle opzioni di gestione dell'eccezione di Visual Studio per modificarne il comportamento.

### 7.3.2. Costrutti per la programmazione concorrente – uso di Task

#### 7.3.2.1. Costrutto fork/join

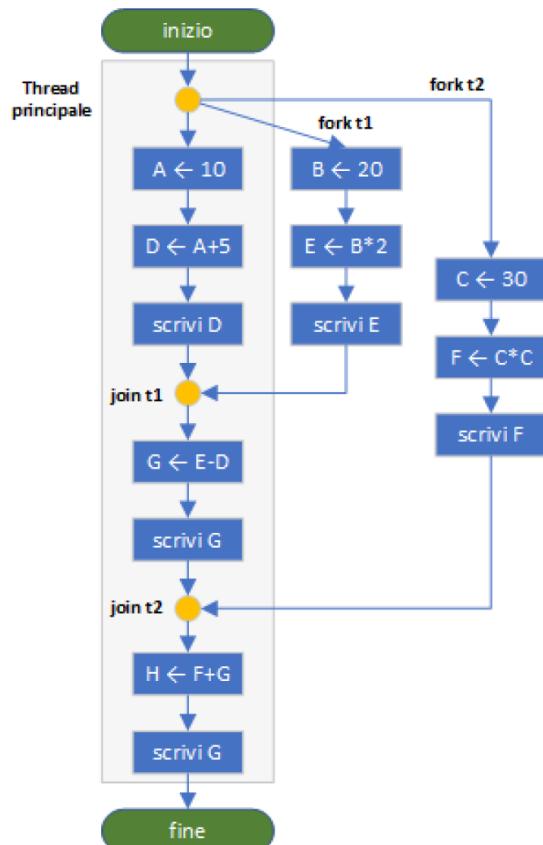
L'esempio di fork/join della sezione 7.2.1 può essere codificato in C# con l'utilizzo della libreria TPL, come mostrato nel seguente codice:

```

inizio
  A ← 10
  B ← 20
  C ← 30
  D ← A + 5
  scrivi D
  E ← B * 2
  scrivi E
  F ← C * C
  scrivi F
  G ← E - D
  scrivi G
  H ← F + G
  scrivi H
fine

```

## Scomposizione dell'algoritmo mediante fork/join



```

using System;
using System.Threading.Tasks;

namespace CostruttiProgrammazioneConcorrenteTask
{
    class Program
    {
        static int A, B, C, D, E, F, G, H;
        static void Main(string[] args)
        {
            // Metodo classico di avvio di un nuovo task
            Task tsk1 = Task.Factory.StartNew(Proc1);
            Task tsk2 = Task.Factory.StartNew(Proc2);

            A = 10;
            D = A + 5;
            Console.WriteLine("D: {0}", D);
            tsk1.Wait(); // Attende il completamento del task tsk1
            G = E - D;
            Console.WriteLine("G: {0}", G);
            tsk2.Wait(); // Attende il completamento del task tsk2
            H = F + G;
            Console.WriteLine("H: {0}", H);
        }

        static void Proc1()
        {
            B = 20;
            E = B * 2;
            Console.WriteLine("E: {0}", E);
        }
        static void Proc2()
        {
            C = 30;
            F = C * C;
            Console.WriteLine("F: {0}", F);
        }
    }
}

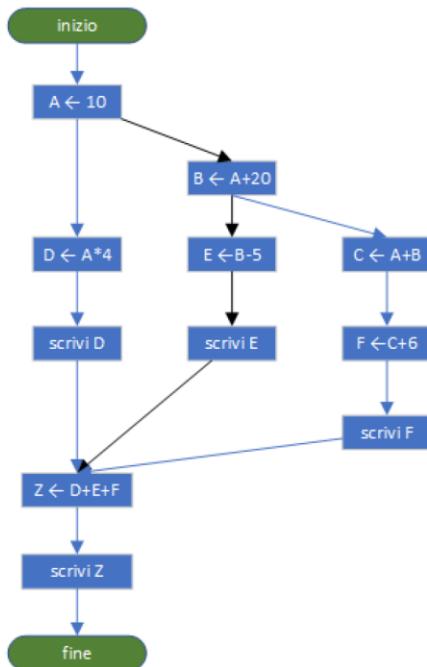
```

}

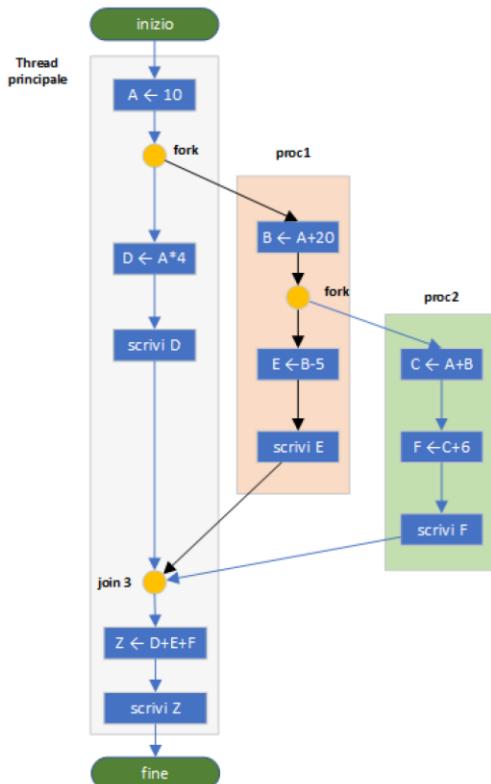
### 7.3.2.2. Costrutto join(count)

L'esempio di join(count) della sezione 7.2.2 può essere codificato in C# con l'utilizzo della libreria TPL, come mostrato nel seguente codice:

Diagramma iniziale



join(count)



```

using System;
using System.Threading;
using System.Threading.Tasks;

namespace JoinCountDemoTask
{
    class Program
    {
        static int A, B, C, D, E, F, Z;
        static CountdownEvent L = new CountdownEvent(3);
        static void Main(string[] args)
        {
            A = 10;
            Task t = Task.Factory.StartNew(Proc1);
            D = A * 4;
            Console.WriteLine("D: {0}", D);
            L.Signal(); // Decrementa il contatore L
            L.Wait(); // Attende che il contatore L sia uguale a zero
            Z = D + E + F;
            Console.WriteLine("Z: {0}", Z);
        }
        static void Proc1()
        {
            Console.WriteLine("Procedura n. 1");
            B = A + 20;
            Task t = Task.Factory.StartNew(Proc2);
            E = B - 5;
        }
        static void Proc2()
        {
            Console.WriteLine("Procedura n. 2");
            C = B + A;
            F = C + 6;
            Console.WriteLine("F: {0}", F);
        }
    }
}
    
```

```

        Console.WriteLine("E: {0}", E);
        L.Signal(); // Decrementa il contatore L
    }
    static void Proc2()
    {
        Console.WriteLine("Procedura n. 2");
        C = A + B;
        F = C + 6;
        Console.WriteLine("F: {0}", F);
        L.Signal(); // Decrementa il contatore L
    }
}
}

```

### 7.3.2.3. Costrutto cobegin/coend

L'esempio di cobegin/coend della sezione 7.2.3 può essere codificato in C# con l'utilizzo della libreria TPL, come mostrato nel seguente codice:

Diagramma iniziale

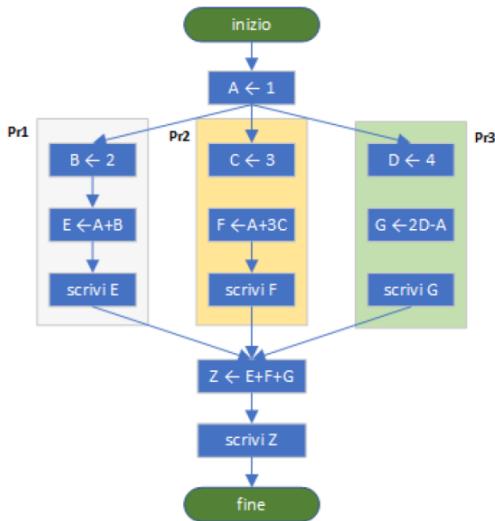
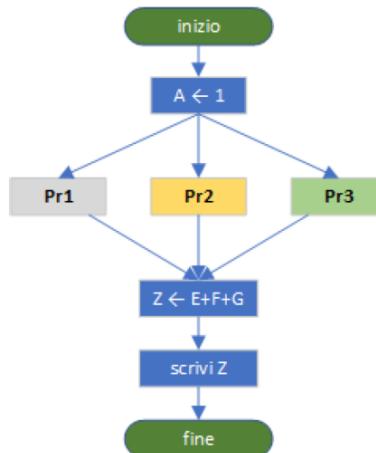


Diagramma trasformato



```

using System;
using System.Threading.Tasks;

namespace ParallelDemoTask
{
    class Program
    {
        static int A, B, C, D, E, F, G, Z;
        static void Main(string[] args)
        {
            A = 1;
            // Esecuzione parallela delle tre procedure, equivalente a:
            // cobegin
            // Proc1();
            // Proc2();
            // Proc3();
            // coend
            // Il metodo Invoke esegue le tre procedure, normalmente in parallelo.
            // Il main thread attende il completamento di tutte le procedure indicate.
            Task tsk1 = Task.Factory.StartNew(Proc1);
            Task tsk2 = Task.Factory.StartNew(Proc2);
            Task tsk3 = Task.Factory.StartNew(Proc3);
            // Il main thread attende il completamento di tutti i task
            Task.WaitAll(tsk1, tsk2, tsk3);
            Z = E + F + G;
            Console.WriteLine("Z: {0}", Z);
        }
    }
}

```

```

        }

        static void Proc1()
        {
            B = 2;
            E = A + B;
            Console.WriteLine("E: {0}", E);
        }
        static void Proc2()
        {
            C = 3;
            F = A + 3 * C;
            Console.WriteLine("F: {0}", F);
        }
        static void Proc3()
        {
            D = 4;
            G = 2 * D - A;
            Console.WriteLine("G: {0}", G);
        }
    }
}

```

### 7.3.3. C# Task: Parallelismo dei dati (Task Parallel Library)

<https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/data-parallelism-task-parallel-library>

Con l'espressione *parallelismo dei dati* ci si riferisce a scenari in cui la stessa operazione viene eseguita contemporaneamente (ovvero in parallelo) sugli elementi di una matrice o una raccolta di origine. Nelle operazioni in parallelo su dati la raccolta di origine viene suddivisa in partizioni in modo che più thread possano agire simultaneamente su segmenti diversi.

La libreria Task Parallel Library (TPL) supporta il parallelismo dei dati tramite la classe [System.Threading.Tasks.Parallel](#). Questa classe fornisce le implementazioni in parallelo basate su metodo dei cicli [for](#) e [foreach](#). La scrittura della logica di un ciclo [Parallel.For](#) o [Parallel.ForEach](#) è molto simile a quella della logica di un ciclo sequenziale. Non è necessario creare thread o accodare elementi di lavoro. Nei cicli di base non è necessario acquisire blocchi. La libreria TPL gestisce automaticamente tutto il lavoro di basso livello. Per informazioni dettagliate sull'uso di [Parallel.For](#) e [Parallel.ForEach](#), scaricare il documento [Patterns for Parallel Programming: Understanding and Applying Parallel Patterns with the .NET Framework 4](#) (Modelli per la programmazione parallela: informazioni sui modelli paralleli con .NET Framework 4 e su come applicarli).

#### 7.3.3.1. Esempio 1 di [Parallel.For](#):

<https://docs.microsoft.com/it-it/dotnet/standard/parallel-programming/how-to-write-a-simple-parallel-for-loop>

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.IO;

namespace ParallelForDemo
{
    class Program
    {
        public static void Main()
        {
            long totalSize = 0;

```

```

String[] args = Environment.GetCommandLineArgs();
if (args.Length == 1)
{
    Console.WriteLine("There are no command line arguments.");
    return;
}
if (!Directory.Exists(args[1]))
{
    Console.WriteLine("The directory does not exist.");
    return;
}

String[] files = Directory.GetFiles(args[1]);
Parallel.For(0, files.Length,
            (index) => {
                FileInfo fi = new FileInfo(files[index]);
                long size = fi.Length;
                Interlocked.Add(ref totalSize, size);
            });
Console.WriteLine("Directory '{0}':", args[1]);
Console.WriteLine("{0:N0} files, {1:N0} bytes", files.Length, totalSize);
}
}
}

```

### 7.3.3.2. Esempio 2 di Parallel.For:

<https://docs.microsoft.com/it-it/dotnet/standard/parallel-programming/how-to-write-a-simple-parallel-for-loop>

```

using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Threading.Tasks;

namespace ParallelMatricesMultiply
{
    class Program
    {
        #region Sequential_Loop
        static void MultiplyMatricesSequential(double[,] matA, double[,] matB,
                                                double[,] result)
        {
            int matACols = matA.GetLength(1);
            int matBCols = matB.GetLength(1);
            int matARows = matA.GetLength(0);

            for (int i = 0; i < matARows; i++)
            {
                for (int j = 0; j < matBCols; j++)
                {
                    double temp = 0;
                    for (int k = 0; k < matACols; k++)
                    {
                        temp += matA[i, k] * matB[k, j];
                    }
                    result[i, j] += temp;
                }
            }
        }
        #endregion

        #region Parallel_Loop

```

```

        static void MultiplyMatricesParallel(double[,] matA, double[,] matB, double[,] result)
    {
        int matACols = matA.GetLength(1);
        int matBCols = matB.GetLength(1);
        int matARows = matA.GetLength(0);

        // A basic matrix multiplication.
        // Parallelize the outer loop to partition the source array by rows.
        Parallel.For(0, matARows, i =>
    {
        for (int j = 0; j < matBCols; j++)
    {
        double temp = 0;
        for (int k = 0; k < matACols; k++)
        {
            temp += matA[i, k] * matB[k, j];
        }
        result[i, j] = temp;
    }
}); // Parallel.For
}
#endregion

#region Main
static void Main(string[] args)
{
    // Set up matrices. Use small values to better view
    // result matrix. Increase the counts to see greater
    // speedup in the parallel loop vs. the sequential loop.
    int colCount = 1800;
    int rowCount = 2000;
    int colCount2 = 270;
    double[,] m1 = InitializeMatrix(rowCount, colCount);
    double[,] m2 = InitializeMatrix(colCount, colCount2);
    double[,] result = new double[rowCount, colCount2];

    // First do the sequential version.
    Console.Error.WriteLine("Executing sequential loop...");
    Stopwatch stopwatch = new Stopwatch();
    stopwatch.Start();

    MultiplyMatricesSequential(m1, m2, result);
    stopwatch.Stop();
    long sequentialTime = stopwatch.ElapsedMilliseconds;
    Console.Error.WriteLine("Sequential loop time in milliseconds: {0}",
                           sequentialTime);

    // For the skeptics.
    //OfferToPrint(rowCount, colCount2, result);

    // Reset timer and results matrix.
    stopwatch.Reset();
    result = new double[rowCount, colCount2];

    // Do the parallel loop.
    Console.Error.WriteLine("Executing parallel loop...");
    stopwatch.Start();
    MultiplyMatricesParallel(m1, m2, result);
    stopwatch.Stop();
    long parallelTime = stopwatch.ElapsedMilliseconds;
    Console.Error.WriteLine("Parallel loop time in milliseconds: {0}",
                           parallelTime);
}

```

```

        //OfferToPrint(rowCount, colCount2, result);
        //calculate speedup factor
        double speedup = (double)sequentialTime / parallelTime;
        Console.WriteLine($"Speedup = {speedup:F2}; il calcolo parallelo è {speedup:F2}
volte più veloce di quello sequenziale");
        // Keep the console window open in debug mode.
        Console.Error.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
#endregion

#region Helper_Methods
static double[,] InitializeMatrix(int rows, int cols)
{
    double[,] matrix = new double[rows, cols];

    Random r = new Random();
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            matrix[i, j] = r.Next(100);
        }
    }
    return matrix;
}

private static void OfferToPrint(int rowCount, int colCount, double[,] matrix)
{
    Console.Error.Write("Computation complete. Print results (y/n)? ");
    char c = Console.ReadKey(true).KeyChar;
    Console.Error.WriteLine(c);
    if (Char.ToUpperInvariant(c) == 'Y')
    {
        if (!Console.IsOutputRedirected &&
RuntimeInformation.IsOSPlatform(OSPlatform.Windows)) Console.WindowWidth = 180;
        Console.WriteLine();
        for (int x = 0; x < rowCount; x++)
        {
            Console.WriteLine("ROW {0}: ", x);
            for (int y = 0; y < colCount; y++)
            {
                Console.Write("{0:#.##} ", matrix[x, y]);
            }
            Console.WriteLine();
        }
    }
}
#endregion
}

```

### 7.3.3.3. Esempio 3 di Parallel.For

<https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallel.for>

Questo esempio mostra l'utilizzo di un parametro di stato (di tipo ParallelLoopState <https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallelloopstate>) che permette a ciascuna iterazione di un parallel for di interagire con le altre iterazioni.

In particolare è possibile utilizzare:

La property ShouldExitCurrentIteration

<https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallelloopstate.shouldexitcurrentiteration>

per sapere se l'iterazione corrente deve terminare prima del previsto.

La property **LowestBreakIteration**

<https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallelloopstate.lowestbreakiteration>

per conoscere l'indice più piccolo rispetto al quale si è verificata la condizione di break

Il metodo **Break**

<https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallelloopstate.break>

per indicare che il parallel for dovrebbe terminare l'esecuzione delle iterazioni con indice superiore a quello corrente. Nota: il break è invocato da un thread che esegue il ciclo i-mo e permette di terminare tutti i thread che stanno eseguendo il parallel for con indici superiori a quello corrente.

Il metodo **Stop**

<https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallelloopstate.stop>

per indicare che il Parallel.For dovrebbe terminare il prima possibile l'esecuzione di tutti i thread

Si noti che il Parallel.For restituisce una struct ParallelLoopResult (<https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallelloopresult>) che contiene informazioni sull'esecuzione in parallelo: IsCompleted e LowestBreakIteration:

If **IsCompleted** returns true, then the loop ran to completion, such that all iterations of the loop were executed. If **IsCompleted** returns false and **LowestBreakIteration** returns null, a call to **Stop** was used to end the loop prematurely. If **IsCompleted** returns false and **LowestBreakIteration** returns a non-null integral value, **Break** was used to end the loop prematurely.

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace ParallelLoopResultDemo1
{
    class Program
    {
        //la Console è condivisa da più thread, che in alcune situazioni, alterano il colore
        //del testo.
        //il lock è necessario solo perché si vuole che il testo in alcune situazioni
        //specifiche sia di un certo colore,
        //ad esempio di colore rosso quando il lavoro non è portato a termine oppure di
        //colore verde quando è portato a termine
        //Per evitare che un thread scriva con un colore impostato da un altro thread si
        //acquisisce il lock sulla console per
        //poter scrivere
        private static readonly object _consoleColorLock = new object();
        public static void Main()
        {
            var rnd = new Random(50);
            int breakIndex = rnd.Next(1, 11); //se si prova un valore del break index
            superiore a 100 tutto il lavoro sarà completato, ovviamente
        }
}
```

```

        Console.WriteLine($"I am the Main Thread with Thread Id =
{Thread.CurrentThread.ManagedThreadId} and I will call Break at iteration {breakIndex}\n");

        var result = Parallel.For(1, 101, (i, state) =>
    {
        lock (_consoleColorLock)
            Console.WriteLine($"Beginning iteration {i}");
        if (i == breakIndex)
        {
            state.Break();
            lock (_consoleColorLock)
                Console.WriteLine($"Break in iteration {i}");

        }
        //simulazione della prima parte del lavoro
        int delay;
        lock (rnd)
            delay = rnd.Next(1, 1001);
        Thread.Sleep(delay);
        lock (_consoleColorLock)
            Console.WriteLine($"Indice {i} su Thread Id
={Thread.CurrentThread.ManagedThreadId}; eseguita la prima parte del lavoro");

        //controllo se l'iterazione i-ma deve fermarsi o proseguire
        if (state.ShouldExitCurrentIteration)
        {
            if (state.LowestBreakIteration < i)
                lock (_consoleColorLock)
                {
                    ConsoleColor previousForegroundColor =
Console.ForegroundColor;
                    ConsoleColor foregroundColor = ConsoleColor.Red;
                    Console.WriteLine($"Indice {i} su Thread Id
={Thread.CurrentThread.ManagedThreadId}; NON ho completato il lavoro");
                    Console.ForegroundColor = previousForegroundColor;
                }
            //il thread corrente deve terminare perché il suo indice è superiore a
            quello in cui si è verificato il Break
            return;
        }
        //se sono qui devo continuare la seconda parte del lavoro
        //simulazione della seconda parte del lavoro
        lock (rnd)
            delay = rnd.Next(1, 1001);
        Thread.Sleep(delay);
        lock (_consoleColorLock)
        {
            ConsoleColor previousForegroundColor = Console.ForegroundColor;
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine($"Indice {i} su Thread Id
={Thread.CurrentThread.ManagedThreadId}; eseguito il lavoro fino alla fine");
            Console.ForegroundColor = previousForegroundColor;
        }

    });
    //ritorno al Main
    if (!result.IsCompleted)
    {
        if (result.LowestBreakIteration.HasValue)
            Console.WriteLine($"\\nLowest Break Iteration:
{result.LowestBreakIteration}");
        else
            Console.WriteLine($"\\nNo lowest break iteration.");
    }
}

```

```

        else
        {
            Console.WriteLine($"\\nThe job was completed.");
        }

    }

}

//Output:
I am the Main Thread with Thread Id = 1 and I will call Break at iteration 9

Beginning iteration 1
Beginning iteration 7
Beginning iteration 13
Beginning iteration 19
Beginning iteration 25
Beginning iteration 31
Beginning iteration 37
Beginning iteration 43
Beginning iteration 49
Beginning iteration 55
Beginning iteration 61
Beginning iteration 67
Beginning iteration 73
Beginning iteration 79
Beginning iteration 85
Beginning iteration 91
Beginning iteration 97
Indice 67 su Thread Id =11; eseguita la prima parte del lavoro
Indice 67 su Thread Id =11; eseguito il lavoro fino alla fine
Beginning iteration 68
Indice 55 su Thread Id =14; eseguita la prima parte del lavoro
Indice 43 su Thread Id =9; eseguita la prima parte del lavoro
Indice 13 su Thread Id =5; eseguita la prima parte del lavoro
Indice 25 su Thread Id =7; eseguita la prima parte del lavoro
Indice 43 su Thread Id =9; eseguito il lavoro fino alla fine
Beginning iteration 44
Indice 85 su Thread Id =18; eseguita la prima parte del lavoro
Indice 31 su Thread Id =8; eseguita la prima parte del lavoro
Indice 1 su Thread Id =1; eseguita la prima parte del lavoro
Indice 73 su Thread Id =15; eseguita la prima parte del lavoro
Indice 61 su Thread Id =13; eseguita la prima parte del lavoro
Indice 37 su Thread Id =10; eseguita la prima parte del lavoro

```

Indice 55 su Thread Id =14; eseguito il lavoro fino alla fine  
Beginning iteration 56  
Indice 13 su Thread Id =5; eseguito il lavoro fino alla fine  
Beginning iteration 14  
Indice 7 su Thread Id =4; eseguita la prima parte del lavoro  
Indice 49 su Thread Id =12; eseguita la prima parte del lavoro  
Indice 25 su Thread Id =7; eseguito il lavoro fino alla fine  
Beginning iteration 26  
Indice 68 su Thread Id =11; eseguita la prima parte del lavoro  
Indice 19 su Thread Id =6; eseguita la prima parte del lavoro  
Indice 49 su Thread Id =12; eseguito il lavoro fino alla fine  
Beginning iteration 50  
Indice 61 su Thread Id =13; eseguito il lavoro fino alla fine  
Indice 73 su Thread Id =15; eseguito il lavoro fino alla fine  
Beginning iteration 62  
Beginning iteration 74  
Indice 44 su Thread Id =9; eseguita la prima parte del lavoro  
Indice 91 su Thread Id =19; eseguita la prima parte del lavoro  
Indice 7 su Thread Id =4; eseguito il lavoro fino alla fine  
Beginning iteration 8  
Indice 85 su Thread Id =18; eseguito il lavoro fino alla fine  
Beginning iteration 86  
Indice 79 su Thread Id =16; eseguita la prima parte del lavoro  
Indice 97 su Thread Id =17; eseguita la prima parte del lavoro  
Indice 14 su Thread Id =5; eseguita la prima parte del lavoro  
Indice 1 su Thread Id =1; eseguito il lavoro fino alla fine  
Beginning iteration 2  
Indice 2 su Thread Id =1; eseguita la prima parte del lavoro  
Indice 37 su Thread Id =10; eseguito il lavoro fino alla fine  
Beginning iteration 38  
Indice 68 su Thread Id =11; eseguito il lavoro fino alla fine  
Beginning iteration 69  
Indice 31 su Thread Id =8; eseguito il lavoro fino alla fine  
Beginning iteration 32  
Indice 19 su Thread Id =6; eseguito il lavoro fino alla fine  
Indice 56 su Thread Id =14; eseguita la prima parte del lavoro  
Indice 62 su Thread Id =13; eseguita la prima parte del lavoro  
Beginning iteration 20  
Indice 86 su Thread Id =18; eseguita la prima parte del lavoro  
Indice 86 su Thread Id =18; eseguito il lavoro fino alla fine  
Beginning iteration 87  
Indice 14 su Thread Id =5; eseguito il lavoro fino alla fine

Beginning iteration 15  
Indice 97 su Thread Id =17; eseguito il lavoro fino alla fine  
Beginning iteration 98  
Indice 38 su Thread Id =10; eseguita la prima parte del lavoro  
Indice 62 su Thread Id =13; eseguito il lavoro fino alla fine  
Beginning iteration 63  
Indice 69 su Thread Id =11; eseguita la prima parte del lavoro  
Indice 44 su Thread Id =9; eseguito il lavoro fino alla fine  
Beginning iteration 45  
Indice 98 su Thread Id =17; eseguita la prima parte del lavoro  
Indice 87 su Thread Id =18; eseguita la prima parte del lavoro  
Indice 98 su Thread Id =17; eseguito il lavoro fino alla fine  
Beginning iteration 99  
Indice 69 su Thread Id =11; eseguito il lavoro fino alla fine  
Indice 91 su Thread Id =19; eseguito il lavoro fino alla fine  
Beginning iteration 92  
Beginning iteration 70  
Indice 99 su Thread Id =17; eseguita la prima parte del lavoro  
Indice 26 su Thread Id =7; eseguita la prima parte del lavoro  
Indice 50 su Thread Id =12; eseguita la prima parte del lavoro  
Indice 99 su Thread Id =17; eseguito il lavoro fino alla fine  
Beginning iteration 100  
Indice 38 su Thread Id =10; eseguito il lavoro fino alla fine  
Beginning iteration 39  
Indice 74 su Thread Id =15; eseguita la prima parte del lavoro  
Indice 79 su Thread Id =16; eseguito il lavoro fino alla fine  
Beginning iteration 80  
Indice 100 su Thread Id =17; eseguita la prima parte del lavoro  
Indice 8 su Thread Id =4; eseguita la prima parte del lavoro  
Indice 56 su Thread Id =14; eseguito il lavoro fino alla fine  
Beginning iteration 57  
Indice 15 su Thread Id =5; eseguita la prima parte del lavoro  
Indice 20 su Thread Id =6; eseguita la prima parte del lavoro  
Indice 2 su Thread Id =1; eseguito il lavoro fino alla fine  
Beginning iteration 3  
Indice 70 su Thread Id =11; eseguita la prima parte del lavoro  
Indice 87 su Thread Id =18; eseguito il lavoro fino alla fine  
Beginning iteration 88  
Indice 63 su Thread Id =13; eseguita la prima parte del lavoro  
Indice 50 su Thread Id =12; eseguito il lavoro fino alla fine  
Beginning iteration 51  
Indice 57 su Thread Id =14; eseguita la prima parte del lavoro

Indice 32 su Thread Id =8; eseguita la prima parte del lavoro  
Indice 63 su Thread Id =13; eseguito il lavoro fino alla fine  
Beginning iteration 64  
Indice 100 su Thread Id =17; eseguito il lavoro fino alla fine  
Beginning iteration 4  
Indice 64 su Thread Id =13; eseguita la prima parte del lavoro  
Indice 45 su Thread Id =9; eseguita la prima parte del lavoro  
Indice 20 su Thread Id =6; eseguito il lavoro fino alla fine  
Beginning iteration 21  
Indice 8 su Thread Id =4; eseguito il lavoro fino alla fine  
Beginning iteration 9  
Break in iteration 9  
Indice 92 su Thread Id =19; eseguita la prima parte del lavoro  
**Indice 92 su Thread Id =19; NON ho completato il lavoro**  
Indice 39 su Thread Id =10; eseguita la prima parte del lavoro  
**Indice 39 su Thread Id =10; NON ho completato il lavoro**  
Indice 26 su Thread Id =7; eseguito il lavoro fino alla fine  
Indice 45 su Thread Id =9; eseguito il lavoro fino alla fine  
Indice 51 su Thread Id =12; eseguita la prima parte del lavoro  
**Indice 51 su Thread Id =12; NON ho completato il lavoro**  
Indice 74 su Thread Id =15; eseguito il lavoro fino alla fine  
Indice 88 su Thread Id =18; eseguita la prima parte del lavoro  
**Indice 88 su Thread Id =18; NON ho completato il lavoro**  
Indice 4 su Thread Id =17; eseguita la prima parte del lavoro  
Indice 4 su Thread Id =17; eseguito il lavoro fino alla fine  
Beginning iteration 5  
Indice 70 su Thread Id =11; eseguito il lavoro fino alla fine  
Indice 3 su Thread Id =1; eseguita la prima parte del lavoro  
Indice 80 su Thread Id =16; eseguita la prima parte del lavoro  
**Indice 80 su Thread Id =16; NON ho completato il lavoro**  
Indice 15 su Thread Id =5; eseguito il lavoro fino alla fine  
Indice 57 su Thread Id =14; eseguito il lavoro fino alla fine  
Indice 32 su Thread Id =8; eseguito il lavoro fino alla fine  
Indice 5 su Thread Id =17; eseguita la prima parte del lavoro  
Indice 3 su Thread Id =1; eseguito il lavoro fino alla fine  
Indice 21 su Thread Id =6; eseguita la prima parte del lavoro  
**Indice 21 su Thread Id =6; NON ho completato il lavoro**  
Indice 64 su Thread Id =13; eseguito il lavoro fino alla fine  
Indice 9 su Thread Id =4; eseguita la prima parte del lavoro  
Indice 5 su Thread Id =17; eseguito il lavoro fino alla fine  
Beginning iteration 6  
Indice 9 su Thread Id =4; eseguito il lavoro fino alla fine

Indice 6 su Thread Id =17; eseguita la prima parte del lavoro

Indice 6 su Thread Id =17; eseguito il lavoro fino alla fine

Lowest Break Iteration: 9

Come si vede dagli indici delle iterazioni che non sono state portate a termine, nel caso di utilizzo del metodo Break() NON è possibile che vengano interrotte iterazioni con indice minore rispetto a quella nella quale avviene la segnalazione di Break: saranno interrotte solo le iterazioni con indice maggiore a quello in cui è stato segnalato il Break.

Si noti anche che, nonostante sia stato utilizzato un seed per il generatore di numeri casuali, il risultato della simulazione non è sempre lo stesso perché i thread che vengono eseguiti di volta in volta sono soggetti all'algoritmo dello schedulatore del runtime.

#### 7.3.3.4. Esempio 4 di Parallel.For

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace ParallelLoopResultDemo2
{
    class Program
    {
        //la Console è condivisa da più thread, che in alcune situazioni, alterano il colore
        //del testo.
        //il lock è necessario solo perché si vuole che il testo in alcune situazioni
        //specifiche sia di un certo colore,
        //ad esempio di colore rosso quando il lavoro non è portato a termine oppure di
        //colore verde quando è portato a termine
        //Per evitare che un thread scriva con un colore impostato da un altro thread si
        //acquisisce il lock sulla console per
        //poter scrivere
        private static readonly object _consoleColorLock = new object();
        public static void Main()
        {
            var rnd = new Random(50);
            int breakIndex = rnd.Next(1, 11); //se si prova un valore del break index
            superiore a 100 tutto il lavoro sarà completato, ovviamente
            Console.WriteLine($"I am the Main Thread with Thread Id =
            {Thread.CurrentThread.ManagedThreadId} and I will call Stop at iteration {breakIndex}\n");

            var result = Parallel.For(1, 101, (i, state) =>
            {
                lock (_consoleColorLock)
                    Console.WriteLine($"Beginning iteration {i}");
                if (i == breakIndex)
                {
                    state.Stop();
                    lock (_consoleColorLock)
                        Console.WriteLine($"Stop in iteration {i}");

                }
                //simulazione della prima parte del lavoro
                int delay;
                lock (rnd)
                    delay = rnd.Next(1, 1001);
                Thread.Sleep(delay);
                lock (_consoleColorLock)
```

```

        Console.WriteLine($"Indice {i} su Thread Id
={Thread.CurrentThread.ManagedThreadId}; eseguita la prima parte del lavoro");

        //controllo se l'iterazione i-ma deve fermarsi o proseguire
        if (state.ShouldExitCurrentIteration)
        {
            if (state.IsStopped)
                lock (_consoleColorLock)
            {
                ConsoleColor previousForegroundColor =
Console.ForegroundColor;
                ConsoleColor = ConsoleColor.Red;
                Console.WriteLine($"Indice {i} su Thread Id
={Thread.CurrentThread.ManagedThreadId}; NON ho completato il lavoro");
                ConsoleColor = previousForegroundColor;
            }
            //il thread corrente deve terminare perché è stato segnalato uno Stop
            return;
        }
        //se sono qui devo continuare la seconda parte del lavoro
        //simulazione della seconda parte del lavoro
        lock (rnd)
            delay = rnd.Next(1, 1001);
        Thread.Sleep(delay);
        lock (_consoleColorLock)
        {
            ConsoleColor previousForegroundColor = Console.ForegroundColor;
            ConsoleColor = ConsoleColor.Green;
            Console.WriteLine($"Indice {i} su Thread Id
={Thread.CurrentThread.ManagedThreadId}; eseguito il lavoro fino alla fine");
            ConsoleColor = previousForegroundColor;
        }

    });
    //ritorno al Main
    if (!result.IsCompleted)
    {
        if (result.LowestBreakIteration.HasValue)
            Console.WriteLine($"\\nLowest Break Iteration:
{result.LowestBreakIteration}");
        else
            Console.WriteLine($"\\nNo lowest break iteration.");
    }
    else
    {
        Console.WriteLine($"\\nThe job was completed.");
    }
}
}

//Output:
I am the Main Thread with Thread Id = 1 and I will call Break at iteration 9

```

Beginning iteration 1  
Beginning iteration 7  
Beginning iteration 13  
Beginning iteration 19  
Beginning iteration 25  
Beginning iteration 31  
Beginning iteration 37  
Beginning iteration 43  
Beginning iteration 49

Beginning iteration 67  
Beginning iteration 55  
Beginning iteration 73  
Beginning iteration 61  
Beginning iteration 79  
Beginning iteration 85  
Beginning iteration 91  
Beginning iteration 97  
Indice 73 su Thread Id =14; eseguita la prima parte del lavoro  
Indice 73 su Thread Id =14; eseguito il lavoro fino alla fine  
Beginning iteration 74  
Indice 67 su Thread Id =12; eseguita la prima parte del lavoro  
Indice 43 su Thread Id =8; eseguita la prima parte del lavoro  
Indice 13 su Thread Id =5; eseguita la prima parte del lavoro  
Indice 25 su Thread Id =7; eseguita la prima parte del lavoro  
Indice 85 su Thread Id =18; eseguita la prima parte del lavoro  
Indice 31 su Thread Id =9; eseguita la prima parte del lavoro  
Indice 43 su Thread Id =8; eseguito il lavoro fino alla fine  
Beginning iteration 44  
Indice 1 su Thread Id =1; eseguita la prima parte del lavoro  
Indice 61 su Thread Id =15; eseguita la prima parte del lavoro  
Indice 55 su Thread Id =13; eseguita la prima parte del lavoro  
Indice 37 su Thread Id =11; eseguita la prima parte del lavoro  
Indice 67 su Thread Id =12; eseguito il lavoro fino alla fine  
Beginning iteration 68  
Indice 13 su Thread Id =5; eseguito il lavoro fino alla fine  
Beginning iteration 14  
Indice 7 su Thread Id =4; eseguita la prima parte del lavoro  
Indice 74 su Thread Id =14; eseguita la prima parte del lavoro  
Indice 49 su Thread Id =10; eseguita la prima parte del lavoro  
Indice 25 su Thread Id =7; eseguito il lavoro fino alla fine  
Beginning iteration 26  
Indice 19 su Thread Id =6; eseguita la prima parte del lavoro  
Indice 74 su Thread Id =14; eseguito il lavoro fino alla fine  
Beginning iteration 75  
Indice 61 su Thread Id =15; eseguito il lavoro fino alla fine  
Beginning iteration 62  
Indice 85 su Thread Id =18; eseguito il lavoro fino alla fine  
Indice 55 su Thread Id =13; eseguito il lavoro fino alla fine  
Beginning iteration 86  
Beginning iteration 56  
Indice 91 su Thread Id =17; eseguita la prima parte del lavoro  
Indice 7 su Thread Id =4; eseguito il lavoro fino alla fine  
Beginning iteration 8  
Indice 31 su Thread Id =9; eseguito il lavoro fino alla fine  
Indice 79 su Thread Id =16; eseguita la prima parte del lavoro  
Beginning iteration 32  
Indice 97 su Thread Id =19; eseguita la prima parte del lavoro  
Indice 14 su Thread Id =5; eseguita la prima parte del lavoro  
Indice 1 su Thread Id =1; eseguito il lavoro fino alla fine  
Beginning iteration 2  
Indice 2 su Thread Id =1; eseguita la prima parte del lavoro  
Indice 37 su Thread Id =11; eseguito il lavoro fino alla fine  
Beginning iteration 38  
Indice 62 su Thread Id =15; eseguita la prima parte del lavoro  
Indice 26 su Thread Id =7; eseguita la prima parte del lavoro  
Indice 19 su Thread Id =6; eseguito il lavoro fino alla fine  
Indice 44 su Thread Id =8; eseguita la prima parte del lavoro  
Indice 68 su Thread Id =12; eseguita la prima parte del lavoro  
Beginning iteration 20  
Indice 79 su Thread Id =16; eseguito il lavoro fino alla fine  
Beginning iteration 80  
Indice 80 su Thread Id =16; eseguita la prima parte del lavoro  
Indice 14 su Thread Id =5; eseguito il lavoro fino alla fine

```
Beginning iteration 15
Indice 97 su Thread Id =19; eseguito il lavoro fino alla fine
Beginning iteration 98
Indice 38 su Thread Id =11; eseguita la prima parte del lavoro
Indice 68 su Thread Id =12; eseguito il lavoro fino alla fine
Beginning iteration 69
Indice 62 su Thread Id =15; eseguito il lavoro fino alla fine
Beginning iteration 63
Indice 98 su Thread Id =19; eseguita la prima parte del lavoro
Indice 56 su Thread Id =13; eseguita la prima parte del lavoro
Indice 80 su Thread Id =16; eseguito il lavoro fino alla fine
Beginning iteration 81
Indice 56 su Thread Id =13; eseguito il lavoro fino alla fine
Beginning iteration 57
Indice 91 su Thread Id =17; eseguito il lavoro fino alla fine
Beginning iteration 92
Indice 63 su Thread Id =15; eseguita la prima parte del lavoro
Indice 49 su Thread Id =10; eseguito il lavoro fino alla fine
Beginning iteration 50
Indice 57 su Thread Id =13; eseguita la prima parte del lavoro
Indice 50 su Thread Id =10; eseguita la prima parte del lavoro
Indice 75 su Thread Id =14; eseguita la prima parte del lavoro
Indice 38 su Thread Id =11; eseguito il lavoro fino alla fine
Beginning iteration 39
Indice 86 su Thread Id =18; eseguita la prima parte del lavoro
Indice 32 su Thread Id =9; eseguita la prima parte del lavoro
Indice 75 su Thread Id =14; eseguito il lavoro fino alla fine
Beginning iteration 76
Indice 8 su Thread Id =4; eseguita la prima parte del lavoro
Indice 44 su Thread Id =8; eseguito il lavoro fino alla fine
Beginning iteration 45
Indice 15 su Thread Id =5; eseguita la prima parte del lavoro
Indice 20 su Thread Id =6; eseguita la prima parte del lavoro
Indice 2 su Thread Id =1; eseguito il lavoro fino alla fine
Beginning iteration 3
Indice 63 su Thread Id =15; eseguito il lavoro fino alla fine
Beginning iteration 64
Indice 81 su Thread Id =16; eseguita la prima parte del lavoro
Indice 69 su Thread Id =12; eseguita la prima parte del lavoro
Indice 45 su Thread Id =8; eseguita la prima parte del lavoro
Indice 26 su Thread Id =7; eseguito il lavoro fino alla fine
Beginning iteration 27
Indice 50 su Thread Id =10; eseguito il lavoro fino alla fine
Beginning iteration 51
Indice 69 su Thread Id =12; eseguito il lavoro fino alla fine
Indice 76 su Thread Id =14; eseguita la prima parte del lavoro
Beginning iteration 70
Indice 98 su Thread Id =19; eseguito il lavoro fino alla fine
Beginning iteration 99
Indice 76 su Thread Id =14; eseguito il lavoro fino alla fine
Beginning iteration 77
Indice 20 su Thread Id =6; eseguito il lavoro fino alla fine
Beginning iteration 21
Indice 8 su Thread Id =4; eseguito il lavoro fino alla fine
Beginning iteration 9
Stop in iteration 9
Indice 92 su Thread Id =17; eseguita la prima parte del lavoro
Indice 92 su Thread Id =17; NON ho completato il lavoro
Indice 39 su Thread Id =11; eseguita la prima parte del lavoro
Indice 39 su Thread Id =11; NON ho completato il lavoro
Indice 57 su Thread Id =13; eseguito il lavoro fino alla fine
Indice 77 su Thread Id =14; eseguita la prima parte del lavoro
Indice 77 su Thread Id =14; NON ho completato il lavoro
Indice 45 su Thread Id =8; eseguito il lavoro fino alla fine
```

```

Indice 86 su Thread Id =18; eseguito il lavoro fino alla fine
Indice 81 su Thread Id =16; eseguito il lavoro fino alla fine
Indice 70 su Thread Id =12; eseguita la prima parte del lavoro
Indice 70 su Thread Id =12; NON ho completato il lavoro
Indice 3 su Thread Id =1; eseguita la prima parte del lavoro
Indice 3 su Thread Id =1; NON ho completato il lavoro
Indice 64 su Thread Id =15; eseguita la prima parte del lavoro
Indice 64 su Thread Id =15; NON ho completato il lavoro
Indice 32 su Thread Id =9; eseguito il lavoro fino alla fine
Indice 15 su Thread Id =5; eseguito il lavoro fino alla fine
Indice 27 su Thread Id =7; eseguita la prima parte del lavoro
Indice 27 su Thread Id =7; NON ho completato il lavoro
Indice 51 su Thread Id =10; eseguita la prima parte del lavoro
Indice 51 su Thread Id =10; NON ho completato il lavoro
Indice 21 su Thread Id =6; eseguita la prima parte del lavoro
Indice 21 su Thread Id =6; NON ho completato il lavoro
Indice 99 su Thread Id =19; eseguita la prima parte del lavoro
Indice 99 su Thread Id =19; NON ho completato il lavoro
Indice 9 su Thread Id =4; eseguita la prima parte del lavoro
Indice 9 su Thread Id =4; NON ho completato il lavoro

```

No lowest break iteration.

Come si vede dagli indici delle iterazioni che non sono state portate a termine, nel caso di utilizzo del metodo Stop() è possibile che vengano interrotte anche iterazioni con indice minore rispetto a quella nella quale avviene la segnalazione di Stop.

Si noti anche che, nonostante sia stato utilizzato un seed per il generatore di numeri casuali, il risultato della simulazione non è sempre lo stesso perché i thread che vengono eseguiti di volta in volta sono soggetti all'algoritmo dello scheduler del runtime.

### 7.3.3.5. Esempio 1 Parallel.ForEach

<https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/how-to-write-a-simple-parallel-foreach-loop>

Ricerca dei numeri primi con il calcolo parallelo e confronto con il calcolo sequenziale:

```

using System.Collections.Concurrent;
using System.Diagnostics;

class Program
{
    //https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/how-to-write-a-simple-parallel-foreach-loop
    static void Main()
    {
        // 2 million
        var limit = 2_000_000;
        var numbers = Enumerable.Range(0, limit).ToList(); //creo una lista con i valori da 0 a limit

        var watch = Stopwatch.StartNew();
        var primeNumbersFromForeach = GetPrimeList(numbers); //restituisco la lista con i numeri primi
        watch.Stop();

        var watchForParallel = Stopwatch.StartNew();
        var primeNumbersFromParallelForeach = GetPrimeListWithParallel(numbers); //restituisco la lista
        con i numeri primi con il calcolo parallelo
        watchForParallel.Stop();

        Console.WriteLine($"Classical foreach loop | Total prime numbers : {primeNumbersFromForeach.Count} | Time Taken : {watch.ElapsedMilliseconds} ms.");
        Console.WriteLine($"Parallel.ForEach loop | Total prime numbers : {primeNumbersFromParallelForeach.Count} | Time Taken : {watchForParallel.ElapsedMilliseconds} ms.");
        //calcoliamo il fattore di speedup
        double speedup = (double)(watch.ElapsedMilliseconds)/watchForParallel.ElapsedMilliseconds;
        Console.WriteLine($"Il fattore di speedup è {speedup:F2}. Dunque Il calcolo parallelo è
{speedup:F2} volte più veloce di quello sequenziale");
    }
}

```

```

}

/// <summary>
/// GetPrimeList returns Prime numbers by using sequential ForEach
/// </summary>
/// <param name="inputs"></param>
/// <returns></returns>
private static IList<int> GetPrimeList(IList<int> numbers) =>
numbers.Where(IsPrime).ToList(); //filtro i numeri contenuti nella lista in base al metodo/predicato
IsPrime

/// <summary>
/// GetPrimeListWithParallel returns Prime numbers by using Parallel.ForEach
/// </summary>
/// <param name="numbers"></param>
/// <returns></returns>
private static IList<int> GetPrimeListWithParallel(IList<int> numbers)
{
    //https://learn.microsoft.com/en-us/dotnet/api/system.collections.concurrent.concurrentbag-1?view=net-7.0
    //Represents a thread-safe, unordered collection of objects.
    var primeNumbers = new ConcurrentBag<int>();

    Parallel.ForEach(numbers, number =>
    {
        if (IsPrime(number))
        {
            primeNumbers.Add(number);
        }
    });
}

return primeNumbers.ToList();
}

/// <summary>
/// IsPrime returns true if number is Prime, else
false.(https://en.wikipedia.org/wiki/Prime_number)
/// </summary>
/// <param name="number"></param>
/// <returns></returns>
private static bool IsPrime(int number)
{
    if (number < 2)
    {
        return false;
    }

    for (var divisor = 2; divisor <= Math.Sqrt(number); divisor++)
    {
        if (number % divisor == 0)
        {
            return false;
        }
    }
    return true;
}
}

```

### 7.3.3.6. Esempio 5 di Parallel.For (con Thread-local data)

<https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/how-to-write-a-parallel-for-loop-with-thread-local-variables>

Questo esempio illustra come usare le variabili locali di thread per archiviare e recuperare lo stato in ogni attività separata creata da un ciclo `For`. L'uso dei dati locali di thread permette di evitare il sovraccarico dovuto alla sincronizzazione di un numero elevato di accessi a uno stato condiviso. Invece di scrivere in una risorsa condivisa a ogni iterazione, si calcola e si archivia il valore fino al completamento di tutte le iterazioni per l'attività. È quindi possibile scrivere il risultato finale una volta sola nella risorsa condivisa oppure passarlo a un altro metodo.

```

namespace ParallelForWithThreadLocalVariables
{
    internal class Program

```

```

{
    static void Main(string[] args)
    {
        const int Numbers = 1000000;
        int[] nums = Enumerable.Range(1, Numbers).ToArray(); //creo un array con i valori da 1 a N
        long total = 0;
        //https://learn.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallel.for?view=net-7.0
        //https://learn.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallel.for?f1url=%3FappId%3DDev16IDEF1%26l%3DEN-US%26k%3Dk(System.Threading.Tasks.Parallel.For%2560%25601)%3Bk(DevLang-csharp)%26rd%3Dtrue&view=net-7.0#system-threading-tasks-parallel-for-1(system-int32-system-int32-system-threading-tasks-paralleloptions-system-func((-0))-system-func((system-int32-system-threading-tasks-paralleloopstate-0-0))-system-action((-0)))
        // Use type parameter to make subtotal a long, not an int
        //()=>0 inizializza la variabile locale al Task, subtotal in questo caso. La variabile
        subtotal viene inizializzata a 0
        Parallel.For<long>(0, nums.Length, () => 0, (j, loopState, subtotal) =>
        {
            subtotal += nums[j]; //modify local variable
            return subtotal; // value to be passed to next iteration
        },
        //la variabile x è il risultato dell'elaborazione del Task. Corrisponde a subtotal quando
        il task finisce la sua elaborazione
        //Questa è la action che viene eseguita alla fine dell'elaborazione del Task
        (x) => Interlocked.Add(ref total, x)
    );

    Console.WriteLine("The total is {0:N0}", total);
    //confrontiamo il risultato con la formula di Gauss
    long sommaGauss = (long)Numbers * (Numbers + 1) / 2;
    Console.WriteLine("Il totale con la formula di Gauss {0:N0}", sommaGauss);
}
}

```

The body delegate is invoked once for each value in the iteration range (fromInclusive, toExclusive). It is provided with the following parameters: the iteration count ([Int32](#)), a [ParallelLoopState](#) instance that may be used to break out of the loop prematurely, and some local state that may be shared amongst iterations that execute on the same thread.

The localInit delegate is invoked once for each task that participates in the loop's execution and returns the initial local state for each of those tasks. These initial states are passed to the first body invocations on each task. Then, every subsequent body invocation returns a possibly modified state value that is passed to the next body invocation. Finally, the last body invocation on each task returns a state value that is passed to the localFinally delegate. The localFinally delegate is invoked once per task to perform a final action on each task's local state. This delegate might be invoked concurrently on multiple tasks; therefore, you must synchronize access to any shared variables.

The [Parallel.For](#) method may use more tasks than threads over the lifetime of its execution, as existing tasks complete and are replaced by new tasks. This gives the underlying [TaskScheduler](#) object the chance to add, change, or remove threads that service the loop.

If fromInclusive is greater than or equal to toExclusive, then the method returns immediately without performing any iterations.

For an example that uses this method, see [How to: Write a Parallel.ForEach with Thread-Local Variables](#).

### 7.3.3.7. Esempio 2 di [Parallel.ForEach](#) (con Partition-local data)

<https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/how-to-write-a-parallel-foreach-loop-with-partition-local-variables>

L'esempio seguente illustra come scrivere un metodo [ForEach](#) che usa variabili partition-local.

Quando viene eseguito un ciclo [ForEach](#), la relativa raccolta di origine viene divisa in più partizioni. Ogni partizione ha la propria copia della variabile partition-local. Una variabile partition-local è simile a una [variabile thread-local](#), tranne per il fatto che in un singolo thread possono essere eseguite più partizioni.

Il codice e i parametri riportati in questo esempio, somigliano molto al metodo [For](#) corrispondente. Per altre informazioni, vedere [Procedura: scrivere un ciclo Parallel.For con variabili di thread locali](#).

Per usare una variabile partition-local in un ciclo [ForEach](#), è necessario chiamare uno degli overload del metodo che accetta due parametri di tipo. Il primo parametro di tipo, TSource, specifica il tipo di elemento di origine, mentre il secondo parametro di tipo, TLocal, specifica il tipo di variabile [partition-local](#).

```
namespace ParallelForEachWithPartitionLocalVariables
{
    internal class Program
    {
        static void Main(string[] args)
        {
            const int Numbers = 1000000;
            int[] nums = Enumerable.Range(1, Numbers).ToArray();
            long total = 0;

            // First type parameter is the type of the source elements
            // Second type parameter is the type of the thread-local variable (partition subtotal),
finalResult in questo esempio
            Parallel.ForEach<int, long>(nums, // source collection
                                         () => 0, // method to initialize the local variable, subtotal
in questo caso
                                         (item, loopState, subtotal) => // method invoked by the loop
on each iteration
                                         {
                                             subtotal += item; //modify local variable
                                             return subtotal; // value to be passed to next iteration
                                         },
                                         // Method to be executed when each partition has completed.
                                         // finalResult is the final value of subtotal for a particular
partition.
                                         (finalResult) => Interlocked.Add(ref total, finalResult)
                                         );

            Console.WriteLine("The total from Parallel.ForEach is {0:N0}", total);
            //confrontiamo il risultato con la formula di Gauss
            long sommaGauss = (long)Numbers * (Numbers + 1) / 2;
            Console.WriteLine("Il totale con la formula di Gauss {0:N0}", sommaGauss);
        }
    }
}
```

### 7.3.3.8. Un esempio di confronto tra calcolo sequenziale e calcolo parallelo

```
using System.Diagnostics;

namespace ElaborazioneParallelSuCollection1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            //creare una collection di numeri e usare il Parallel.For per sfruttare il calcolo
parallelo
            // 2 million
            var limit = 2_000_000;
            var numbers = Enumerable.Range(1, limit).ToList(); //creo una lista con i valori da 0 a
limit
            const int Numbers = 100000000;
            int[] nums = Enumerable.Range(1, Numbers).ToArray();
            long total = 0;
            Stopwatch stopwatch = Stopwatch.StartNew();
            //vogliamo calcolare la somma dei valori dell'array nums
            //nel caso sequenziale
```

```

stopwatch.Start();
for (int i = 0; i < nums.Length; i++)
{
    total += nums[i];
}
stopwatch.Stop();
Console.WriteLine($"Somma sequenziale = {total}. Tempo impiegato =
{stopwatch.ElapsedMilliseconds} ms");
total = 0;//riazzeriamo total
stopwatch.Restart();//riazzeriamo il cronometro
//nel caso parallelo, senza l'utilizzo di variabili thread-local
Parallel.For(0, nums.Length, i =>
{
    //codice eseguito in maniera parallela
    Interlocked.Add(ref total, nums[i]);
});

stopwatch.Stop();
Console.WriteLine($"Somma parallela = {total}. Tempo impiegato =
{stopwatch.ElapsedMilliseconds} ms");

//versione parallela con l'utilizzo di variabili thread local
total = 0;//riazzeriamo total
stopwatch.Restart();//riazzeriamo il cronometro
Parallel.For<long>(0, nums.Length, ()=>0, (i, loopState, subtotal) =>
{
    subtotal +=nums[i];
    return subtotal;
},
//ogni volta che un Task finisce le sue iterazioni sulla collection passa il valore finale
di subtotal all'action local final
//la somma di tutti i valori di uscita dei thread restituisce il risultato finale
(finalSubtotal) => Interlocked.Add(ref total, finalSubtotal)
);
stopwatch.Stop();
Console.WriteLine($"Somma parallela ottimizzata con Parallel.For = {total}. Tempo
impiegato = {stopwatch.ElapsedMilliseconds} ms");

//versione parallela con l'utilizzo di variabili partition local
total = 0;//riazzeriamo total
stopwatch.Restart();//riazzeriamo il cronometro
Parallel.ForEach<int, long>(nums,//source collection
    ()=>0,//method to initialize the local variable
    (item, loopState, subtotal) =>// method invoked by the loop on each iteration
    {
        subtotal += item;
        return subtotal;
},
// Method to be executed when each partition has completed.
// finalResult is the final value of subtotal for a particular partition.
(finalResult) => Interlocked.Add(ref total, finalResult)
);
stopwatch.Stop();
Console.WriteLine($"Somma parallela ottimizzata con Parallel.ForEach = {total}. Tempo
impiegato = {stopwatch.ElapsedMilliseconds} ms");
//confrontiamo il risultato con la formula di Gauss
long sommaGauss = (long)Numbers * (Numbers + 1) / 2;
Console.WriteLine("Il totale con la formula di Gauss {0}", sommaGauss);
}
}

```

### **7.3.4. Attached and Detached Child Tasks**

<https://docs.microsoft.com/it-it/dotnet/standard/parallel-programming/attached-and-detached-child-tasks>

A *child task* (or *nested task*) is a [System.Threading.Tasks.Task](#) instance that is created in the user delegate of another task, which is known as the *parent task*. A child task can be either detached or attached. A *detached child task* is a task that executes independently of its parent. An *attached child task* is a nested task that is created with the [TaskCreationOptions.AttachedToParent](#) option whose

parent does not explicitly or by default prohibit it from being attached. A task may create any number of attached and detached child tasks, limited only by system resources.

The following table lists the basic differences between the two kinds of child tasks.

Category	Detached child tasks	Attached child tasks
Parent waits for child tasks to complete.	No	Yes
Parent propagates exceptions thrown by child tasks.	No	Yes
Status of parent depends on status of child.	No	Yes

In most scenarios, we recommend that you use detached child tasks, because their relationships with other tasks are less complex. That is why tasks created inside parent tasks are detached by default, and you must explicitly specify the `TaskCreationOptions.AttachedToParent` option to create an attached child task.

#### 7.3.4.1. Detached child tasks

Although a child task is created by a parent task, by default it is independent of the parent task. In the following example, a parent task creates one simple child task. If you run the example code multiple times, you may notice that the output from the example differs from that shown, and also that the output may change each time you run the code. This occurs because the parent task and child tasks execute independently of each other; the child is a detached task. The example waits only for the parent task to complete, and the child task may not execute or complete before the console app terminates.

```
using System;
using System.Threading;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        var parent = Task.Factory.StartNew(() => {
            Console.WriteLine("Outer task executing.");

            var child = Task.Factory.StartNew(() => {
                Console.WriteLine("Nested task starting.");
                Thread.Sleep(500000);
                Console.WriteLine("Nested task completing.");
            });
        });

        parent.Wait();
        Console.WriteLine("Outer has completed.");
    }
}

// The example produces output like the following:
```

```
//      Outer task executing.  
//      Nested task starting.  
//      Outer has completed.  
//      Nested task completing.
```

If the child task is represented by a [Task<TResult>](#) object rather than a [Task](#) object, you can ensure that the parent task will wait for the child to complete by accessing the [Task<TResult>.Result](#) property of the child even if it is a detached child task. The [Result](#) property blocks until its task completes, as the following example shows.

```
using System;  
using System.Threading;  
using System.Threading.Tasks;  
  
class Example  
{  
    static void Main()  
    {  
        var outer = Task<int>.Factory.StartNew(() => {  
            Console.WriteLine("Outer task executing.");  
  
            var nested = Task<int>.Factory.StartNew(() => {  
                Console.WriteLine("Nested task starting.");  
                Thread.Sleep(5000000);  
                Console.WriteLine("Nested task completing.");  
                return 42;  
            });  
  
            // Parent will wait for this detached child.  
            return nested.Result;  
        });  
  
        Console.WriteLine("Outer has returned {0}.", outer.Result);  
    }  
}  
// The example displays the following output:  
//      Outer task executing.  
//      Nested task starting.  
//      Nested task completing.  
//      Outer has returned 42.
```

### 7.3.4.2. Attached child tasks

Unlike detached child tasks, attached child tasks are closely synchronized with the parent. You can change the detached child task in the previous example to an attached child task by using the [TaskCreationOptions.AttachedToParent](#) option in the task creation statement, as shown in the following example. In this code, the attached child task completes before its parent. As a result, the output from the example is the same each time you run the code.

```
using System;  
using System.Threading;  
using System.Threading.Tasks;  
  
public class Example  
{  
    public static void Main()  
    {  
        var parent = Task.Factory.StartNew(() => {  
            Console.WriteLine("Parent task executing.");  
        });  
  
        var child = Task.Factory.StartNew(() => {  
            Console.WriteLine("Child task executing.");  
        }, TaskCreationOptions.AttachedToParent);  
  
        child.Wait();  
  
        Console.WriteLine("Parent has returned {0}.", parent.Result);  
    }  
}
```

```

        var child = Task.Factory.StartNew(() => {
            Console.WriteLine("Attached child starting.");
            Thread.Sleep(5000000);
            Console.WriteLine("Attached child completing.");
        }, TaskCreationOptions.AttachedToParent);
    });
    parent.Wait();
    Console.WriteLine("Parent has completed.");
}
}

// The example displays the following output:
//      Parent task executing.
//      Attached child starting.
//      Attached child completing.
//      Parent has completed.

```

You can use attached child tasks to create tightly synchronized graphs of asynchronous operations.

However, a child task can attach to its parent only if its parent does not prohibit attached child tasks. Parent tasks can explicitly prevent child tasks from attaching to them by specifying the [TaskCreationOptions.DenyChildAttach](#) option in the parent task's class constructor or the [TaskFactory.StartNew](#) method. Parent tasks implicitly prevent child tasks from attaching to them if they are created by calling the [Task.Run](#) method. The following example illustrates this. It is identical to the previous example, except that the parent task is created by calling the [Task.Run\(Action\)](#) method rather than the [TaskFactory.StartNew\(Action\)](#) method. Because the child task is not able to attach to its parent, the output from the example is unpredictable. Because the default task creation options for the [Task.Run](#) overloads include [TaskCreationOptions.DenyChildAttach](#), this example is functionally equivalent to the first example in the "Detached child tasks" section.

```

using System;
using System.Threading;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        var parent = Task.Run(() => {
            Console.WriteLine("Parent task executing.");
            var child = Task.Factory.StartNew(() => {
                Console.WriteLine("Child starting.");
                Thread.Sleep(5000000);
                Console.WriteLine("Child completing.");
            }, TaskCreationOptions.AttachedToParent);
        });
        parent.Wait();
        Console.WriteLine("Parent has completed.");
    }
}

// The example displays output like the following:
//      Parent task executing
//      Parent has completed.
//      Attached child starting.

```

### 7.3.5. Cancellazione di un'attività

<https://docs.microsoft.com/it-it/dotnet/standard/threading/cancellation-in-managed-threads>

A partire da .NET Framework 4, .NET Framework usa un modello unificato per l'annullamento cooperativo di operazioni asincrone o di operazioni sincrone a esecuzione prolungata. Questo modello è basato su un oggetto leggero chiamato token di annullamento. L'oggetto che richiama una o più operazioni annullabili, ad esempio tramite la creazione di nuovi thread o attività, passa il token a ogni operazione. Singole operazioni possono a loro volta passare copie del token ad altre operazioni. In un secondo momento, l'oggetto che ha creato il token può usarlo per richiedere che le operazioni arrestino le rispettive attività. Solo l'oggetto richiedente può inviare la richiesta di annullamento e ogni listener è responsabile del rilevamento della richiesta e della relativa risposta in modo appropriato e tempestivo.

Il criterio generale per implementare il modello di annullamento cooperativo è il seguente:

- Creare un'istanza di un oggetto [CancellationSource](#), che gestisce e invia la notifica di annullamento ai singoli token di annullamento.
- Passare il token restituito dalla proprietà [CancellationSource.Token](#) a ogni attività o thread in attesa di annullamento.
- Specificare un meccanismo per ogni attività o thread per rispondere all'annullamento.
- Chiamare il metodo [CancellationSource.Cancel](#) per fornire la notifica di annullamento.

#### 7.3.5.1. Un esempio di cancellazione di attività

<https://docs.microsoft.com/it-it/dotnet/standard/parallel-programming/task-cancellation>

```
using System;
using System.Collections.Concurrent;
using System.Threading;
using System.Threading.Tasks;

namespace TaskCancellationDemo
{
    class Program
    {
        //usare async Task al posto di void se si usa await task al posto di task.Wait()
        static async Task Main(string[] args)
        {
            var tokenSource = new CancellationTokenSource();
            CancellationToken ct = tokenSource.Token;

            var task = Task.Run(() =>
            {
                // Were we already canceled?
                ct.ThrowIfCancellationRequested();

                bool moreToDo = true;
                while (moreToDo)
                {
                    // Poll on this property if you have to do
                    // other cleanup before throwing.
                    if (ct.IsCancellationRequested)
                    {
                        // Clean up here, then...
                        ct.ThrowIfCancellationRequested();
                    }
                }
            });
        }
    }
}
```

```

        }, ct); // Pass same token to Task.Run.
        Thread.Sleep(7000);
        tokenSource.Cancel();

        // Just continue on this thread, or await with try-catch:
        try
        {
            //https://stackoverflow.com/questions/13140523/await-vs-task-wait-deadlock
            //https://stackoverflow.com/questions/7340309/throw-exception-inside-a-task-
await-vs-wait
            //task.Wait();
            await task;
        }
        catch (OperationCanceledException e)
        {
            Console.WriteLine($"{nameof(OperationCanceledException)} thrown with
message: {e.Message}");
        }
        catch (AggregateException e)
        {
            Console.WriteLine($"{nameof(AggregateException)} thrown with message:
{e.Message}");
        }
        finally
        {
            tokenSource.Dispose();
        }

        Console.ReadKey();
    }

}

```

Lanciare l'esempio precedente con CTRL+F5 (Avvia senza Debug), oppure se si lancia con F5 (Start Debug), Visual Studio si ferma sull'eccezione OperationCanceledException a meno che non si modifichi la configurazione di default di Visual Studio per le eccezioni.

### 7.3.5.2. Cancellare un task e i suoi figli

<https://docs.microsoft.com/it-it/dotnet/standard/parallel-programming/how-to-cancel-a-task-and-its-children>

These examples show how to perform the following tasks:

1. Create and start a cancelable task.
2. Pass a cancellation token to your user delegate and optionally to the task instance.
3. Notice and respond to the cancellation request in your user delegate.
4. Optionally notice on the calling thread that the task was canceled.

The calling thread does not forcibly end the task; it only signals that cancellation is requested. If the task is already running, it is up to the user delegate to notice the request and respond appropriately. If cancellation is requested before the task runs, then the user delegate is never executed and the task object transitions into the Canceled state.

This example shows how to terminate a [Task](#) and its children in response to a cancellation request. It also shows that when a user delegate terminates by throwing a [TaskCanceledException](#), the calling

thread can optionally use the [Wait](#) method or [WaitAll](#) method to wait for the tasks to finish. In this case, you must use a try/catch block to handle the exceptions on the calling thread.

```
using System;
using System.Collections.Concurrent;
using System.Threading;
using System.Threading.Tasks;

public class Program
{
    public static async Task Main()
    {
        var tokenSource = new CancellationTokenSource();
        var token = tokenSource.Token;

        // Store references to the tasks so that we can wait on them and
        // observe their status after cancellation.
        Task t;
        //https://docs.microsoft.com/en-
        us/dotnet/api/system.collections.concurrent.concurrentbag-1
        var tasks = new ConcurrentBag<Task>();

        Console.WriteLine("Press any key to begin tasks...");
        Console.ReadKey(true);
        Console.WriteLine("To terminate the example, press 'c' to cancel and
exit...");  
        Console.WriteLine();

        // Request cancellation of a single task when the token source is canceled.
        // Pass the token to the user delegate, and also to the task so it can
        // handle the exception correctly.
        t = Task.Run(() => DoSomeWork(1, token), token);
        Console.WriteLine("Task {0} executing", t.Id);
        tasks.Add(t);

        // Request cancellation of a task and its children. Note the token is
passed
        // to (1) the user delegate and (2) as the second argument to Task.Run, so
        // that the task instance can correctly handle the
OperationCanceledException.
        t = Task.Run(() =>
        {
            // Create some cancelable child tasks.
            Task tc;
            for (int i = 3; i <= 10; i++)
            {
                // For each child task, pass the same token
                // to each user delegate and to Task.Run.
                tc = Task.Run(() => DoSomeWork(i, token), token);
                Console.WriteLine("Task {0} executing", tc.Id);
                tasks.Add(tc);
                // Pass the same token again to do work on the parent task.
                // All will be signaled by the call to tokenSource.Cancel below.
                //DoSomeWork(2, token);
            }
        }, token);

        Console.WriteLine("Task {0} executing", t.Id);
    }
}
```

```

    tasks.Add(t);

    // Request cancellation from the UI thread.
    char ch = Console.ReadKey().KeyChar;
    if (ch == 'c' || ch == 'C')
    {
        tokenSource.Cancel();
        Console.WriteLine("\nTask cancellation requested.");

        // Optional: Observe the change in the Status property on the task.
        // It is not necessary to wait on tasks that have canceled. However,
        // if you do wait, you must enclose the call in a try-catch block to
        // catch the TaskCanceledExceptions that are thrown. If you do
        // not wait, no exception is thrown if the token that was passed to the
        // Task.Run method is the same token that requested the cancellation.
    }

    try
    {
        await Task.WhenAll(tasks.ToArray());
    }
    //in alternativa all'uso di await si può usare una Wait, ma in tal caso bisogna
gestire l'AggregateException
    //try
    //{
    //    Task myTask = Task.WhenAll(tasks.ToArray());
    //    myTask.Wait();
    //}
    //catch (AggregateException ae)
    //{
    //    foreach (var e in ae.Flatten().InnerExceptions)
    //    {
    //        if (e is TaskCanceledException)
    //        {
    //            Console.WriteLine($"\\n{nameof(TaskCanceledException)} thrown\\n");
    //        }
    //        else
    //        {
    //            throw;
    //        }
    //    }
    //}        catch (OperationCanceledException)
    {
        Console.WriteLine($"\\n{nameof(OperationCanceledException)} thrown\\n");
    }
    finally
    {
        tokenSource.Dispose();
    }

    // Display status of all tasks.
    foreach (var task in tasks)
        Console.WriteLine("Task {0} status is now {1}", task.Id, task.Status);
}

static void DoSomeWork(int taskNum, CancellationToken ct)
{
    // Was cancellation already requested?
    if (ct.IsCancellationRequested)
    {

```

```

        Console.WriteLine("Task {0} was cancelled before it got started.",
                           taskNum);
        ct.ThrowIfCancellationRequested();
    }

    int maxIterations = 100;

    // NOTE!!! A "TaskCanceledException was unhandled
    // by user code" error will be raised here if "Just My Code"
    // is enabled on your computer. On Express editions JMC is
    // enabled and cannot be disabled. The exception is benign.
    // Just press F5 to continue executing your code.
    for (int i = 0; i <= maxIterations; i++)
    {
        // Do a bit of work. Not too much.
        var sw = new SpinWait();
        for (int j = 0; j <= 100; j++)
            sw.SpinOnce();

        if (ct.IsCancellationRequested)
        {
            Console.WriteLine("Task {0} cancelled", taskNum);
            ct.ThrowIfCancellationRequested();
        }
    }
}

// The example displays output like the following:
//      Press any key to begin tasks...
//      To terminate the example, press 'c' to cancel and exit...
//
//      Task 1 executing
//      Task 2 executing
//      Task 3 executing
//      Task 4 executing
//      Task 5 executing
//      Task 6 executing
//      Task 7 executing
//      Task 8 executing
//      c
//      Task cancellation requested.
//      Task 2 cancelled
//      Task 7 cancelled
//
//      OperationCanceledException thrown
//
//      Task 2 status is now Canceled
//      Task 1 status is now RanToCompletion
//      Task 8 status is now Canceled
//      Task 7 status is now Canceled
//      Task 6 status is now RanToCompletion
//      Task 5 status is now RanToCompletion
//      Task 4 status is now RanToCompletion
//      Task 3 status is now RanToCompletion

```

### 7.3.5.3. Un esempio di cancellazione di attività con il Parallel.ForEach

<https://docs.microsoft.com/it-it/dotnet/standard/parallel-programming/how-to-cancel-a-parallel-for-or-foreach-loop>

```
namespace CancelParallelLoops
{
    using System;
    using System.Linq;
    using System.Threading;
    using System.Threading.Tasks;

    class Program
    {
        static void Main()
        {
            int[] nums = Enumerable.Range(0, 10000000).ToArray();
            CancellationTokenSource cts = new CancellationTokenSource();

            // Use ParallelOptions instance to store the CancellationToken
            ParallelOptions po = new ParallelOptions();
            po.CancellationToken = cts.Token;
            po.MaxDegreeOfParallelism = System.Environment.ProcessorCount;
            Console.WriteLine("Press any key to start. Press 'c' to cancel.");
            Console.ReadKey();

            // Run a task so that we can cancel from another thread.
            Task.Factory.StartNew(() =>
            {
                if (Console.ReadKey().KeyChar == 'c')
                    cts.Cancel();
                Console.WriteLine("press any key to exit");
            });

            try
            {
                Parallel.ForEach(nums, po, (num) =>
                {
                    double d = Math.Sqrt(num);
                    Console.WriteLine("{0} on {1}", d,
Thread.CurrentThread.ManagedThreadId);
                    po.CancellationToken.ThrowIfCancellationRequested();
                });
            }
            catch (OperationCanceledException e)
            {
                Console.WriteLine(e.Message);
            }
            finally
            {
                cts.Dispose();
            }

            Console.ReadKey();
        }
    }
}
```

### 7.3.6. Exception handling (Task Parallel Library)

<https://docs.microsoft.com/it-it/dotnet/standard/parallel-programming/exception-handling-task-parallel-library>

Unhandled exceptions that are thrown by user code that is running inside a task are propagated back to the calling thread, except in certain scenarios that are described later in this topic. Exceptions are propagated when you use one of the static or instance [Task.Wait](#) methods, and you handle them by enclosing the call in a try/catch statement. If a task is the parent of attached child tasks, or if you are waiting on multiple tasks, multiple exceptions could be thrown.

To propagate all the exceptions back to the calling thread, the Task infrastructure wraps them in an [AggregateException](#) instance. The [AggregateException](#) exception has an [InnerExceptions](#) property that can be enumerated to examine all the original exceptions that were thrown, and handle (or not handle) each one individually. You can also handle the original exceptions by using the [AggregateException.Handle](#) method.

Even if only one exception is thrown, it is still wrapped in an [AggregateException](#) exception, as the following example shows.

```
using System;
using System.Threading.Tasks;

public class Program
{
    public static void Main()
    {
        var task1 = Task.Run( () => { throw new CustomException("This exception is
expected!"); } );

        try
        {
            task1.Wait();
        }
        catch (AggregateException ae)
        {
            foreach (var e in ae.InnerExceptions) {
                // Handle the custom exception.
                if (e is CustomException) {
                    Console.WriteLine(e.Message);
                }
                // Rethrow any other exception.
                else {
                    throw;
                }
            }
        }
    }
}

public class CustomException : Exception
{
    public CustomException(String message) : base(message)
    {}
}
// The example displays the following output:
```

```
// This exception is expected!
```

You could avoid an unhandled exception by just catching the [AggregateException](#) and not observing any of the inner exceptions. However, we recommend that you do not do this because it is analogous to catching the base [Exception](#) type in non-parallel scenarios. To catch an exception without taking specific actions to recover from it can leave your program in an indeterminate state.

If you do not want to call the [Task.Wait](#) method to wait for a task's completion, you can also retrieve the [AggregateException](#) exception from the task's [Exception](#) property, as the following example shows. For more information, see the [Observing exceptions by using the Task.Exception property](#) section in this topic.

```
using System;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        var task1 = Task.Run( () => { throw new CustomException("This exception is
expected!"); } );

        while(! task1.IsCompleted) {}

        if (task1.Status == TaskStatus.Faulted) {
            foreach (var e in task1.Exception.InnerExceptions) {
                // Handle the custom exception.
                if (e is CustomException) {
                    Console.WriteLine(e.Message);
                }
                // Rethrow any other exception.
                else {
                    throw e;
                }
            }
        }
    }
}

public class CustomException : Exception
{
    public CustomException(String message) : base(message)
    {}
}
// The example displays the following output:
//      This exception is expected!
```

### Nota

When "Just My Code" is enabled, Visual Studio in some cases will break on the line that throws the exception and display an error message that says "exception not handled by user code." This error is benign. You can press F5 to continue and see the exception-handling behavior that is demonstrated in these examples. To prevent Visual Studio from breaking on the first error, just uncheck the **Enable Just My Code** checkbox under **Tools, Options, Debugging, General**.

#### 7.3.6.1. Attached child tasks and nested AggregateExceptions

If a task has an attached child task that throws an exception, that exception is wrapped in

an [AggregateException](#) before it is propagated to the parent task, which wraps that exception in its own [AggregateException](#) before it propagates it back to the calling thread. In such cases, the [InnerExceptions](#) property of the [AggregateException](#) exception that is caught at the [Task.Wait](#), [WaitAny](#), or [WaitAll](#) method contains one or more [AggregateException](#) instances, not the original exceptions that caused the fault. To avoid having to iterate over nested [AggregateException](#) exceptions, you can use the [Flatten](#) method to remove all the nested [AggregateException](#) exceptions, so that the [AggregateException.InnerExceptions](#) property contains the original exceptions. In the following example, nested [AggregateException](#) instances are flattened and handled in just one loop.

```
using System;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        var task1 = Task.Factory.StartNew(() => {
            var child1 = Task.Factory.StartNew(() => {
                var child2 = Task.Factory.StartNew(() => {
                    // This exception is nested inside three
                    AggregateExceptions.
                    throw new CustomException("Attached child2 faulted.");
                }, TaskCreationOptions.AttachedToParent);

                    // This exception is nested inside two AggregateExceptions.
                    throw new CustomException("Attached child1 faulted.");
                }, TaskCreationOptions.AttachedToParent);
            });

        try {
            task1.Wait();
        }
        catch (AggregateException ae) {
            foreach (var e in ae.Flatten().InnerExceptions) {
                if (e is CustomException) {
                    Console.WriteLine(e.Message);
                }
                else {
                    throw;
                }
            }
        }
    }
}

public class CustomException : Exception
{
    public CustomException(String message) : base(message)
    {}
}
// The example displays the following output:
//     Attached child1 faulted.
//     Attached child2 faulted.
```

### 7.3.6.2. Exceptions from detached child tasks

By default, child tasks are created as detached. Exceptions thrown from detached tasks must be

handled or rethrown in the immediate parent task; they are not propagated back to the calling thread in the same way as attached child tasks propagated back. The topmost parent can manually rethrow an exception from a detached child to cause it to be wrapped in an [AggregateException](#) and propagated back to the calling thread.

```
using System;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        var task1 = Task.Run(() => {
            var nested1 = Task.Run(() => {
                throw new CustomException("Detached child
task faulted.");
            });
            // Here the exception will be escalated back to the calling thread.
            // We could use try/catch here to prevent that.
            nested1.Wait();
        });

        try {
            task1.Wait();
        }
        catch (AggregateException ae) {
            foreach (var e in ae.Flatten().InnerExceptions) {
                if (e is CustomException) {
                    Console.WriteLine(e.Message);
                }
            }
        }
    }
}

public class CustomException : Exception
{
    public CustomException(String message) : base(message)
    {}
}
// The example displays the following output:
// Detached child task faulted.
```

### 7.3.7. Dataflow (Task Parallel Library – solo per approfondimento)

<https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/dataflow-task-parallel-library>

#### 7.3.7.1. How to: Write Messages to and Read Messages from a Dataflow Block

<https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/how-to-write-messages-to-and-read-messages-from-a-dataflow-block>

#### 7.3.7.2. How to: Implement a Producer-Consumer Dataflow Pattern

<https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/how-to-implement-a-producer-consumer-dataflow-pattern>

### **7.3.7.3. How to: Perform Action When a Dataflow Block Receives Data**

<https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/how-to-perform-action-when-a-dataflow-block-receives-data>

### **7.3.7.4. Walkthrough: Creating a Dataflow Pipeline**

<https://docs.microsoft.com/it-it/dotnet/standard/parallel-programming/walkthrough-creating-a-dataflow-pipeline>

## **7.3.8. Programmazione asincrona con async e await**

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/>

### **Molto importante! Da studiare tutto.**

Il modello di programmazione asincrona Task (TAP) offre un'astrazione su codice asincrono. È possibile leggere il codice come se ogni istruzione venisse completata prima che venga iniziata quella successiva. Il compilatore esegue una serie di trasformazioni poiché alcune delle istruzioni potrebbero essere eseguite e restituire [Task](#) che rappresenta il lavoro in corso.

L'obiettivo di questa sintassi consiste nell'abilitare un codice che viene letto come una sequenza di istruzioni ma viene eseguito in un ordine più complesso in base all'allocazione delle risorse esterne e al completamento dell'attività. Si tratta di un funzionamento analogo a quello in cui gli utenti specificano istruzioni per i processi che includono attività asincrone. In questo articolo verrà usato un esempio di istruzioni per la preparazione di una colazione per osservare in che modo le parole chiave `async` e `await` consentono di motivare in modo più semplice un codice che include una serie di istruzioni asincrone. Si procederà a scrivere istruzioni come quelle dell'elenco seguente per descrivere come preparare una colazione:

1. Versare una tazza di caffè.
2. Scaldate una padella e friggere due uova.
3. Friggere tre fette di pancetta.
4. Tostare due fette di pane.
5. Aggiungere burro e marmellata alla fetta di pane tostata.
6. Versare un bicchiere di succo d'arancia.

Se si ha esperienza in cucina, queste istruzioni verranno eseguite **in modo asincrono**. Si inizierà a scaldatare la padella per le uova e si inizierà a cuocere la pancetta. Si inserirà il pane nel tostapane, quindi si inizieranno a cuocere le uova. A ogni passaggio del processo si inizia un'attività, quindi ci si dedica alle attività che man mano richiedono attenzione.

La preparazione della colazione è un buon esempio di lavoro asincrono non parallelo. Tutte le attività possono essere gestite da una sola persona (o thread). Continuando con l'analogia della colazione, una sola persona può preparare la colazione in modo asincrono iniziando l'attività successiva prima che l'attività precedente venga completata. La preparazione procede indipendentemente dal fatto che venga controllata da qualcuno. Non appena si inizia a scaldatare la padella per le uova, è possibile iniziare a friggere la pancetta. Dopo aver iniziato a cuocere la pancetta, è possibile inserire il pane nel tostapane.

In un algoritmo parallelo sarebbero necessari più cuochi (o thread). Un cuoco cucinerebbe le uova, un cuoco cucinerebbe la pancetta e così via. Ogni cuoco si dedicherebbe a una singola attività. Ogni cuoco (o thread) verrebbe bloccato in modo sincrono in attesa che la pancetta sia pronta per essere girata o che la tostatura del pane venga completata.

A questo punto, si prendano in esame le stesse istruzioni scritte sotto forma di istruzioni C#:

```

using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using System.Diagnostics;
namespace AsyncBreakfast
{
    internal class Bacon
    {
    }
    internal class Egg
    {
    }
    internal class Coffee
    {
    }
    internal class Juice
    {
    }
    internal class Toast
    {
    }
    class Program
    {
        static void Main(string[] args)
        {
            //Prepariamo la colazione - versione sincrona
            Console.WriteLine("Prepariamo la colazione - versione sincrona");
            ColazioneSincrona();

            //Console.WriteLine("\n\nPrepariamo la colazione - versione parallela");
            //ColazioneParallelta();

            ////Prepariamo la colazione - versione asincrona
            //Console.WriteLine("\n\nPrepariamo la colazione - versione asincrona");
            //await ColazioneAsincrona();

            ////Prepariamo la colazione - versione asincrona ottimizzata
            //Console.WriteLine("\n\nPrepariamo la colazione - versione asincrona
ottimizzata");
            //await ColazioneAsincronaOttimizzata();
        }

        private static void ColazioneSincrona()
        {
            Stopwatch sw = new Stopwatch();
            sw.Start();
            Coffee cup = PourCoffee();
            Console.WriteLine("coffee is ready");
            List<Egg> eggs = FryEggs(2);
            Console.WriteLine("eggs are ready");
            List<Bacon> bacon = FryBacon(3);
            Console.WriteLine("bacon is ready");
            List<Toast> toast = ToastBread(2);
            ApplyButter(toast);
            ApplyJam(toast);
            Console.WriteLine("toast is ready");
            Juice oj = PourOJ();
            Console.WriteLine("oj is ready");
            Console.WriteLine("Breakfast is ready!");
            sw.Stop();
            Console.WriteLine($"Il tempo per la colazione sincrona è
{sw.ElapsedMilliseconds} ms");
        }
    }
}

```

```

}

private static Coffee PourCoffee()
{
    Console.WriteLine($"Sto iniziando a preparare il caffè");
    Task.Delay(1000).Wait();
    return new Coffee();
}

private static List<Egg> FryEggs(int v)
{
    Console.WriteLine($"Sto iniziando a friggere {v} uova");

    List<Egg> uova = new List<Egg>();
    for (int i = 0; i < v; i++)
    {
        // https://stackoverflow.com/questions/20082221/when-to-use-task-delay-when-to-use-thread-sleep
        Task.Delay(200).Wait();
        uova.Add(new Egg());
    }
    return uova;
}

private static List<Bacon> FryBacon(int v)
{
    Console.WriteLine($"Sto iniziando a friggere {v} fette di pancetta");

    List<Bacon> fetteDiPancetta = new List<Bacon>();
    for (int i = 0; i < v; i++)
    {
        Task.Delay(200).Wait();
        fetteDiPancetta.Add(new Bacon());
    }
    return fetteDiPancetta;
}

private static List<Toast> ToastBread(int v)
{
    Console.WriteLine($"Sto iniziando a tostare {v} fette di pane");
    List<Toast> toasts = new List<Toast>();
    for (int i = 0; i < v; i++)
    {
        Console.WriteLine($"{i+1}-ma fetta");
        Task.Delay(200).Wait();
        toasts.Add(new Toast());
    }
    return toasts;
}

private static void ApplyButter(List<Toast> toast)
{
    Console.WriteLine("Sto iniziando a spalmare il burro ");
    for (int i = 0; i < toast.Count; i++)
    {
        Task.Delay(300).Wait();
        Console.WriteLine($"{i + 1}-ma fetta ");
    }
}

```

```

private static void ApplyJam(List<Toast> toast)
{
    Console.WriteLine("Sto iniziando a spalmare la marmellata ");
    for (int i = 0; i < toast.Count; i++)
    {
        Task.Delay(500).Wait();
        Console.WriteLine($"\\tSto spalmando la marmellata sulla {i+1}-ma fetta");
    }
}

private static Juice PourOJ()
{
    Console.WriteLine("Sto iniziando a spremere le arance");
    Task.Delay(1000).Wait();
    return new Juice();
}

}
}

```

I computer non interpretano le istruzioni allo stesso modo delle persone. Il computer si bloccherà in corrispondenza di ogni istruzione fino a quando non verrà completata prima di passare all'istruzione successiva. In questo modo non verrà preparata una colazione soddisfacente. Le attività successive non verranno iniziate prima del completamento delle attività precedenti. La preparazione della colazione richiederà più tempo e alcuni alimenti si raffredderanno prima di essere serviti.

Se si vuole che il computer esegua le istruzioni precedenti in modo asincrono, è necessario scrivere codice asincrono.

Queste considerazioni sono importanti per l'attuale scrittura dei programmi. Quando si scrivono programmi client, si vuole che l'interfaccia utente risponda all'input dell'utente. L'applicazione non deve bloccare l'uso del telefono durante il download di dati dal Web. Quando si scrivono programmi server, non si vuole che i thread vengano bloccati. I thread potrebbero essere impegnati a rispondere ad altre richieste. L'uso di codice sincrono quando sono presenti alternative asincrone riduce la possibilità di aumentare le istanze in modo meno costoso. I thread bloccati hanno un costo.

Per applicazioni moderne efficienti è necessario creare codice asincrono. Senza supporto del linguaggio, la scrittura di codice asincrono richiedeva callback, eventi di completamento o altri elementi che nascondevano la finalità originale del codice. Il vantaggio del codice sincrono risiede nella semplicità di comprensione. Le azioni passo passo rendono più semplice l'analisi e la comprensione. Nei modelli asincroni tradizionali era necessario porre l'attenzione sulla natura asincrona del codice anziché sulle azioni fondamentali del codice.

### **7.3.8.1. Versione parallela, veloce ma non sempre ottimizzata**

Utilizzando una versione parallela del precedente programma è come se avessimo a disposizione tanti cuochi che in parallelo preparano alcune delle attività previste per la colazione, come, ad esempio, friggere le uova, preparare il bacon e tostare il pane. Questo modo di procedere sicuramente riduce i tempi, ma a scapito delle risorse utilizzate, soprattutto quando i singoli thread non sono sempre impegnati ad utilizzare cicli macchina del processore, ma attendono il verificarsi di un evento asincrono. Ad esempio, i cuochi che preparano la colazione starebbero ad attendere che il pane si tosti, che le uova si friggano e così via. In questo caso ci sarebbe un grande dispendio di thread che per gran parte del tempo starebbero solo ad attendere il verificarsi di eventi esterni. La versione

seguente del metodo che prepara la colazione usa i Task per far partire in parallelo le attività relative alla preparazione delle uova, del bacon e dei toast. Viene fatto un wait sul task relativo al toast perché le attività sincrone successive (ApplyButter e ApplyJam) richiedono che i toast siano pronti per poter procedere. L'uso dei task è già un miglioramento rispetto a quello dell'uso diretto dei thread perché la libreria TPL ottimizza l'utilizzo dei thread sottostanti ai task impiegati, tuttavia il modello di programmazione seguito nell'esempio della programmazione parallela è quello di un cobegin/coend della programmazione parallela.

```

private static void ColazioneParallela()
{
    //in questo esempio ci sono alcune attività che sono svolte in parallelo
    //e alcune attività che sono svolte in maniera sincrona
    Stopwatch sw = new Stopwatch();
    sw.Start();
    PourCoffee(); //attività sincrona
    Console.WriteLine("coffee is ready");
    Task<List<Egg>> eggs = Task.Factory.StartNew(() => FryEggs(2));
    Task<List<Bacon>> bacon = Task.Factory.StartNew(() => FryBacon(3));
    Task<List<Toast>> toast = Task.Factory.StartNew(() => ToastBread(2));
    toast.Wait();
    ApplyButter(toast); //attività sincrona
    ApplyJam(toast); //attività sincrona
    Console.WriteLine("toast is ready");
    Juice oj = PourOJ(); //attività sincrona
    Console.WriteLine("oj is ready");
    //quando tutto è pronto la colazione è pronta
    Task.WaitAll(eggs, bacon, toast);
    Console.WriteLine("eggs are ready");
    Console.WriteLine("bacon is ready");
    Console.WriteLine("Breakfast is ready!");
    sw.Stop();
    Console.WriteLine($"Il tempo per la colazione parallela è
{sw.ElapsedMilliseconds} ms");
}

//usa gli stessi metodi della versione sincrona, ma in task paralleli, ad eccezione dei
//seguenti metodi:
private static void ApplyButter(Task<List<Toast>> toast)
{
    Console.WriteLine("Sto iniziando a spalmare il burro ");
    for (int i = 0; i < toast.Result.Count; i++)
    {
        Task.Delay(300).Wait();
        Console.WriteLine($"\\tSto spalmando il burro sulla {i + 1}-ma fetta ");
    }
}

private static void ApplyJam(Task<List<Toast>> toast)
{
    Console.WriteLine("Sto iniziando a spalmare la marmellata ");
    for (int i = 0; i < toast.Result.Count; i++)
    {
        Task.Delay(500).Wait();
        Console.WriteLine($"\\tSto spalmando la marmellata sulla {i + 1}-ma fetta");
    }
}

```

### 7.3.8.2. Non bloccare, ma attendere

Si procederà ora ad aggiornare il codice in modo che il thread non venga bloccato mentre sono in

esecuzione altre attività. La parola chiave await consente di iniziare un'attività senza alcun blocco e di continuare l'esecuzione al completamento dell'attività. Una versione asincrona semplice del codice della preparazione della colazione sarebbe simile al frammento seguente:

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using System.Diagnostics;
namespace AsyncBreakfast
{
    internal class Bacon
    {
    }
    internal class Egg
    {
    }
    internal class Coffee
    {
    }
    internal class Juice
    {
    }
    internal class Toast
    {
    }
    class Program
    {
        static async Task Main(string[] args)
        {
            //Prepariamo la colazione - versione sincrona
            Console.WriteLine("Prepariamo la colazione - versione sincrona");
            ColazioneSincrona();

            //Prepariamo la colazione - versione parallela
            Console.WriteLine("\n\nPrepariamo la colazione - versione parallela");
            ColazioneParallela();

            //Prepariamo la colazione - versione asincrona
            Console.WriteLine("\n\nPrepariamo la colazione - versione asincrona");
            await ColazioneAsincrona();

            //Prepariamo la colazione - versione asincrona
            //Console.WriteLine("\n\nPrepariamo la colazione - versione asincrona ottimizzata");
            //await ColazioneAsincronaOttimizzata();
        }

        private static async Task ColazioneAsincrona()
        {
            Stopwatch sw = new Stopwatch();
            sw.Start();
            PourCoffee();
            //un metodo asincrono restituisce un Task
            //Il task può non restituire nulla, nel caso di azione asincrona (operazione che non restituisce alcun valore al chiamante)
            //oppure può restituire un oggetto, come ad esempio nel caso di una lista di Egg o di Bacon

            //ci sono due modi di utilizzare un metodo asincrono:
```

```

        //1) aspettare il completamento del task restituito dal metodo asincrono, come
nel caso di:
        //Task<List<Egg>> eggs = FryEggsAsync(2);
        //2) aspettare in maniera asincrona direttamente il risultato del task, come nel
caso di
        //List<Toast> toast = await MakeToastWithButterAndJamAsync();
        //in questo secondo caso si utilizza la keyword await, che vuol dire asynchronous
wait.
        //con l'utilizzo della keyword await dopo il segno di = si ottiene non un Task,
ma direttamente il valore restituito
        //dal task, se esiste, oppure si attende la fine del task se questo non
restituisce nessun valore
    Task<List<Egg>> eggs = FryEggsAsync(2);
    Task<List<Bacon>> bacon = FryBaconAsync(3);
    List<Toast> toast = await MakeToastWithButterAndJamAsync(2);
    await bacon;
    Console.WriteLine("bacon is ready");
    await eggs;
    Console.WriteLine("eggs are ready");

    Console.WriteLine("toast is ready");

Juice oj = PourOJ();
Console.WriteLine("oj is ready");
//quando tutto è pronto la colazione è pronta
Console.WriteLine("Breakfast is ready!");
sw.Stop();
Console.WriteLine($"Il tempo per la colazione asincrona è
{sw.ElapsedMilliseconds} ms");
}

private static Coffee PourCoffee()
{
    Console.WriteLine($"Sto iniziando a preparare il caffè");
    Task.Delay(1000).Wait();
    return new Coffee();
}

//Un metodo asincrono è definito tramite la keyword async e restituisce un
Task<TResult> oppure Task se non restituisce nulla
//la keyword async si utilizza in combinazione con la keyword await all'interno del
metodo
private static async Task<List<Egg>> FryEggsAsync(int v)
{
    Console.WriteLine($"Sto iniziando a friggere {v} uova");

    List<Egg> uova = new List<Egg>();
    for (int i = 0; i < v; i++)
    {
        await Task.Delay(200);
        uova.Add(new Egg());
    }
    return uova;
}

private static async Task<List<Bacon>> FryBaconAsync(int v)
{
    Console.WriteLine($"Sto iniziando a friggere {v} fette di pancetta");

    List<Bacon> fetteDiPancetta = new List<Bacon>();
    for (int i = 0; i < v; i++)
    {
        await Task.Delay(200);

```

```

        fetteDiPancetta.Add(new Bacon());
    }
    return fetteDiPancetta;
}

private static async Task<List<Toast>> MakeToastWithButterAndJamAsync(int v)
{
    List<Toast> toast = await ToastBreadAsync(v);
    //await toast;
    ApplyButter(toast); //attività sincrona
    ApplyJam(toast); //attività sincrona
    return toast;
}
private static async Task<List<Toast>> ToastBreadAsync(int v)
{
    Console.WriteLine($"Sto iniziando a tostare {v} fette di pane");
    List<Toast> toasts = new List<Toast>();
    for (int i = 0; i < v; i++)
    {
        Console.WriteLine($"\\tTosto la {i + 1}-ma fetta");
        await Task.Delay(200);
        toasts.Add(new Toast());
    }
    return toasts;
}

private static Juice PourOJ()
{
    Console.WriteLine("Sto iniziando a spremere le arance");
    Task.Delay(1000).Wait();
    return new Juice();
}

}
}

```

Questo codice non si blocca durante la cottura delle uova o della pancetta. Il codice tuttavia non inizia altre attività. Si inserisce il pane nel tostapane e si rimane a osservarlo fino al completamento della cottura. Ma almeno si risponde a un interlocutore che richiede attenzione. In un ristorante in cui vengono fatte più ordinazioni, il cuoco può iniziare a preparare un'altra colazione mentre la prima è in cottura.

Il thread impegnato nella preparazione della colazione non è bloccato in attesa che venga completata un'attività iniziata. Per alcune applicazioni, questa modifica è tutto ciò che serve. Un'applicazione GUI risponde sempre all'utente solo con questa modifica. Tuttavia, per questo scenario si desidera un altro funzionamento. Non si vuole che ogni attività del componente venga eseguita in modo sequenziale. È preferibile iniziare ogni attività del componente prima del completamento dell'attività precedente.

### 7.3.8.3. Iniziare più attività contemporaneamente

In molti scenari si vuole iniziare immediatamente più attività indipendenti. Quindi, man mano che ogni attività viene terminata, è possibile passare ad altre operazioni da eseguire. Nell'analogia della colazione, questa modalità consente di preparare la colazione più rapidamente. Inoltre, tutte le operazioni vengono terminate quasi nello stesso momento. Si otterrà una colazione calda.

[System.Threading.Tasks.Task](#) e i tipi correlati sono classi che è possibile usare per gestire le attività in corso. In questo modo è possibile scrivere codice più simile al modo in cui effettivamente si prepara una colazione. Si inizia a cuocere uova, pancetta e pane contemporaneamente. Man mano che ogni

attività richiederà un'azione, si porrà l'attenzione su quell'attività, quindi sull'azione successiva e infine si rimarrà in attesa di altra attività da eseguire.

Si inizia un'attività e la si mantiene nell'oggetto [Task](#) che rappresenta il lavoro. Si rimarrà in attesa (await) di ogni attività prima di utilizzarne il risultato.

Il codice precedente ha un funzionamento migliore. Tutte le attività asincrone vengono iniziare contemporaneamente. Si rimane in attesa di ogni attività solo quando è necessario avere a disposizione il risultato dell'attività. Il codice precedente potrebbe essere simile al codice di un'applicazione Web che effettua le richieste di diversi microservizi, quindi unisce i risultati in una singola pagina. Si eseguiranno tutte le richieste immediatamente, quindi si rimarrà in attesa (await) di tutte le attività e si comporrà la pagina Web.

#### 7.3.8.4. Composizione di attività

Tutti gli alimenti della colazione sono pronti contemporaneamente ad eccezione del pane. La preparazione del pane rappresenta la composizione di un'operazione asincrona (tostatura del pane) e di operazioni sincrone (aggiunta del burro e della marmellata). L'aggiornamento di questo codice illustra un concetto importante:

##### Importante

La composizione di un'operazione asincrona, seguita da un lavoro sincrono è un'operazione asincrona. In altre parole, se una parte di un'operazione è asincrona, l'intera operazione è asincrona.

Il codice precedente ha mostrato che è possibile usare gli oggetti [Task](#) o [Task<TResult>](#) per attività in esecuzione. Si rimane in attesa (await) di ogni attività prima di usarne il risultato. Il passaggio successivo consiste nel creare metodi che rappresentano la combinazione di altre operazioni. Prima di servire la colazione, si vuole attendere l'attività che rappresenta la tostatura del pane prima dell'aggiunta del burro e della marmellata. È possibile rappresentare queste operazioni con il codice seguente:

```
private static async Task<List<Toast>> MakeToastWithButterAndJamAsync(int v)
{
    List<Toast> toast = await ToastBreadAsync(v);
    ApplyButter(toast); //attività sincrona
    ApplyJam(toast); //attività sincrona
    return toast;
}

private static async Task<List<Toast>> ToastBreadAsync(int v)
{
    Console.WriteLine($"Sto iniziando a tostare {v} fette di pane");
    List<Toast> toasts = new List<Toast>();
    for (int i = 0; i < v; i++)
    {
        Console.WriteLine($"\\tTosto la {i + 1}-ma fetta");
        await Task.Delay(200);
        toasts.Add(new Toast());
    }
    return toasts;
}
```

Il metodo precedente include il modificatore `async` nella firma. Il modificatore segnala al compilatore che il metodo contiene un'istruzione `await`; contiene operazioni asincrone. Questo metodo rappresenta l'attività di tostatura del pane, quindi aggiunge il burro e la marmellata. Questo metodo restituisce [Task<TResult>](#) che rappresenta la composizione di queste tre operazioni.

La modifica precedente ha illustrato una tecnica importante per l'uso di codice asincrono. Si

compongono le attività separando le operazioni in un nuovo metodo che restituisce un'attività. È possibile scegliere quando rimanere in attesa dell'attività. È possibile iniziare altre attività contemporaneamente.

### 7.3.8.5. Attendere le attività in modo efficiente

La serie di istruzioni await alla fine del codice precedente può essere migliorata usando i metodi della classe Task. Una delle API è [WhenAll](#) che restituisce [Task](#) che viene completata quando tutte le attività del relativo elenco di argomenti sono state completate, come illustrato nel codice seguente:

```
await Task.WhenAll(eggsTask, baconTask, toastTask);
Console.WriteLine("eggs are ready");
Console.WriteLine("bacon is ready");
Console.WriteLine("toast is ready");
Console.WriteLine("Breakfast is ready!");
```

Un'altra opzione consiste nell'usare [WhenAny](#) che restituisce Task<Task> che viene completata quando una delle attività fornite sarà completata. È possibile attendere l'attività restituita, sapendo che è già stata completata. Il codice seguente illustra come è possibile usare [WhenAny](#) per attendere il completamento della prima attività e quindi elaborarne il risultato. Dopo aver elaborato il risultato dell'attività completata, si rimuove l'attività completata dall'elenco delle attività passate a WhenAny.

```
private static async Task ColazioneAsincronaOttimizzata()
{
    Stopwatch sw = new Stopwatch();
    sw.Start();
    Coffee cup = PourCoffee();
    Console.WriteLine("coffee is ready");

    var eggsTask = FryEggsAsync(2);
    var baconTask = FryBaconAsync(3);
    var toastTask = MakeToastWithButterAndJamAsync(2);

    var breakfastTasks = new List<Task> { eggsTask, baconTask, toastTask };
    while (breakfastTasks.Count > 0)
    {
        Task finishedTask = await Task.WhenAny(breakfastTasks);
        if (finishedTask == eggsTask)
        {
            Console.WriteLine("eggs are ready");
        }
        else if (finishedTask == baconTask)
        {
            Console.WriteLine("bacon is ready");
        }
        else if (finishedTask == toastTask)
        {
            Console.WriteLine("toast is ready");
        }
        breakfastTasks.Remove(finishedTask);
    }

    Juice oj = PourOJ();
    Console.WriteLine("oj is ready");
    Console.WriteLine("Breakfast is ready!");
    sw.Stop();
    Console.WriteLine($"Il tempo per la colazione asincrona è
{sw.ElapsedMilliseconds} ms");
}
```

Il codice finale è asincrono. Riflette con maggior precisione il modo in cui viene preparata una colazione. Confrontare il codice precedente con il primo esempio di codice di questo articolo. Le

azioni principali risultano ancora chiare dalla lettura del codice. È possibile leggere il codice allo stesso modo in cui si leggerebbero le istruzioni per preparare una colazione riportate all'inizio di questo articolo. Le funzionalità del linguaggio per `async` e `await` offrono la traduzione che ogni persona farebbe per seguire le istruzioni scritte: iniziare le attività non appena possibile e non bloccarsi in attesa del completamento delle attività.

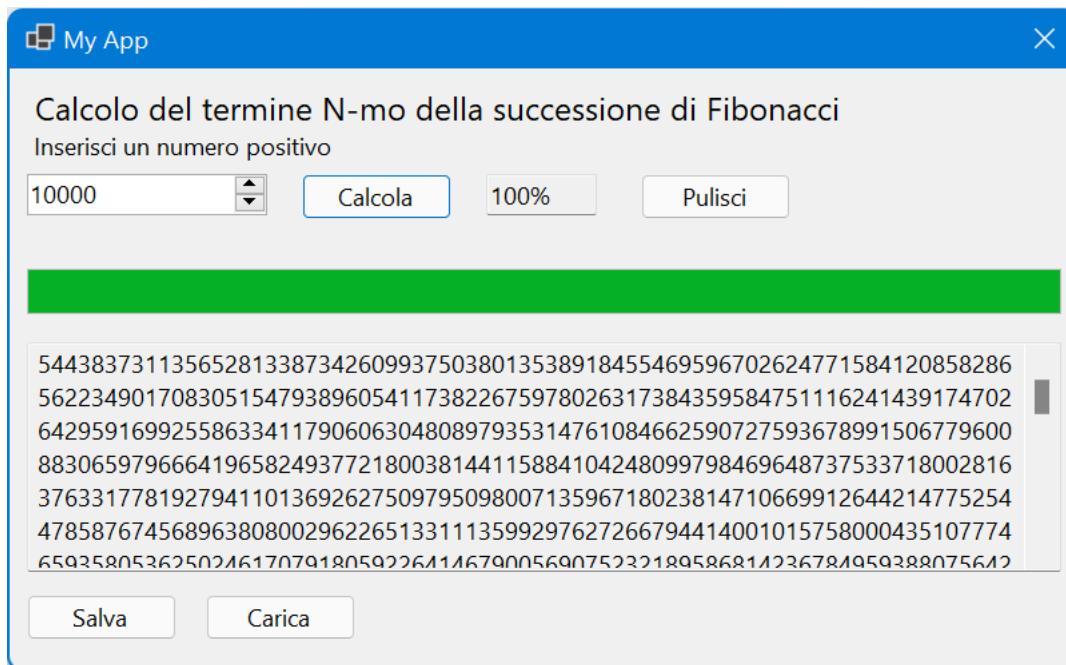
### 7.3.9. Flusso di controllo in programmi asincroni (C#)

<https://docs.microsoft.com/it-it/dotnet/csharp/programming-guide/concepts/async/control-flow-in-async-programs>

Le parole chiave `async` e `await` consentono di scrivere e gestire più facilmente i programmi asincroni. Tuttavia, i risultati potrebbero creare perplessità se non si conosce il funzionamento del programma. Questo argomento descrive il flusso di controllo attraverso un programma asincrono semplice per indicare quando il controllo si sposta da un metodo a un altro e quali informazioni vengono trasferite ogni volta.

In generale, si contrassegnano i metodi che contengono codice asincrono con il modificatore [async \(C#\)](#). In un metodo contrassegnato con un modificatore `async` è possibile usare un operatore [await \(C#\)](#) per specificare dove il metodo viene sospeso in attesa del completamento di un processo asincrono chiamato. Per altre informazioni, vedere [Programmazione asincrona con `async` e `await \(C#\)`](#).

Un esempio con Windows Forms: Calcolo del termine N-mo della successione di Fibonacci



L'aspetto più importante che mette in evidenza questo esempio è come sia possibile utilizzare i costrutti della programmazione asincrona per eseguire operazioni che siano I/O-bound (ossia operazioni che richiedono diverse interazioni con elementi esterni al thread/task), oppure CPU-bound (ossia che richiedono diverse risorse di calcolo del processore) su Task separati dal UI Thread, su cui gira l'interfaccia grafica dell'applicazione. In questo modo è possibile realizzare operazioni che richiedono molto tempo e/o risorse di calcolo senza bloccare l'interfaccia grafica dell'App.

Nell'esempio viene mostrato anche l'utilizzo dell'interfaccia `IProgress` per comunicare aggiornamenti dal Task asincrono sul Thread della user interface.

Codice del Form:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace TaskAsyncDemo
{
    public partial class MyApp : Form
    {
        public MyApp()
        {
            InitializeComponent();
        }

        private async void calcolaButton_Click(object sender, EventArgs e)
        {
            outputProgressBar.Value = 0;
            outputTextBox.Text = "";
            outputProgressBar.Maximum = 100;
            outputProgressBar.Step = 1;
            //https://stackoverflow.com/questions/12126889/how-to-use-winforms-progress-bar
            //https://stackoverflow.com/a/35113843

            var progress = new Progress<int>(v =>
            {
                // This lambda is executed in context of UI thread,
                // so it can safely update form controls
                outputProgressBar.Value = v;
                percentageTextBox.Text = v + "%";
            });
            int valore = (int)InputNumericUpDown.Value;
            // Run operation in another thread
            BigInteger risultato = await Task.Run(() => Fibo(valore, progress));
            // TODO: Do something after all calculations
            outputTextBox.Text = risultato.ToString();
            saveButton.Enabled = true;
        }

        private BigInteger Fibo(int nth, IProgress<int> progress)
        {
            int i;
            BigInteger x = 0, y = 1, z = 0;

            try
            {
                int percentageOld = 0;
                int percentage = percentageOld;
                for (i = 1; i <= nth; i++)
                {
                    z = checked(x + y);
                    x = y;
                    y = z;
                    // Use progress to notify UI thread that progress has
                    // changed
                    progress?.Report(z);
                }
            }
        }
    }
}

```

```

        //https://stackoverflow.com/questions/12126889/how-to-use-winforms-
progress-bar
    percentage = i * 100 / nth;
    if (percentage >= percentageOld + 1)
    {
        percentageOld = percentage;
        if (progress != null)
            progress.Report(percentage);
    }
}
catch (System.OverflowException e)
{
    // The following line displays information about the error.
    Debug.WriteLine("CHECKED and CAUGHT: " + e.ToString());
}

return z;
}

private async void saveButton_Click(object sender, EventArgs e)
{
    //https://stackoverflow.com/questions/60029221/save-file-to-a-specific-folder-
in-windows-form-c-sharp
    var assemblyPath = System.Reflection.Assembly.GetExecutingAssembly().Location;
    var assemblyParentPath = Path.GetDirectoryName(assemblyPath);
    var resultsDir = Path.Combine(assemblyParentPath, "FibonacciResults");

    if (!Directory.Exists(resultsDir))
        Directory.CreateDirectory(resultsDir);

    SaveFileDialog f = new SaveFileDialog();
    f.InitialDirectory = resultsDir;
    string fileName= "Fibonacci of " + InputNumericUpDown.Value + "
"+DateTime.Today.ToShortDateString() + ".txt";
    f.FileName = fileName;
    if (f.ShowDialog() == DialogResult.OK)
    {
        using (Stream s = File.Open(resultsDir + "\\ " + Path.GetFileName(fileName),
 FileMode.Create))
        using (StreamWriter sw = new StreamWriter(s))
        {
            await sw.WriteAsync("Fibonacci of " + InputNumericUpDown.Value +
"=" +outputTextBox.Text);
        }
    }
}

private async void loadButton_Click(object sender, EventArgs e)
{
    string fileContent;
    string filePath;
    var assemblyPath = System.Reflection.Assembly.GetExecutingAssembly().Location;
    var assemblyParentPath = Path.GetDirectoryName(assemblyPath);
    var resultsDir = Path.Combine(assemblyParentPath, "FibonacciResults");

    if (!Directory.Exists(resultsDir))
        Directory.CreateDirectory(resultsDir);

    using (OpenFileDialog openFileDialog = new OpenFileDialog())
    {
        openFileDialog.InitialDirectory = resultsDir;

```

```

openFileDialog.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
openFileDialog.FilterIndex = 2;
openFileDialog.RestoreDirectory = true;

if (openFileDialog.ShowDialog() == DialogResult.OK)
{
    //Get the path of specified file
    filePath = openFileDialog.FileName;

    //Read the contents of the file into a stream
    var fileStream = openFileDialog.OpenFile();

    using (StreamReader reader = new StreamReader(fileStream))
    {
        //lettura del contenuto del file
        fileContent = await reader.ReadToEndAsync();

        //estrapolazione del calcolo e del numero di sequenza
        int posizioneNumeroInFile = fileContent.IndexOf("=") + 1;
        int posizioneNumeroSequenza = "Fibonacci of ".Length - 1;
        int numberofCharToRead = posizioneNumeroInFile -
posizioneNumeroSequenza-1;
        outputTextBox.Text = fileContent.Substring(posizioneNumeroInFile);
        string numeroSequenzaString = fileContent.Substring("Fibonacci of ".
Length-1, numberofCharToRead);
        if(int.TryParse(numeroSequenzaString, out int numero))
        {
            InputNumericUpDown.Value = numero;
        }
    }
}

private void clearButton_Click(object sender, EventArgs e)
{
    outputProgressBar.Value = 0;
    percentageTextBox.Text = "";
    InputNumericUpDown.Value = 1;
    outputTextBox.Text = "";
}
}
}

```

### 7.3.10. Modelli di programmazione asincrona

<https://docs.microsoft.com/it-it/dotnet/standard/asynchronous-programming-patterns/>

#### 7.3.10.1. Task-based asynchronous pattern (TAP)

Il TAP (Task Based asynchronous Pattern) è il pattern di riferimento in .NET.

<https://docs.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/task-based-asynchronous-pattern-tap>

#### 7.3.10.2. Event-based Asynchronous Pattern (EAP) – (legacy model)

<https://docs.microsoft.com/it-it/dotnet/standard/asynchronous-programming-patterns/event-based-asynchronous-pattern-eap>

Starting with the .NET Framework 4, the Task Parallel Library provides a new model for asynchronous and parallel programming. For more information, see [Task Parallel Library \(TPL\)](#) and [Task-based Asynchronous Pattern \(TAP\)](#).

### 7.3.11. I/O di file asincrono

<https://docs.microsoft.com/it-it/dotnet/standard/io/>

<https://docs.microsoft.com/it-it/dotnet/standard/io/asynchronous-file-i-o>

A partire da .NET Framework 4.5, i tipi di I/O includono metodi `async` per semplificare le operazioni asincrone. Un metodo asincrono contiene `Async` nel nome, ad esempio `ReadAsync`, `WriteAsync`, `CopyToAsync`, `FlushAsync`, `ReadLineAsync` e `ReadToEndAsync`. Questi metodi asincroni sono implementati nelle classi di flusso, come `Stream`, `FileStreame`, `MemoryStream`, e nelle classi usate per la lettura o la scrittura nei flussi, come `TextReader` e `TextWriter`.

In .NET Framework 4 e versioni precedenti è necessario usare metodi quali `BeginRead` e `EndRead` per implementare operazioni di I/O asincrone. Questi metodi sono ancora disponibili in .NET Framework 4.5 per supportare il codice legacy basato sull'event-based asynchronous model (EAP). Tuttavia, i metodi `async` consentono di implementare più facilmente le operazioni di I/O asincrone.

L'esempio seguente mostra come usare due oggetti `FileStream` per copiare i file in modo asincrono da una directory a un'altra.

#### 7.3.11.1. Copiare file in modalità asincrona

```
using System;
using System.IO;
using System.Threading;
using System.Threading.Tasks;

namespace CopyAsyncDemo
{
    class Program
    {
        static async Task CopyFiles(string startDirectory, string endDirectory)
        {
            foreach (string fileName in Directory.EnumerateFiles(startDirectory))
            {

                using (FileStream SourceStream = File.Open(fileName, FileMode.Open))
                {
                    using (FileStream DestinationStream = File.Create(endDirectory +
fileName.Substring(fileName.LastIndexOf('\\'))))
                    {
                        await SourceStream.CopyToAsync(DestinationStream);
                        //introduco appositamente un ritardo per poter apprezzare il tempo
che passa
                        //Thread.Sleep(20);
                        await Task.Delay(20);
                    }
                }
            }

            foreach (string directoryName in Directory.EnumerateDirectories(startDirectory))
            {
                //creo la cartella destinazione
                DirectoryInfo destinationDir = Directory.CreateDirectory(endDirectory +
directoryName.Substring(directoryName.LastIndexOf('\\')));
                //chiamata ricorsiva del metodo
                await CopyFiles(directoryName, destinationDir.FullName);
            }
        }
    }
}
```

```

//la copia poteva essere fatta anche come descritto qui:
// https://docs.microsoft.com/en-us/dotnet/api/system.io.directoryinfo

}

static async Task Main(string[] args)
{
    //creo un task che copia i file
    //i percorsi sorgente e destinazione devono esistere
    string StartDirectory = @"C:\Users\genna\source\repos\2020-2021";
    string EndDirectory = @"C:\Users\genna\Documents\Temp";
    Task copyTask = CopyFiles(StartDirectory, EndDirectory);

    //creo un task che perde tempo, ma che interrompo appena ho finito a copiare i
task
    var tokenSource = new CancellationTokenSource();
    CancellationToken ct = tokenSource.Token;
    //lancio il task perdiTempo
    Task perdiTempo = Task.Run(() => {
        ct.ThrowIfCancellationRequested();
        int count = 0;
        while (true)
        {
            Console.SetCursorPosition(0, 0);
            Console.WriteLine($"Sono le ore {DateTime.Now} e tutto va bene, è la
{count++} volta");
            Thread.Sleep(100);
            // Poll on this property if you have to do
            // other cleanup before throwing.
            if (ct.IsCancellationRequested)
            {
                // Clean up here, then...
                ct.ThrowIfCancellationRequested();
            }
        }
    }, ct);
    //avvio il copytask
    await copyTask;
    Console.WriteLine("Copia effettuata correttamente");
    //appena ho finito di copiare interrompo il task perdiTempo
    tokenSource.Cancel();
    try
    {
        await perdiTempo;
    }
    catch (OperationCanceledException)
    {
        Console.WriteLine($"Interrotto il Task Perditempo");
    }
    finally
    {
        tokenSource.Dispose();
    }
}

}
}

```

### 7.3.11.2. Processare a blocchi un file di testo

In questo esempio viene copiato un file di testo con un metodo asincrono che effettua un processamento a blocchi.

Il file è preso da: <https://raw.githubusercontent.com/dlang/druntime/master/benchmark/extr-files/dante.txt>

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ProcessAsyncFile
{
    class Program
    {
        /// <summary>
        /// Copia un file di testo da Source a Destination effettuando un processamento a
        blocchi del file
        /// </summary>
        /// <param name="Source"></param>
        /// <param name="Destination"></param>
        /// <returns></returns>
        public static async Task CopyFilesAsync(StreamReader Source, StreamWriter
        Destination)
        {
            //buffer di char utilizzato per copiare il file di testo
            char[] buffer = new char[0x1000];
            int numRead;
            //ReadAsync legge un blocco di byte di grandezza massima buffer.Length dallo
            stream Souce
            //e lo inserisce in buffer a partire dalla posizione 0 di buffer. ReadAsync
            restituisce
            //il numero di byte letti e avanza nello stream del numero di byte letti
            while ((numRead = await Source.ReadAsync(buffer, 0, buffer.Length)) != 0)
            {
                //WriteDestination scrive in Destination i byte di buffer dalla
                //posizione 0 fino a numRead
                await Destination.WriteAsync(buffer, 0, numRead);
            }
        }
        static async Task Main(string[] args)
        {

            string UserDirectory = @"C:\Users\genna\Documents\Temp\";
            //il file sorgente deve esistere
            using StreamReader SourceReader = File.OpenText(UserDirectory + "dante.txt");
            using StreamWriter DestinationWriter = File.CreateText(UserDirectory + "dante-
            copia.txt");
            await CopyFilesAsync(SourceReader, DestinationWriter);
        }
    }
}
```

### 7.3.12. Async/Await - Best Practices in Asynchronous Programming

<https://stackoverflow.com/questions/12144077/async-await-when-to-return-a-task-vs-void>

<https://stackoverflow.com/a/12144426>

<https://docs.microsoft.com/en-us/archive/msdn-magazine/2013/march/async-await-best-practices-in-asynchronous-programming>

<https://channel9.msdn.com/Series/Three-Essential-Tips-for-Async>

<https://johnthiriet.com/configure-await/>

Summary of Asynchronous Programming Guidelines

Name	Description	Exceptions
Avoid async void	Prefer async Task methods over async void methods	Event handlers
Async all the way	Don't mix blocking and async code	Console main method
Configure context	Use ConfigureAwait(false) when you can	Methods that require context

## 8. Introduzione alla programmazione di rete in .NET

### 8.1. Introduzione

<https://docs.microsoft.com/en-us/dotnet/framework/network-programming/>

<https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient>

Microsoft .NET offre un'implementazione a livelli, estendibile e gestita di servizi Internet che possono essere integrati rapidamente e facilmente nelle applicazioni. Le classi di accesso a Internet negli spazi dei nomi [System.Net](#) e [System.Net.Sockets](#) possono essere usate per implementare applicazioni basate sul Web e basate su Internet.

#### 8.1.1. Applicazioni Internet

Le applicazioni Internet possono essere classificate in due tipi generali: applicazioni client che richiedono informazioni e applicazioni server che rispondono alle richieste di informazioni dai client. La classica applicazione client-server Internet è il World Wide Web, in cui gli utenti usano i browser per accedere a documenti e ad altri dati archiviati in server Web in tutto il mondo.

Le applicazioni non si limitano solo a uno di questi ruoli. Ad esempio, il comune server applicazioni di livello intermedio risponde alle richieste dei client richiedendo dati a un altro server, caso in cui agisce sia come server sia come client.

L'applicazione client effettua una richiesta identificando la risorsa Internet richiesta e il protocollo di comunicazione da usare per la richiesta e la risposta. Se necessario, il client fornisce anche tutti i dati aggiuntivi necessari per completare la richiesta, come la posizione del proxy o le informazioni di autenticazione (nome utente, password e così via). Una volta creata, la richiesta può essere inviata al server.

#### 8.1.2. Identificazione delle risorse

.NET Framework usa un URI (Uniform Resource Identifier) per identificare la risorsa Internet richiesta e il protocollo di comunicazione. L'URI è costituito da almeno tre, possibilmente quattro, parti: l'identificatore dello schema, che identifica il protocollo di comunicazione per la richiesta e la risposta, l'identificatore del server, costituito da un nome host DNS (Domain Name System) o un indirizzo TCP che identifica in modo univoco il server in Internet, l'identificatore del percorso, che individua le informazioni richieste nel server e una stringa di query facoltativa, che passa informazioni dal client al server.

Ad esempio, l'URI <http://www.contoso.com/whatsnew.aspx?date=today> è costituito dall'identificatore dello schema http, dall'identificatore del server www.contoso.com, dal percorso /whatsnew.aspx e dalla stringa di query ?date=today.

Dopo che il server ha ricevuto la richiesta e ha elaborato la risposta, restituisce la risposta all'applicazione client. La risposta include informazioni supplementari, tra cui il tipo di contenuto (ad esempio, testo non elaborato o dati XML).

<https://en.wikipedia.org/wiki/URL>:

A **Uniform Resource Locator (URL)**, colloquially termed a **web address**,<sup>[1]</sup> is a reference to a **web resource** that specifies its location on a **computer network** and a mechanism for retrieving it. A URL is a specific type of **Uniform Resource Identifier (URI)**,<sup>[2][3]</sup> although many people use the two terms interchangeably.<sup>[4][5]</sup> URLs occur most commonly to reference web pages ([http](#)), but are also used for file transfer ([ftp](#)), email ([mailto](#)), database access ([JDBC](#)), and many other applications.

Most **web browsers** display the URL of a web page above the page in an **address bar**. A typical URL could have the form <http://www.example.com/index.html>, which indicates a protocol ([http](#)),

a hostname (`www.example.com`), and a file name (`index.html`).

Every URL ultimately breaks down to the following structure, where optional components are designated with square brackets [] :

`scheme://authority/]path[?query][#fragment]`

So here, we see plainly that the only required components of a URL are the scheme and subsequent colon delimiter, and a path. Everything else is optional, and you'll note that each optional component has its presence in the URL indicated by its unique prefix character.

#### **8.1.2.1. Authority specification**

The authority can be broken down as follows: `//[access_credentials][@]host_domain[:port]`

So, if an authority component is present, it will always be prefixed with a ( // ) delimiter, and will always contain the host domain.

Meanwhile, the access credentials component is also broken down as follows: `[user_id][:][password]` Here, only one component is required. However, if either component is present, then the ( @ ) character separating access credentials from the host domain becomes a requirement. And, if both the user\_id and the password functions are present, then the colon ( : ) delimiter between the two components will be required.

#### **8.1.2.2. Query specification**

Finally, the last component that has a well-defined specification for how it can be composed is the query component. It can be broken down as follows:

`?[parameter=value][( ; | &)parameter=value]...`

The sequence of additional delimiters and key-value pairs can extend all the way to the maximum allowable length of a valid URL. By following these syntax specifications, you can decompose any URL you are presented with into its component parts, and meaningfully leverage it to access the resource it identifies.

#### **8.1.2.3. The URL as a sub-type of the URI**

A URL is actually a single, specific kind of something known as a Uniform Resource Identifier ( URI ), which is a string of characters adhering to a well-defined syntax that universally and uniquely identifies a resource on a network. The distinction between a URL and a URI is subtle, and almost entirely conceptual. The simplest way to characterize that distinction is to note that, by using a URL, we are guaranteed to be able to identify and locate a requested resource. The only thing we are guaranteed, given a simple URI, is an ability to identify it that is to distinguish the resource from any other arbitrary resource. In practice, however, the terms URL and URI are frequently used interchangeably. However, the class exposed by .NET Core for constructing, decomposing, and leveraging these addresses is named for the more generic URI specification.

#### **8.1.2.4. The `System.Net.UriBuilder` class**

Vediamo un esempio di creazione di URI in .NET Core

```
using System;
using System.Threading;

namespace UriTest
{
    public class UriTestProgram
```

```

{
    public static Uri GetSimpleUri() {
        var builder = new UriBuilder();
        builder.Scheme = "http";
        builder.Host = "packt.com";
        return builder.Uri;
    }

    public static Uri GetSimpleUri_Constructor() {
        var builder = new UriBuilder("http", "packt.com");
        return builder.Uri;
    }

    public static void Main(string[] args)
    {
        var simpleUri = GetSimpleUri();
        Console.WriteLine(simpleUri.ToString());
        // Expected output: http://packt.com

        var constructorUri = GetSimpleUri_Constructor();
        Console.WriteLine(constructorUri.ToString());
        // Expected output: http://packt.com

        Thread.Sleep(10000);
    }
}
}

```

### 8.1.2.5. Hosts – domain names and IPs

An IP address is the underlying numeric address used by routing hardware and software to navigate to a resource on a network. It's the unique ID, specific to a piece of hardware at a specific location. A domain name, however, is the human-readable string of words and alpha-numeric characters used to make addressing easier and more consistent. It is more consistent, easily remembered, and less prone to error than a raw IP address. What's interesting, however, is that domain names and their IP addresses are actually functionally interchangeable. In any context in which you can use one, you can always safely substitute the other.

The DNS is a distributed, decentralized network of authoritative servers that hosts a directory of all sub-domain servers, as well as any domain names that can be resolved by that authoritative server. Any domain name that has been registered with a certified domain name registrar, and which meets the syntax standards of a domain name (and which hasn't already been registered), is considered valid. Valid domain names are added to the distributed registry hosted by authoritative servers. Between your computer and any other network node you hope to interact with using a valid, registered domain name, your request will have to interact with one or more of these name servers. Each server will inspect the domain name given, and look up the domain in its own directory of names and IP address mappings. Naturally, the server will first determine if the given name can be resolved by that server, or at least by one of its subordinate servers. If so, the authoritative server simply replaces the domain name in the request with the IP address to which it maps, and forwards the request along accordingly. If the current server cannot resolve the domain name, however, it will forward it along up the hierarchy of name servers to a more general, parent domain. This process continues up to the root name server, or until the name is resolved.

It is occasionally necessary to identify the underlying IP address for a domain name from within the context of our software. For that, .NET Core provides the static Dns class as part of the System.Net namespace. With the Dns class, we can access directory information as returned by the nearest downstream name server capable of resolving the given name. We can request an instance of the IPHostEntry class, containing all of the relevant directory information of a DNS entry, or simply an array of IP addresses registered to resolve requests against the domain name. To see this in action,

simply invoke any of the methods exposed by the static Dns class in a sample program as follows:

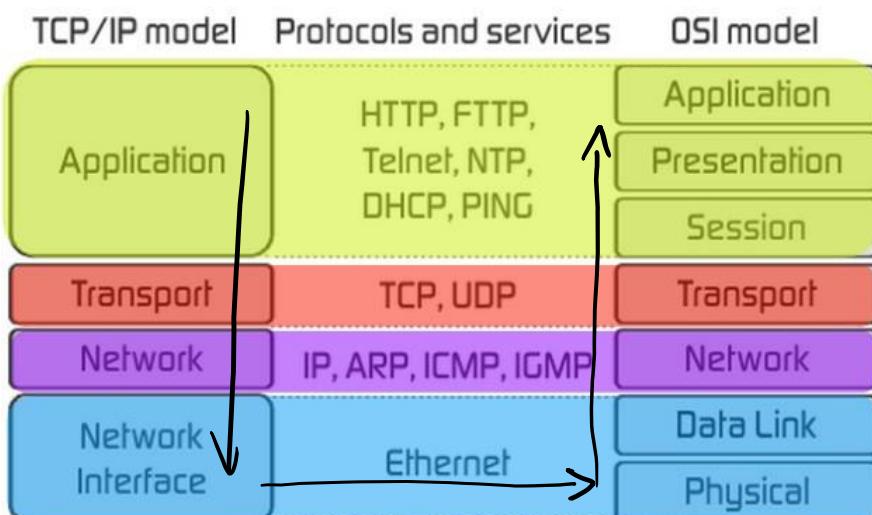
```
using System;
using System.Net;
using System.Threading;

namespace DnsTest
{
    class DnsTestProgram {
        static void Main(string[] args) {
            var domainEntry = Dns.GetHostEntry("google.com");
            Console.WriteLine(domainEntry.HostName);
            foreach (var ip in domainEntry.AddressList) {
                Console.WriteLine(ip);
            }

            var domainEntryByAddress = Dns.GetHostEntry("127.0.0.1");
            Console.WriteLine(domainEntryByAddress.HostName);
            foreach (var ip in domainEntryByAddress.AddressList) {
                Console.WriteLine(ip);
            }
        }
    }
}
```

### 8.1.2.6. The Network Reference Models

[https://it.wikipedia.org/wiki/Suite\\_di\\_protocolli\\_Internet](https://it.wikipedia.org/wiki/Suite_di_protocolli_Internet)



The application layer is where the vast majority of day-to-day network programming will take place. This is especially true within the .NET Core framework, since the libraries provided by that framework deliver a wide array of clean, easy-to-use abstractions for entities or responsibilities that must be programmed lower in the stack.

This might feel redundant at this point, but it really is worth driving home that the application layer is where the vast majority of .NET developers are going to be doing their network programming. Since that accounts for most of you, we're going to keep talking about it. But why is the application layer so important? The crux of it is that the application layer serves as the gateway to network activities for your business logic. This becomes very apparent as you explore how thoroughly .NET has hidden the implementation details of any of the responsibilities of lower levels of the network stack. Essentially, if there is something that you need to specify about how your application should behave anywhere below the stack, you'll be doing so through a .NET library class.

### 8.1.3. List of Well-Known Ports

Port numbers range from 0 to 65535, but only port numbers 0 to 1023 are reserved for privileged services and designated as well-known ports. The following list of *well-known port numbers* specifies the port used by the server process as its contact port.

Port Number	Description
1	<a href="#">TCP</a> Port Service Multiplexer (TCPMUX)
5	Remote Job Entry (RJE)
7	ECHO
18	Message Send Protocol (MSP)
20	<a href="#">FTP</a> -- Data
21	<a href="#">FTP</a> -- Control
22	<a href="#">SSH</a> Remote Login Protocol
23	<a href="#">Telnet</a>
25	<a href="#">Simple Mail Transfer Protocol (SMTP)</a>
29	MSG ICP
37	Time
42	Host Name Server (Nameserv)
43	Whols
49	Login Host Protocol (Login)
53	<a href="#">Domain Name System (DNS)</a>
69	<a href="#">Trivial File Transfer Protocol (TFTP)</a>
70	<a href="#">Gopher</a> Services
79	<a href="#">Finger</a>
80	<a href="#">HTTP</a>
103	<a href="#">X.400</a> Standard
108	SNA Gateway Access Server
109	POP2
110	<a href="#">POP3</a>
115	Simple File Transfer Protocol (SFTP)
118	SQL Services
119	Newsgroup ( <a href="#">NNTP</a> )
137	<a href="#">NetBIOS</a> Name Service
139	NetBIOS Datagram Service
143	Interim Mail Access Protocol (IMAP)

150	NetBIOS Session Service
156	<a href="#">SQL Server</a>
161	<a href="#">SNMP</a>
179	<a href="#">Border Gateway Protocol</a> (BGP)
190	Gateway Access Control Protocol (GACP)
194	<a href="#">Internet Relay Chat</a> (IRC)
197	Directory Location Service (DLS)
389	<a href="#">Lightweight Directory Access Protocol</a> (LDAP)
396	Novell Netware over IP
443	<a href="#">HTTPS</a>
444	Simple Network Paging Protocol (SNPP)
445	Microsoft-DS
458	Apple <a href="#">QuickTime</a>
546	<a href="#">DHCP</a> Client
547	DHCP Server
563	SNEWS
569	MSN
1080	Socks

Well-known ports range from 0 through 1023.

Registered ports are 1024 to 49151.

Dynamic ports (also called private ports) are 49152 to 65535.

For further information, see [RFC 1700](#).

#### 8.1.4. HTTP – application to application communication

Welcome to the bread and butter of almost every .NET Core developer working today. HTTP is by far the most common and useful protocol for applications to interact over networks currently in use today. Why is that? Because HTTP is the protocol that almost every single web page on the internet is served up on by remote hosts, and requested by local clients. That alone is reason enough to call it the most common protocol in use. If you want more evidence, though, consider that most native mobile applications that serve up web-hosted data request this data from APIs that are exposed via HTTP.

<https://developer.mozilla.org/en-US/docs/Web/HTTP>

What is HTTP? As should already be obvious, HTTP is a protocol implemented and leveraged by software that lives in the application layer of the OSI network stack. It's the primary mechanism of communication for applications exposed through the internet, and is designed for the transfer of hypermedia over a network. Hypermedia typically refers to hypertext documents that contain multimedia information, as well as hyperlinks that can be used to navigate to and load additional resources from other remote hosts. The transfer component of HTTP is, fundamentally, a request/response protocol that assumes a client-server relationship between hosts in an active HTTP session. To understand how this is done, let's start with the notion of a client-server relationship.

#### **8.1.4.1. Request/response**

In describing the nature of the client-server relationship, we've also touched on the nature of the HTTP request/response protocol. This protocol, as a way of serving up information, is fairly intuitive to understand. When a client makes a request of a server (the request part of request/response), the server, assuming it meets the specifications of the protocol, is expected to respond with meaningful information about the success or failure of that request, as well as by providing the specific data initially requested. Sometimes, the complete process of requesting information and receiving a meaningful response requires several intermediary round-trips between the client and server to establish initial connections, determine the ability of the server to service the request, and then submit the information necessary to initiate the request. This entire process, however, will be considered a single request/response session from the perspective of application-layer software. This leads us nicely onto the subject of just how those sessions are initially established in the first place.

#### **8.1.4.2. HTTP sessions**

So far, we've talked about the back and forth of the request/response communication patterns of HTTP, but we've neglected the context that allows that chatter to happen so seamlessly. This fluid interaction is facilitated by an underlying session established prior to satisfying the first request made by a client. Historically, this session has been provided by a Transmission Control Protocol (TCP) connection established against a specific port on the host server. This port can be specified in the URI when designating your target host, but typically will use default ports for HTTP, such as 80, 8080, or 443 (for HTTPS, which we'll cover later in this book). Once the connection is established, round trips of HTTP communication can proceed freely until the session is terminated.

#### **8.1.4.3. Request methods**

When a client wants to make a request of a server, it must specify the method by which the server will be expected to respond to the given request. These method specifications are typically called HTTP verbs, since most of them describe an action to be taken by the server when processing a request sent by the client. The standard methods are as follows:

- **OPTIONS** : This returns the list of other HTTP methods supported by the server at the given URL.
- **TRACE** : This is a utility method that will simply echo the original request as received by the server. It is useful for identifying any modifications made to the request by entities on the network while the request is in transit.
- **CONNECT** : CONNECT requests establish a transparent TCP/IP tunnel between the originating host and the remote host.
- **GET** : This retrieves a copy of the resource specified by the URL to which the HTTP request was sent. By convention, GET requests will only ever retrieve the resource, with no side-effects on the state of the resources on the server (however, these conventions can be broken by poor programming practices, as we'll see later in the book).
- **HEAD** : This method requests the same response as a GET request to a given URL, but without the body of the response. What is returned is only the response headers.
- **POST** : The POST method transmits data in the body of the request, and requests that the server store the content of the request body as a new resource hosted by the server.
- **PUT** : The PUT method is similar to the POST method in that the client is requesting that the server store the content of the request body. However, in the case of a PUT operation, if there is already content at the requested URL, this content is then modified and updated with the contents of the request body.
- **PATCH** : The PATCH method will perform partial updates of the resource at the requested URL,

modifying it with the contents of the request body. A PATCH request will typically fail if there is no resource already on the server to be patched.

- **DELETE** : The DELETE method will permanently delete the resource at the specified URL.

A server will not respond to a request method invoked against a given location unless the server has been configured to do so. This is because some of the methods defined by the HTTP standard can permanently impact the state of resources on that server, and so should only be invoked and processed when it is safe to irrevocably update that state. There are, however, a number of methods designated as safe, by convention. This simply means that they can be processed by a server without having any side-effects on the state of the resources on that server. HEAD, GET, OPTIONS, and TRACE are all conventionally designated as safe.

#### 8.1.4.4. Status codes

Even when an application has constructed a valid HTTP request object, and submits that request to a valid path on an active host, it's not uncommon for the server to fail to properly respond. For this reason, HTTP designates, as part of a response object, a status code to communicate the ability of the server to properly service the request.

HTTP status codes are, by convention, 3-digit numeric codes returned as part of every response. The first digit indicates the general nature of the response, and the second and third digits will tell you the exact issue encountered. In this way, we can say that status codes are categorized by their first digits. When you're writing software that responds to HTTP requests, it's important to send accurate status codes in response to different errors. HTTP is a standard that must be adhered to by developers in order to remain useful. There are only five valid values for the first digit of an HTTP status code, and thus, five categories of responses; they are as follows:

- **1XX: Informational status code.** This indicates that the request was in fact received, and the processing of that request is continuing.
- **2XX: Success status code.** This indicates that the request was successfully received and responded to.
- **3XX: Redirection.** This indicates that the requesting host must send their request to a new location for it to be successfully processed.
- **4XX: Client Error.** An error that is produced by the actions of the client, such as sending a malformed request or attempting to access resources from the wrong location.
- **5XX: Server Error.** There was a fault on the server preventing it from being able to fulfill a request. The client submitted the request correctly, but the server failed to satisfy it. Status codes are returned by servers for every HTTP request made against the server, and so can be very useful for building resiliency into your client software.

#### 8.1.4.5. The HTTP message format

Requests and responses in HTTP are always sent as plain text messages. Those plain text messages consist of a well-ordered, and well-structured, series of message segments that can be reliably parsed by the recipient.

In requests, messages consist of three required message components and one optional component:

- The request line consists of the method, the path to the requested resource, and the specific protocol version that should be used to determine the validity of the rest of the message; for example, GET /users/id/12 HTTP/1.1 .
- A series of request headers and their values, for example, Accept: application/json .
- An empty line.

- (Optional) A request message body. This consists of content headers that provide metadata about the content type, as well as the content itself. Each segment is delineated by a <CR> carriage return character and an <LF> line feed character; these are special white-space characters whose specific American Standard Code for Information Interchange ( ASCII ) values allow them to reliably be used to indicate the breaks between segments in a message stream.

Meanwhile, an HTTP response consists of its own series of almost identically structured segments, each also delimited by the <CR><LF> characters. Just as with the request message, it contains three required segments and one optional message body segment, as follows:

- A status line consisting of the specific protocol, the HTTP status code, and the reason phrase associated with that status code, e.g.:
  - HTTP/1.1 401 Bad Request : A response containing the 401 client error status code (indicating that the client sent an improper request message for the resource that it was looking for).
  - HTTP/2.0 201 Created : A response indicating the 201 success status code, meaning that the desired resource has been created on the server.
- Headers, as with the request message segment, providing metadata about how the response should be parsed.
- An empty line.
- An optional message body.

Those simple segments fully define every valid HTTP message sent across the internet. This accounts for millions of requests per second, between millions or billions of devices. It's the simplicity of the message specification that makes that kind of scale possible.

#### **8.1.4.6. HTTP in C#**

Remembering the proper character delimiters and order for segments in an HTTP message, anyone should be able to build a request from scratch. Thankfully though, you don't have to remember those details; .NET Core has you covered with the System.Net.Http namespace. We'll explore this namespace in much greater detail later, but for now, just trust that any feature or detail you find yourself needing to leverage HTTP communication in your application is exposed through that namespace. This namespace exposes enum types for status codes and header values, and an HttpMethod class to specify your message verb. As a library, it's rich with out-of-the-box features while remaining flexible and extensible enough to be leveraged in any use case.

#### **8.1.5. FTP and SFTP**

[https://it.wikipedia.org/wiki/File\\_Transfer\\_Protocol](https://it.wikipedia.org/wiki/File_Transfer_Protocol)

<https://it.wikipedia.org/wiki/FTPS>

#### **8.1.6. SMTP**

[https://it.wikipedia.org/wiki/Simple\\_Mail\\_Transfer\\_Protocol](https://it.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol)

#### **8.1.7. TCP**

[https://it.wikipedia.org/wiki/Transmission\\_Control\\_Protocol](https://it.wikipedia.org/wiki/Transmission_Control_Protocol)

#### **8.1.8. UDP**

[https://it.wikipedia.org/wiki/User\\_Datagram\\_Protocol](https://it.wikipedia.org/wiki/User_Datagram_Protocol)

## 8.2. Richieste e risposte HTTP in .NET

<https://docs.microsoft.com/en-us/dotnet/framework/network-programming/introducing-pluggable-protocols>

### 8.2.1. Generalità

#### 8.2.1.1. Gestione del DNS

<https://aws.amazon.com/it/route53/what-is-dns/>

<https://youtu.be/e2xLV7pCOLI>

Vedere anche il file hosts in C:\Windows\System32\Drivers\etc\hosts

Comando nslookup: <https://ss64.com/nt/nslookup.html>

Comando ping: <https://ss64.com/nt/ping.html>

<https://serverfault.com/questions/698058/nslookup-not-using-hosts-file>

Nel seguente esempio si vede come è possibile risolvere un nome DNS oppure un indirizzo ip tramite il metodo Dns.GetHostEntry().

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Threading.Tasks;
using System.Threading;

namespace DnsTest
{
    class DnsTestProgram
    {
        static void Main(string[] args)
        {
            var domainEntry = Dns.GetHostEntry("www.istitutogreppi.edu.it");
            Console.WriteLine(domainEntry.HostName);
            foreach (var ip in domainEntry.AddressList)
            {
                Console.WriteLine(ip);
            }

            var domainEntryByAddress = Dns.GetHostEntry("127.0.0.1");
            Console.WriteLine(domainEntryByAddress.HostName);
            foreach (var ip in domainEntryByAddress.AddressList)
            {
                Console.WriteLine(ip);
            }
        }
    }
}
```

Provare a confrontare il risultato con quanto ottenuto dal comando nslookup della shell di Windows:

```

C:\WINDOWS\system32\cmd.exe
www.istitutogreppi.edu.it
89.46.105.74
kubernetes.docker.internal
127.0.0.1

Microsoft Windows [Version 10.0.18363.535]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\genna>nslookup www.istitutogreppi.edu.it
Server: UnKnown
Address: 192.168.43.1

Non-authoritative answer:
Name: www.istitutogreppi.edu.it
Address: 89.46.105.74

C:\Users\genna>nslookup 127.0.0.1
Server: UnKnown
Address: 192.168.43.1

Name: localhost
Address: 127.0.0.1

```

### 8.2.1.2. Costruzione di un oggetto URI

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;

namespace UriTest
{
    public class UriTestProgram
    {
        public static Uri GetSimpleUri()
        {
            var builder = new UriBuilder();
            builder.Scheme = "http";
            builder.Host = "packt.com";
            return builder.Uri;
        }

        public static Uri GetSimpleUri_Constructor()
        {
            var builder = new UriBuilder("http", "packt.com");
            return builder.Uri;
        }

        public static void Main(string[] args)
        {
            var simpleUri = GetSimpleUri();
            Console.WriteLine(simpleUri.ToString());
            // Expected output: http://packt.com

            var constructorUri = GetSimpleUri_Constructor();
            Console.WriteLine(constructorUri.ToString());
            // Expected output: http://packt.com
        }
    }
}

```

### **8.2.1.3. Caricamento e download dei dati da un server Internet**

<https://docs.microsoft.com/en-us/dotnet/framework/network-programming/requesting-data>

Note importanti:

<https://docs.microsoft.com/en-us/dotnet/api/system.net.webclient?view=net-6.0>

Non è consigliabile usare la classe WebClient per il nuovo sviluppo. Usare invece la classe **System.Net.Http.HttpClient**.

<https://docs.microsoft.com/en-us/dotnet/api/system.net.httpwebrequest?view=net-6.0>

Non è consigliabile usare HttpWebRequest per il nuovo sviluppo. Usare invece la classe **System.Net.Http.HttpClient**.

<https://docs.microsoft.com/en-us/dotnet/api/system.net.ftpwebrequest?view=net-6.0>

Non è consigliabile usare la FtpWebRequest classe per nuove attività di sviluppo. Per altre informazioni e alternative a FtpWebRequest, vedere la pagina relativa all' [uso di WebRequest](#) in GitHub.

<https://github.com/dotnet/platform-compat/blob/master/docs/DE0003.md>

WebRequest-based APIs are on life-support only (that is, only critical fixes, no new improvements, enhancements).

For HttpWebRequest: use [HttpClient](#) instead.

For FtpWebRequest: use third party FTP client (e.g. from [this list](#)).

## **8.2.2. HttpClient**

<https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=net-6.0>

### **8.2.2.1. Scaricare del testo dalla rete**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;

namespace HttpClientDemo1
{
    class Program
    {
        // HttpClient is intended to be instantiated once per application, rather than per-
use. See Remarks.
        static readonly HttpClient client = new HttpClient();
        static async Task Main()
        {
            // Call asynchronous network methods in a try/catch block to handle exceptions.
            try
            {
                HttpResponseMessage response = await
client.GetAsync("https://www.istitutogreppi.edu.it/");
                response.EnsureSuccessStatusCode();
                string responseBody = await response.Content.ReadAsStringAsync();
                // Above three lines can be replaced with new helper method below
                // string responseBody = await client.GetStringAsync(uri);

                Console.WriteLine(responseBody);
            }
        }
}
```

```
        catch (HttpRequestException e)
    {
        Console.WriteLine("\nException Caught!");
        Console.WriteLine("Message :{0} ", e.Message);
    }
}
```

The [HttpClient](#) class instance acts as a session to send HTTP requests. An [HttpClient](#) instance is a collection of settings applied to all requests executed by that instance. In addition, every [HttpClient](#) instance uses its own connection pool, isolating its requests from requests executed by other [HttpClient](#) instances.

The [HttpClient](#) also acts as a base class for more specific HTTP clients. An example would be a [FacebookHttpClient](#) providing additional methods specific to a Facebook web service (a `GetFriends` method, for instance). Derived classes should not override the virtual methods on the class. Instead, use a constructor overload that accepts [HttpMessageHandler](#) to configure any pre- or post-request processing instead.

### **Importante:**

[HttpClient](#) is intended to be instantiated once and re-used throughout the life of an application. Instantiating an HttpClient class for every request will exhaust the number of sockets available under heavy loads. This will result in `SocketException` errors. Below is an example using HttpClient correctly.

```
public class GoodController : ApiController
{
    private static readonly HttpClient HttpClient;

    static GoodController()
    {
        HttpClient = new HttpClient();
    }
}
```

### **8.2.2.2. Uso di HttpClient con proxy**

Nel caso in cui la connessione con l'endpoint remoto avvenga per mezzo di un proxy occorre configurare l'oggetto HttpClient per utilizzare il proxy. Ci sono diversi meccanismi per utilizzare il proxy; quello che viene riportato di seguito è stato testato su Windows 11 e dovrebbe funzionare anche su altri sistemi operativi (a patto che sul guest OS sia stato configurato il proxy di sistema).

```
using System;
using System.Net.Http;
using System.Runtime.InteropServices;
using Microsoft.Win32;
using System.Net;
using System.Threading.Tasks;

namespace HttpClientDemo1
{
    class Program
    {
        // HttpClient is intended to be instantiated once per application, rather than per-
use. See Remarks.
        static HttpClient client;

        /// <summary>
        /// Effettua il setup del client Http con l'eventuale proxy (se presente)
        /// Richiede:
```

```

/// 1) l'installazione del pacchetto Microsoft.Win32.Registry tramite NuGet
/// 2) using Microsoft.Win32;
/// 3) using System.Runtime.InteropServices;
/// 4) using System.Net;
/// </summary>
/// <param name="client"></param>
static void HttpClientProxySetup(out HttpClient client)
{
    Uri proxy;
    //https://docs.microsoft.com/en-
us/dotnet/api/system.net.http.httpClient.defaultproxy?view=net-6.0
    //https://medium.com/@sddkal/net-core-interaction-with-registry-4d7fcabc7a6b
    //https://www.shellhacks.com/windows-show-proxy-settings-cmd-powershell/
    //https://stackoverflow.com/a/63884955

    if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
    {
        //ottengo lo user specific proxy che si ottiene con il comando:
        //C:\> reg query
        "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings"
        //per poter utilizzare Registry occorre:
        //1) --> installare il pacchetto Microsoft.Win32.Registry tramite NuGet
        //2) --> using Microsoft.Win32;

        //leggiamo lo user specific proxy direttamente dal registro di sistema di
        Windows
        string userProxy =
Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
Settings").GetValue("ProxyServer") as string;
        //il proxy viene abilitato mediante il valore della chiave di registro
        ProxyEnable
        int? proxyEnable =
Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
Settings").GetValue("ProxyEnable") as int?;

        //impostiamo proxy
        proxy = proxyEnable > 0 ? new Uri(userProxy) : null;
    }

    else //se il sistema operativo è diverso da Windows procediamo con la
    determinazione del system wide proxy (se impostato)
    {
        //questa è la procedura per ottenere il system proxy
        Uri destinationUri = new Uri("https://www.google.it");
        //Ottiene il default proxy quando si prova a contattare la destinationUri
        //Se il proxy non è impostato si ottiene null
        //Uri proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
        //Con il proxy calcolato in automatico si crea l'handler da passare
        all'oggetto HttpClient e
        //funziona sia che il proxy sia configurato sia che non lo sia
        proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
    }

    //con il proxy ottenuto con il codice precedente
    HttpClientHandler httpHandler = new HttpClientHandler()
    {
        Proxy = new WebProxy(proxy, true),
        UseProxy = true,
        PreAuthenticate = false,
        UseDefaultCredentials = false,
    };
    client = new HttpClient(httpHandler);
}

```

```

        }
        static async Task Main()
        {
            //client = new HttpClient();
            HttpClientProxySetup(out client);
            // Call asynchronous network methods in a try/catch block to handle exceptions.
            try
            {
                HttpResponseMessage response = await
client.GetAsync("https://www.istitutogreppi.edu.it/");
                response.EnsureSuccessStatusCode();
                string responseBody = await response.Content.ReadAsStringAsync();
                // Above three lines can be replaced with new helper method below
                // string responseBody = await client.GetStringAsync(uri);

                Console.WriteLine(responseBody);
            }
            catch (HttpRequestException e)
            {
                Console.WriteLine("\nException Caught!");
                Console.WriteLine("Message :{0} ", e.Message);
            }
        }
    }
}

```

### 8.2.2.3. Come eseguire più richieste web in parallelo tramite `async` e `await` (C#)

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/>

The [Task asynchronous programming model \(TAP\)](#) provides an abstraction over asynchronous code. You write code as a sequence of statements, just like always. You can read that code as though each statement completes before the next begins. The compiler performs many transformations because some of those statements may start work and return a [Task](#) that represents the ongoing work.

```

using System;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Win32;
using System.Runtime.InteropServices;
using System.Net;

namespace MultipleWebRequestsDemo1
{
    class Program
    {
        /// <summary>
        /// Effettua il setup del client Http con l'eventuale proxy (se presente)
        /// Richiede:
        /// 1) l'installazione del pacchetto Microsoft.Win32.Registry tramite NuGet
        /// 2) using Microsoft.Win32;
        /// 3) using System.Runtime.InteropServices;
        /// 4) using System.Net;
        /// </summary>
        /// <param name="client"></param>
        static void HttpClientProxySetup(out HttpClient client)
        {

            Uri proxy;

```

```

//https://docs.microsoft.com/en-
us/dotnet/api/system.net.http.httpclient.defaultproxy?view=net-6.0
//https://medium.com/@sddkal/net-core-interaction-with-registry-4d7fcabc7a6b
//https://www.shellhacks.com/windows-show-proxy-settings-cmd-powershell/
//https://stackoverflow.com/a/63884955

if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
{
    //ottengo lo user specific proxy che si ottiene con il comando:
    //C:\> reg query
"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings"
    //per poter utilizzare Registry occorre:
    //1) --> installare il pacchetto Microsoft.Win32.Registry tramite NuGet
    //2) --> using Microsoft.Win32;

    //leggiamo lo user specific proxy direttamente dal registro di sistema di
Windows
    string userProxy =
Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
Settings").GetValue("ProxyServer") as string;
        //il proxy viene abilitato mediante il valore della chiave di registro
ProxyEnable
    int? proxyEnable =
Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
Settings").GetValue("ProxyEnable") as int?;

    //impostiamo proxy
    proxy = proxyEnable > 0 ? new Uri(userProxy) : null;

}

else //se il sistema operativo è diverso da Windows procediamo con la
determinazione del system wide proxy (se impostato)
{
    //questa è la procedura per ottenere il system proxy
    Uri destinationUri = new Uri("https://www.google.it");
    //Ottiene il default proxy quando si prova a contattare la destinationUri
    //Se il proxy non è impostato si ottiene null
    //Uri proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
    //Con il proxy calcolato in automatico si crea l'handler da passare
all'oggetto HttpClient e
    //funziona sia che il proxy sia configurato sia che non lo sia
    proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
}

//con il proxy ottenuto con il codice precedente
HttpClientHandler httpHandler = new HttpClientHandler()
{
    Proxy = new WebProxy(proxy, true),
    UseProxy = true,
    PreAuthenticate = false,
    UseDefaultCredentials = false,
};
client = new HttpClient(httpHandler);
}

/// <summary>
/// Scarica un file dalla rete e restituisce la lunghezza in byte
/// </summary>
/// <param name="url"></param>
/// <param name="client"></param>
/// <returns></returns>
static async Task<int> ProcessURLAsync(string url, HttpClient client)

```

```

{
    var byteArray = await client.GetByteArrayAsync(url);
    DisplayResults(url, byteArray);
    return byteArray.Length;
}
/// <summary>
/// Dati url e l'array di byte di una pagina stampa l'url e la dimensione in byte
/// </summary>
/// <param name="url"></param>
/// <param name="content"></param>
static void DisplayResults(string url, byte[] content)
{
    // Display the length of each website.
    var bytes = content.Length;
    // Strip off the "https://" or https://.
    var displayURL = url.Replace("https://", "").Replace("http://", "");
    Console.WriteLine($"{displayURL,-75} {bytes,10}");
}

/// <summary>
/// Crea più task per scaricare in parallelo più file dalla rete
/// </summary>
/// <returns></returns>
static async Task CreateMultipleTasksAsync()
{
    HttpClientProxySetup(out HttpClient client); //impostiamo il proxy - serve a
scuola

    //The maximum number of bytes to buffer when reading the response content.
    //The default value for this property is 2 gigabytes.
    //An app can set the MaxResponseContentBufferSize property to a lower value to
limit the size of the response to buffer when reading the response.
    //If the size of the response content is greater than the
MaxResponseContentBufferSize property, an exception is thrown.
    // https://docs.microsoft.com/en-
us/dotnet/api/system.net.http.httpclient.maxresponsecontentbuffersize?view=net-6.0
    //Qui impostiamo il Buffer per mostrare che si può fare, ma di solito non è
richiesto
    //Anzi se la dimensione della risposta è maggiore del valore da noi impostato
come MaxResponseContentBufferSize viene sollevata un'eccezione
    client.MaxResponseContentBufferSize = 10000000;

    // Create and start the tasks. As each task finishes, DisplayResults
    // displays its length.
    Task<int> download1 =
        ProcessURLAsync("https://docs.microsoft.com/en-us/welcome-to-docs", client);
    Task<int> download2 =
        ProcessURLAsync("https://docs.microsoft.com/en-us/dotnet/csharp/language-
reference/operators/await", client);
    Task<int> download3 =
        ProcessURLAsync("https://docs.microsoft.com/en-us/dotnet/csharp/language-
reference/keywords/async", client);

    // Await each task.
    int length1 = await download1;
    int length2 = await download2;
    int length3 = await download3;

    int total = length1 + length2 + length3;

    // Display the total count for the downloaded websites.
    Console.WriteLine($"Total bytes returned: {total}\r\n");
}

```

```

        static async Task Main(string[] args)
    {
        await CreateMultipleTasksAsync();
    }
}

```

Vediamo una versione ottimizzata del precedente codice che fa uso del costrutto WhenAll:

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/>

```

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Win32;
using System.Runtime.InteropServices;
using System.Net;

namespace MultipleWebRequestsDemo2
{
    class Program
    {
        /// <summary>
        /// Effettua il setup del client Http con l'eventuale proxy (se presente)
        /// Richiede:
        /// 1) l'installazione del pacchetto Microsoft.Win32.Registry tramite NuGet
        /// 2) using Microsoft.Win32;
        /// 3) using System.Runtime.InteropServices;
        /// 4) using System.Net;
        /// </summary>
        /// <param name="client"></param>
        static void HttpClientProxySetup(out HttpClient client)
        {

            Uri proxy;
            //https://docs.microsoft.com/en-
us/dotnet/api/system.net.http.httpclient.defaultproxy?view=net-6.0
            //https://medium.com/@sddkal/net-core-interaction-with-registry-4d7fcabc7a6b
            //https://www.shellhacks.com/windows-show-proxy-settings-cmd-powershell/
            //https://stackoverflow.com/a/63884955

            if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
            {
                //ottengo lo user specific proxy che si ottiene con il comando:
                //C:\> reg query
                "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings"
                    //per poter utilizzare Registry occorre:
                    //1) --> installare il pacchetto Microsoft.Win32.Registry tramite NuGet
                    //2) --> using Microsoft.Win32;

                //leggiamo lo user specific proxy direttamente dal registro di sistema di
                Windows
                string userProxy =
                    Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
                    Settings").GetValue("ProxyServer") as string;
                    //il proxy viene abilitato mediante il valore della chiave di registro
                    ProxyEnable
                    int? proxyEnable =
                        Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
                        Settings").GetValue("ProxyEnable") as int?;

```

```

        //impostiamo proxy
        proxy = proxyEnable > 0 ? new Uri(userProxy) : null;

    }

    else //se il sistema operativo è diverso da Windows procediamo con la
determinazione del system wide proxy (se impostato)
    {

        //questa è la procedura per ottenere il system proxy
        Uri destinationUri = new Uri("https://www.google.it");
        //Ottiene il default proxy quando si prova a contattare la destinationUri
        //Se il proxy non è impostato si ottiene null
        //Uri proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
        //Con il proxy calcolato in automatico si crea l'handler da passare
all'oggetto HttpClient e
        //funziona sia che il proxy sia configurato sia che non lo sia
        proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
    }

    //con il proxy ottenuto con il codice precedente
HttpClientHandler httpHandler = new HttpClientHandler()
{
    Proxy = new WebProxy(proxy, true),
    UseProxy = true,
    PreAuthenticate = false,
    UseDefaultCredentials = false,
};

client = new HttpClient(httpHandler);
}

/// <summary>
/// Scarica un file dalla rete e restituisce la lunghezza in byte
/// </summary>
/// <param name="url"></param>
/// <param name="client"></param>
/// <returns></returns>
static async Task<int> ProcessURLAsync(string url, HttpClient client)
{
    var sw = new Stopwatch();
    sw.Start();
    var byteArray = await client.GetByteArrayAsync(url);
    sw.Stop();
    DisplayResults(url, "https://docs.microsoft.com/en-us/",
byteArray, sw.ElapsedMilliseconds );
    return byteArray.Length;
}
/// <summary>
/// Stampa una parte dell'url, la dimensione in byte di una pagina e il tempo
impiegato per il download
/// </summary>
/// <param name="url"></param>
/// <param name="urlHeadingStrip"></param>
/// <param name="content"></param>
/// <param name="elapsedMillis"></param>
static void DisplayResults(string url, string urlHeadingStrip, byte[] content, long
elapsedMillis)
{
    // Display the length of each website.
    var bytes = content.Length;
    // Strip off the "urlHeadingStrip" part from url
    var displayURL = url.Replace(urlHeadingStrip, "");
}

```

```

        Console.WriteLine($"\"{displayURL},-80} bytes: {bytes,-10} ms: {elapsedMillis,-10}");
    }

    /// <summary>
    /// Restituisce una lista di url
    /// </summary>
    /// <returns></returns>
    static List<string> SetUpURLList()
    {
        List<string> urls = new List<string>
        {
            "https://docs.microsoft.com/en-us/welcome-to-docs",
            "https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/",
            "https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/",
            "https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/linq-to-objects",
            "https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/linq-and-strings",
            "https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/linq-to-xml-overview",
            "https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/async-return-types",
            "https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/linq-to-xml-vs-dom",
            "https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/reflection"
        };
        return urls;
    }

    /// <summary>
    /// Effettua il setup di una lista di url e per ognuno di essi avvia un download
    /// asincrono su un task separato
    /// </summary>
    /// <returns></returns>
    static async Task SumPageSizesAsync()
    {
        // Make a list of web addresses.
        List<string> urlList = SetUpURLList();
        //setup del client con eventuale Proxy
        HttpClientProxySetup(out HttpClient client);

        //// Create a query.
        //IEnumerable<Task<int>> downloadTasksQuery =
        //    from url in urlList select ProcessURLAsync(url, client);

        //misuriamo il tempo complessivo per scaricare tutte le pagine
        var swGlobal = new Stopwatch();
        swGlobal.Start();
        //processiamo in parallelo una lista di URL
        IEnumerable<Task<int>> downloadTasks = urlList.Select(u => ProcessURLAsync(u, client));

        ////altro modo per processare in parallelo più attività è il seguente:
        //var completedDownloads = new List<Task<int>>();
        //Parallel.For(0, urlList.Count, (index) =>
        //{
        //    completedDownloads.Add(ProcessURLAsync(urlList[index], client));
        //});
        //// You can do other work here before awaiting.
        //int[] lengthsParallel = await Task.WhenAll(completedDownloads);
        //int totalParallel = lengthsParallel.Sum();
    }
}

```

```
    ////altro modo per processare in parallelo più attività è il seguente:  
    //var completedDownloads2 = new List<Task<int>>();  
    //Parallel.ForEach(urlList, url =>  
    //{
//        completedDownloads2.Add(ProcessURLAsync(url, client));  
    });  
    //// You can do other work here before awaiting.  
    //int[] lengthsParallelForEach = await Task.WhenAll(completedDownloads2);  
    //int totalParallelForEach = lengthsParallelForEach.Sum();  
  
    // Await the completion of all the running tasks.  
    int[] lengths = await Task.WhenAll(downloadTasks);  
    //// The previous line is equivalent to the following two statements.  
    //Task<int[]> whenAllTask = Task.WhenAll(downloadTasks);  
    //int[] lengths = await whenAllTask;  
    swGlobal.Stop();  
    long elapsedTotalMs = swGlobal.ElapsedMilliseconds;  
    int total = lengths.Sum();  
  
    // Display the total count for all of the web addresses.  
    Console.WriteLine($"\\r\\n\\r\\nTotal bytes returned: {total}\\r\\n");  
    Console.WriteLine($"Tempo complessivo di scaricamento = {elapsedTotalMs}");  
  
}  
  
static async Task Main(string[] args)  
{  
    //imposto la dimensione della console - vale solo per Windows  
    if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))  
    {  
        Console.WindowWidth = 120;  
    }  
    await SumPageSizesAsync();  
}  
}
```

#### **8.2.2.4. Analizzare la risposta di una Get http mediante le properties della response**

```
using System;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Win32;
using System.Runtime.InteropServices;
using System.Net;

namespace HttpResponseMessageAnalysisDemo1
{
    class Program
    {
        /// <summary>
        /// Effettua il setup del client Http con l'eventuale proxy (se presente)
        /// Richiede:
        /// 1) l'installazione del pacchetto Microsoft.Win32.Registry tramite NuGet
        /// 2) using Microsoft.Win32;
        /// 3) using System.Runtime.InteropServices;
        /// 4) using System.Net;
        /// </summary>
```

```

/// <param name="client"></param>
static void HttpClientProxySetup(out HttpClient client)
{
    Uri proxy;
    //https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpClient.defaultproxy?view=net-6.0
    //https://medium.com/@sddkal/net-core-interaction-with-registry-4d7fcabc7a6b
    //https://www.shellhacks.com/windows-show-proxy-settings-cmd-powershell/
    //https://stackoverflow.com/a/63884955

    if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
    {
        //ottengo lo user specific proxy che si ottiene con il comando:
        //C:\> reg query
        "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings"
        //per poter utilizzare Registry occorre:
        //1) --> installare il pacchetto Microsoft.Win32.Registry tramite NuGet
        //2) --> using Microsoft.Win32;

        //leggiamo lo user specific proxy direttamente dal registro di sistema di
        Windows
        string userProxy =
Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
Settings").GetValue("ProxyServer") as string;
        //il proxy viene abilitato mediante il valore della chiave di registro
        ProxyEnable
        int? proxyEnable =
Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
Settings").GetValue("ProxyEnable") as int?;

        //impostiamo proxy
        proxy = proxyEnable > 0 ? new Uri(userProxy) : null;

    }
    else //se il sistema operativo è diverso da Windows procediamo con la
determinazione del system wide proxy (se impostato)
    {
        //questa è la procedura per ottenere il system proxy
        Uri destinationUri = new Uri("https://www.google.it");
        //Ottiene il default proxy quando si prova a contattare la destinationUri
        //Se il proxy non è impostato si ottiene null
        //Uri proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
        //Con il proxy calcolato in automatico si crea l'handler da passare
        all'oggetto HttpClient e
        //funziona sia che il proxy sia configurato sia che non lo sia
        proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
    }

    //con il proxy ottenuto con il codice precedente
    HttpClientHandler httpHandler = new HttpClientHandler()
    {
        Proxy = new WebProxy(proxy, true),
        UseProxy = true,
        PreAuthenticate = false,
        UseDefaultCredentials = false,
    };
    client = new HttpClient(httpHandler);
}

static async Task Main()
{

```

```

        // HttpClient is intended to be instantiated once per application, rather than
per-use. See Remarks.
        HttpClientProxySetup(out HttpClient client);

        // Call asynchronous network methods in a try/catch block to handle exceptions.
try
{
    //invio richiesta Get in modalità Async e ottengo la risposta
    HttpResponseMessage response = await
client.GetAsync("http://www.istitutogreppi.edu.it");
    //stampo lo status code
    Console.WriteLine($"status code = {response.StatusCode}");
    //stampo gli headers http della risposta
    Console.WriteLine("\nstampa gli headers http della risposta\n");
    foreach (var header in response.Headers)
    {
        Console.Write($"{header.Key} : ");
        foreach (var val in header.Value)
        {
            Console.Write($"{val} ");
        }
        Console.WriteLine();
    }
}

//stampo gli headers http della risposta usando il
response.Headers.ToString()
//Console.WriteLine("\nstampa di ToString() di response.Headers \n");
//Console.WriteLine( $" { response.Headers.ToString()}\n");
//stampo gli headers del content
Console.WriteLine("\nstampa degli headers del content della risposta\n");
foreach (var header in response.Content.Headers)
{
    Console.Write($"{header.Key} : ");
    foreach (var val in header.Value)
    {
        Console.Write($"{val} ");
    }
    Console.WriteLine();
}

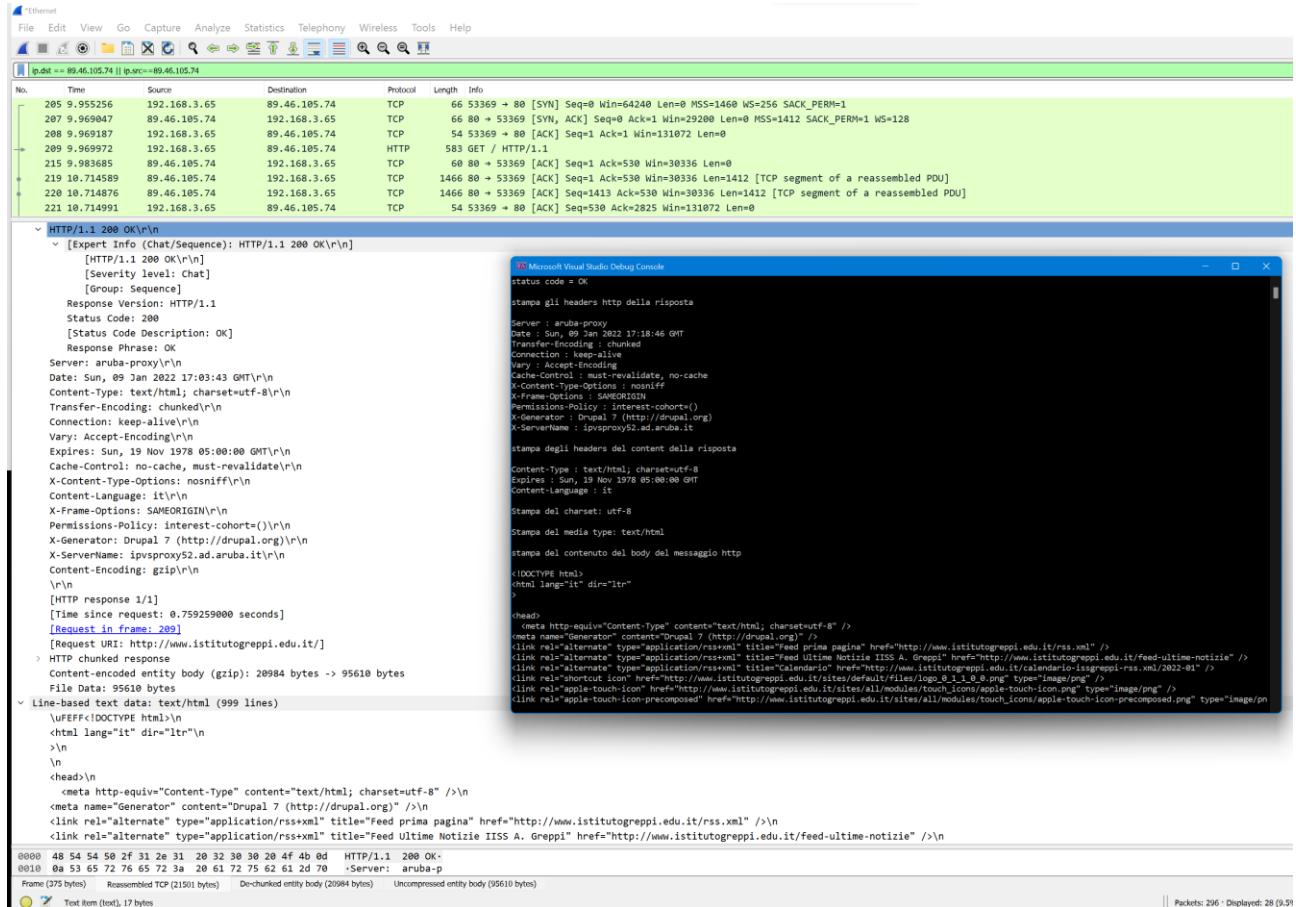
//ottenere il charset
Console.Write("\nStampa del charset: ");
Console.WriteLine(response.Content.Headers.ContentType.CharSet);
Console.Write("\nStampa del media type: ");
Console.WriteLine(response.Content.Headers.ContentType.MediaType);

response.EnsureSuccessStatusCode();
string responseBody = await response.Content.ReadAsStringAsync();
// Above three lines can be replaced with new helper method below
// string responseBody = await client.GetStringAsync(uri);
Console.WriteLine("\nstampa del contenuto del body del messaggio http\n");
Console.WriteLine(responseBody);
}

catch (HttpRequestException e)
{
    Console.WriteLine("\nException Caught!");
    Console.WriteLine("Message :{0} ", e.Message);
}
}
}

```

## Confronto tra quello stampato dal programma C# e quanto riportato da Wireshark



La precedente cattura è stata ottenuta attivando l'analizzatore di rete Wireshark mentre il programma demo in C# ha effettuato la richiesta http per lo scaricamento della pagina.

### 8.2.2.5. Esempi di utilizzo di HttpClient

<http://zetcode.com/csharp/httpclient/>

<http://zetcode.com/csharp/readwebpage/>

### 8.2.2.6. Scaricamento e salvataggio di una pagina web

Modifichiamo l'esempio precedente per salvare su file la pagina web scaricata.

Versione semplice, ma non ottimizzata:

```
using System;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Win32;
using System.Runtime.InteropServices;
using System.Net;

namespace HttpClientWebPageDownload
{
    class Program
    {
        /// <summary>
        /// Effettua il setup del client Http con l'eventuale proxy (se presente)
        /// Richiede:
        /// 1) l'installazione del pacchetto Microsoft.Win32.Registry tramite NuGet
        /// 2) using Microsoft.Win32;
```

```

/// 3) using System.Runtime.InteropServices;
/// 4) using System.Net;
/// </summary>
/// <param name="client"></param>
static void HttpClientProxySetup(out HttpClient client)
{
    Uri proxy;
    //https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpClient.defaultproxy?view=net-6.0
    //https://medium.com/@sddkal/net-core-interaction-with-registry-4d7fcabc7a6b
    //https://www.shellhacks.com/windows-show-proxy-settings-cmd-powershell/
    //https://stackoverflow.com/a/63884955

    if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
    {
        //ottengo lo user specific proxy che si ottiene con il comando:
        //C:\> reg query
        "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings"
            //per poter utilizzare Registry occorre:
            //1) --> installare il pacchetto Microsoft.Win32.Registry tramite NuGet
            //2) --> using Microsoft.Win32;

        //leggiamo lo user specific proxy direttamente dal registro di sistema di
        Windows
        string userProxy =
Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
Settings").GetValue("ProxyServer") as string;
            //il proxy viene abilitato mediante il valore della chiave di registro
ProxyEnable
        int? proxyEnable =
Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
Settings").GetValue("ProxyEnable") as int?;

        //impostiamo proxy
        proxy = proxyEnable > 0 ? new Uri(userProxy) : null;
    }

    else //se il sistema operativo è diverso da Windows procediamo con la
determinazione del system wide proxy (se impostato)
    {
        //questa è la procedura per ottenere il system proxy
        Uri destinationUri = new Uri("https://www.google.it");
        //Ottiene il default proxy quando si prova a contattare la destinationUri
        //Se il proxy non è impostato si ottiene null
        //Uri proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
        //Con il proxy calcolato in automatico si crea l'handler da passare
        all'oggetto HttpClient e
        //funziona sia che il proxy sia configurato sia che non lo sia
        proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
    }

    //con il proxy ottenuto con il codice precedente
    HttpClientHandler httpHandler = new HttpClientHandler()
    {
        Proxy = new WebProxy(proxy, true),
        UseProxy = true,
        PreAuthenticate = false,
        UseDefaultCredentials = false,
    };
    client = new HttpClient(httpHandler);
}
static async Task Main(string[] args)

```

```
{  
    HttpClientProxySetup(out HttpClient client);  
    try  
    {  
        string responseBody = await  
client.GetStringAsync("http://www.istitutogreppi.edu.it/");  
        //salvataggio su file  
        string path =  
Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory) +  
System.IO.Path.DirectorySeparatorChar + "pagina.html";  
        File.WriteAllText(path, responseBody);  
        Console.WriteLine("File scaricato, controlla la cartella Desktop");  
    }  
    catch (HttpRequestException e)  
    {  
        Console.WriteLine("\nException Caught!");  
        Console.WriteLine("Message :{0} ", e.Message);  
    }  
}  
}
```

Nell'esempio precedente la scrittura del file sul filesystem è sincrona e quindi bloccante. Vediamo la lettura e scrittura su file usando metodi asincroni.

### **8.2.2.7. Lettura e scrittura di file scaricati da Internet con metodi asincroni**

<https://docs.microsoft.com/it-it/dotnet/csharp/programming-guide/concepts/async/using-async-for-file-access>

Nel seguente esempio si mostra come scaricare un file da Internet e poi salvarlo sul file system. Il file viene poi aperto e letto a console. Le operazioni di scrittura e lettura del file sono gestite mediante chiamate asincrone.

```
using System;
using System.IO;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using Microsoft.Win32;
using System.Runtime.InteropServices;
using System.Net;

namespace FileDownloadAsyncDemo
{
    class Program
    {
        /// <summary>
        /// Effettua il setup del client Http con l'eventuale proxy (se presente)
        /// Richiede:
        /// 1) l'installazione del pacchetto Microsoft.Win32.Registry tramite NuGet
        /// 2) using Microsoft.Win32;
        /// 3) using System.Runtime.InteropServices;
        /// 4) using System.Net;
        /// </summary>
        /// <param name="client"></param>
        static void HttpClientProxySetup(out HttpClient client)
        {
            Uri proxy;
```

```

//https://docs.microsoft.com/en-
us/dotnet/api/system.net.http.httpclient.defaultproxy?view=net-6.0
//https://medium.com/@sddkal/net-core-interaction-with-registry-4d7fcabc7a6b
//https://www.shellhacks.com/windows-show-proxy-settings-cmd-powershell/
//https://stackoverflow.com/a/63884955

if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
{
    //ottengo lo user specific proxy che si ottiene con il comando:
    //C:\> reg query
"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings"
    //per poter utilizzare Registry occorre:
    //1) --> installare il pacchetto Microsoft.Win32.Registry tramite NuGet
    //2) --> using Microsoft.Win32;

    //leggiamo lo user specific proxy direttamente dal registro di sistema di
Windows
    string userProxy =
Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
Settings").GetValue("ProxyServer") as string;
        //il proxy viene abilitato mediante il valore della chiave di registro
ProxyEnable
    int? proxyEnable =
Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
Settings").GetValue("ProxyEnable") as int?;

    //impostiamo proxy
    proxy = proxyEnable > 0 ? new Uri(userProxy) : null;

}

else //se il sistema operativo è diverso da Windows procediamo con la
determinazione del system wide proxy (se impostato)
{
    //questa è la procedura per ottenere il system proxy
    Uri destinationUri = new Uri("https://www.google.it");
    //Ottiene il default proxy quando si prova a contattare la destinationUri
    //Se il proxy non è impostato si ottiene null
    //Uri proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
    //Con il proxy calcolato in automatico si crea l'handler da passare
all'oggetto HttpClient e
    //funziona sia che il proxy sia configurato sia che non lo sia
    proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
}

//con il proxy ottenuto con il codice precedente
HttpClientHandler httpHandler = new HttpClientHandler()
{
    Proxy = new WebProxy(proxy, true),
    UseProxy = true,
    PreAuthenticate = false,
    UseDefaultCredentials = false,
};
client = new HttpClient(httpHandler);
}

/// <summary>
/// Scrive del testo su un file usando la codifica specificata
/// https://stackoverflow.com/questions/11774827/writing-to-a-file-asynchronously/
/// https://stackoverflow.com/a/22617832
/// </summary>
/// <param name="filePath"></param>
/// <param name="text"></param>
/// <param name="encoding"></param>

```

```

    /// <returns></returns>
    static async Task WriteTextAsync(string filePath, string text, Encoding encoding,
int writeBufferSize = 4096)
{
    using FileStream sourceStream = new FileStream(filePath, FileMode.Create,
FileAccess.Write, FileShare.None,
        bufferSize: writeBufferSize, useAsync: true);
    using StreamWriter sw = new StreamWriter(sourceStream, encoding);
    await sw.WriteLineAsync(text);
}

/// <summary>
/// Legge un file di testo usando la codifica UTF8
/// https://stackoverflow.com/questions/11774827/writing-to-a-file-asynchronously/
/// https://stackoverflow.com/a/22617832
/// </summary>
/// <param name="filePath"></param>
/// <returns></returns>
static async Task<string> ReadTextUTF8Async(string filePath, int readBufferSize =
4096)
{
    using FileStream sourceStream = new FileStream(filePath, FileMode.Open,
FileAccess.Read, FileShare.Read,
        bufferSize: readBufferSize, useAsync: true);

    StringBuilder sb = new StringBuilder();

    byte[] buffer = new byte[0x1000];
    int numRead;
    while ((numRead = await sourceStream.ReadAsync(buffer, 0, buffer.Length)) !=
0)
    {
        string text = Encoding.UTF8.GetString(buffer, 0, numRead);
        sb.Append(text);
    }
    return sb.ToString();
}

/// <summary>
/// Legge un file di testo usando la codifica specificata
/// https://stackoverflow.com/questions/11774827/writing-to-a-file-asynchronously/
/// https://stackoverflow.com/a/22617832
/// </summary>
/// <param name="filePath"></param>
/// <param name="encoding"></param>
/// <returns></returns>
static async Task<string> ReadTextAsync(string filePath, Encoding encoding, int
readBufferSize = 4096)
{
    using FileStream sourceStream = new FileStream(filePath,
        FileMode.Open, FileAccess.Read, FileShare.Read,
        bufferSize: readBufferSize, useAsync: true);
    using StreamReader sr = new StreamReader(sourceStream, encoding);
    //https://docs.microsoft.com/it-it/dotnet/standard/io/how-to-read-text-from-a-file
    return await sr.ReadToEndAsync();
}

/// <summary>
/// Recupera il nome del file dall'url
/// https://stackoverflow.com/a/40361205
/// https://stackoverflow.com/questions/1105593/get-file-name-from-uri-string-in-csharp
/// </summary>
/// <param name="url"></param>
/// <returns></returns>

```

```

        static string GetFileNameFromUrl(string url)
    {
        Uri SomeBaseUri = new Uri("http://canbeanything");
        if (!Uri.TryCreate(url, UriKind.Absolute, out Uri uri))
        {
            uri = new Uri(SomeBaseUri, url);
        }
        //Path.GetFileName funziona se ha in input un URL assoluto
        return Path.GetFileName(uri.LocalPath);
    }

        static async Task Main(string[] args)
    {
        HttpClientProxySetup(out HttpClient client);
        try
        {
            //https://www.gutenberg.org/files/1012/1012-0.txt - La Divina Commedia in
            txt
            string fileName =
GetFileNameFromUrl("https://www.gutenberg.org/files/1012/1012-0.txt");
            HttpResponseMessage response = await
client.GetAsync("https://www.gutenberg.org/files/1012/1012-0.txt");
            response.EnsureSuccessStatusCode();
            string responseBody = await response.Content.ReadAsStringAsync();
            //salvataggio su file
            string path =
Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory) +
System.IO.Path.DirectorySeparatorChar + fileName;
            //File.WriteAllText(path, responseBody);
            // https://stackoverflow.com/questions/2223882/whats-the-difference-between-
utf-8-and-utf-8-without-bom
            //encoding senza BOM - https://docs.microsoft.com/en-
us/dotnet/api/system.text.encoding.utf8?view=netframework-4.8
            //BOM https://stackoverflow.com/a/2223926
            //BOM https://it.wikipedia.org/wiki/Byte_Order_Mark
            Console.WriteLine("Salvataggio su file in corso...");
            await WriteTextAsync(path, responseBody, new UTF8Encoding(false));
            Console.WriteLine("Lettura da file in corso... dei primi 10000
caratteri...tutto non entra nel buffer della console");
            string testoLettoDaFile = await ReadTextAsync(path, new
UTF8Encoding(false));
            //string testoLettoDaFile = await ReadTextUTF8Async(path);
            Console.WriteLine(testoLettoDaFile.Substring(0, 10000));
        }
        catch (HttpRequestException e)
        {
            Console.WriteLine("\nException Caught!");
            Console.WriteLine("Message :{0} ", e.Message);
        }
    }
}

```

### 8.2.2.7.1. Riportare lo stato di avanzamento del download di un file

Per approfondimenti su come riportate lo stato di avanzamento del download vedere anche  
<https://stackoverflow.com/questions/20661652/progress-bar-with-httpclient>

```
HttpClient client = //...
```

```
// Must use ResponseHeadersRead to avoid buffering of the content
```

```

using (var response = await client.GetAsync(uri, HttpCompletionOption.ResponseHeadersRead)){
    // You must use as stream to have control over buffering and number of bytes read/received
    using (var stream = await response.Content.ReadAsStreamAsync())
    {
        // Read/process bytes from stream as appropriate

        // Calculated by you based on how many bytes you have read. Likely incremented within a loop.
        long bytesReceived = ...;

        long? totalBytes = response.Content.Headers.ContentLength;
        double? percentComplete = (double)bytesReceived / totalBytes;

        // Do what you want with `percentComplete`
    }
}

```

### 8.2.2.7.2. Gestione di file di grandi dimensioni

If an app using [HttpClient](#) and related classes in the [System.Net.Http](#) namespace intends to download large amounts of data (50 megabytes or more), then the app should stream those downloads and not use the default buffering. If the default buffering is used the client memory usage will get very large, potentially resulting in substantially reduced performance.

Ad esempio per effettuare un download di un file molto grande si può utilizzare l'approccio mostrato qui: <http://www.tugberkugurlu.com/archive/efficiently-streaming-large-http-responses-with-httpclient>

```

static async Task HttpGetForLargeFileInRightWay()
{
    using (HttpClient client = new HttpClient())
    {
        const string url = "https://github.com/tugberkugurlu/ASPNETWebAPISamples/archive/master.zip";
        using (HttpResponseMessage response = await client.GetAsync(url, HttpCompletionOption.ResponseHeadersRead))
        using (Stream streamToReadFrom = await response.Content.ReadAsStreamAsync())
        {
            string fileToWriteTo = Path.GetTempFileName();
            using (Stream streamToWriteTo = File.Open(fileToWriteTo, FileMode.Create))
            {
                await streamToReadFrom.CopyToAsync(streamToWriteTo);
            }
        }
    }
}

```

Notice that we are calling [another overload of the GetAsync method](#) by passing the [HttpCompletionOption](#) enumeration value as [ResponseHeadersRead](#). This switch tells the [HttpClient](#) not to buffer the response. In other words, it will just read the headers and return the control back. This means that the [HttpContent](#) is not ready at the time when you get the control back. Afterwards, we are getting the stream and calling the [CopyToAsync](#) method on it by passing our [FileStream](#). The result is much better

### 8.2.2.7.3. Un esempio di scaricamento di file di grandi dimensioni

Nel seguente esempio viene scaricato un file binario di dimensione superiore a 100 MB.

L'url dell'esempio corrisponde a quello del file della .NET Core SDK 3.1 delle dimensioni di circa 120MB.

Il metodo `WriteBinaryAsync` salva il file nel percorso specificato, mentre il metodo `WriteBinaryAsyncWithProgress` fa la stessa cosa, ma permette anche di stampare la percentuale di progresso di download.

```
using System;
using System.Diagnostics;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Win32;
using System.Runtime.InteropServices;
using System.Net;

namespace BinaryBigFileDownloadAsyncDemo
{

    class Program
    {
        /// <summary>
        /// Effettua il setup del client Http con l'eventuale proxy (se presente)
        /// Richiede:
        /// 1) l'installazione del pacchetto Microsoft.Win32.Registry tramite NuGet
        /// 2) using Microsoft.Win32;
        /// 3) using System.Runtime.InteropServices;
        /// 4) using System.Net;
        /// </summary>
        /// <param name="client"></param>
        static void HttpClientProxySetup(out HttpClient client)
        {

            Uri proxy;
            //https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.defaultproxy?view=net-6.0
            //https://medium.com/@sddkal/net-core-interaction-with-registry-4d7fcabc7a6b
            //https://www.shellhacks.com/windows-show-proxy-settings-cmd-powershell/
            //https://stackoverflow.com/a/63884955

            if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
            {
                //ottengo lo user specific proxy che si ottiene con il comando:
                //C:\> reg query
                "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings"
                    //per poter utilizzare Registry occorre:
                    //1) --> installare il pacchetto Microsoft.Win32.Registry tramite NuGet
                    //2) --> using Microsoft.Win32;

                //leggiamo lo user specific proxy direttamente dal registro di sistema di Windows
                string userProxy =
                    Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet Settings").GetValue("ProxyServer") as string;
                    //il proxy viene abilitato mediante il valore della chiave di registro ProxyEnable
                    int? proxyEnable =
                    Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet Settings").GetValue("ProxyEnable") as int?;
```

```

        //impostiamo proxy
        proxy = proxyEnable > 0 ? new Uri(userProxy) : null;

    }

    else //se il sistema operativo è diverso da Windows procediamo con la
determinazione del system wide proxy (se impostato)
{
    //questa è la procedura per ottenere il system proxy
    Uri destinationUri = new Uri("https://www.google.it");
    //Ottiene il default proxy quando si prova a contattare la destinationUri
    //Se il proxy non è impostato si ottiene null
    //Uri proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
    //Con il proxy calcolato in automatico si crea l'handler da passare
all'oggetto HttpClient e
        //funziona sia che il proxy sia configurato sia che non lo sia
        proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
}

//con il proxy ottenuto con il codice precedente
HttpClientHandler httpHandler = new HttpClientHandler()
{
    Proxy = new WebProxy(proxy, true),
    UseProxy = true,
    PreAuthenticate = false,
    UseDefaultCredentials = false,
};
client = new HttpClient(httpHandler);
}

static long bytesReceived = 0;
static long? totalBytes;
static int left;
static int top;
/// <summary>
/// Scrive un file nel percorso di destinazione a partire da uno stream di input
/// </summary>
/// <param name="filePathDestination"></param>
/// <param name="inputStream"></param>
/// <returns></returns>
static async Task WriteBinaryAsync(string filePathDestination, Stream inputStream,
int writeBufferSize = 4096)
{
    using FileStream outputStream = new FileStream(filePathDestination,
        FileMode.Create, FileAccess.Write, FileShare.None,
        bufferSize: writeBufferSize, useAsync: true);

    await inputStream.CopyToAsync(outputStream);
}

/// <summary>
/// Scrive un file nel percorso destinazione a partire da uno stream di input.
/// Questa versione stampa a console la percentuale di progress.
/// Utilizza le variabili totalBytes e bytesReceived
/// </summary>
/// <param name="filePathDestination"></param>
/// <param name="inputStream"></param>
/// <returns></returns>
static async Task WriteBinaryAsyncWithProgress(string filePathDestination, Stream
inputStream)
{
    if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))

```

```

{
    Console.CursorVisible = false;
}
using FileStream outputStream = new FileStream(filePathDestination,
 FileMode.Create, FileAccess.Write, FileShare.None,
 bufferSize: 4096, useAsync: true);

byte[] buffer = new byte[0x100000];//circa 1MB di buffer
int numRead;
while ((numRead = await inputStream.ReadAsync(buffer, 0, buffer.Length)) != 0)
{
    await outputStream.WriteAsync(buffer, 0, numRead);
    bytesRecieved += numRead;
    double? percentComplete = (double)bytesRecieved / totalBytes;
    Console.SetCursorPosition(left, top);
    Console.WriteLine($"download al {percentComplete * 100:F2}%");
}

if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
{
    Console.CursorVisible = true;
}
}

/// <summary>
/// Recupera il nome del file dall'url
/// https://stackoverflow.com/a/40361205
/// https://stackoverflow.com/questions/1105593/get-file-name-from-uri-string-in-c-
sharp
/// </summary>
/// <param name="url"></param>
/// <returns></returns>
static string GetFileNameFromUrl(string url)
{
    Uri SomeBaseUri = new Uri("http://canbeanything");
    if (!Uri.TryCreate(url, UriKind.Absolute, out Uri uri))
        uri = new Uri(SomeBaseUri, url);
    //Path.GetFileName funziona se ha in input un URL assoluto
    return Path.GetFileName(uri.LocalPath);
}
static async Task Main(string[] args)
{
    //const string url =
"https://github.com/tugberkugurlu/ASPNETWebAPISamples/archive/master.zip";
    const string url =
"https://download.visualstudio.microsoft.com/download/pr/639f7cfa-84f8-48e8-b6c9-
82634314e28f/8eb04e1b5f34df0c840c1bffa363c101/dotnet-sdk-3.1.100-win-x64.exe";
    HttpClientProxySetup(out HttpClient client);

    try
    {

        //leggo solo l'header HTTP, il resto verrà scaricato successivamente in
        maniera asincrona
        HttpResponseMessage response = await client.GetAsync(url,
HttpCompletionOption.ResponseHeadersRead);
        response.EnsureSuccessStatusCode();
        var sw = new Stopwatch();
        sw.Start();
        Console.WriteLine("Salvataggio su file in corso...");
        //ottengo il nome del file dall'url
        string fileName = GetFileNameFromUrl(url);
        //definisco il path complessivo del file
    }
}

```

```
        string path =
Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory) +
System.IO.Path.DirectorySeparatorChar + fileName;
        using (Stream streamToReadFrom = await response.Content.ReadAsStreamAsync())
{
    //copio in modalità async il file
    //await WriteBinaryAsync(path, streamToReadFrom);
    totalBytes = response.Content.Headers.ContentLength;
    left = Console.CursorLeft;
    top = Console.CursorTop;
    await WriteBinaryAsyncWithProgress(path, streamToReadFrom);
}
long elapsedMs = sw.ElapsedMilliseconds;
Console.WriteLine($"Salvataggio terminato...in {elapsedMs} ms");
}
catch (HttpRequestException e)
{
    Console.WriteLine("\nException Caught!");
    Console.WriteLine("Message :{0} ", e.Message);
}

}
}
```

### 8.2.3. Extension Methods e librerie

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods>

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/how-to-implement-and-call-a-custom-extension-method>

Extension methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type. Extension methods are static methods, but they're called as if they were instance methods on the extended type. For client code written in C#, F# and Visual Basic, there's no apparent difference between calling an extension method and the methods defined in a type.

The most common extension methods are the LINQ standard query operators that add query functionality to `System.Collections.IEnumerable` and `System.Collections.Generic.IEnumerable<T>` types.

To use the standard query operators, first bring them into scope with a using System.Linq directive. Then any type that implements `IEnumerable<T>` appears to have instance methods such as `GroupBy`, `OrderBy`, `Average`, and so on. You can see these additional methods in IntelliSense statement completion when you type "dot" after an instance of an `IEnumerable<T>` type such as `List<T>` or `Array`.

### **8.2.3.1. How to write custom Extension Methods?**

Client code can use your extension methods by adding a reference to the DLL that contains them, and adding a [using](#) directive that specifies the namespace in which the extension methods are defined.

#### **8.2.3.1.1. To define and call the extension method**

1. Define a static `class` to contain the extension method.

The class must be visible to client code. For more information about accessibility rules, see [Access Modifiers](#).

2. Implement the extension method as a static method with at least the same visibility as the containing class.
3. The first parameter of the method specifies the type that the method operates on; it must be preceded with the `this` modifier.
4. In the calling code, add a using directive to specify the [namespace](#) that contains the extension method class.
5. Call the methods as if they were instance methods on the type.

Note that the first parameter is not specified by calling code because it represents the type on which the operator is being applied, and the compiler already knows the type of your object. You only have to provide arguments for parameters 2 through n.

#### **8.2.3.1.1. Esempio di Extension Method**

The following example implements an extension method named WordCount in the CustomExtensions.StringExtension class. The method operates on the `String` class, which is specified as the first method parameter. The CustomExtensions namespace is imported into the application namespace, and the method is called inside the Main method.

```
using System;
namespace CustomExtensions
{
    // Extension methods must be defined in a static class.
    public static class StringExtension
    {
        // This is the extension method.
        // The first parameter takes the "this" modifier
        // and specifies the type for which the method is defined.
        public static int WordCount(this String str)
        {
            return str.Split(new char[] { ' ', '.', '?' },
StringSplitOptions.RemoveEmptyEntries).Length;
        }
    }
}
namespace ExtensionMethodsDemo
{
    // Import the extension method namespace.
    using CustomExtensions;
    class Program
    {

        static void Main(string[] args)
        {
            string s = "The quick brown fox jumped over the lazy dog.";
            // Call the method as if it were an
            // instance method on the type. Note that the first
            // parameter is not specified by the calling code.

            // Il metodo WordCount non esiste nella classe String, ma è stato aggiunto come
metodo di estensione
            int i = s.WordCount();
            Console.WriteLine("Word count of s is {0}", i);
        }
    }
}
```

#### **8.2.3.1.2. Creazione di una libreria con metodi di estensione per**

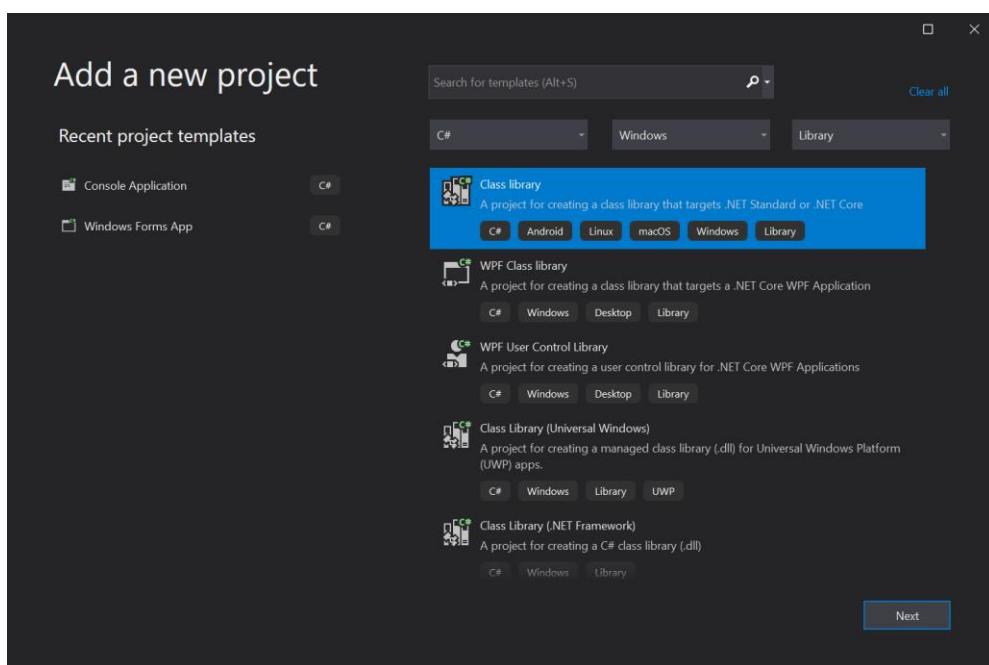
## ***la gestione del grado di parallelismo dei Task che elaborano una collection di dati***

Quando si invoca il metodo Task.WhenAll(taskCollection) tutti i task della taskCollection vengono avviati contemporaneamente. Come avviene nell'esempio del paragrafo 8.2.2.6 nel quale si scarica in contemporanea una lista di URL con il codice:

```
IEnumerable<Task<int>> downloadTasks = urlList.Select(u => ProcessURLAsync(u, client));  
// Await the completion of all the running tasks.  
int[] lengths = await Task.WhenAll(downloadTasks);
```

Per gestire il grado di parallelismo massimo dei task che sono eseguiti in parallelo su una collection di oggetti è possibile implementare un paio di semplici metodi che utilizzano una coda Thread-safe per processare gli elementi della collection. Svilupperemo questo progetto come libreria con metodi di estensione per poter poi includere la libreria nei nostri progetti.

### **8.2.3.1.1.2.1. Step N1: creazione di un progetto di tipo libreria in Visual Studio**



Creiamo un progetto chiamato TaskParallelismControl e al suo interno creiamo la classe TaskConcurrencyHelper così definita:

```
using System;  
using System.Collections.Concurrent;  
using System.Collections.Generic;  
using System.Linq;  
using System.Threading.Tasks;  
  
namespace TaskParallelismControl  
{  
    //Questa classe implementa metodi di estensione:  
    //https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-  
    //structs/how-to-implement-and-call-a-custom-extension-method  
    //https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-  
    //structs/extension-methods  
  
    public static class TaskConcurrencyHelper  
    {  
        //esempio preso da  
        //https://medium.com/@nirinchev/executing-a-collection-of-tasks-in-parallel-with-  
        //control-over-the-degree-of-parallelism-in-c-508d59ddfffc6  
        //con alcuni adattamenti
```

```

    /// <summary>
    /// Permette di eseguire in parallelo un metodo che restituisce un Task su una
    collection, specificando il numero massimo di task da impiegare
    /// </summary>
    /// <typeparam name="T">Tipo degli elementi della collection</typeparam>
    /// <param name="collection">Collection da processare</param>
    /// <param name="processor">metodo da eseguire su ciascun elemento della
    collection</param>
    /// <param name="degreeOfParallelism">Grado di parallelismo</param>
    /// <returns>Restituisce un task che termina quando tutti i task che processano la
    collection hanno terminato</returns>
    public static async Task ExecuteInParallel<T>(this IEnumerable<T> collection,
                                                Func<T, Task> processor,
                                                int degreeOfParallelism)
    {
        //creo una coda Thrade-safe a partire dalla collection
        var queue = new ConcurrentQueue<T>(collection);
        //creo tanti task (esecutori) quanto è il grado di parallelismo richiesto
        var tasks = Enumerable.Range(0, degreeOfParallelism).Select(async _ =>
        {
            //ogni task cerca di svuotare la coda, prelevando un elemento e
processandolo
            while (queue.TryDequeue(out T item))
            {
                await processor(item);
            }
        });
        //attendo che tutti i task finiscano
        await Task.WhenAll(tasks);
    }

    public static async Task<ConcurrentBag<TResult>> ExecuteInParallel<T, TResult>(this
    IEnumerable<T> collection,
                                                Func<T, Task<TResult>> processor,
                                                int degreeOfParallelism)
    {
        //creo una coda Thrade-safe a partire dalla collection
        var queue = new ConcurrentQueue<T>(collection);
        //creo una collection Thrade-safe dove inserire i risultati dell'elaborazione
dei Task
        var results = new ConcurrentBag<TResult>();

        var tasks = Enumerable.Range(0, degreeOfParallelism).Select(async _ =>
        {
            //ogni task cerca di svuotare la coda, prelevando un elemento e
processandolo
            while (queue.TryDequeue(out T item))
            {
                results.Add(await processor(item));
            }
        });
        //attendo che tutti i task finiscano
        await Task.WhenAll(tasks);
        return results;
    }
}
}

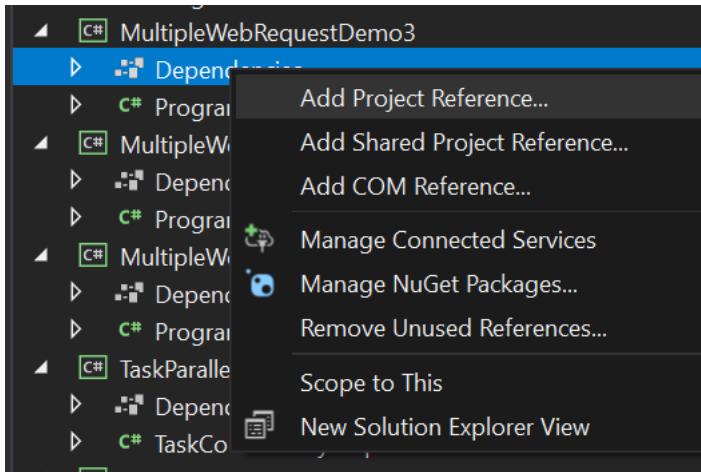
```

Compiliamo la libreria con il comando Build.

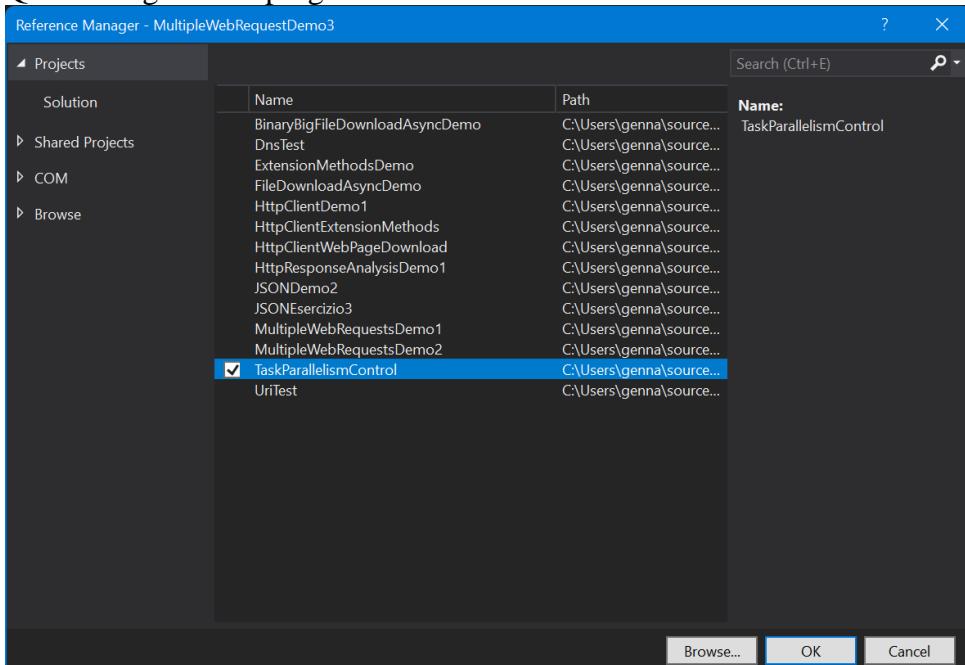
### 8.2.3.1.1.2.2. Step N2: utilizzo della libreria in un altro progetto di

## Visual Studio

Creiamo il progetto `MultipleWebRequestDemo3` che utilizza la libreria `TaskParallelismControl`. Dopo aver creato un progetto Console chiamato `MultipleWebRequestDemo3`, aggiungiamo la dipendenza dal progetto libreria appena creato:



Quindi scegliamo il progetto `TaskParallelismControl`.



Nel file della classe `Program` aggiungiamo il namespace della libreria `TaskParallelismControl` e scriviamo il codice della classe:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Win32;
using System.Runtime.InteropServices;
using System.Net;
using System.Collections.Concurrent;

//namespace della libreria creata
using TaskParallelismControl;
```

```

namespace MultipleWebRequestsDemo3
{
    class Program
    {
        /// <summary>
        /// Effettua il setup del client Http con l'eventuale proxy (se presente)
        /// Richiede:
        /// 1) l'installazione del pacchetto Microsoft.Win32.Registry tramite NuGet
        /// 2) using Microsoft.Win32;
        /// 3) using System.Runtime.InteropServices;
        /// 4) using System.Net;
        /// </summary>
        /// <param name="client"></param>
        static void HttpClientProxySetup(out HttpClient client)
        {

            Uri proxy;
            //https://docs.microsoft.com/en-
us/dotnet/api/system.net.http.httpclient.defaultproxy?view=net-6.0
            //https://medium.com/@sddkal/net-core-interaction-with-registry-4d7fcabc7a6b
            //https://www.shellhacks.com/windows-show-proxy-settings-cmd-powershell/
            //https://stackoverflow.com/a/63884955

            if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
            {
                //ottengo lo user specific proxy che si ottiene con il comando:
                //C:\> reg query
                "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings"
                //per poter utilizzare Registry occorre:
                //1) --> installare il pacchetto Microsoft.Win32.Registry tramite NuGet
                //2) --> using Microsoft.Win32;

                //leggiamo lo user specific proxy direttamente dal registro di sistema di
                Windows
                string userProxy =
                    Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
                    Settings").GetValue("ProxyServer") as string;
                    //il proxy viene abilitato mediante il valore della chiave di registro
                    ProxyEnable
                    int? proxyEnable =
                    Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
                    Settings").GetValue("ProxyEnable") as int?;

                    //impostiamo proxy
                    proxy = proxyEnable > 0 ? new Uri(userProxy) : null;

            }
            else //se il sistema operativo è diverso da Windows procediamo con la
            determinazione del system wide proxy (se impostato)
            {
                //questa è la procedura per ottenere il system proxy
                Uri destinationUri = new Uri("https://www.google.it");
                //Ottiene il default proxy quando si prova a contattare la destinationUri
                //Se il proxy non è impostato si ottiene null
                //Uri proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
                //Con il proxy calcolato in automatico si crea l'handler da passare
                all'oggetto HttpClient e
                //funziona sia che il proxy sia configurato sia che non lo sia
                proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
            }

            //con il proxy ottenuto con il codice precedente
            HttpClientHandler httpHandler = new HttpClientHandler()

```

```

{
    Proxy = new WebProxy(proxy, true),
    UseProxy = true,
    PreAuthenticate = false,
    UseDefaultCredentials = false,
};

client = new HttpClient(httpHandler);
}

/// <summary>
/// Scarica un file dalla rete e restituisce la lunghezza in byte
/// </summary>
/// <param name="url"></param>
/// <param name="client"></param>
/// <returns></returns>
static async Task<int> ProcessURLAsync(string url, HttpClient client)
{
    var sw = new Stopwatch();
    sw.Start();
    var byteArray = await client.GetByteArrayAsync(url);
    sw.Stop();
    DisplayResults(url, "https://docs.microsoft.com/en-us/", byteArray,
    sw.ElapsedMilliseconds);
    return byteArray.Length;
}
/// <summary>
/// Stampa una parte dell'url, la dimensione in byte di una pagina e il tempo
impiegato per il download
/// </summary>
/// <param name="url"></param>
/// <param name="urlHeadingStrip"></param>
/// <param name="content"></param>
/// <param name="elapsedMillis"></param>
static void DisplayResults(string url, string urlHeadingStrip, byte[] content, long
elapsedMillis)
{
    // Display the length of each website.
    var bytes = content.Length;
    // Strip off the "urlHeadingStrip" part from url
    var displayURL = url.Replace(urlHeadingStrip, "");

    Console.WriteLine($"\\n{displayURL,-80} bytes: {bytes,-10} ms: {elapsedMillis,-
10}");
}

/// <summary>
/// Restituisce una lista di url
/// </summary>
/// <returns></returns>
static List<string> SetUpURLList()
{
    List<string> urls = new List<string>
    {
        "https://docs.microsoft.com/en-us/welcome-to-docs",
        "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/async/",
        "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/linq/",
        "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/linq/linq-to-objects",
        "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/linq/linq-and-strings",
    };
}

```

```

        "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/linq/linq-to-xml-overview",
        "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/async/async-return-types",
        "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/linq/linq-to-xml-vs-dom",
        "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/reflection"
    };
    return urls;
}
/// <summary>
/// Effettua il setup di una lista di url e per ognuno di essi avvia un download
asincrono su un task separato
/// </summary>
/// <returns></returns>
static async Task SumPageSizesAsync()
{
    // Make a list of web addresses.
    List<string> urlList = SetUpURLList();
    //setup del client con eventuale Proxy
    HttpClientProxySetup(out HttpClient client);

    //misuriamo il tempo complessivo per scaricare tutte le pagine
    var swGlobal = new Stopwatch();
    swGlobal.Start();
    //processiamo in parallelo una lista di URL
    //Ienumerable<Task<int>> downloadTasks = urlList.Select(u => ProcessURLAsync(u,
client));
    //ConcurrentBag<int> bag = new ConcurrentBag<int>();
    //await urlList.ExecuteInParallel(async u => { bag.Add(await ProcessURLAsync(u,
client)); }, 10);
    //var theTotal = bag.ToArray().Sum();
    //await urlList.ExecuteInParallel(async u => { await Task.Delay(10); }, 10);

    //definiamo il grado di parallelismo
    const int numberOfParallelThreads = 5;
    //processiamo tutti gli oggetti della collection con il grado di parallelismo
    massimo predefinito
    ConcurrentBag<int> concurrentBagOfResults = await urlList.ExecuteInParallel(u =>
ProcessURLAsync(u, client), numberOfParallelThreads);
    //sommiamo tutti i valori restituiti dai thread
    var theTotal = concurrentBagOfResults.ToArray().Sum();
    Console.WriteLine($"Somma = {theTotal}");
    swGlobal.Stop();
    long elapsedTotalMs = swGlobal.ElapsedMilliseconds;

    // Display the total count for all of the web addresses.
    Console.WriteLine($"\\r\\n\\r\\nTotal bytes returned: {theTotal}\\r\\n");
    Console.WriteLine($"Tempo complessivo di scaricamento = {elapsedTotalMs}");

}

static async Task Main(string[] args)
{
    //imposto la dimensione della console - vale solo per Windows
    if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
    {
        Console.WindowWidth = 120;
    }
    await SumPageSizesAsync();
}
}

```

### **8.2.3.1.1.3. Libreria per la gestione del Proxy con l'oggetto HttpClient**

Il codice per il setup del proxy e le relative dipendenze può utilmente essere inserito all'interno di una libreria richiamata in tutti i progetti che ne fanno uso.

Creiamo la libreria HttpProxyControl da usare per i futuri progetti.

```
using Microsoft.Win32;
using System;
using System.Net;
using System.Net.Http;
using System.Runtime.InteropServices;
using System.Linq;
using System.Collections.Generic;

namespace HttpProxyControl
{
    public struct ProxyParams
    {
        public string ProxyAddress { get; set; }
        public int ProxyPort { get; set; }
    }

    public class ProxyHelperException : Exception
    {
        //https://docs.microsoft.com/en-us/dotnet/standard/exceptions/how-to-create-user-defined-exceptions
        public ProxyHelperException()
        {
        }

        public ProxyHelperException(string message)
            : base(message)
        {
        }

        public ProxyHelperException(string message, Exception inner)
            : base(message, inner)
        {
        }
    }

    /// Richiede:
    /// 1) l'installazione del pacchetto Microsoft.Win32.Registry tramite NuGet
    /// 2) using Microsoft.Win32;
    /// 3) using System.Runtime.InteropServices;
    /// 4) using System.Net;
    public static class HttpProxyHelper
    {
        /// <summary>
        ///
        /// Restituisce il proxy attualmente in uso (se presente)
        /// Il proxy è un Uri nella forma proxy_address:proxy_port
        /// </summary>
        /// <returns>Il proxy attualmente in uso. Restituisce null se nessun proxy è in uso</returns>
        public static Uri GetHttpClientProxy()
        {
            Uri proxy;
            //https://docs.microsoft.com/en-us/dotnet/api/system.net.http.defaultproxy?view=net-6.0
            //https://medium.com/@sddkal/net-core-interaction-with-registry-4d7fcabc7a6b
            //https://www.shellhacks.com/windows-show-proxy-settings-cmd-powershell/
            //https://stackoverflow.com/a/63884955
```

```

if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
{
    //ottengo lo user specific proxy che si ottiene con il comando:
    //C:\> reg query
"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings"
    //per poter utilizzare Registry occorre:
    //1) --> installare il pacchetto Microsoft.Win32.Registry tramite NuGet
    //2) --> using Microsoft.Win32;

    //leggiamo lo user specific proxy direttamente dal registro di sistema di Windows
    string userProxy =
Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
Settings").GetValue("ProxyServer") as string;
    //il proxy viene abilitato mediante il valore della chiave di registro
ProxyEnable
    int? proxyEnable =
Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
Settings").GetValue("ProxyEnable") as int?;

    //impostiamo proxy
    proxy = proxyEnable > 0 ? new Uri(userProxy) : null;

}
else //se il sistema operativo è diverso da Windows procediamo con la determinazione
del system wide proxy (se impostato)
{
    //questa è la procedura per ottenere il system proxy
    Uri destinationUri = new Uri("https://www.google.it");
    //Ottiene il default proxy quando si prova a contattare la destinationUri
    //Se il proxy non è impostato si ottiene null
    //Uri proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
    //Con il proxy calcolato in automatico si crea l'handler da passare all'oggetto
HttpClient e
    //funziona sia che il proxy sia configurato sia che non lo sia
    proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
}
return proxy;
}
/// <summary>
/// Effettua il setup del client Http con l'eventuale proxy (se impostato)
/// </summary>
/// <param name="client"></param>
public static void HttpClientProxySetup(out HttpClient client)
{
    Uri proxy = GetHttpClientProxy();
    if (proxy != null)
    {
        HttpClientHandler httpHandler = new HttpClientHandler()
        {
            Proxy = new WebProxy(proxy, true),
            UseProxy = true,
            PreAuthenticate = false,
            UseDefaultCredentials = false,
        };
        client = new HttpClient(httpHandler);
    }
    else
    {
        client = new HttpClient();
    }
}

/// <summary>
/// Restituisce i parametri del proxy attualmente in uso, altrimenti null
/// </summary>

```

```
/// <returns></returns>
public static ProxyParams? GetHttpClientProxyParams()
{
    Uri proxy = GetHttpClientProxy();
    if (proxy != null)
    {
        string proxyString = proxy.ToString();
        //rimuovo eventuale slash finale
        int lastSlash = proxyString.LastIndexOf('/');
        proxyString = (lastSlash > -1) ? proxyString.Substring(0, lastSlash) :
proxyString;
        //rimuovo eventuale http:// oppure https:// iniziale
        List<string> protocolSchemas = new List<string>() { "http://", "https://" };
        protocolSchemas.ForEach(_ =>
        {
            if (proxyString.StartsWith(_))
            {
                proxyString = proxyString.Substring(_.Length);
            }
        });
        //individuo la posizione del :
        int positionOfColons = proxyString.LastIndexOf(":");
        string proxyAddress = (positionOfColons != -1) ? proxyString.Substring(0,
positionOfColons) : proxyString;
        //estraggo il numero di porta proxyPort
        if (int.TryParse(proxyString.Substring(positionOfColons + 1), out int proxyPort))
        {
            return new ProxyParams() { ProxyAddress = proxyAddress, ProxyPort = proxyPort
};
        }
        else
        {
            //se non trovo il proxyPort c'è stato un errore nella ricerca del proxy
            throw new ProxyHelperException("Could not retrieve proxy port");
        }
    }
    else
    {
        return null;
    }
}
```

Usiamo la libreria `HttpProxyControl` per una versione aggiornata del progetto precedente che, oltre ad utilizzare la libreria `TaskParallelismControl` utilizza anche la libreria `HttpProxyControl`.

Il programma seguente `MultipleWebRequestsDemo4` scarica in parallelo le pagine di più URL, gestendo il proxy http e fissando il massimo grado di parallelismo dei Task, mediante l'utilizzo di due librerie specifiche.

```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;
using System.Runtime.InteropServices;

//using di librerie proprie
using TaskParallelismControl;
using HttpProxyControl;
```

```

namespace MultipleWebRequestsDemo4
{
    class Program
    {

        /// <summary>
        /// Scarica un file dalla rete e restituisce la lunghezza in byte
        /// </summary>
        /// <param name="url"></param>
        /// <param name="client"></param>
        /// <returns></returns>
        static async Task<int> ProcessURLAsync(string url, HttpClient client)
        {
            var sw = new Stopwatch();
            sw.Start();
            var byteArray = await client.GetByteArrayAsync(url);
            sw.Stop();
            DisplayResults(url, "https://docs.microsoft.com/en-us/", byteArray,
sw.ElapsedMilliseconds);
            return byteArray.Length;
        }
        /// <summary>
        /// Stampa una parte dell'url, la dimensione in byte di una pagina e il tempo
        impiegato per il download
        /// </summary>
        /// <param name="url"></param>
        /// <param name="urlHeadingStrip"></param>
        /// <param name="content"></param>
        /// <param name="elapsedMillis"></param>
        static void DisplayResults(string url, string urlHeadingStrip, byte[] content, long
elapsedMillis)
        {
            // Display the length of each website.
            var bytes = content.Length;
            // Strip off the "urlHeadingStrip" part from url
            var displayURL = url.Replace(urlHeadingStrip, "");

            Console.WriteLine($"\\n{displayURL,-80} bytes: {bytes,-10} ms: {elapsedMillis,-
10}");
        }
        /// <summary>
        /// Restituisce una lista di url
        /// </summary>
        /// <returns></returns>
        static List<string> SetUpURLList()
        {
            List<string> urls = new List<string>
            {
                "https://docs.microsoft.com/en-us/welcome-to-docs",
                "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/async/",
                "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/linq/",
                "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/linq/linq-to-objects",
                "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/linq/linq-and-strings",
                "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/linq/linq-to-xml-overview",
                "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/async/async-return-types",
            };
        }
    }
}

```

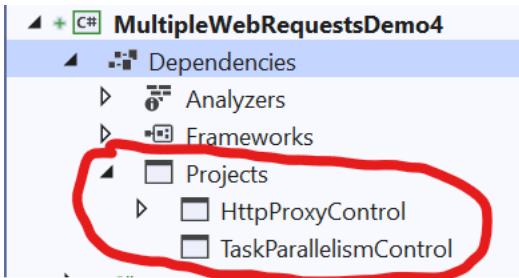
```
        "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/linq/linq-to-xml-vs-dom",
        "https://docs.microsoft.com/en-us/dotnet/csharp/programming-
guide/concepts/reflection"
    };
    return urls;
}
/// <summary>
/// Effettua il setup di una lista di url e per ognuno di essi avvia un download
asincrono su un task separato
/// </summary>
/// <returns></returns>
static async Task SumPageSizesAsync()
{
    // Make a list of web addresses.
    List<string> urlList = SetUpURLList();
    //setup del client con eventuale Proxy
    HttpProxyHelper.HttpClientProxySetup(out HttpClient client);

    //misuriamo il tempo complessivo per scaricare tutte le pagine
    var swGlobal = new Stopwatch();
    swGlobal.Start();
    //processiamo in parallelo una lista di URL
    //IEnumarable<Task<int>> downloadTasks = urlList.Select(u => ProcessURLAsync(u,
client));
    //ConcurrentBag<int> bag = new ConcurrentBag<int>();
    //await urlList.ExecuteInParallel(async u => { bag.Add(await ProcessURLAsync(u,
client)); }, 10);
    //var theTotal = bag.ToArray().Sum();
    //await urlList.ExecuteInParallel(async u => { await Task.Delay(10); }, 10);

    //definiamo il grado di parallelismo
    const int numberOfParallelThreads = 5;
    //processiamo tutti gli oggetti della collection con il grado di parallelismo
    massimo predefinito
    ConcurrentBag<int> concurrentBagOfResults = await urlList.ExecuteInParallel(u =>
ProcessURLAsync(u, client), numberOfParallelThreads);
    //sommiamo tutti i valori restituiti dai thread
    var theTotal = concurrentBagOfResults.ToArray().Sum();
    Console.WriteLine($"Somma = {theTotal}");
    swGlobal.Stop();
    long elapsedTotalMs = swGlobal.ElapsedMilliseconds;

    // Display the total count for all of the web addresses.
    Console.WriteLine($"r\nr\nTotal bytes returned: {theTotal}\r\n");
    Console.WriteLine($"Tempo complessivo di scaricamento = {elapsedTotalMs}");
}

static async Task Main(string[] args)
{
    //imposto la dimensione della console - vale solo per Windows
    if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
    {
        Console.WindowWidth = 120;
    }
    await SumPageSizesAsync();
}
```



Si osservi il file di configurazione di un progetto che utilizza i riferimenti a progetti di tipo libreria:

```
<Project Sdk="Microsoft.NET.Sdk">

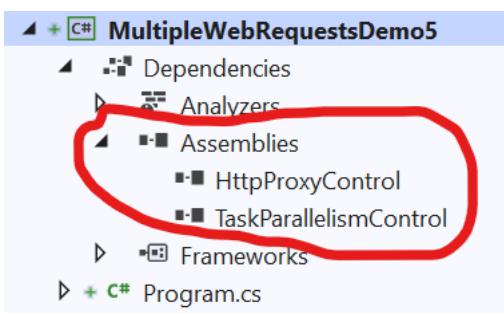
<PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>net5.0</TargetFramework>
</PropertyGroup>

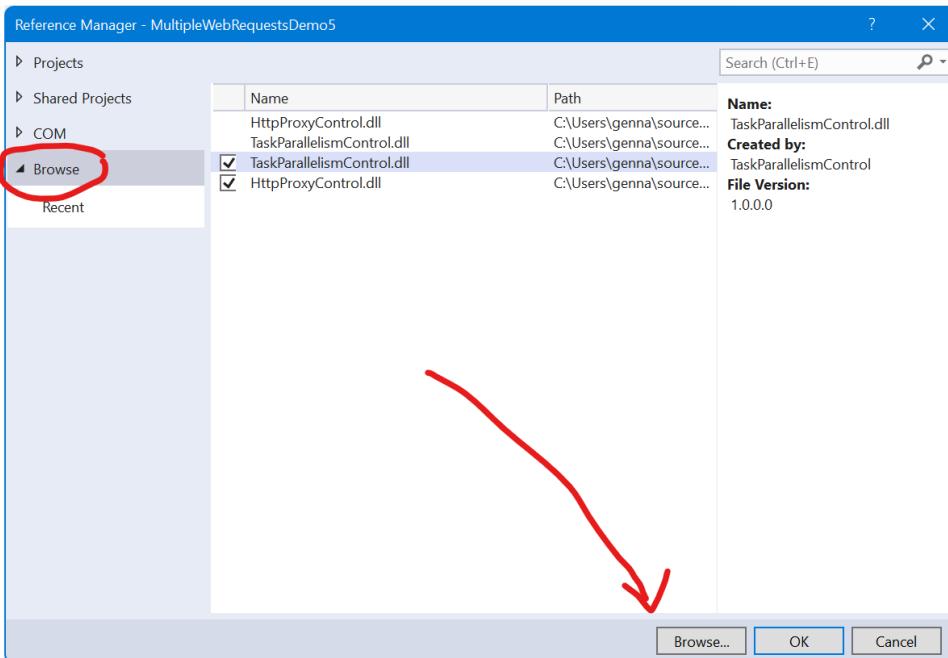
<ItemGroup>
  <ProjectReference Include="..\HttpProxyControl\HttpProxyControl.csproj" />
  <ProjectReference Include="..\TaskParallelismControl\TaskParallelismControl.csproj" />
</ItemGroup>

</Project>
```

#### **8.2.3.1.1.4. Utilizzo di un Assembly compilato di una libreria**

Quando si crea una progetto di tipo libreria (.NET Standard/.NET Core) e si effettua il Build del progetto, viene creato un file con estensione .DLL che contiene il codice compilato della libreria. Questo file può essere collegato direttamente a un progetto che usa tale libreria come assembly compilato. Ad esempio il progetto MultipleWebRequests5 utilizza direttamente i riferimenti alle DLL. Per aggiungere una DLL precompilata si può procedere come segue:





Si osservi che il file di configurazione di un progetto utilizza direttamente gli assembly precompilati di tipo DLL:

```

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <Reference Include="HttpProxyControl">
      <HintPath>..\HttpProxyControl\bin\Debug\net5.0\HttpProxyControl.dll</HintPath>
    </Reference>
    <Reference Include="TaskParallelismControl">
      <HintPath>..\TaskParallelismControl\bin\Debug\net5.0\TaskParallelismControl.dll</HintPath>
    </Reference>
  </ItemGroup>

</Project>

```

#### 8.2.3.1.1.5. Creazione di una libreria e pubblicazione su NuGet

<https://docs.microsoft.com/en-us/nuget/quickstart/create-and-publish-a-package-using-visual-studio?tabs=netcore-cli>

<https://www.c-sharpcorner.com/article/create-and-publish-a-net-core-class-library-package-into-nuget-org/>

## 8.2.4. JSON (JavaScript Object Notation)

### 8.2.4.1. Concetti di base

<https://www.digitalocean.com/community/tutorials/an-introduction-to-json>

<https://techcommunity.microsoft.com/t5/microsoft-365-pnp-blog/introduction-to-json/ba-p/2049369>

[https://www.w3schools.com/js/js\\_json\\_intro.asp](https://www.w3schools.com/js/js_json_intro.asp)

A JSON object is a key-value data format that is typically rendered in curly braces. When you're working with JSON, you'll likely see JSON objects in a .json file, but they can also exist as a JSON object or string within the context of a program.

A JSON object looks something like this:

```
{  
  "first_name" : "Sammy",  
  "last_name" : "Shark",  
  "location" : "Ocean",  
  "online" : true,  
  "followers" : 987  
}
```

Although this is a very short example, and JSON could be many lines long, this shows that the format is generally set up with two curly braces (or curly brackets) that look like this {} on either end of it, and with key-value pairs populating the space between. Most data used in JSON ends up being encapsulated in a JSON object.

Key-value pairs have a colon between them as in "key" : "value". Each key-value pair is separated by a comma, so the middle of a JSON looks like this: "key" : "value", "key" : "value", "key": "value". In our example above, the first key-value pair is "first\_name" : "Sammy".

JSON **keys** are on the left side of the colon. They need to be wrapped in double quotation marks, as in "key", and can be any valid string. Within each object, keys need to be unique. These key strings *can* include whitespaces, as in "first name", but that can make it harder to access when you're programming, so it's best to use underscores, as in "first\_name".

JSON **values** are found to the right of the colon. At the granular level, these need to be one of 6 simple data types:

- strings
- numbers
- objects
- arrays
- Booleans (true or false)
- null

Each of the data types that are passed in as values into JSON will maintain their own syntax, so strings will be in quotes, but numbers will not be.

Writing the JSON format on multiple lines often makes it much more readable, especially when dealing with a large data set. Because JSON ignores whitespace between its elements, you can space out your colons and key-value pairs in order to make the data even more human readable:

```
{
  "first_name" : "Sammy",
  "last_name" : "Shark",
  "online" : true
}
```

It is important to keep in mind that though they look similar, a JSON object is not the same format as a JavaScript object, so, though you can use functions within JavaScript objects, you cannot use them as values in JSON. The most important attribute of JSON is that it can be readily transferred between programming languages in a format that all of the participating languages can work with. JavaScript objects can only be worked with directly through the JavaScript programming language.

#### **8.2.4.2. Working with Complex Types in JSON**

JSON can store nested objects in JSON format in addition to nested arrays. These objects and arrays will be passed as values assigned to keys, and typically will be comprised of key-value pairs as well.

##### **8.2.4.2.1. Nested Objects**

In the users.json file below, for each of the four users ("sammy", "jesse", "drew", "jamie") there is a nested JSON object passed as the value for each of the users, with its own nested keys of "username" and "location" that relate to each of the users. The first nested JSON object is highlighted below.

Users.json

```
{
  "sammy" : [
    {
      "username" : "SammyShark",
      "location" : "Indian Ocean",
      "online" : true,
      "followers" : 987
    },
    "jesse" : {
      "username" : "JesseOctopus",
      "location" : "Pacific Ocean",
      "online" : false,
      "followers" : 432
    },
    "drew" : {
      "username" : "DrewSquid",
      "location" : "Atlantic Ocean",
      "online" : false,
      "followers" : 321
    }
},
```

```

"jamie" : {
    "username" : "JamieMantisShrimp",
    "location" : "Pacific Ocean",
    "online" : true,
    "followers" : 654
}
}

```

In the example above, curly braces are used throughout to form a nested JSON object with associated username and location data for each of four users. Just like any other value, when using objects, commas are used to separate elements.

#### 8.2.4.2.2. Nested Arrays

Data can also be nested within the JSON format by using JavaScript arrays that are passed as a value. JavaScript uses square brackets [ ] on either end of its array type. Arrays are ordered collections and can contain values of differing data types.

We may use an array when we are dealing with a lot of data that can be easily grouped together, like when there are various websites and social media profiles associated with a single user.

With the first nested array highlighted, a user profile for Sammy may look like this:

User\_profile.json

```

{
    "first_name" : "Sammy",
    "last_name" : "Shark",
    "location" : "Ocean",
    "websites" : [
        {
            "description" : "work",
            "URL" : "https://www.digitalocean.com/"
        },
        {
            "description" : "tutorials",
            "URL" : "https://www.digitalocean.com/community/tutorials"
        }
    ],
    "social_media" : [
        {
            "description" : "twitter",
            "link" : "https://twitter.com/digitalocean"
        },
        {
            "description" : "facebook",

```

```

        "link" : "https://www.facebook.com/DigitalOceanCloudHosting"
    },
    {
        "description" : "github",
        "link" : "https://github.com/digitalocean"
    }
]
}

```

The "websites" key and "social\_media" key each use an array to nest information belonging to Sammy's 2 website links and 3 social media profile links. We know that those are arrays because of the use of square brackets.

Using nesting within our JSON format lets us work with more complicated and hierarchical data.

#### **8.2.4.3. Comparison to XML**

XML, or eXtensible Markup Language, is a way to store accessible data that can be read by both humans and machines. The XML format is available for use across many programming languages.

In many ways, XML is very similar to JSON, but it requires much more text so is longer in length and more time consuming to read and write. XML must be parsed with an XML parser, but JSON can be parsed with a standard function. Also unlike JSON, XML cannot use arrays.

We'll look at an example of the XML format and then look at the same data rendered in JSON.

`users.xml`

```

<users>
    <user>
        <username>SammyShark</username> <location>Indian Ocean</location>
    </user>
    <user>
        <username>JesseOctopus</username> <location>Pacific Ocean</location>
    </user>
    <user>
        <username>DrewSquirr</username> <location>Atlantic Ocean</location>
    </user>
    <user>
        <username>JamieMantisShrimp</username> <location>Pacific Ocean</location>
    </user>
</users>

```

`users.json`

```
{
  "users": [
    {"username" : "SammyShark", "location" : "Indian Ocean"},
    {"username" : "JesseOctopus", "location" : "Pacific Ocean"},
    {"username" : "DrewSquid", "location" : "Atlantic Ocean"},
    {"username" : "JamieMantisShrimp", "location" : "Pacific Ocean"}
  ]
}
```

#### **8.2.4.4. Utility and resources**

You can convert CSV or tab-delimited data that you may find in spreadsheet programs into JSON by using the open-source tool [Mr. Data Converter](#).

You can convert XML to JSON and vice versa with the Creative Commons-licensed [utilities-online.info site](#).

You can validate your JSON with [JSONLint](#), and can test your JSON in a web development context with [JSFiddle](#)

#### **8.2.4.5. Serializzazione e deserializzazione di oggetti JSON in .NET con System.Text.Json (la libreria di riferimento per i nuovi progetti)**

<https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to?pivots=dotnet-6-0>

Esempio di serializzazione in JSON

<https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to?pivots=dotnet-6-0#how-to-write-net-objects-as-json-serialize>

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text.Json;
using System.Threading.Tasks;
//esempio di serializzazione da oggetti .NET a oggetti JSON
//https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to?pivots=dotnet-6-0

namespace JSONDemo1
{
    //il modello dei dati
    public class WeatherForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string Summary { get; set; }
    }
    class Program
    {

        static async Task Main(string[] args)
        {
            //versione sincrona
```

```

//JSONDemoEsempio1();

//Versione asincrona
await JSONDemoAsyncEsempio1();
}

private static void JSONDemoEsempio1()
{
    var weatherForecast = new WeatherForecast
    {
        Date = DateTime.Parse("2019-08-01"),
        TemperatureCelsius = 25,
        Summary = "Hot"
    };
    var weatherForecast2 = new WeatherForecast
    {
        Date = DateTime.Parse("2019-08-02"),
        TemperatureCelsius = 30,
        Summary = "Very Hot"
    };
    List<WeatherForecast> previsioni = new List<WeatherForecast>();
    previsioni.Add(weatherForecast);
    previsioni.Add(weatherForecast2);

    //serializzazione: da oggetto .NET a oggetto JSON
    var options = new JsonSerializerOptions { WriteIndented = true };
    string jsonString = JsonSerializer.Serialize(weatherForecast, options);
    Console.WriteLine(jsonString);
    //serializzazione di una collection in JSON
    Console.WriteLine("\n");
    string jsonCollectionString = JsonSerializer.Serialize(previsioni, options);
    Console.WriteLine(jsonCollectionString);
    //salviamo su file la collection
    string fileName = "WeatherForecasts.json";
    File.WriteAllText(fileName, jsonCollectionString);
    //leggo dal file e stampo a console
    Console.WriteLine("Lettura del JSON da file");
    Console.WriteLine(File.ReadAllText(fileName));
}

private static async Task JSONDemoAsyncEsempio1()
{
    var weatherForecast = new WeatherForecast
    {
        Date = DateTime.Parse("2019-08-01"),
        TemperatureCelsius = 25,
        Summary = "Hot"
    };
    var weatherForecast2 = new WeatherForecast
    {
        Date = DateTime.Parse("2019-08-02"),
        TemperatureCelsius = 30,
        Summary = "Very Hot"
    };
    List<WeatherForecast> previsioni = new List<WeatherForecast>();
    previsioni.Add(weatherForecast);
    previsioni.Add(weatherForecast2);

    //serializzazione: da oggetto .NET a oggetto JSON
    var options = new JsonSerializerOptions { WriteIndented = true };
    string jsonString = JsonSerializer.Serialize(weatherForecast, options);
    Console.WriteLine(jsonString);
    //serializzazione di una collection in JSON
    Console.WriteLine("\n");
}

```

```

        string jsonCollectionString = JsonSerializer.Serialize(previsioni, options);
        Console.WriteLine(jsonCollectionString);
        //salviamo su file la collection
        string fileName = "WeatherForecasts.json";
        FileStream fileStream = File.Create(fileName);
        //Impostiamo il flusso di serializzazione JSON direttamente sullo stream che
punta al file
        await JsonSerializer.SerializeAsync(fileStream, previsioni, options);
        await fileStream.DisposeAsync();
        //File.WriteAllText(fileName, jsonCollectionString);
        //leggo dal file e stampo a console
        Console.WriteLine("Lettura del JSON da file");

        Console.WriteLine(await File.ReadAllTextAsync(fileName));

    }

}

```

### Esempi di deserializzazione in JSON

<https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to?pivots=dotnet-6-0#how-to-read-json-as-net-objects-deserialize>

```

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Text.Json;
using System.Threading.Tasks;

//esempio di deserializzazione da oggetti JSON a oggetti .NET
//https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to?pivots=dotnet-6-0

namespace JSONDemo2
{
    public class WeatherForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string Summary { get; set; }
        public string SummaryField;
        public IList<DateTimeOffset> DatesAvailable { get; set; }
        public Dictionary<string, HighLowTemps> TemperatureRanges { get; set; }
        public string[] SummaryWords { get; set; }
    }

    public class HighLowTemps
    {
        public int High { get; set; }
        public int Low { get; set; }
    }

    class Program
    {
        static async Task Main(string[] args)
        {
            DeserializzazioneJSONEsempio1();

            //DeserializzazioneJSONDaFileEsempio1();
        }
    }
}

```

```

        //await DeserializzazioneJSONDaFileAsyncEsempio1();

    }

    private static async Task DeserializzazioneJSONDaFileAsyncEsempio1()
    {
        //il file WeatherForecasts.json deve esistere e deve contenere un array di
oggetti JSON di tipo WeatherForecast
        try
        {
            string fileName = "WeatherForecasts.json";
            using FileStream fileStream = File.OpenRead(fileName);
            List<WeatherForecast> weatherForecasts =
                await
JsonSerializer.DeserializeAsync<List<WeatherForecast>>(fileStream);
            //List<WeatherForecast> weatherForecasts =
JsonSerializer.Deserialize<List<WeatherForecast>>(jsonString);
            foreach (var weatherForecast in weatherForecasts)
            {
                Console.WriteLine($"Date: {weatherForecast.Date}");
                Console.WriteLine($"TemperatureCelsius:
{weatherForecast.TemperatureCelsius}");
                Console.WriteLine($"Summary: {weatherForecast.Summary}");
                Console.WriteLine("-----");
            }
        }
        catch (JsonException e)
        {

            Debug.WriteLine(e.Message);
        }
    }

    private static void DeserializzazioneJSONDaFileEsempio1()
    {
        //il file WeatherForecasts.json deve esistere e deve contenere un array di
oggetti JSON di tipo WeatherForecast
        try
        {
            string fileName = "WeatherForecasts.json";
            string jsonString = File.ReadAllText(fileName);
            List<WeatherForecast> weatherForecasts =
JsonSerializer.Deserialize<List<WeatherForecast>>(jsonString);
            foreach (var weatherForecast in weatherForecasts)
            {
                Console.WriteLine($"Date: {weatherForecast.Date}");
                Console.WriteLine($"TemperatureCelsius:
{weatherForecast.TemperatureCelsius}");
                Console.WriteLine($"Summary: {weatherForecast.Summary}");
                Console.WriteLine("-----");
            }
        }
        catch (JsonException e)
        {

            Debug.WriteLine(e.Message);
        }
    }

    private static void DeserializzazioneJSONEsempio1()
    {
        string jsonString =
        @{
            ""Date"": "2019-08-01T00:00:00-07:00",

```

```

        "TemperatureCelsius": 25,
        "Summary": "Hot",
        "DatesAvailable": [
            "2019-08-01T00:00:00-07:00",
            "2019-08-02T00:00:00-07:00"
        ],
        "TemperatureRanges": {
            "Cold": {
                "High": 20,
                "Low": -10
            },
            "Hot": {
                "High": 60,
                "Low": 20
            }
        },
        "SummaryWords": [
            "Cool",
            "Windy",
            "Humid"
        ]
    };
}

WeatherForecast weatherForecast =
    JsonSerializer.Deserialize<WeatherForecast>(jsonString);

Console.WriteLine($"Date: {weatherForecast.Date}");
Console.WriteLine($"TemperatureCelsius: {weatherForecast.TemperatureCelsius}");
Console.WriteLine($"Summary: {weatherForecast.Summary}");
}
}
}

```

#### 8.2.4.5.1. Deserialization behavior

<https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to?pivots=dotnet-6-0#deserialization-behavior>

The following behaviors apply when deserializing JSON:

- By default, property name matching is case-sensitive. You can [specify case-insensitivity](#).
- If the JSON contains a value for a read-only property, the value is ignored and no exception is thrown.
- Non-public constructors are ignored by the serializer.
- Deserialization to immutable objects or properties that don't have public set accessors is supported. See [Immutable types and Records](#).
- By default, enums are supported as numbers. You can [serialize enum names as strings](#).
- By default, fields are ignored. You can [include fields](#).
- By default, comments or trailing commas in the JSON throw exceptions. You can [allow comments and trailing commas](#).
- The [default maximum depth](#) is 64.

When you use System.Text.Json indirectly in an ASP.NET Core app, some default behaviors are different. For more information, see [Web defaults for JsonSerializerOptions](#).

You can [implement custom converters](#) to provide functionality that isn't supported by the built-in

converters.

#### 8.2.4.5.2. Serialize to formatted JSON

<https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to?pivots=dotnet-6-0#serialize-to-formatted-json>

```
using System;
using System.Text.Json;

namespace SerializeWriteIndented
{
    public class WeatherForecast
    {
        public DateTimeOffset Date { get; set; }
        public int TemperatureCelsius { get; set; }
        public string Summary { get; set; }
    }

    public class Program
    {
        public static void Main()
        {
            var weatherForecast = new WeatherForecast
            {
                Date = DateTime.Parse("2019-08-01"),
                TemperatureCelsius = 25,
                Summary = "Hot"
            };

            var options = new JsonSerializerOptions { WriteIndented = true };
            string jsonString = JsonSerializer.Serialize(weatherForecast, options);

            Console.WriteLine(jsonString);
        }
    }
}

// output:
//{
//    "Date": "2019-08-01T00:00:00-07:00",
//    "TemperatureCelsius": 25,
//    "Summary": "Hot"
//}
```

#### 8.2.4.5.3. HttpClient and HttpContent extension methods

Serializing and deserializing JSON payloads from the network are common operations. Extension methods on [HttpClient](#) and [HttpContent](#) let you do these operations in a single line of code. These extension methods use [web defaults for JsonSerializerOptions](#).

Un esempio di Serializzazione e Deserializzazione con HttpClient

```
using System;
using System.Net.Http;
```

```

using System.Net.Http.Json;
using System.Threading.Tasks;
using Microsoft.Win32;
using System.Runtime.InteropServices;
using System.Net;

namespace HttpClientExtensionMethods
{
    public class User
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Username { get; set; }
        public string Email { get; set; }
    }

    public class Program
    {
        /// <summary>
        /// Effettua il setup del client Http con l'eventuale proxy (se presente)
        /// Richiede:
        /// 1) l'installazione del pacchetto Microsoft.Win32.Registry tramite NuGet
        /// 2) using Microsoft.Win32;
        /// 3) using System.Runtime.InteropServices;
        /// 4) using System.Net;
        /// </summary>
        /// <param name="client"></param>
        static void HttpClientProxySetup(out HttpClient client)
        {

            Uri proxy;
            //https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.defaultproxy?view=net-6.0
            //https://medium.com/@sddkal/net-core-interaction-with-registry-4d7fcabc7a6b
            //https://www.shellhacks.com/windows-show-proxy-settings-cmd-powershell/
            //https://stackoverflow.com/a/63884955

            if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
            {
                //ottengo lo user specific proxy che si ottiene con il comando:
                //C:\> reg query
                "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings"
                    //per poter utilizzare Registry occorre:
                    //1) --> installare il pacchetto Microsoft.Win32.Registry tramite NuGet
                    //2) --> using Microsoft.Win32;

                //leggiamo lo user specific proxy direttamente dal registro di sistema di Windows
                string userProxy =
                    Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet Settings").GetValue("ProxyServer") as string;
                    //il proxy viene abilitato mediante il valore della chiave di registro ProxyEnable
                int? proxyEnable =
                    Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet Settings").GetValue("ProxyEnable") as int?;

                //impostiamo proxy
                proxy = proxyEnable > 0 ? new Uri(userProxy) : null;
            }
            else //se il sistema operativo è diverso da Windows procediamo con la determinazione del system wide proxy (se impostato)
        }
    }
}

```

```

{
    //questa è la procedura per ottenere il system proxy
    Uri destinationUri = new Uri("https://www.google.it");
    //Ottiene il default proxy quando si prova a contattare la destinationUri
    //Se il proxy non è impostato si ottiene null
    //Uri proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
    //Con il proxy calcolato in automatico si crea l'handler da passare
all'oggetto HttpClient e
    //funziona sia che il proxy sia configurato sia che non lo sia
    proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
}

//con il proxy ottenuto con il codice precedente
HttpClientHandler httpHandler = new HttpClientHandler()
{
    Proxy = new WebProxy(proxy, true),
    UseProxy = true,
    PreAuthenticate = false,
    UseDefaultCredentials = false,
};
client = new HttpClient(httpHandler);
}
public static async Task Main()
{
    HttpClientProxySetup(out HttpClient client);
    client.BaseAddress = new Uri("https://jsonplaceholder.typicode.com");

    // Get the user information.
    User user = await client.GetFromJsonAsync<User>("users/1");
    Console.WriteLine($"Id: {user.Id}");
    Console.WriteLine($"Name: {user.Name}");
    Console.WriteLine($"Username: {user.Username}");
    Console.WriteLine($"Email: {user.Email}");

    // Post a new user.
    HttpResponseMessage response = await client.PostAsJsonAsync("users", user);
    Console.WriteLine(
        $"{(response.IsSuccessStatusCode ? "Success" : "Error")} -
{response.StatusCode}");
}
}

// Produces output like the following example but with different names:
// 
//Id: 1
//Name: Tyler King
//Username: Tyler
//Email: Tyler @contoso.com
//Success - Created

```

#### **8.2.4.5.4. Metodi di estensione per inviare e ricevere contenuto HTTP sotto forma di oggetti JSON**

<https://docs.microsoft.com/en-us/dotnet/api/system.net.http.json.httpclientjsonextensions?view=net-6.0>

#### **8.2.4.6. Serializzazione e deserializzazione con la libreria Newtonsoft.Json**

<https://www.newtonsoft.com/json>

<https://www.newtonsoft.com/json/help/html/Introduction.htm>

#### 8.2.4.7. JSON New Features in .NET 6

<https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-6#systemtextjson-apis>

<https://www.c-sharpcorner.com/article/new-programming-model-for-handling-json-in-net-6/>

#### 8.2.4.8. Esercizi sulla serializzazione e deserializzazione di oggetti JSON

##### 8.2.4.9. Esercizi su JSON

Esercizio N. 1

Seguendo l'esempio riportato qui:

<https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to?pivots=dotnet-6-0#httpclient-and-httpcontent-extension-methods>

Scrivere un programma C# che scarica dall'endpoint <https://jsonplaceholder.typicode.com/todos> gli oggetti Todo e li inserisce nella lista **List<Todo> todoList**. Il programma stampa a console la lista ottenuta.

Esercizio N. 2

Il programma si comporta come nell'esercizio precedente, con la differenza che la lista **List<Todo> todoList** degli oggetti Todo, anziché essere stampata a console, è salvata (in modo asincrono) su un file di testo chiamato **todos.json**

Esercizio N.3

Scrivere un programma C# che scarica dall'endpoint <https://jsonplaceholder.typicode.com/photos> l'array JSON delle foto e di queste processa solo i primi 10 oggetti JSON deserializzandoli nella lista **List<Photo> photos**. Il programma crea anche la lista **List<Photo> cachedPhotos** che contiene per i campi url e thumbnailUrl degli URI che puntano a risorse locali anziché alle risorse web. Suggerimento: scaricare le immagini in una cartella cachedPhotos e le thumbnails in una cartella cachedThumbnails, conservando lo stesso nome del file. Ad esempio, la foto descritta dall'URI: <https://via.placeholder.com/600/92c952> viene mappata nel file locale descritto dall'URI: <file:///C:/ProgramWorkingDir/cachedPhotos/600/92c952.png>.

Risorse per l'esercizio:

<https://docs.microsoft.com/en-us/dotnet/api/system.uri?view=net-6.0>

<https://stackoverflow.com/a/1546420>

<https://stackoverflow.com/questions/1546419/convert-file-path-to-a-file-uri>

<https://zetcode.com/csharp/httpclient/> (per l'esempio di download di un'immagine)

Soluzione esercizio N. 3

```
//file Photo.cs
namespace JSONEsercizio3
{
    public class Photo
    {
        public int AlbumId { get; set; }
        public int Id { get; set; }
```

```

        public string Title { get; set; }
        public string Url { get; set; }
        public string ThumbnailUrl { get; set; }

        public override string ToString()
        {
            return $"{nameof(AlbumId)}={AlbumId}, {nameof(Id)}={Id},
{nameof>Title}={Title}, {nameof(Url)}={Url}, {nameof(ThumbnailUrl)}={ThumbnailUrl}";
        }
    }

}

//file Program.cs
using Microsoft.Win32;
using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using System.Net.Http;
using System.Net.Http.Json;
using System.Runtime.InteropServices;
using System.Text.Json;
using System.Threading.Tasks;

namespace JSONEsercizio3
{
    //public class User
    //{
    //    public int Id { get; set; }
    //    public string Name { get; set; }
    //    public string Username { get; set; }
    //    public string Email { get; set; }
    //}
    class Program
    {
        /// <summary>
        /// Effettua il setup del client Http con l'eventuale proxy (se presente)
        /// Richiede:
        /// 1) l'installazione del pacchetto Microsoft.Win32.Registry tramite NuGet
        /// 2) using Microsoft.Win32;
        /// 3) using System.Runtime.InteropServices;
        /// 4) using System.Net;
        /// </summary>
        /// <param name="client"></param>
        static void HttpClientProxySetup(out HttpClient client)
        {

            Uri proxy;
            //https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.defaultproxy?view=net-6.0
            //https://medium.com/@sddkal/net-core-interaction-with-registry-4d7fcabc7a6b
            //https://www.shellhacks.com/windows-show-proxy-settings-cmd-powershell/
            //https://stackoverflow.com/a/63884955

            if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
            {
                //ottengo lo user specific proxy che si ottiene con il comando:
                //C:\> reg query
                "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings"
                    //per poter utilizzare Registry occorre:
                    //1) --> installare il pacchetto Microsoft.Win32.Registry tramite NuGet
                    //2) --> using Microsoft.Win32;
            }
        }
    }
}

```

```

        //leggiamo lo user specific proxy direttamente dal registro di sistema di
Windows
        string userProxy =
Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
Settings").GetValue("ProxyServer") as string;
            //il proxy viene abilitato mediante il valore della chiave di registro
ProxyEnable
        int? proxyEnable =
Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Internet
Settings").GetValue("ProxyEnable") as int?;

        //impostiamo proxy
proxy = proxyEnable > 0 ? new Uri(userProxy) : null;

    }
else //se il sistema operativo è diverso da Windows procediamo con la
determinazione del system wide proxy (se impostato)
{
    //questa è la procedura per ottenere il system proxy
Uri destinationUri = new Uri("https://www.google.it");
    //Ottiene il default proxy quando si prova a contattare la destinationUri
    //Se il proxy non è impostato si ottiene null
    //Uri proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
    //Con il proxy calcolato in automatico si crea l'handler da passare
all'oggetto HttpClient e
    //funziona sia che il proxy sia configurato sia che non lo sia
    proxy = HttpClient.DefaultProxy.GetProxy(destinationUri);
}

//con il proxy ottenuto con il codice precedente
HttpClientHandler httpHandler = new HttpClientHandler()
{
    Proxy = new WebProxy(proxy, true),
    UseProxy = true,
    PreAuthenticate = false,
    UseDefaultCredentials = false,
};
client = new HttpClient(httpHandler);
}

/// <summary>
/// Recupera il nome del file dall'url
/// https://stackoverflow.com/a/40361205
/// https://stackoverflow.com/questions/1105593/get-file-name-from-uri-string-in-c-
sharp
/// </summary>
/// <param name="url"></param>
/// <returns></returns>
static string GetFileNameFromUrl(string url)
{
    Uri SomeBaseUri = new Uri("http://canbeanything");
    if (!Uri.TryCreate(url, UriKind.Absolute, out Uri uri))
    {
        uri = new Uri(SomeBaseUri, url);
    }
    //Path.GetFileName funziona se ha in input un URL assoluto
    return Path.GetFileName(uri.LocalPath);
}
static async Task Main(string[] args)
{
    HttpClientProxySetup(out HttpClient client);
    client.BaseAddress = new Uri("https://jsonplaceholder.typicode.com");

    try

```

```

{
    // Get the Photo information.
    //https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-
text-json-how-to?pivots=dotnet-6-0#httpclient-and-httpcontent-extension-methods
    //Sends a GET request to the specified Uri and returns the value that
results from deserializing the response body as JSON in an asynchronous operation.
    List<Photo> photoListComplete = await
client.GetFromJsonAsync<List<Photo>>("photos");
    //prendo solo i primi dieci elementi
    List<Photo> photoList10 = photoListComplete.GetRange(0, 10);
    //photoList.ForEach(p => Console.WriteLine(JsonSerializer.Serialize(p)));
    var options = new JsonSerializerOptions { WriteIndented = true };
    //stampo i primi 10 oggetti
    for (int i = 0; i < 10 && i < photoListComplete.Count; i++)
    {
        Console.WriteLine(JsonSerializer.Serialize(photoList10[i], options));
    }

    //per ogni elemento della lista scarico l'immagine in locale
    //ottengo il path alla cartella dove si trova l'eseguibile del programma
    //https://stackoverflow.com/a/7771944
    string path = System.Reflection.Assembly.GetExecutingAssembly().Location;
    //once you have the path you get the directory with:
    string appDir = Path.GetDirectoryName(path);
    //creo la cartella cachedPhotos se non esiste
    string photoSubDir = "cachedPhotos";
    string photoDir = appDir + Path.DirectorySeparatorChar + photoSubDir;
    if (!Directory.Exists(photoDir))
    {
        Directory.CreateDirectory(photoDir);
    }
    //creo la cartella cachedThumbnails se non esiste
    string thumbnailSubDir = "cachedThumbnails";
    string thumbnailDir = appDir + Path.DirectorySeparatorChar +
    thumbnailSubDir;
    if (!Directory.Exists(thumbnailDir))
    {
        Directory.CreateDirectory(thumbnailDir);
    }
    //creo la lista cachedPhotos
    List<Photo> cachedPhotos = new List<Photo>();
    //scarico le immagini nella cartella cachedPhotos e le thumbnail nella
    cartella cachedThumbnail
    foreach (var photo in photoList10)
    {
        //scarico la foto
        byte[] imageBytes = await client.GetByteArrayAsync(photo.Url);
        //scarico la thumbnail
        byte[] thumbnailBytes = await
client.GetByteArrayAsync(photo.ThumbnailUrl);

        //salvo la foto
        string localFileNamePhoto = GetFileNameFromUrl(photo.Url) + ".png";
        string localPathPhoto = Path.Combine(photoDir, localFileNamePhoto);
        await File.WriteAllBytesAsync(localPathPhoto, imageBytes);

        //salvo la thumbnail
        string localFileNameThumb = GetFileNameFromUrl(photo.ThumbnailUrl) +
        ".png";
        string localPathThumb = Path.Combine(thumbnailDir, localFileNameThumb);
        await File.WriteAllBytesAsync(localPathThumb, thumbnailBytes);
    }
}

```

```
//aggiungo l'oggetto con i riferimenti nella cache nella lista
cachedPhotos
    cachedPhotos.Add(new Photo() {
        Id = photo.Id,
        AlbumId = photo.AlbumId,
        Title = photo.Title,
        Url = new Uri(localPathPhoto).AbsoluteUri,
        ThumbnailUrl = new Uri(localPathThumb).AbsoluteUri });
}

//stampo la lista cachedPhotos
Console.WriteLine("\n\nCached List:");
cachedPhotos.ForEach(p => Console.WriteLine(JsonSerializer.Serialize(p,
options)));

//se dell'URI volessimo ottenere il path dovremmo usare la property
LocalPath oppure AbsolutePath
//ad esempio, stampo i path delle foto
Console.WriteLine("\n\nStampa degli absolute path delle foto a partire
dagli Uri");
    cachedPhotos.ForEach(f => Console.WriteLine(new
Uri(f.Url).AbsolutePath));
}

}
catch (HttpRequestException e)
{
    Console.WriteLine("\nException Caught!");
    Console.WriteLine("Message :{0} ", e.Message);
}

}
}
```

## 9. REST Api

### 9.1. REST 101

<https://restfulapi.net/>

[https://it.wikipedia.org/wiki/Representational\\_state\\_transfer](https://it.wikipedia.org/wiki/Representational_state_transfer)

<https://www.html.it/pag/19596/i-principi-dellarchitettura-restful/>

<https://www.html.it/articoli/cos-e-rest-caratteristiche/>

<https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

<https://www.guru99.com/restful-web-services.html>

<https://smartbear.com/blog/an-introduction-to-restful-apis-infographic/>

<https://blog.postman.com/rest-api-definition/>

**REST** definisce un insieme di principi architetturali per la progettazione di un sistema. Rappresenta uno stile architettonico, cioè non si riferisce ad un sistema concreto e ben definito né si tratta di uno standard stabilito da un organismo di standardizzazione.

La sua definizione è apparsa per la prima volta nel 2000 nella tesi di **Roy Fielding**, Architectural Styles and the Design of Network-based Software Architectures, discussa presso l'Università della California, ad Irvine. In questa tesi si analizzavano alcuni principi alla base di diverse architetture software, tra cui appunto i principi di un'architettura software che consentisse di vedere il Web come una piattaforma per l'elaborazione distribuita.

È bene precisare che i principi **REST** non sono necessariamente legati al Web, nel senso che si tratta di principi astratti di cui il **World Wide Web** ne risulta essere un esempio concreto.

#### 9.1.1. Principi

REST prevede che la scalabilità del [Web](#) e la crescita siano risultati di pochi principi chiave di progettazione:

- lo stato dell'applicazione e le funzionalità sono divisi in *risorse web*
- ogni risorsa è unica e indirizzabile usando *sintassi universale* per uso nei link ipertestuali
- tutte le risorse sono condivise come **interfaccia uniforme** per il trasferimento di stato tra client e risorse, questo consiste in:
  - un insieme vincolato di operazioni ben definite
  - un insieme vincolato di contenuti, optionalmente supportato da codice a richiesta
  - un protocollo che è:
    - [client-server](#)
    - privo di stato (*stateless*)
    - memorizzabile in cache
    - a livelli.

#### 9.1.2. Vincoli

L'approccio architettonico REST è definito dai seguenti sei vincoli applicati ad un'architettura, mentre lascia libera l'implementazione dei singoli componenti<sup>[2]</sup>.

**Client-server.** Un insieme di interfacce uniformi separa i *client* dai *server*. Questa separazione di ruoli e compiti significa che, per esempio, il client non si deve preoccupare del salvataggio delle informazioni, che rimangono all'interno dei singoli server. In questo modo la portabilità del codice del client ne trae vantaggio. I server non si devono fare carico dell'interfaccia grafica o dello stato dell'utente, in questo modo l'[hardware](#) può essere più semplice e maggiormente scalabile. Server e client possono essere sostituiti e sviluppati indipendentemente fintanto che l'interfaccia non viene modificata.

**Stateless.** La comunicazione client-server è vincolata in modo che nessun contesto client venga memorizzato sul server tra le richieste. Ciascuna richiesta dai vari client contiene tutte le informazioni necessarie per richiedere il servizio e lo stato della sessione è contenuto nel client. Lo stato della sessione può anche essere trasferito al server attraverso un altro servizio di memorizzazione persistente, per esempio un [database](#).

**Cacheable.** Come nel World Wide Web, i client possono mettere in [cache](#) le risposte. Queste devono in ogni modo definirsi implicitamente o esplicitamente *cacheable* o no, in modo da prevenire che i client possano riutilizzare stati vecchi e dati errati. Una corretta gestione della cache può ridurre o parzialmente eliminare le comunicazioni client-server migliorando scalabilità e prestazioni.

**Layered system.** La struttura del sistema "a strati" (letteralmente dall'inglese) rende possibile per esempio pubblicare le API in un server, memorizzare i dati in un secondo server e gestire l'autenticazione delle richieste in un terzo server. Questo comporta che un client non può sapere se è connesso direttamente a un server finale oppure a uno della catena.

**Code on demand.** (opzionale) I server possono temporaneamente estendere o personalizzare le funzionalità del client trasferendo del codice eseguibile. Per esempio questo può includere componenti compilati come [Applet Java](#) o linguaggi di scripting lato *client* come per esempio [JavaScript](#). Il concetto di *Code on demand* è l'unico vincolo opzionale per la definizione di un'architettura REST.

**Uniform interface.** Un'interfaccia di comunicazione omogenea tra client e server permette di semplificare e disaccoppiare l'architettura, per poterla modificare separatamente a blocchi.

### 9.1.3. Il principio fondamentale di REST: le risorse

Un concetto importante in REST è l'esistenza di **risorse** (fonti di informazioni), a cui si può accedere tramite un identificatore globale (un [URI](#)).

Per utilizzare le risorse, le *componenti* di una rete (componenti client e server) comunicano attraverso un'interfaccia standard (per esempio HTTP) per scambiare *rappresentazioni* di queste risorse, ovvero il documento che trasmette le *informazioni*. Per esempio, una risorsa [cerchio](#) potrebbe accettare e restituire una rappresentazione che specifica un punto per il centro e il raggio in formato [SVG](#), ma potrebbe anche accettare e restituire una rappresentazione che specifica tre punti distinti qualsiasi lungo la circonferenza nel formato [CSV](#).

Un numero qualsiasi di *connettori* ([client](#), [server](#), [cache](#), [tunnel](#) ecc.) può mediare la richiesta, ma ogni connettore interviene senza conoscere la "storia passata" delle altre richieste; proprio per questo motivo l'architettura REST si definisce *stateless*, in opposizione ad altre architetture o protocolli *stateful*. Di conseguenza un'applicazione può interagire con una risorsa conoscendo due cose: l'identificatore della risorsa e l'azione richiesta. Non serve sapere se ci sono [proxy](#), [gateway](#), [firewall](#), tunnel o altri meccanismi intermedi tra essa e il server in cui è presente l'informazione necessaria.

L'applicazione comunque deve conoscere il formato dell'informazione restituita, ovvero la sua *rappresentazione*. Tipicamente è un documento [HTML](#), [XML](#) o [JSON](#), ma possono essere anche immagini o altri contenuti.

## 9.1.4. REST Resource Naming Guide

<https://restfulapi.net/resource-naming/>

### 9.1.4.1. What is a Resource?

In REST, the primary data representation is called **resource**. Having a consistent and robust REST resource naming strategy – will prove one of the best design decisions in the long term.

The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service (e.g. “today’s weather in Los Angeles”), a collection of other resources, a non-virtual object (e.g., a person), and so on.

In other words, any concept that might be the target of an author’s hypertext reference must fit within the definition of a resource.

A resource is a conceptual mapping to a set of entities, not the entity that corresponds to the mapping at any particular point in time.

— Roy Fielding’s dissertation

### 9.1.4.2. Singleton and Collection Resources

A **resource can be a singleton or a collection.**

For example, “**customers**” is a collection resource and “**customer**” is a singleton resource (in a banking domain).

We can identify “**customers**” collection resource using the URI “**/customers**”. We can identify a single “**customer**” resource using the URI “**/customers/{customerId}**”.

### 9.1.4.3. Collection and Sub-collection Resources

A **resource may contain sub-collection resources** also.

For example, sub-collection resource “**accounts**” of a particular “**customer**” can be identified using the URN “**/customers/{customerId}/accounts**” (in a banking domain).

Similarly, a singleton resource “**account**” inside the sub-collection resource “**accounts**” can be identified as follows: “**/customers/{customerId}/accounts/{accountId}**”.

### 9.1.4.4. URI

REST APIs use Uniform Resource Identifiers (URIs) to address resources. REST API designers should create URIs that convey a REST API’s resource model to the potential clients of the API. When resources are named well, an API is intuitive and easy to use. If done poorly, that same API can be challenging to use and understand.

*The constraint of a uniform interface is partially addressed by the combination of URIs and HTTP verbs and using them in line with the standards and conventions.*

Below are a few tips to get you going when creating the resource URIs for your new API.

### 9.1.5. Best Practices

#### 9.1.5.1. Use nouns to represent resources

RESTful URI should refer to a resource that is a thing (noun) instead of referring to an action (verb) because nouns have properties that verbs do not have – similarly, resources have attributes. Some examples of a resource are:

- Users of the system
- User Accounts
- Network Devices etc.

and their resource URIs can be designed as below:

```
http://api.example.com/device-management/managed-devices
```

```
http://api.example.com/device-management/managed-devices/{device-id}
```

```
http://api.example.com/user-management/users
```

```
http://api.example.com/user-management/users/{id}
```

For more clarity, let's divide the **resource archetypes** into four categories (document, collection, store, and controller). Then it would be best if **you always targeted to put a resource into one archetype and then use its naming convention consistently**.

*For uniformity's sake, resist the temptation to design resources that are hybrids of more than one archetype.*

#### 9.1.5.2. document

A document resource is a singular concept that is akin to an object instance or database record.

In REST, you can view it as a single resource inside resource collection. A document's state representation typically includes both fields with values and links to other related resources.

Use "singular" name to denote document resource archetype.

```
http://api.example.com/device-management/managed-devices/{device-id}
```

```
http://api.example.com/user-management/users/{id}
```

```
http://api.example.com/user-management/users/admin
```

#### 9.1.5.3. *collection*

A collection resource is a server-managed directory of resources.

Clients may propose new resources to be added to a collection. However, it is up to the collection resource to choose to create a new resource or not.

A collection resource chooses what it wants to contain and also decides the URIs of each contained resource.

Use the “plural” name to denote the collection resource archetype.

```
http://api.example.com/device-management/managed-devices
```

```
http://api.example.com/user-management/users
```

```
http://api.example.com/user-management/users/{id}/accounts
```

#### 9.1.5.4. *store*

A store is a client-managed resource repository. A store resource lets an API client put resources in, get them back out, and decide when to delete them.

A store never generates new URIs. Instead, each stored resource has a URI. The URI was chosen by a client when the resource initially put it into the store.

Use “plural” name to denote store resource archetype.

```
http://api.example.com/song-management/users/{id}/playlists
```

#### 9.1.5.5. *controller*

A controller resource models a procedural concept. Controller resources are like executable functions, with parameters and return values, inputs, and outputs.

Use “verb” to denote controller archetype.

```
http://api.example.com/cart-management/users/{id}/cart/checkout
```

```
http://api.example.com/song-management/users/{id}/playlist/play
```

### **9.1.5.6. *Consistency is the key***

Use consistent resource naming conventions and URI formatting for minimum ambiguity and maximum readability and maintainability. You may implement the below design hints to achieve consistency:

#### **9.1.5.6.1. Use forward slash (/) to indicate hierarchical relationships**

The forward-slash (/) character is used in the path portion of the URI to indicate a hierarchical relationship between resources. e.g.

```
http://api.example.com/device-management
```

```
http://api.example.com/device-management/managed-devices
```

```
http://api.example.com/device-management/managed-devices/{id}
```

```
http://api.example.com/device-management/managed-devices/{id}/scripts
```

```
http://api.example.com/device-management/managed-devices/{id}/scripts/{id}
```

#### **9.1.5.6.2. Do not use trailing forward slash (/) in URIs**

As the last character within a URI's path, a forward slash (/) adds no semantic value and may confuse. It's better to drop it from the URI.

```
http://api.example.com/device-management/managed-devices/  
http://api.example.com/device-management/managed-devices /*This is much  
better version*/
```

#### **9.1.5.6.3. Use hyphens (-) to improve the readability of URIs**

To make your URIs easy for people to scan and interpret, use the hyphen (-) character to improve the readability of names in long path segments.

```
http://api.example.com/device-management/managed-devices/
```

```
http://api.example.com/device-management/managed-devices /*This is  
much better version*/
```

#### **9.1.5.6.4. Do not use underscores (\_)**

It's possible to use an underscore in place of a hyphen to be used as a separator – But depending on the application's font, it is possible that the underscore (\_) character can either get partially obscured or completely hidden in some browsers or screens.

To avoid this confusion, use hyphens (-) instead of underscores (\_).

```
http://api.example.com/inventory-management/managed-entities/{id}/install-script-location //More readable
```

```
http://api.example.com/inventory-management/managedEntities/{id}/installScriptLocation //Less readable
```

#### 9.1.5.6.5. Use lowercase letters in URIs

When convenient, lowercase letters should be consistently preferred in URI paths.

```
http://api.example.org/my-folder/my-doc //1
```

```
HTTP://API.EXAMPLE.ORG/my-folder/my-doc //2
```

```
http://api.example.org/My-Folder/my-doc //3
```

In the above examples, 1 and 2 are the same but 3 is not as it uses **My-Folder** in capital letters.

#### 9.1.5.6.6. Do not use file extensions

File extensions look bad and do not add any advantage. Removing them decreases the length of URIs as well. No reason to keep them.

Apart from the above reason, if you want to highlight the media type of API using file extension, then you should rely on the media type, as communicated through the **Content-Type** header, to determine how to process the body's content.

```
http://api.example.com/device-management/managed-devices.xml /*Do not use it*/
```

```
http://api.example.com/device-management/managed-devices /*This is correct URI*/
```

#### 9.1.5.6.7. Never use CRUD function names in URIs

We should not use URIs to indicate a CRUD function. URIs should only be used to uniquely identify the resources and not any action upon them.

We should use HTTP request methods to indicate which CRUD function is performed.

```
HTTP GET http://api.example.com/device-management/managed-devices //Get all devices

HTTP POST http://api.example.com/device-management/managed-devices //Create new Device

HTTP GET http://api.example.com/device-management/managed-devices/{id} //Get device for given Id

HTTP PUT http://api.example.com/device-management/managed-devices/{id} //Update device for given Id

HTTP DELETE http://api.example.com/device-management/managed-devices/{id} //Delete device for given Id
```

#### 9.1.5.6.8. Use query component to filter URI collection

Often, you will encounter requirements where you will need a collection of resources sorted, filtered, or limited based on some specific resource attribute.

For this requirement, do not create new APIs – instead, enable sorting, filtering, and pagination capabilities in resource collection API and pass the input parameters as query parameters. e.g.

```
http://api.example.com/device-management/managed-devices

http://api.example.com/device-management/managed-devices?region=USA

http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ

http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ&sort=installation-date
```

#### 9.1.6. HTTP Methods

<https://restfulapi.net/http-methods/>

RESTful APIs enable you to develop any kind of web application having all possible CRUD (create, retrieve, update, delete) operations. REST guidelines suggest using a specific HTTP method on a specific type of call made to the server.

- **HTTP GET**

Use GET requests **to retrieve resource representation/information only** – and not to modify it in any way. As GET requests do not change the state of the resource, these are said to be **safe methods**. Additionally, GET APIs should be **idempotent**, which means that making multiple identical requests must produce the same result every time until another API (POST or PUT) has changed the state of the resource on the server.

If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

For any given HTTP GET API, if the resource is found on the server, then it must return HTTP response code 200 (OK) – along with the response body, which is usually either XML or JSON content (due to their platform-independent nature).

In case resource is NOT found on server then it must return HTTP response code 404 (NOT FOUND). Similarly, if it is determined that GET request itself is not correctly formed then server will return HTTP response code 400 (BAD REQUEST).

- **Example request URIs**

```
HTTP GET http://www.appdomain.com/users
```

```
HTTP GET http://www.appdomain.com/users?size=20&page=5
```

```
HTTP GET http://www.appdomain.com/users/123
```

```
HTTP GET http://www.appdomain.com/users/123/address
```

- **HTTP POST**

Use POST APIs **to create new subordinate resources**, e.g., a file is subordinate to a directory containing it or a row is subordinate to a database table. Talking strictly in terms of REST, **POST** methods are used to create a new resource into the collection of resources.

Ideally, if a resource has been created on the origin server, the response **SHOULD** be HTTP response code 201 (Created) and contain an entity which describes the status of the request and refers to the new resource, and a [Location](#) header.

Many times, the action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either HTTP response code 200 (OK) or 204 (No Content) is the appropriate response status.

Responses to this method are **not cacheable**, unless the response includes appropriate [Cache-Control](#) or [Expires](#) header fields.

Please note that POST is **neither safe nor idempotent**, and invoking two identical POST requests will result in two different resources containing the same information (except resource ids).

- **Example request URIs**

```
HTTP POST http://www.appdomain.com/users
```

```
HTTP POST http://www.appdomain.com/users/123/accounts
```

- **HTTP PUT**

Use PUT APIs primarily **to update existing resource** (if the resource does not exist, then API may decide to create a new resource or not). If a new resource has been created by the PUT API, the origin server **MUST** inform the user agent via the HTTP response code 201 (Created) response and if an

existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are **not cacheable**.

The difference between the POST and PUT APIs can be observed in request URIs. POST requests are made on resource collections, whereas PUT requests are made on an individual resource.

- **Example request URIs**

HTTP PUT http://www.appdomain.com/users/123

HTTP PUT http://www.appdomain.com/users/123/accounts/456

- **HTTP DELETE**

As the name applies, DELETE APIs are used **to delete resources** (identified by the Request-URI).

A successful response of DELETE requests SHOULD be HTTP response code 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has been queued, or 204 (No Content) if the action has been performed but the response does not include an entity.

DELETE operations are **idempotent**. If you DELETE a resource, it's removed from the collection of resources. Repeatedly calling DELETE API on that resource will not change the outcome – however calling DELETE on a resource a second time will return a 404 (NOT FOUND) since it was already removed. Some may argue that it makes the DELETE method non-idempotent. It's a matter of discussion and personal opinion.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are **not cacheable**.

- **Example request URIs**

HTTP DELETE http://www.appdomain.com/users/123

HTTP DELETE http://www.appdomain.com/users/123/accounts/456

- **HTTP PATCH**

HTTP PATCH requests are **to make partial update on a resource**. If you see PUT requests also modify a resource entity so to make more clear – PATCH method is the correct choice for partially updating an existing resource and PUT should only be used if you're replacing a resource in its entirety.

Please note that there are some challenges if you decide to use PATCH APIs in your application:

- Support for PATCH in browsers, servers, and web application frameworks is not universal. IE8, PHP, Tomcat, Django, and lots of other software has missing or broken support for it.
- Request payload of PATCH request is not straightforward as it is for PUT request. e.g.

HTTP GET /users/1

produces below response:

```
{id: 1, username: 'admin', email: 'email@example.org'}
```

A sample patch request to update the email will be like this:

HTTP PATCH /users/1

```
[  
 { "op": "replace", "path": "/email", "value": "new.email@example.org" }  
]
```

There may be following possible operations are per the HTTP specification.

```
[  
 { "op": "test", "path": "/a/b/c", "value": "foo" },  
 { "op": "remove", "path": "/a/b/c" },  
 { "op": "add", "path": "/a/b/c", "value": [ "foo", "bar" ] },  
 { "op": "replace", "path": "/a/b/c", "value": 42 },  
 { "op": "move", "from": "/a/b/c", "path": "/a/b/d" },  
 { "op": "copy", "from": "/a/b/d", "path": "/a/b/e" }  
]
```

PATCH method is not a replacement for the POST or PUT methods. It applies a delta (diff) rather than replacing the entire resource.

- **Summary of HTTP Methods for RESTful APIs**

The below table summarises the use of HTTP methods discussed above.

HTTP METHOD	CRUD	ENTIRE COLLECTION (E.G. /USERS)	SPECIFIC ITEM (E.G. /USERS/123)
POST	Create	201 (Created), 'Location' header with link to /users/{id} containing new ID.	Avoid using POST on single resource
GET	Read	200 (OK), list of users. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single user. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	404 (Not Found), unless you want to update every resource in the entire collection of resource.	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID not found or invalid.
PATCH	Partial Update/Modify	404 (Not Found), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID not found or invalid.
DELETE	Delete	404 (Not Found), unless you want to delete the whole collection — use with caution.	200 (OK). 404 (Not Found), if ID not found or invalid.

### 9.1.7. HATEOAS Driven REST APIs

<https://restfulapi.net/hateoas/>

**HATEOAS (Hypermedia as the Engine of Application State)** is a constraint of the REST application architecture that keeps the RESTful style architecture unique from most other network application architectures. The term "**hypermedia**" refers to any content that contains links to other forms of media such as images, movies, and text.

REST architectural style lets you use hypermedia links in the response contents so that the client can dynamically navigate to the appropriate resource by traversing the hypermedia links. Above is conceptually the same as a web user browsing through web pages by clicking the relevant hyperlinks to achieve a final goal.

For example, below given JSON response may be from an API like

HTTP GET <http://api.domain.com/management/departments/10>

```
{  
   "departmentId": 10,
```

```

"departmentName": "Administration",
"locationId": 1700,
"managerId": 200,
"links": [
{
  "href": "10/employees",
  "rel": "employees",
  "type" : "GET"
}
]
}

```

In the preceding example, the response returned by the server contains hypermedia links to employee resources 10/employees, which can be traversed by the client to read employees belonging to the department.

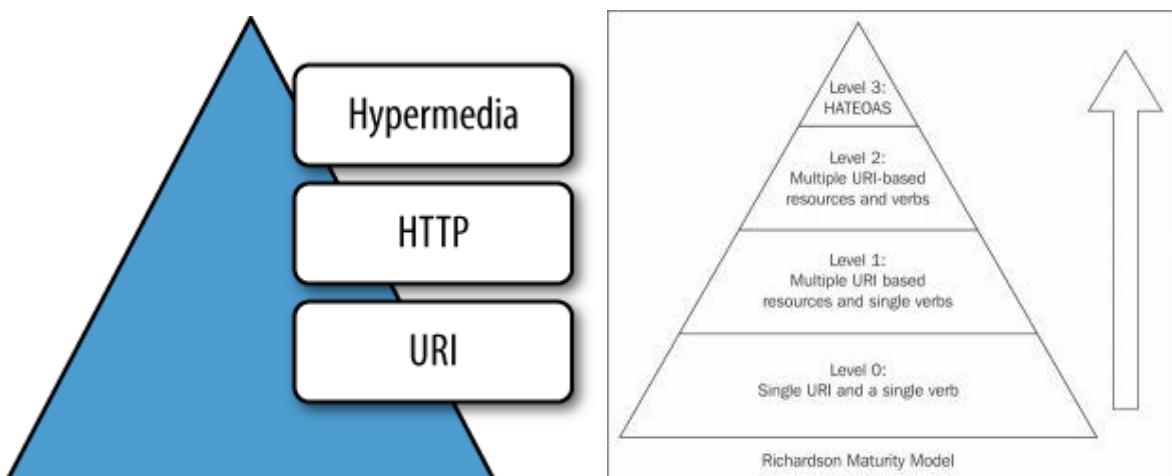
The advantage of the above approach is that hypermedia links returned from the server drive the application's state and not the other way around.

### 9.1.8. Richardson Maturity Model

<https://restfulapi.net/richardson-maturity-model/>

Leonard Richardson analyzed a hundred different web service designs and divided them into four categories based on how much they are REST compliant. This model of division of REST services to identify their maturity level – is called **Richardson Maturity Model**.

Richardson used three factors to decide the maturity of a service i.e. [URL](#), [HTTP Methods](#) and [HATEOAS](#) (Hypermedia as the Engine of Application State). The more a service employs these technologies – more mature it shall be considered.



In this analysis, Richardson described these maturity levels as below:

- [Level Zero](#)

Level zero of maturity does not make use of any of URI, HTTP Methods, and HATEOAS capabilities. These services have a single URI and use a single HTTP method (typically POST)

- [Level One](#)

Level one of maturity **makes use of URIs**. These services employ many URIs but only a single HTTP verb – generally HTTP POST. They give each individual resource in their universe a URI. Every resource is separately identified by a unique URI – and that makes them better than level zero.

- [Level Two](#)

Level two of maturity **makes use of URIs and HTTP**. Level two services host numerous URI-addressable resources. Such services support several of the HTTP verbs on each exposed resource – Create, Read, Update and Delete (CRUD) services.

- [Level Three](#)

Level three of maturity **makes use of all three i.e. URIs and HTTP and HATEOAS**.

This is the most mature level of Richardson's model which encourages easy discoverability and makes it easy for the responses to be self-explanatory by using HATEOAS.

## 9.2. Consumare REST Api mediante l'oggetto HttpClient

<https://docs.microsoft.com/en-us/aspnet/web-api/overview/advanced/calling-a-web-api-from-a-net-client>

<https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/console-webapiclient>

<https://zetcode.com/csharp/httpclient/>

### 9.2.1. Un esempio di programma che utilizza le API di Github

<https://github.com/dotnet/samples/tree/main/csharp/getting-started/console-webapiclient>

Per i Media-Type vedere quanto riportato nella documentazione ufficiale di GitHub:

<https://docs.github.com/en/rest/overview/media-types>

Questo progetto utilizza la libreria HttpProxyControl.

```
//file Repository.cs

using System;
using System.Globalization;
using System.Text.Json.Serialization;

namespace GithubAPIClient
{
    public class Repository
    {
        [JsonPropertyName("name")]
        public string Name { get; set; }

        [JsonPropertyName("description")]
        public string Description { get; set; }

        [JsonPropertyName("html_url")]
        public Uri GitHubHomeUrl { get; set; }
    }
}
```

```

[JsonPropertyName("homepage")]
public Uri Homepage { get; set; }

[JsonPropertyName("watchers")]
public int Watchers { get; set; }

[JsonPropertyName("pushed_at")]
public string JsonDate { get; set; }

public DateTime LastPush =>
    DateTime.ParseExact(JsonDate, "yyyy-MM-ddTHH:mm:ssZ",
CultureInfo.InvariantCulture.InvariantCulture);
}

//file Program.cs

using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text.Json;
using System.Threading.Tasks;
using HttpProxyControl;

namespace GithubAPIClient
{
    class Program
    {

        static async Task Main(string[] args)
        {

            var repositories = await ProcessRepositories();

            foreach (var repo in repositories)
            {
                Console.WriteLine(repo.Name);
                Console.WriteLine(repo.Description);
                Console.WriteLine(repo.GitHubHomeUrl);
                Console.WriteLine(repo.Homepage);
                Console.WriteLine(repo.Watchers);
                Console.WriteLine(repo.LastPush);
                Console.WriteLine();
            }
        }

        private static async Task<List<Repository>> ProcessRepositories()
        {
            HttpProxyHelper.HttpClientProxySetup(out HttpClient client);
            client.DefaultRequestHeaders.Accept.Clear();
            client.DefaultRequestHeaders.Accept.Add(
                new MediaTypeWithQualityHeaderValue("application/vnd.github.v3+json"));
            client.DefaultRequestHeaders.Add("User-Agent", ".NET Foundation Repository
Reporter");

            var streamTask =
client.GetStreamAsync("https://api.github.com/orgs/dotnet/repos");
            var repositories = await JsonSerializer.DeserializeAsync<List<Repository>>(await
streamTask);
            return repositories;
        }
    }
}

```

```
    }  
}
```

## 9.2.2. Principali classi di riferimento per la gestione delle richieste HTTP

Classe HttpClient:

<https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient>

Classe HttpClientHandler:

<https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclienthandler>

Classe HttpContent:

<https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpcontent>

Classe StringContent

<https://docs.microsoft.com/en-us/dotnet/api/system.net.http.stringcontent>

Classe HttpClientJsonExtensions

<https://docs.microsoft.com/en-us/dotnet/api/system.net.http.json.httpclientjsonextensions>

### 9.2.3. Postman

<https://www.getpostman.com/>

<https://portapps.io/app/postman-portable/>

<https://learning.getpostman.com/>

<https://docs.postman-echo.com/?version=latest>

### 9.2.4. Testing APIs with Postman

Introduzione a Postman:

<https://learning.postman.com/docs/getting-started/introduction/>

Inviare la prima richiesta con Postman:

<https://learning.postman.com/docs/getting-started/sending-the-first-request/>

Utilizzare i modelli di collections di Postman Echo:

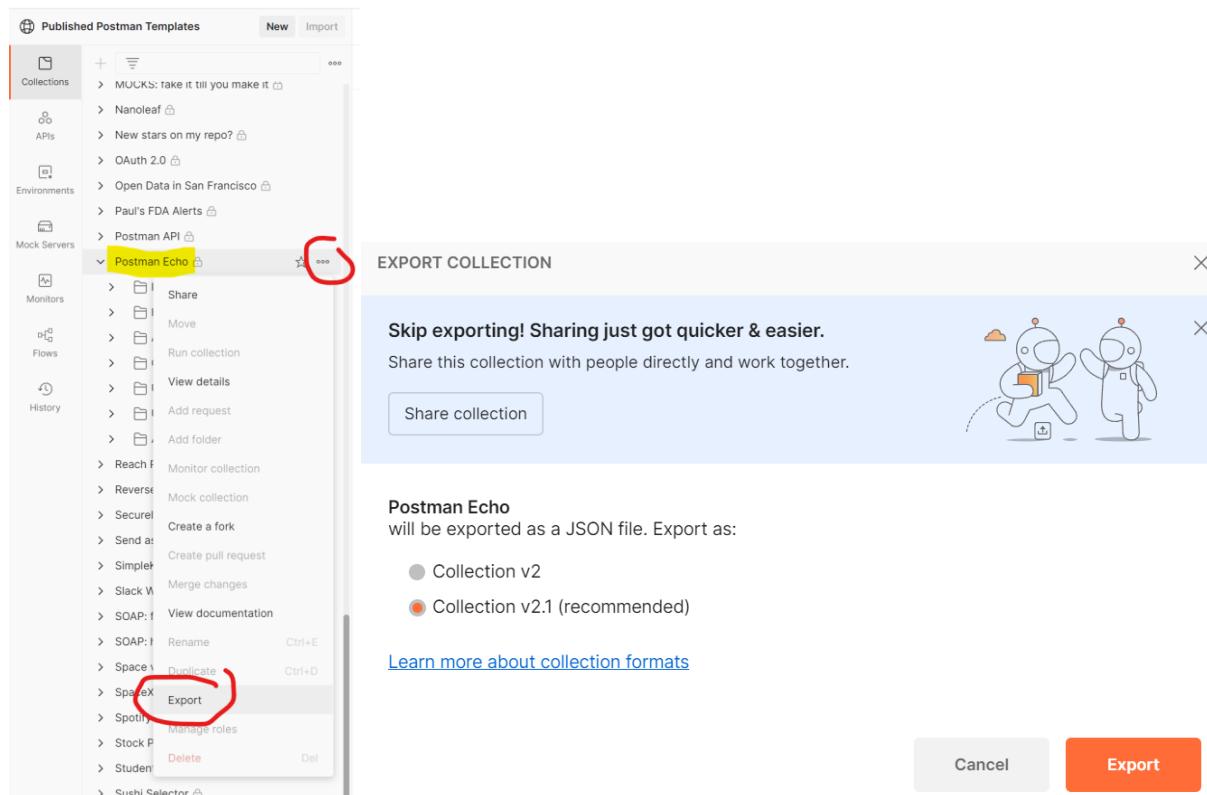
<https://www.postman.com/postman/workspace/published-postman-templates/documentation/631643-f695cab7-6878-eb55-7943-ad88e1ccfd65?ctx=documentation>

Per utilizzare i modelli di richieste di Postman già pronti ci sono almeno due possibilità:

- 1) esportare il modello in un file locale e poi importarlo nel proprio workspace di Postman
- 2) effettuare il fork di una collection (richiede un profilo pubblico di Postman)

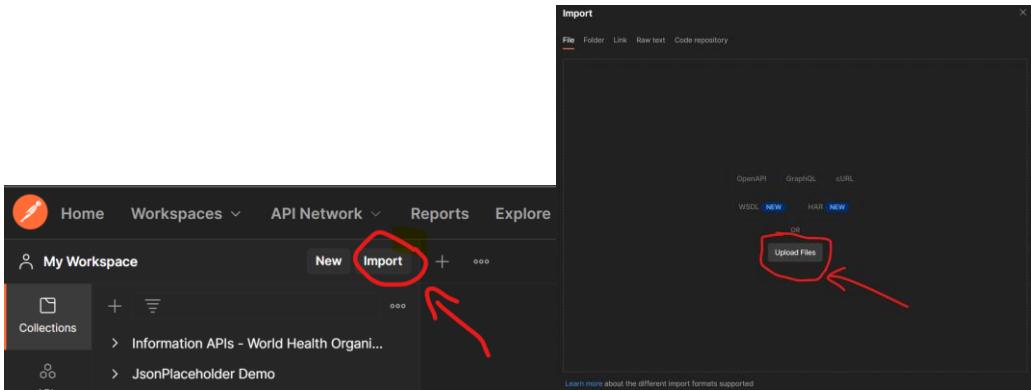
Di seguito si mostrano entrambe le procedure.

#### Esportazione e successiva importazione di una collection pubblica.



Salvare localmente il file in formato JSON.

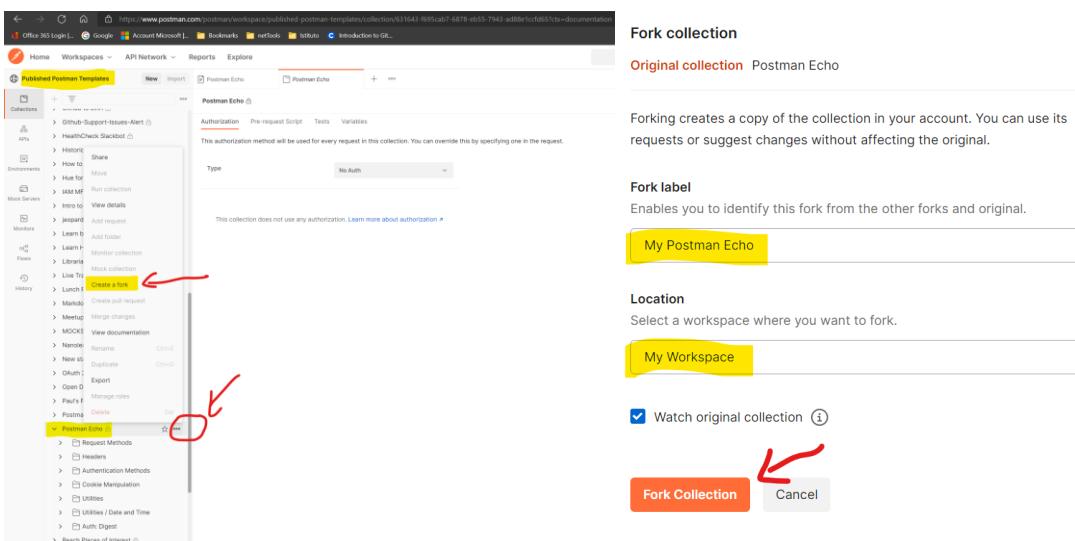
Per importare la collection nel proprio workspace:



Selezionare “Upload Files” e quindi scegliere il file scaricato precedentemente mediante l’export. Dopo questa azione la collection comparirà nel proprio workspace.

### Fork di una collection

Nell’elenco dei modelli a disposizione selezionare Postman Echo e quindi “Create a Fork”. In questo modo verrà creata una copia della collection di richieste, con un nome di propria scelta, nel proprio workspace all’interno del proprio account di Postman. Per poter effettuare il fork di una collection pubblica occorre rendere pubblico il proprio profilo: <https://blog.postman.com/how-to-create-your-postman-public-profile/>



A questo punto la collection verrà copiata nel workspace del proprio account.

Avere un profilo pubblico di Postman permette di avere una pagina di descrizione del proprio profilo al link <https://www.postman.com/UserName>. Su questa pagina è possibile pubblicare le collection di Postman che si vuole rendere pubbliche e che altri possono copiare mediante fork o mediante import.

### 9.2.5. Impariamo ad usare Postman con la collection di Postman Echo

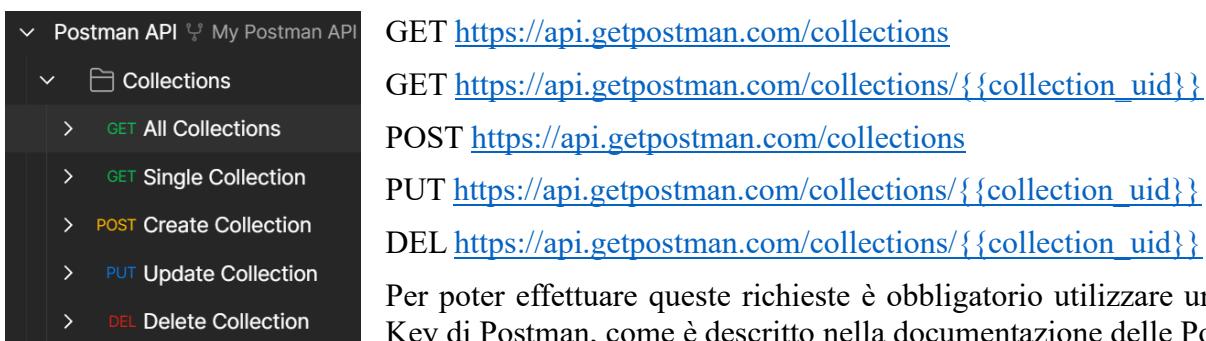
<https://learning.postman.com/docs/sending-requests/requests/>

Dopo aver importato nel proprio workspace la collection di Postman Echo (in uno dei modi descritti precedentemente), iniziamo a vedere come utilizzare questo potente strumento per il testing delle API. Effettuiamo le seguenti richieste, contenute nella collection di Postman Echo:



### 9.2.6. Usiamo la collection Postman API

Dopo aver importato nel proprio workspace la collection di [Postman API](#), effettuiamo le seguenti richieste:



Per poter effettuare queste richieste è obbligatorio utilizzare una API Key di Postman, come è descritto nella documentazione delle Postman API:

An API Key is required to be sent as part of every request to the Postman API, in the form of an X-Api-Key request header.

If you do not have an API Key, you can easily generate one by heading over to the [Postman Integrations Dashboard](#).

An API Key tells our API server that the request it received came from you. Everything that you have access to in Postman is accessible with an API Key that is generated by you.

For ease of use inside Postman, you could store your API key in an [environment variable](#) called postman\_api\_key and this [Collection](#) will automatically use it to make API calls.

Per creare un'API Key di Postman nel proprio account aprire la pagina: <https://web.postman.co/settings/me/api-keys>

Cliccare sul pulsante Generate API Key, inserire un nome per la chiave e salvare il valore della chiave in un file di testo in modo da poterla recuperare successivamente. La chiave è visibile solo nel momento in cui è generata, successivamente no!

The screenshot shows the 'API keys' section of the Postman interface. A modal window titled 'Generate an API key' is open, prompting the user to 'Name your key'. The input field contains 'MyPostmanKey'. Below the input are two buttons: 'Close' and 'Generate API Key'.

The screenshot shows a modal window titled 'Copy your key'. It contains a warning message: '⚠️ Be sure to copy the key now. This is the only time you'll be able to see it unencrypted.' Below the message is the generated API key value: 'PMAK-6200027ea98cf00...'. At the bottom are 'Close' and 'Copy to Clipboard' buttons.

Con il valore della chiave, appena generata, procediamo alla creazione di una variabile di ambiente per le richieste da effettuare sulle Postman API.

The screenshot shows the 'Variables' section of the Postman interface. A new variable 'postman\_api\_key' has been created and is listed under the environment 'My Postman API'. The variable has a type of 'default' and a current value of 'PMAK-61f4ee187...'. Another variable, 'collection\_uid', is also listed with a current value of '1643371-196c9eb1-39d7-4724-a789-44a90f858db9'.

La variabile d'ambiente si chiama `postman_api_key` e il suo valore è richiamato nelle richieste della collection di Postman API. Ad esempio:

GET <https://api.getpostman.com/collections>

The screenshot shows a Postman request configuration for a GET request to 'https://api.getpostman.com/collections'. The 'Headers' tab is selected, showing six headers: Host, User-Agent, Accept, Accept-Encoding, Connection, and X-Api-Key. The X-Api-Key header is highlighted with a red box and has a value of '{{postman\_api\_key}}'.

Dalla risposta alla richiesta precedente otteniamo la lista degli id delle collections, ad esempio:

```
"collections": [
    {
        "id": "196c9bb1-39d7-4724-a769-44a90f858db9",
        "name": "Sample Collection 25",
        "owner": "1643371",
        "createdAt": "2022-02-06T16:17:49.000Z",
        "updatedAt": "2022-02-06T16:45:08.000Z",
        "uid": "1643371-196c9bb1-39d7-4724-a769-44a90f858db9",
        "isPublic": false
    }
]
```

Con il valore del campo uid creiamo un'altra variabile d'ambiente chiamata collection\_uid. Con questa variabile è possibile eseguire una richiesta come le seguenti:

GET [https://api.getpostman.com/collections/{collection\\_uid}](https://api.getpostman.com/collections/{collection_uid})

PUT [https://api.getpostman.com/collections/{collection\\_uid}](https://api.getpostman.com/collections/{collection_uid})

DEL [https://api.getpostman.com/collections/{collection\\_uid}](https://api.getpostman.com/collections/{collection_uid})

### 9.2.7. Autorizzazione delle API in Postman

<https://learning.postman.com/docs/sending-requests/authorization/>

no-auth: <https://learning.postman.com/docs/sending-requests/authorization/#no-auth>

API key: <https://learning.postman.com/docs/sending-requests/authorization/#api-key>

Basic Authentication:

<https://learning.postman.com/docs/sending-requests/authorization/#basic-auth>

[https://it.wikipedia.org/wiki/Basic\\_access\\_authentication](https://it.wikipedia.org/wiki/Basic_access_authentication)

Bearer Token: <https://learning.postman.com/docs/sending-requests/authorization/#bearer-token>

### 9.2.8. Curl (solo per approfondimento)

<https://curl.haxx.se/>

#### 9.2.8.1. *Curl basics*

<https://ec.haxx.se/cmdline>

#### 9.2.8.2. *Http with curl*

<https://ec.haxx.se/http/http-basics>

#### 9.2.8.3. *Curl with TLS*

<https://curl.se/docs/sslcerts.html>

<https://stackoverflow.com/questions/54938026/curl-unknown-error-0x80092012-the-revocation-function-was-unable-to-check-r>

## 9.2.9. Mock Server di REST API

[https://it.wikipedia.org/wiki/Mock\\_object](https://it.wikipedia.org/wiki/Mock_object)

Nella programmazione orientata agli oggetti, i **mock object** (*oggetti simulati od oggetti mock*) sono degli oggetti simulati che riproducono il comportamento degli oggetti reali in modo controllato. Un programmatore crea un oggetto **mock** per testare il comportamento di altri oggetti, reali, ma legati ad un oggetto inaccessibile o non implementato. Allora quest'ultimo verrà sostituito da un mock.

### 9.2.9.1. Mockaroo

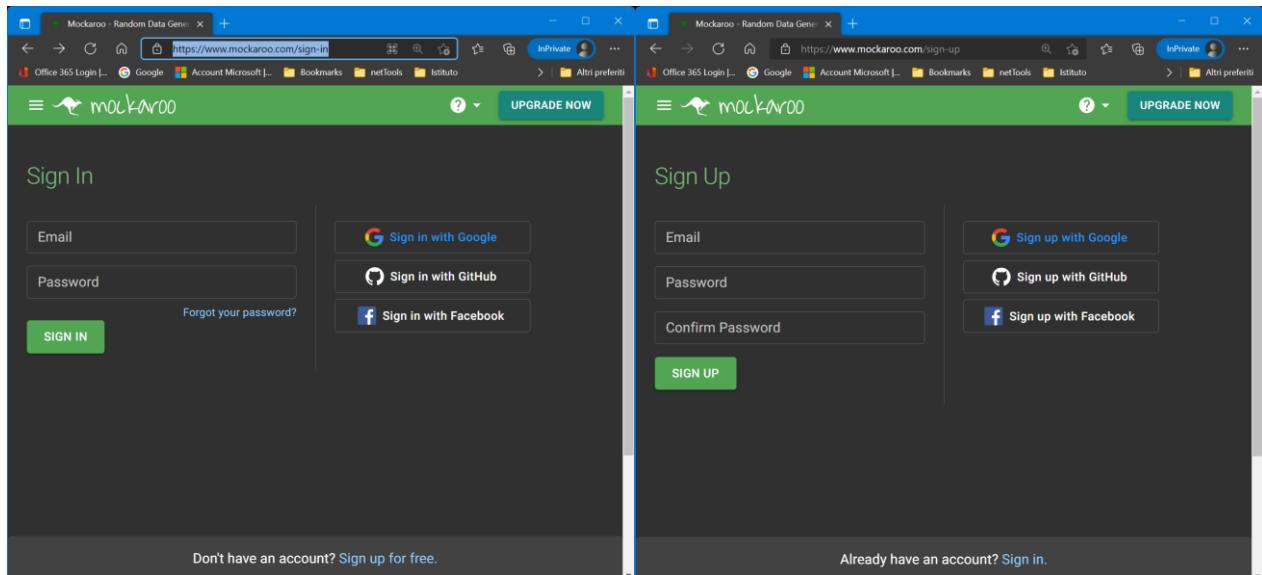
<https://www.mockaroo.com/>

Mockaroo è un servizio web che già dal piano di utilizzo gratuito consente di generare dati fintizi (fake) a partire da uno **Schema**. Inoltre, una volta generato un **Data Set**, è possibile creare un **Mock Server di REST API**, in maniera piuttosto semplice. Ovviamente, non si ha la stessa flessibilità di un'applicazione vera, ma il servizio permette di testare un'applicazione client, quando il server reale non è ancora pronto. Si pensi, ad esempio, al caso in cui si debba realizzare una nuova applicazione e si debba scrivere sia la parte client che la parte server. Se il team di sviluppo è diviso in due sotto-team, uno che realizza l'applicazione client e uno che realizza l'applicazione server, è possibile che, ad un certo punto, il sotto-team che realizza la parte client, sia più avanti nello sviluppo del sotto-team che realizza la parte server. In questo caso, per non rallentare lo sviluppo della parte client, è possibile ricorrere ad un Mock server per iniziare il testing della parte client.

#### 9.2.9.1.1. Utilizzo di Mockaroo per generare DataSet a partire da uno schema

Mockaroo tutorials: <https://www.youtube.com/playlist?list=PLKMZcxOsC3u0Y-4CHg5SDpVjTcrvGttTt>

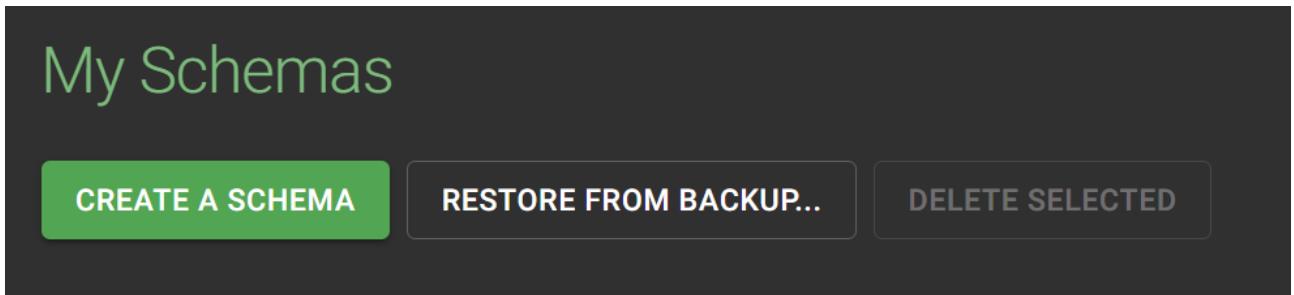
Il primo passo per iniziare ad usare Mockaroo è fare il **sign-in** con un proprio account social oppure Github al link <https://www.mockaroo.com/sign-in>, oppure fare il **sign-up** per crearne uno specifico per al link <https://www.mockaroo.com/sign-up>. Per gli scopi scolastici è consigliabile utilizzare l'account Github.



Una volta fatto il sign-in ci si ritrova nella home page del sito da cui è possibile accedere a diversi servizi:



Per generare dati fittizi, occore prima di tutto creare uno schema da cui poi derivare i dati.



## Schema Companies

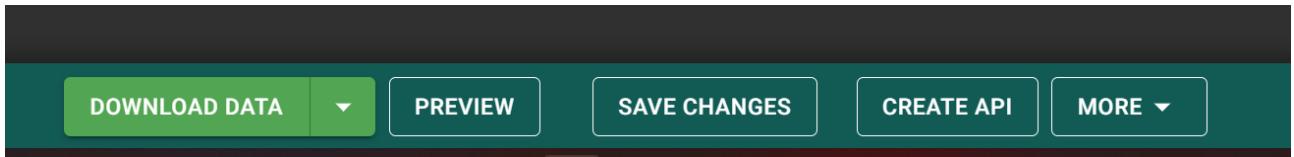
Scriviamo lo schema per la creazione di un data set che simuli un insieme di aziende. Per creare i campi all'interno dello schema è possibile modificare quelli proposti alla creazione dello schema e/o aggiungerne altri attraverso il pulsante “add another field”. Quando si clicca sul campo type di un campo viene mostrato un pannello dal quale è possibile scegliere attraverso alcune centinaia di tipi diversi di dati, per ognuno dei quali è possibile far generare automaticamente dati fittizi.

Seguendo i semplici tutorial online della piattaforma si scopre, ad esempio, che è possibile creare campi strutturati, usando la notazione JSON. Per esempio, per creare un campo headquarter che ha al suo interno i campi lat e lon che rappresentano coordinate cartesiane è possibile aggiungere i campi headquarter.lat e headquarter.lon. Per generare campi che sono a loro volta collections di lunghezza variabile è possibile ricorrere al concetto di JSON array. Ad esempio, per generare un campo che è un array di oggetti, ognuno dei quali contiene i campi city, employee\_number e state, è possibile creare un campo locations di tipo JSON array e poi aggiungere i campi locations.city, locations.employee\_number e locations.state.

E' possibile specificare il numero di rows (righe) da generare e il formato dei dati, ad esempio JSON.

Field Name	Type	Options
id	Row Number	blank: 0 % $\Sigma$ X
name	Company Name	blank: 0 % $\Sigma$ X
revenue	Number	min: 1000000 max: 100000000 decimals: 2 blank: 0 % $\Sigma$ X
headquarter.lat	Latitude	blank: 0 % $\Sigma$ X
headquarter.lon	Longitude	blank: 0 % $\Sigma$ X
locations	JSON Array	min elements: 1 max elements: 5 blank: 0 % $\Sigma$ X
locations.city	City	blank: 0 % $\Sigma$ X
locations.employee	Number	min: 10 max: 10000 decimals: 0 blank: 0 % $\Sigma$ X
locations.state	State	restrict states... Only US blank: 0 % $\Sigma$ X

Dopo aver creato lo schema è possibile avere una preview dei dati che verranno generati e salvare le modifiche fatte.



Si ottiene un array di oggetti JSON del tipo seguente:

```
{  
    "id": 1,  
    "name": "Dablist",  
    "revenue": 45925261.83,  
    "headquarter": {  
        "lat": 40.0678351,  
        "lon": 44.0788197  
    },  
    "locations": [  
        {  
            "city": "Atlanta",  
            "employee_number": 5392,  
            "state": "Georgia"  
        },  
        {  
            "city": "Austin",  
            "employee_number": 3492,  
            "state": "Texas"  
        }  
    ]  
}
```

I dati possono essere scaricati localmente oppure possono essere inseriti all'interno di un Data Set disponibile online:

My Datasets / Companies

## Companies

File  
Companies.json

Generated from Schema  
Companies

### Values

Showing only the first 20 of 100 total rows

```
{  
    "id": 1,  
    "name": "Oloo",  
    "revenue": 13034938.92,  
    "headquarter": {  
        "lat": 43.0429124,  
        "lon": 1.9038837  
    },  
    "locations": [  
        {  
            "city": "Paris",  
            "employee_number": 4392,  
            "state": "France"  
        },  
        {  
            "city": "London",  
            "employee_number": 3492,  
            "state": "United Kingdom"  
        }  
    ]  
}
```

### Update Dataset

Update the dataset associated with this schema with newly generated data.

DOWNLOAD DATA ▲

## Schema Product

E' possibile scrivere uno schema che richiama campi di un altro schema, implementando di fatto il concetto di chiave esterna (foreign key). Ad esempio, scriviamo lo schema Product che rappresenta prodotti (alimentari). Per ognuno dei prodotti c'è un campo company\_id che rappresenta la chiave

esterna sullo schema Companies

The screenshot shows the Mockaroo schema configuration interface. At the top, there's a navigation bar with 'My Schemas / Products'. Below it, the title 'Products' is displayed. The main area contains a table for defining fields:

Field Name	Type	Options
id	Row Number	blank: 0 % $\Sigma$ X
name	Product (Grocery)	blank: 0 % $\Sigma$ X
price	Number	min: 1 max: 100 decimals: 2 blank: 0 % $\Sigma$ X
company_id	Dataset Column	Companies id random blank: 0 % $\Sigma$ X

Below the table is a button 'ADD ANOTHER FIELD'. Further down, there are settings for '# Rows: 100', 'Format: JSON', and checkboxes for 'array' (checked) and 'include null values' (unchecked). A hint at the bottom says: 'Hint: Use "." in column names to generate nested json objects, brackets to generate arrays. [More information...](#)'.

Per aggiungere un campo che è chiave esterna su un altro schema basta scegliere come type di quel campo **Dataset Column** e selezionare quindi lo schema e il campo a cui puntare. Si ottengono oggetti del tipo seguente:

```
{  
  "id": 1,  
  "name": "Soup - French Onion, Dry",  
  "price": 36.27,  
  "company_id": 51  
}
```

### 9.2.9.1.2. Utilizzo di Mockaroo per la creazione di un Mock Server di REST API

Una volta creati i Data Set è possibile fare il Mock di REST API, utilizzando la sezione Mock Apis della dashboard. Ad esempio, per simulare una get sulla rotta /products.json è possibile scrivere il seguente script:

```
# Use the schema named "Products".  
schema "Products"  
  
# Generate rows.  
generate 100
```

Viene anche creato un link che contiene la rotta sull'endpoint e la chiave privata delle API. Si noti che, almeno nella versione gratuita testata, non è possibile cambiare l'APIKey e che c'è un numero massimo di richieste giornaliere che possono essere fatte con la stessa chiave (200 API requests / day).

<https://my.api.mockaroo.com/products.json?key=APIKey>

The screenshot shows the Mockaroo API editor interface. At the top, it says "My APIs / GET /products.json". Below that is a section titled "Editing API" with a "Route" header. A dropdown menu under "Route" is set to "GET" and has selected the path "/products.json". Other options in the dropdown include POST, PUT, PATCH, DELETE, and OPTIONS. The "OPTIONS" option has a tooltip: "GET /users.json" and "GET /users.json?min\_age=10&max\_age=80". Below the route configuration, there is a code editor area containing the following script:

```
# Use the schema named "Users".
schema "Products"

# Generate ten rows.
generate 100
```

Per simulare una get sulla rotta /products/:id.json, ossia per simulare la get di uno specifico prodotto di cui viene fornito l'id, si può scrivere:

The screenshot shows the Mockaroo API editor interface. At the top, it says "Route" and has a dropdown menu set to "GET" with the path "/products/:id.json". Below that is a section titled "Handler Script" containing the following script:

```
schema "Products"
data = generate 1
data['id']=params['id']
data
```

params is an hash containing the URL and query string parameters passed in on the request.

Lo script genera un oggetto di tipo Products e assegna al campo id il valore ottenuto dalla richiesta. L'ultima istruzione serve a restituire tutto l'oggetto nel body della risposta.

Si ottine un link alla rotta specificata: <https://my.api.mockaroo.com/products/123.json?key=APIKey>

Ad esempio, per accedere al prodotto con id pari a 15, la rotta è:

<https://my.api.mockaroo.com/products/15.json?key=APIKey>

Per simulare la POST sulla rotta /products.json, ossia il caricamento di un nuovo oggetto dal client verso il server si può scrivere:

## Route

POST ▾ /products.json

## Handler Script

```
schema "Products"
generate 1
data = entity
data['id'] = 1
status(201)
header 'location', "https://my.api.mockaroo.com/" + params['path']+"/1.json"
data
```

entity is a hash containing the data in the entity body of the request.

Si noti come in questo caso la risposta fornisca il codice http 201 che vuol dire “risorsa creata” e, oltre a restituire l’oggetto creato, fornisca anche l’URI dell’oggetto creato attraverso il campo location dell’header del protocollo http.

Si ottiene il link: <https://my.api.mockaroo.com/products.json?key=APIKey&method=POST>

Per simulare la PATCH sulla rotta /products/:id.json, ossia la modifica di un oggetto esistente sul server, fornendo solo la parte che va modificata nella richiesta:

## Route

PATCH ▾ /products/:id.json

## Handler Script

```
schema "Products"
data = generate 1
data['id'] = params['id']
data['name'] = entity['name']
data
```

Si ottiene il link:

<https://my.api.mockaroo.com/products/123.json?key=APIKey&method=PATCH>

Per simulare la PUT sulla rotta /products/:id.json, ossia la modifica di un oggetto esistente sul server, fornendo tutto l’oggetto da modificare nella richiesta:

**Route**

PUT ▾ /products/:id.json

**Handler Script**

```
schema "Products"
data = generate 1
data['id'] = params['id']
data['name'] = entity['name']
data['price'] = entity['price']
data['company_id'] = entity['company_id']
data
```

Si ottiene il link: <https://my.api.mockaroo.com/products/123.json?key=APIKey&method=PUT>

Per simulare la DELETE sulla rotta /products/:id.json, ossia la cancellazione sul server della risorsa con id specificato nella richiesta:

**Route**

DELETE ▾ /products/:id.json

**Handler Script**

```
{success: true}
```

Si ottiene il link:

<https://my.api.mockaroo.com/products/123.json?key=APIKey&method=DELETE>

In maniera analoga si creano le rotte /companies.json e /companies/:id.json:

Route	Route
GET ▾ /companies.json	GET ▾ /companies/:id.json
<b>Handler Script</b>	<b>Handler Script</b>
<pre>schema "Companies" generate 100</pre>	<pre>schema "Companies" data = generate 1 data['id'] = params['id'] data</pre>

Si ottengono i link:

<https://my.api.mockaroo.com/companies.json?key=APIKey>

<https://my.api.mockaroo.com/companies/123.json?key=APIKey>

### 9.2.9.1.3. Scrrittura di un client console per le Mock API di Mockaroo

<https://docs.microsoft.com/en-us/aspnet/web-api/overview/advanced/calling-a-web-api-from-a-net-client>

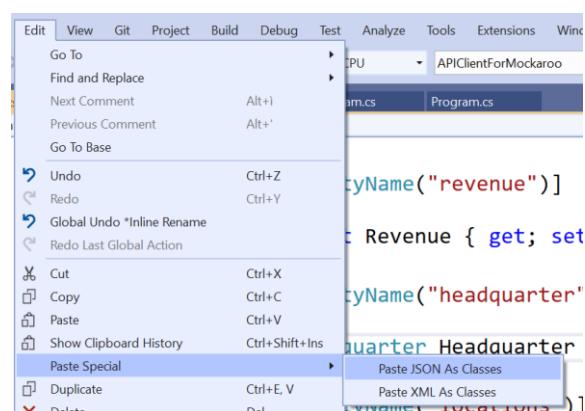
Creare una cartella Model e al suo interno creare le classi corrispondenti agli schemi Companies e Products. Si noti che le classi del Model si scrivono al singolare, mentre gli schemi sono al plurale. Per creare le classi Company e Products si può utilmente sfruttare la funzionalità di Visual Studio che permette di scrivere in automatico il model a partire da un oggetto JSON. Ad esempio, per creare la classe Company, basta copiare un oggetto Company tra quelli generati da Mockaroo:

```
{  
    "id": 1,  
    "name": "Edgetag",  
    "revenue": 42443721.05,  
    "headquarter": {  
        "lat": 14.4712676,  
        "lon": 121.0441201  
    },  
    "locations": [  
        {  
            "city": "San Diego",  
            "employee_number": 1804,  
            "state": "California"  
        },  
        {  
            "city": "Seattle",  
            "employee_number": 1842,  
            "state": "Washington"  
        },  
        {  
            "city": "Pensacola",  
            "employee_number": 1470,  
            "state": "Florida"  
        },  
        {  
            "city": "Columbus",  
            "employee_number": 8631,  
            "state": "Ohio"  
        }  
    ]  
}
```

Quindi creare un file per una nuova classe (cliccando su Add → Class) e poi, dopo aver cancellato il template della classe creata da Visual Studio, cliccare su **Edit → Paste Special → Paste JSON as Classes**. In questo modo vengono create in automatico le classi corrispondenti al model del JSON incollato. Per gestire la corrispondenza tra i nomi JSON e i nomi delle property si possono aggiungere le annotazioni come, ad esempio, `[JsonPropertyName("employee_number")]`, includendo il namespace:

```
System.Text.Json.Serialization;
```

```
//file: Company.cs
```



```

using System.Text.Json.Serialization;

namespace APIClientForMockaroo.Model
{
    public class Company
    {
        [JsonPropertyName("id")]
        public int Id { get; set; }

        [JsonPropertyName("name")]
        public string Name { get; set; }

        [JsonPropertyName("revenue")]
        public float Revenue { get; set; }

        [JsonPropertyName("headquarter")]
        public Headquarter Headquarter { get; set; }

        [JsonPropertyName("locations")]
        public Location[] Locations { get; set; }
    }

    public class Headquarter
    {
        [JsonPropertyName("lat")]
        public float Lat { get; set; }

        [JsonPropertyName("lon")]
        public float Lon { get; set; }
    }

    public class Location
    {
        [JsonPropertyName("city")]
        public string City { get; set; }

        [JsonPropertyName("employee_number")]
        public int EmployeeNumber { get; set; }

        [JsonPropertyName("state")]
        public string State { get; set; }
    }
}

//file: Product.cs

using System.Text.Json.Serialization;

namespace APIClientForMockaroo.Model
{
    public class Product
    {
        [JsonPropertyName("id")]
        public int Id { get; set; }

        [JsonPropertyName("name")]
        public string Name { get; set; }

        [JsonPropertyName("price")]

```

```

        public decimal Price { get; set; }

        [JsonPropertyName("company_id")]
        public int CompanyId { get; set; }
    }
}

//file: Program.cs

using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Net.Http.Json;
using System.Text;
using System.Text.Json;
using System.Threading.Tasks;
using APIClientForMockaroo.Model;
using HttpProxyControl;

namespace APIClientForMockaroo
{

    class Program
    {

        const string APIKeyName = "X-API-Key";
        readonly static string APIKeyValue = GetKeyFromStore();
        static string baseAddress = "https://my.api.mockaroo.com";
        static string acceptedMediaType = "application/json";
        static HttpClient client;

        static string GetKeyFromStore()
        {
            //il file è nella cartella che contiene la soluzione corrente
            //il file contiene una sola riga corrispondente alla API Key di Mockaroo
            string keyStorePath = "../../../../../MyMockarooKey.txt";
            string key = File.ReadAllText(keyStorePath);
            return key;
        }

        static void Main()
        {
            RunAsync().GetAwaiter().GetResult();
        }

        static async Task RunAsync()
        {
            HttpProxyHelper.HttpClientProxySetup(out client);

```

```

client.BaseAddress = new Uri(baseAddress);
client.DefaultRequestHeaders.Accept.Clear();
client.DefaultRequestHeaders.Accept.Add(
    new MediaTypeWithQualityHeaderValue(acceptedMediaType));
//https://stackoverflow.com/questions/53551361/how-to-add-api-key-in-request-header-using-
web-api
//https://stackoverflow.com/questions/14627399/setting-authorization-header-of-httpclient
client.DefaultRequestHeaders.Add(APIKeyName, APIKeyValue);

try
{
    Console.WriteLine("Test di POST");
    Console.WriteLine("Simuliamo la POST per inserire un prodotto sull'endpoint remoto");
    // Create a new product
    Product product = new Product
    {
        Name = "Insalata verde",
        Price = 1.55m,
        CompanyId = 34
    };
    //la POST dovrebbe restituire l'uri dell'oggetto creato e il suo id
    var url = await PostProductAsync(product);
    Console.WriteLine($"Created at {url}");
    // Get the product
    //con l'uri della risorsa creata sul server remoto recuperiamo l'oggetto
    Console.WriteLine("Il prodotto caricato sul server mediante la POST non è realmente
salvato, \n" +
                    "quindi quando recuperiamo il valore tramite la sua url otteniamo un valore
diverso");
    product = await GetProductAsync(url.PathAndQuery);
    ShowProduct(product);

    // Update the product
    Console.WriteLine("\n\nTest di PUT");
    Console.WriteLine("Updating price...");
    product.Price = 80;
    Product updatedProduct = await PutProductAsync(product);
    // Get the updated product
    //Console.WriteLine("Il prodotto aggiornato");
    //product = await GetProductAsync(url.PathAndQuery);
    ShowProduct(updatedProduct);

    // Delete the product
    Console.WriteLine("\n\nTest di DELETE");
    Console.WriteLine($"Eliminiamo il prodotto con id = {product.Id}");
    var statusCode = await DeleteProductAsync(product.Id);
    Console.WriteLine($"Deleted (HTTP Status = {(int)statusCode})");

    //otteniamo tutti i prodotti
}

```

```

        Console.WriteLine("\n\nTest di GET di tutti i prodotti");
        Console.WriteLine("Elenco dei prodotti");
        var products = await GetAllProductsAsync("products.json");
        ShowProducts(products);

        //dato un prodotto, ottenere i dati dell'azienda che produce tale prodotto
        Console.Write("Inserisci l'id di un prodotto: ");
        int productId = int.Parse(Console.ReadLine());
        Console.WriteLine("il prodotto corrispondente all'id inserito è:");
        string productUri = $"products/{productId}.json";
        product = await GetProductAsync(productUri);
        JsonSerializerOptions options = new JsonSerializerOptions() { WriteIndented = true };
        Console.WriteLine(JsonSerializer.Serialize(product, options));
        //si noti che il prodotto ottenuto non corrisponde a quello riportato con
        //lo stesso id nell'elenco dei prodotti, poiché viene generato a caso ogni volta
        string companyUri = $"companies/{product.CompanyId}.json";
        Company company = await GetCompanyAsync(companyUri);
        Console.WriteLine("L'azienda che produce il prodotto scelto è:");
        Console.WriteLine(JsonSerializer.Serialize(company, options));

        Console.WriteLine("Elenco di tutte le compagnie");
        var companies = await GetAllCompaniesAsync("/companies.json");

        companies.ForEach(c => Console.WriteLine(JsonSerializer.Serialize(c, options)+"\n"));
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }

    Console.ReadLine();
}

static void ShowProduct(Product product)
{
    Console.WriteLine($"Id: {product.Id} Name: {product.Name,-50} Price: " +
        $"{product.Price}\tCompanyId: {product.CompanyId}");
}

static void ShowProducts(List<Product> products)
{
    products.ForEach(p => ShowProduct(p));
}

#region REST_Methods_Products
/// <summary>
/// Effettua la POST di un nuovo prodotto sul server
/// </summary>
/// <param name="product">il prodotto da creare sul server</param>
/// <returns>l'URI della risorsa creata</returns>
static async Task<Uri> PostProductAsync(Product product)

```

```

{
    HttpResponseMessage response = await client.PostAsJsonAsync(
        "products.json", product);
    //verifico che l'operazione abbia avuto successo
    response.EnsureSuccessStatusCode();

    //sull'oggetto response posso fare diverse elaborazioni
    //se il server restituisce l'oggetto creato, posso recuperarlo dal body della risposta
    Product prodottoCreato = await response.Content.ReadFromJsonAsync<Product>();
    Console.WriteLine("Prodotto creato:\n" + JsonSerializer.Serialize(prodottoCreato));
    //oppure posso leggere il body della risposta come stringa e poi elaborarla
    string responseContentAsstring = await response.Content.ReadAsStringAsync();
    Product prodotto = JsonSerializer.Deserialize<Product>(responseContentAsstring);
    Console.WriteLine("Prodotto creato come stringa\n" + responseContentAsstring);
    // return URI of the created resource.
    Console.WriteLine("Headers location: " + response.Headers.Location);
    return response.Headers.Location;
}

//effettua la POST utilizzando il metodo PostAsync e l'oggetto StringContent
static async Task<Uri> PostProductAsync2(Product product)
{
    string data = JsonSerializer.Serialize(product);
    // http POST del nuovo post
    var content = new StringContent(data, Encoding.UTF8, "application/json");
    var response = await client.PostAsync($"products.json", content);
    response.EnsureSuccessStatusCode();

    //sull'oggetto response posso fare diverse elaborazioni
    //se il server restituisce l'oggetto creato, posso recuperarlo dal body della risposta
    Product prodottoCreato = await response.Content.ReadFromJsonAsync<Product>();
    Console.WriteLine("Prodotto creato:\n" + JsonSerializer.Serialize(prodottoCreato));
    //oppure posso leggere il body della risposta come stringa e poi elaborarla
    string responseContentAsstring = await response.Content.ReadAsStringAsync();
    Product prodotto = JsonSerializer.Deserialize<Product>(responseContentAsstring);
    Console.WriteLine("Prodotto creato come stringa\n" + responseContentAsstring);
    // return URI of the created resource.
    Console.WriteLine("Headers location: " + response.Headers.Location);
    return response.Headers.Location;
}
/// <summary>
/// Effettua una GET sul server per recuperare un oggetto di cui è specificato l'id nella
rotta
/// </summary>
static async Task<Product> GetProductAsync(string path)
{
    Product product = null;
    HttpResponseMessage response = await client.GetAsync(path);
}

```

```

        if (response.IsSuccessStatusCode)
    {
        product = await response.Content.ReadFromJsonAsync<Product>();
    }
    return product;
}

/// <summary>
/// Effettua una GET sul server per richiedere tutti i prodotti.
/// In questo caso si utilizza il metodo GetStreamAsync di HttpClient in abbinamento al
DeserializeAsync
    /// di JsonSerializer
    /// </summary>
    /// <param name="path"></param>
    /// <returns></returns>
static async Task<List<Product>> GetAllProductsAsync(string path)
{
    var streamTask = client.GetStreamAsync(path);
    var products = await JsonSerializer.DeserializeAsync<List<Product>>(await streamTask);
    return products;
}

static async Task<Product> PutProductAsync(Product product)
{
    HttpResponseMessage response = await client.PutAsJsonAsync(
        $"products/{product.Id}.json", product);
    response.EnsureSuccessStatusCode();
    //Console.WriteLine("Aggiornamento effettuato");
    // Deserialize the updated product from the response body.
    product = await response.Content.ReadFromJsonAsync<Product>();
    //Console.WriteLine("Prodotto aggiornato sul server:\n" +
JsonSerializer.Serialize(product));
    return product;
}

static async Task<HttpStatusCode> DeleteProductAsync(int id)
{
    HttpResponseMessage response = await client.DeleteAsync(
        $"products/{id}.json");
    return response.StatusCode;
}
#endregion REST_Methods_Products

#region REST_Methods_Companies
static async Task<Company> GetCompanyAsync(string path)
{
    return await client.GetFromJsonAsync<Company>(path);
}
static async Task<List<Company>> GetAllCompaniesAsync(string path)

```

```

{
    var streamTask = client.GetStreamAsync(path);
    var companies = await JsonSerializer.DeserializeAsync<List<Company>>(await streamTask);
    return companies;
}
static async Task<List<Company>> GetAllCompaniesAsync2(string path)
{
    var response = await client.GetAsync(path);
    var companies = await response.Content.ReadAsStringAsync();
    return JsonSerializer.Deserialize<List<Company>>(companies);
}
static async Task<List<Company>> GetAllCompaniesAsync3(string path)
{
    return await client.GetFromJsonAsync<List<Company>>(path);
}
#endregion REST_Methods_Companies
}

}

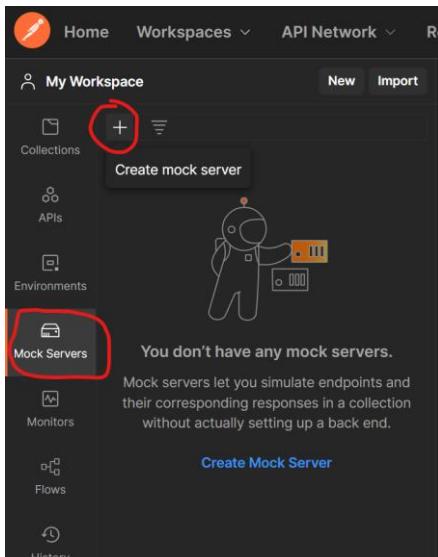
```

### 9.2.10. Postman Mock Server

[https://www.youtube.com/watch?v=n\\_7UUghLpc0](https://www.youtube.com/watch?v=n_7UUghLpc0)

<https://www.youtube.com/watch?v=pAD11I3k9q0&t=157s>

Postman può essere usato anche per fare il Mock di API endpoint, in maniera analoga a quanto fatto con Mockaroo, seppur con qualche differenza. Il primo passo per creare un server Mock in Postman è quello di creare un Mock Server:

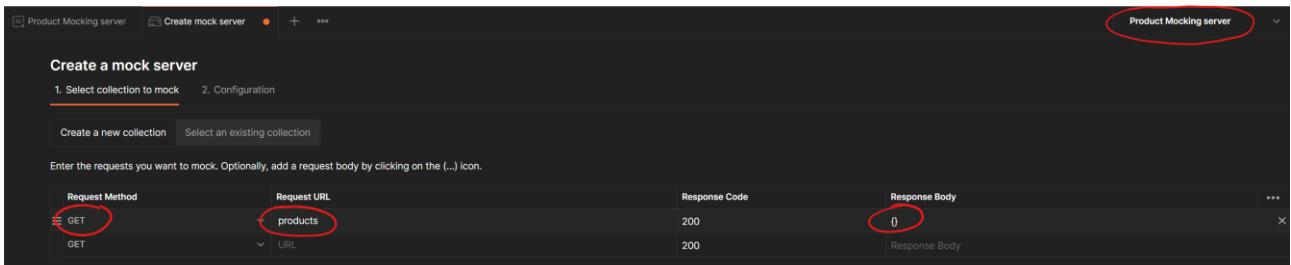


Per creare un Mock server occorre inserire almeno una rotta con un esempio. Un esempio è una descrizione di una richiesta nella quale viene specificato il valore della richiesta e il corrispondente valore della risposta.

In questo esempio vedremo come creare un Mock Server ad accesso libero che consenta di effettuare una GET per ottenere tutti i prodotti sul nostro API endpoint (gli stessi generati con Mackaroo).

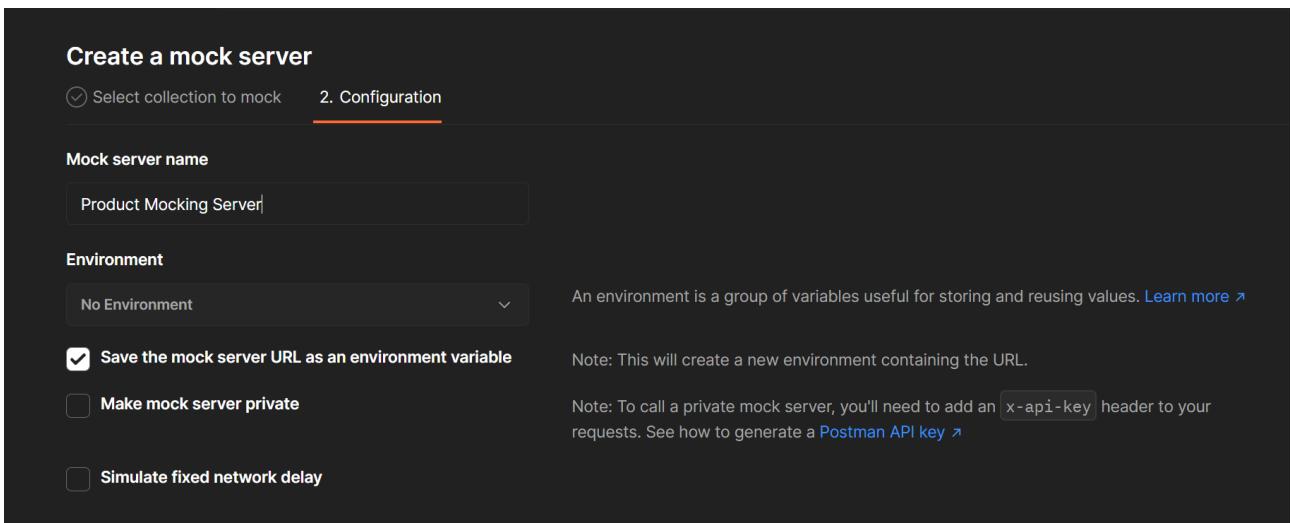
Creiamo un nuovo Mock Server:

Per creare un mock server, bisogna inserire almeno un esempio per una richiesta. Ad esempio, inseriamo una **GET /products** che ottiene come risposta l'oggetto vuoto `{}`. Successivamente andremo a dettagliare meglio la richiesta e la risposta.



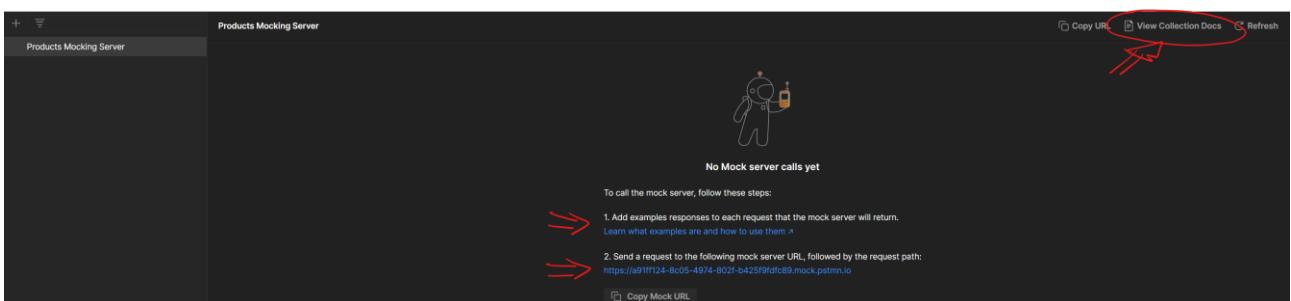
The screenshot shows the 'Create a mock server' interface in Postman. It's step 1: Select collection to mock. The 'Request Method' dropdown has 'GET' selected. The 'Request URL' dropdown has 'products' selected. The 'Response Code' dropdown has '200' selected. The 'Response Body' dropdown has '0' selected. The 'Mock server name' field contains 'Product Mocking Server'. The 'Environment' dropdown is set to 'No Environment'. There are three checkboxes: 'Save the mock server URL as an environment variable' (checked), 'Make mock server private' (unchecked), and 'Simulate fixed network delay' (unchecked). A note says: 'An environment is a group of variables useful for storing and reusing values. [Learn more](#)'.

Dopo aver cliccato su Next vediamo la pagina dove possiamo definire il nome del Mock Server:



The screenshot shows the 'Create a mock server' configuration step. The 'Mock server name' field is filled with 'Product Mocking Server'. The 'Environment' dropdown is set to 'No Environment'. The 'Save the mock server URL as an environment variable' checkbox is checked. A note says: 'Note: This will create a new environment containing the URL.' The 'Make mock server private' and 'Simulate fixed network delay' checkboxes are unchecked. A note says: 'Note: To call a private mock server, you'll need to add an `x-api-key` header to your requests. See how to generate a [Postman API key](#)'.

Chiamiamo il server **Product Mocking Server** e lasciamo la spunta sull'opzione che salva la url del server in una variabile d'ambiente. Viene creato il Mock Server e si presenta la sua pagina iniziale, riportata, a titolo d'esempio, sotto:



The screenshot shows the 'Products Mock Server' initial page. It features a cartoon character, a note 'No Mock server calls yet', and two steps to call the mock server: '1. Add examples responses to each request that the mock server will return.' and '2. Send a request to the following mock server URL, followed by the request path: <https://e9ff1f24-8c05-4974-902f-b425ffdfcd9.mock.pstmn.io>'. There are 'Copy URL' and 'View Collection Docs' buttons. A red arrow points to the 'View Collection Docs' button, and another red arrow points to the 'Copy Mock URL' button.

Postman crea automaticamente un ambiente (Environment), chiamato con lo stesso nome del Mock Server e una collection, sempre con lo stesso nome del Mock Server appena creato.

The screenshot shows the Postman interface. On the left, there's a sidebar with a '+' icon, a list of collections: 'JsonPlaceholder Demo', 'Postman API', 'Postman Echo', 'Postman Echo Copy', and 'Product Mocking Server'. Under 'Product Mocking Server', there's a 'GET products' item with a 'Default' example. The main panel shows a 'Product Mocking Server / products' endpoint with a 'GET' method and a URL template {{url}}/products. Below the URL, there are tabs for 'Params', 'Authorization', 'Headers (5)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. Under 'Headers (5)', there's a 'Query Params' section with a table:

KEY	Value
Key	

At the bottom of the main panel, there's a 'Variables' section titled 'Product Mocking Server' with a table:

VARIABLE	TYPE	INITIAL VALUE	CURRENT VALUE
url	default	https://f656ed25-0808-4eec-b895-d3e84f046cee.mock.pstmn.io	https://f656ed25-0808-4eec-b895-d3e84f046cee.mock.pstmn.io

Ora che il Mock Server è stato creato, è possibile modificare l'esempio relativo alla GET {{url}}/products e aggiungere tutti gli altri endpoint che si desidera con i rispettivi esempi.

Per fare in modo che quando si effettua una richiesta al Mock Server si ottenga come risposta un elenco di prodotti in formato JSON modifichiamo la risposta dell'esempio chiamato default e incolliamo i dati presi da Mockaroo. Impostiamo anche i parametri dell'header della risposta in modo da impostare il Content-Type al valore “application/json”.

The screenshot shows the Product Mocking Server interface with a dark theme. On the left, there is a large preview area and on the right, a smaller configuration area.

**Left Preview Area:**

- Method:** GET
- URL:** {{url}}/products ...
- Params:** Headers Body
- Query Params:**

KEY	VALUE
Key	Value
- Body:** Headers (1)
- Content:**
  - Pretty: [JSON]
  - Raw: [Text]
  - Preview: [Icon]
  - JSON: [Icon]
  - Save: [Icon]

```

1  [
2    {
3      "id": 1,
4      "name": "Onions - Vidalia",
5      "price": 37.54,
6      "company_id": 94
7    },
8    {
9      "id": 2,
10     "name": "Soap - Pine Sol Floor Cleaner",
11     "price": 5.94,
12     "company_id": 95
13   },
14   {
15     "id": 3,
16     "name": "Brandy - Bar",
17     "price": 9.39,
18     "company_id": 15
19   }

```

**Right Configuration Area:**

- Method:** GET
- URL:** {{url}}/products ...
- Params:** Headers Body
- Query Params:**

KEY	VALUE
Key	Value
- Body:** Headers (1)
- Content:**
  - KEY
  - Content-Type
  - Value

Clicchiamo sul pulsante Save per salvare l'esempio relativo alla rotta /products.

Per testare l'esempio andiamo sulla richiesta e clicchiamo sul pulsante Send. Dovremmo vedere la risposta in base all'esempio appena creato.

Possiamo aggiungere un altro esempio sulla rotta /products per andare a prendere uno specifico prodotto, con una rotta del tipo /products/:id. Procediamo con la creazione di altri esempi sulla stessa richiesta:

The image consists of three side-by-side screenshots of the Postman application interface, illustrating the configuration of API endpoints.

- Screenshot 1 (Left):** Shows the left sidebar with collections like "JsonPlaceholder Demo", "Postman API", "Postman Echo", and "Product Mocking Server". The "products" collection is expanded, showing "Get First Product" and "Get Second Product". The main panel shows a GET request for "({url})/products/1" with a response body containing a JSON object with fields "id", "name", "price", and "company\_id".
- Screenshot 2 (Middle):** Shows the "Get First Product" endpoint with a response header "Content-Type: application/json".
- Screenshot 3 (Right):** Shows the "Get Second Product" endpoint with a response header "Content-Type: application/json".

Possiamo aggiungere anche la rotta relativa alle aziende, procedendo in maniera analoga a quanto fatto per products. Avremo dunque la rotta /companies con gli stessi dati presi da Mockaroo:

The screenshot shows the Postman application interface with the following details:

- Left Sidebar:** Shows the "Product Mocking Server" collection expanded, with "GET companies" selected. It includes "Default", "Get First Company", and "TodoItemLite" items.
- Main Panel:**
  - Request:** A GET request to "({url})/companies".
  - Headers:** A single header "Content-Type: application/json" is present.
  - Body:** A response body containing a JSON array with one element. The element has fields "id", "name", "revenue", and "headquarter". The "headquarter" field is itself a nested object with "lat" and "lon" properties.

### **9.2.10.1. Utilizzo di un Postman Mock Server per il testing un'applicazione client**

Un Mock server può essere utilizzato anche al di fuori dell'ambiente Postman, per testare un'applicazione client che stiamo sviluppando. Ad esempio il Mock Server che abbiamo appena creato può essere interrogato da un qualsiasi browser web. Andando sulle proprietà del Mock Server e selezionando la voce "View Documentation" possiamo vedere l'url del mock server e delle relative rotte:

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** Shows collections like "JsonPlaceholder Demo", "Postman API", "Postman Echo", "Postman Echo Copy", and the "Product Mocking Server" collection, which is currently selected.
- Header:** Includes "Release Tag CURRENT", "Language cURL", and a "JUMP TO" dropdown with links to "Introduction", "GET products", and "GET companies".
- Main Content Area:** The title is "Product Mocking Server" with the sub-instruction "Make things easier for your teammates with a complete collection description."
- Request Section:** A "GET products" request is shown with the URL `https://f656ed25-0808-4eec-b895-d3e84f046cee.mock.pstmn.io/products`. A red arrow points from the word "URL" to this URL field.
- Example Section:** Contains a "Request" tab with a "cURL" code block:

```
curl --location --request GET 'https://f656ed25-0808-4eec-b895-d3e84f046cee.mock.pstmn.io/products'
```
- Response Section:** Contains a "Body" tab with a JSON response:

```
[  
  {  
    "id": 1,  
    "name": "Onions - Vidalia",  
    "price": 37.54,  
    "company_id": 94  
  },  
  {  
    "id": 2,  
    "name": "Soap - Pine Sol Floor Cleaner"  
  }]
```

A "View more" link is visible at the bottom of the JSON array.

Se proviamo a copiare l'url nel browser vediamo la risposta del Mock Server:

```
[  
  {  
    "id": 1,  
    "name": "Onions - Vidalia",  
    "price": 37.54,  
    "company_id": 94  
  },  
  {  
    "id": 2,  
    "name": "Soap - Pine Sol Floor Cleaner",  
    "price": 5.94,  
    "company_id": 95  
  },  
  {  
    "id": 3,  
    "name": "Brandy - Bar",  
    "price": 9.39,  
    "company_id": 15  
  },  
  {  
    "id": 4,  
    "name": "Honey - Comb",  
    "price": 26.68,  
    "company_id": 73  
  },  
  {  
    "id": 5,  
    "name": "Remy Red Berry Infusion",  
    "price": 21.53,  
    "company_id": 32  
  },  
  {  
    "id": 6,  
    "name": "Monkfish - Fresh",  
    "price": 48.64,  
    "company_id": 24  
  },  
  {  
    "id": 7,  
    "name": "Chocolate - Chips Compound",  
    "price": 28.12,  
    "company_id": 29  
  },  
  {  
    "id": 8,  
    "name": "Pork - Bacon",  
    "price": 12.5,  
    "company_id": 10  
  }]
```

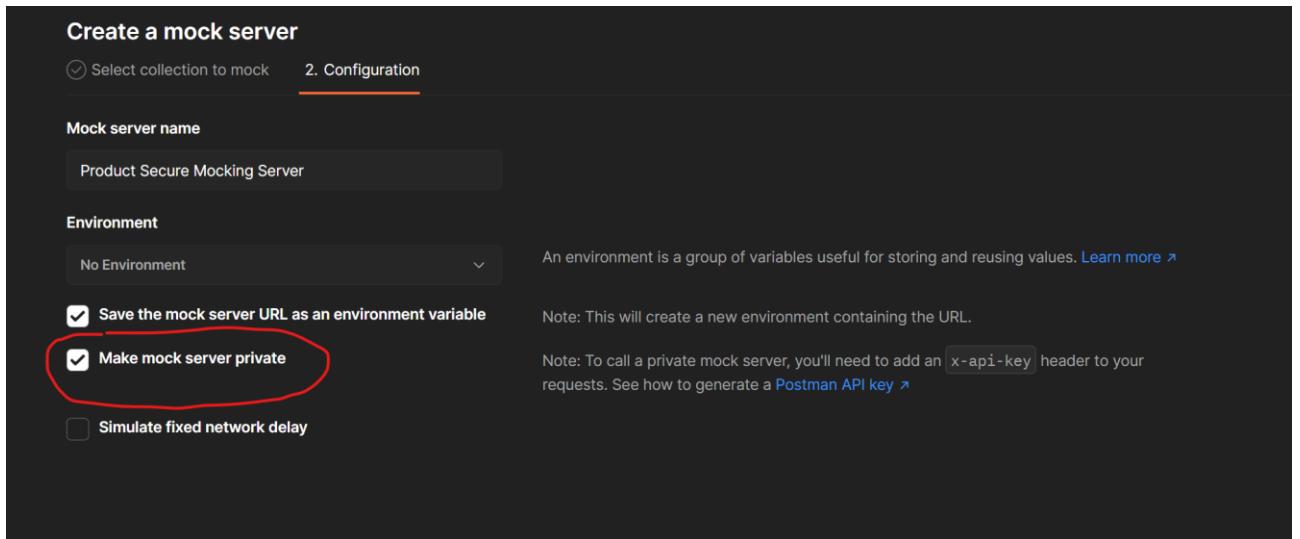
Questo vuol dire che possiamo scrivere un'applicazione web utilizzando in prima battuta il Mock

Server di Postman prima ancora che sia pronto il vero server REST API. Ovviamente non è pensabile usare un mock server in produzione, perché un mock server simula il funzionamento di un server, ma non è un vero server. Ad esempio, un mock server non salva davvero i dati che potremmo caricare con una POST oppure una PUT.

### 9.2.10.2. Postman Mock Server privato

Postman consente anche di creare dei Mock Server privati, che utilizzano un meccanismo di autenticazione basato su credenziali di accesso per poter effettuare chiamate REST agli endpoint del mock server. Ad esempio, per creare un Product Secure Mocking Server possiamo procedere come descritto di seguito:

- 1) creiamo un mock server come visto nei paragrafi precedenti, ma nelle opzioni di creazione del server mettiamo la spunta sull'opzione



- 2) Inseriamo una Postman API Key (generata nel modo mostrato nei paragrafi precedenti) nell'header delle richieste http sotto forma di campo x-api-key
- 3) Procediamo alla creazione delle collection di richieste GET, POST, PUT, DELETE analogamente a quanto fatto negli esempi precedenti.

Ad esempio:

The screenshot shows the Postman application interface. On the left, the sidebar displays 'My Workspace' with a list of collections, APIs, environments, mock servers, monitors, flows, and history. The main workspace shows a collection named 'Product Secure Mocking Server' with several endpoints listed under 'Get Companies'. One endpoint, 'Get Company with Id=73', is selected and expanded, showing its details. The request method is 'GET' with the URL path '{{url}}/companies/73'. The 'Body' tab is selected, displaying the response body in pretty-printed JSON format:

```

1  {
2   "id": 73,
3   "name": "Eare",
4   "revenue": 97669310.01,
5   "headquarter": {
6     "lat": 38.63333,
7     "lon": 21.36667
8   },
9   "locations": [
10    {
11      "city": "Tallahassee",
12      "employee_number": 9897,
13      "state": "Florida"
14    },
15    {
16      "city": "Dallas",
17      "employee_number": 8457,
18      "state": "Texas"
19    },
20    {
21      "city": "Dayton",
22      "employee_number": 7597,
23      "state": "Ohio"
24    }
25  ]
26

```

Tra gli esempi sono stati inseriti la GET di un prodotto con id=4 e la GET di una compagnia con id=73 corrispondente al company\_id del prodotto con id=4.

### 9.2.10.3. Scrrittura di un client console per le Mock API di Postman

<https://learning.postman.com/docs/designing-and-developing-your-api/mocking-data/setting-up-mock/>

<https://learning.postman.com/docs/sending-requests/authorization/>

<https://learning.postman.com/docs/developer/intro-api/>

Usando lo stesso procedimento visto nel paragrafo 9.2.9.1.3 procediamo alla scrittura del client console per il Product Secure Mocking Server. I file del model sono gli stessi utilizzati per il server di Mockaroo.

```
//file Company.cs
using System.Text.Json.Serialization;
```

```
namespace APIClientForPostmanMockServer.Model
```

```

{
    public class Company
    {
        [JsonPropertyName("id")]
        public int Id { get; set; }

        [JsonPropertyName("name")]
        public string Name { get; set; }

        [JsonPropertyName("revenue")]
        public float Revenue { get; set; }

        [JsonPropertyName("headquarter")]
        public Headquarter Headquarter { get; set; }

        [JsonPropertyName("locations")]
        public Location[] Locations { get; set; }
    }

    public class Headquarter
    {
        [JsonPropertyName("lat")]
        public float Lat { get; set; }

        [JsonPropertyName("lon")]
        public float Lon { get; set; }
    }

    public class Location
    {
        [JsonPropertyName("city")]
        public string City { get; set; }

        [JsonPropertyName("employee_number")]
        public int EmployeeNumber { get; set; }

        [JsonPropertyName("state")]
        public string State { get; set; }
    }
}

```

```

//file Product.cs

using System.Text.Json.Serialization;

namespace APIClientForPostmanMockServer.Model
{
    public class Product
    {
        [JsonPropertyName("id")]
        public int Id { get; set; }

        [JsonPropertyName("name")]
        public string Name { get; set; }

        [JsonPropertyName("price")]
        public decimal Price { get; set; }

        [JsonPropertyName("company_id")]
        public int CompanyId { get; set; }
    }
}

```

```

        }
    }

//file Program.cs

using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Net.Http.Json;
using System.Text;
using System.Text.Json;
using System.Threading.Tasks;
using APIClientForPostmanMockServer.Model;
using HttpProxyControl;
using System.Text.Json.Serialization;
namespace APIClientForPostmanMockServer
{

    public class PostmanStore
    {
        [JsonPropertyName("api_key")]
        public string APIKeyValue { get; set; }

        [JsonPropertyName("base_address")]
        public string BaseAddress { get; set; }
    }

    class Program
    {
        const string APIKeyName = "X-API-Key";
        static PostmanStore postmanStore = GetDataFromStore();
        static readonly string APIKeyValue = postmanStore.APIKeyValue;
        static string baseAddress = postmanStore.BaseAddress;
        static string acceptedMediaType = "application/json";
        static HttpClient client;

        static PostmanStore GetDataFromStore()
        {
            //il file è nella cartella che contiene la soluzione corrente
            //il file contiene un oggetto JSON del tipo:
            //{
            //    "api_key": "api_key_value",
            //    "base_address": "base_address_value"
            //}
            string keyStorePath = "../../../../../MyPostmanStore.json";
            string store = File.ReadAllText(keyStorePath);
            PostmanStore postmanStore = JsonSerializer.Deserialize<PostmanStore>(store);
        }
    }
}

```

```

        return postmanStore;
    }

static void Main()
{
    RunAsync().GetAwaiter().GetResult();
}

static async Task RunAsync()
{
    HttpProxyHelper.HttpClientProxySetup(out client);

    client.BaseAddress = new Uri(baseAddress);
    client.DefaultRequestHeaders.Accept.Clear();
    client.DefaultRequestHeaders.Accept.Add(
        new MediaTypeWithQualityHeaderValue(acceptedMediaType));
    //https://stackoverflow.com/questions/53551361/how-to-add-api-key-in-request-header-using-
    web-api
    //https://stackoverflow.com/questions/14627399/setting-authorization-header-of-httpclient
    client.DefaultRequestHeaders.Add(APIKeyName, APIKeyValue);

    try
    {
        Console.WriteLine("Test di POST");
        Console.WriteLine("Simuliamo la POST per inserire un prodotto sull'endpoint remoto");
        // Create a new product
        Product product = new Product
        {
            Name = "Insalata verde",
            Price = 1.55m,
            CompanyId = 34
        };
        //la POST dovrebbe restituire l'uri dell'oggetto creato e il suo id
        var url = await PostProductAsync(product);
        Console.WriteLine($"Created at {url}");
        // Get the product
        //con l'uri della risorsa creata sul server remoto recuperiamo l'oggetto
        Console.WriteLine("Il prodotto caricato sul server mediante la POST non è realmente
salvato, \n" +
            "quindi quando recuperiamo il valore tramite la sua url otteniamo un valore
diverso");
        product = await GetProductAsync(url.PathAndQuery);
        ShowProduct(product);

        // Update the product
        Console.WriteLine("\n\nTest di PUT");
        Console.WriteLine("Updating price...");
        product.Price = 80;
        Product updatedProduct = await PutProductAsync(product);
    }
}

```

```

// Get the updated product
//Console.WriteLine("Il prodotto aggiornato");
//product = await GetProductAsync(url.PathAndQuery);
ShowProduct(updatedProduct);

// Delete the product
Console.WriteLine("\n\nTest di DELETE");
Console.WriteLine($"Eliminiamo il prodotto con id = {product.Id}");
var statusCode = await DeleteProductAsync(product.Id);
Console.WriteLine($"Deleted (HTTP Status = {(int)statusCode})");

//otteniamo tutti i prodotti
Console.WriteLine("\n\nTest di GET di tutti i prodotti");
Console.WriteLine("Elenco dei prodotti");
var products = await GetAllProductsAsync("products");
ShowProducts(products);

//dato un prodotto, ottenere i dati dell'azienda che produce tale prodotto
Console.Write("Recuperiamo il prodotto con id =4: ");
int productId = 4;
Console.WriteLine("il prodotto corrispondente all'id inserito è:");
string productUri = $"products/{productId}";
product = await GetProductAsync(productUri);
JsonSerializerOptions options = new JsonSerializerOptions() { WriteIndented = true };
Console.WriteLine(JsonSerializer.Serialize(product, options));
//si noti che il prodotto ottenuto non corrisponde a quello riportato con
//lo stesso id nell'elenco dei prodotti, poiché viene generato a caso ogni volta
string companyUri = $"companies/{product.CompanyId}";

Company company = await GetCompanyAsync(companyUri);
Console.WriteLine("L'azienda che produce il prodotto scelto è:");
Console.WriteLine(JsonSerializer.Serialize(company, options));

Console.WriteLine("Elenco di tutte le compagnie");
var companies = await GetAllCompaniesAsync("/companies");

companies.ForEach(c => Console.WriteLine(JsonSerializer.Serialize(c, options) +
"\n"));
}

catch (Exception e)
{
    Console.WriteLine(e.Message);
}

Console.ReadLine();
}

static void ShowProduct(Product product)
{
    Console.WriteLine($"Id: {product.Id} Name: {product.Name,-50} Price: " +

```

```

        $"{{product.Price}}\tCompanyId: {{product.CompanyId}}");
    }

    static void ShowProducts(List<Product> products)
    {
        products.ForEach(p => ShowProduct(p));
    }

#region REST_Methods_Products
/// <summary>
/// Effettua la POST di un nuovo prodotto sul server
/// </summary>
/// <param name="product">il prodotto da creare sul server</param>
/// <returns>l'URI della risorsa creata</returns>
static async Task<Uri> PostProductAsync(Product product)
{
    HttpResponseMessage response = await client.PostAsJsonAsync(
        "products", product);
    //verifico che l'operazione abbia avuto successo
    response.EnsureSuccessStatusCode();

    //sull'oggetto response posso fare diverse elaborazioni
    //se il server restituisce l'oggetto creato, posso recuperarlo dal body della risposta
    Product prodottoCreato = await response.Content.ReadFromJsonAsync<Product>();
    Console.WriteLine("Prodotto creato:\n" + JsonSerializer.Serialize(prodottoCreato));
    //oppure posso leggere il body della risposta come stringa e poi elaborarla
    string responseContentAsstring = await response.Content.ReadAsStringAsync();
    Product prodotto = JsonSerializer.Deserialize<Product>(responseContentAsstring);
    Console.WriteLine("Prodotto creato come stringa\n" + responseContentAsstring);
    // return URI of the created resource.
    Console.WriteLine("Headers location: " + response.Headers.Location);
    return response.Headers.Location;
}

//effettua la POST utilizzando il metodo PostAsync e l'oggetto StringContent
static async Task<Uri> PostProductAsync2(Product product)
{
    string data = JsonSerializer.Serialize(product);

    // http POST del nuovo post
    var content = new StringContent(data, Encoding.UTF8, "application/json");

    var response = await client.PostAsync($"products", content);

    response.EnsureSuccessStatusCode();

    //sull'oggetto response posso fare diverse elaborazioni
    //se il server restituisce l'oggetto creato, posso recuperarlo dal body della risposta
}

```

```

        Product prodottoCreato = await response.Content.ReadFromJsonAsync<Product>();
        Console.WriteLine("Prodotto creato:\n" + JsonSerializer.Serialize(prodottoCreato));
        //oppure posso leggere il body della risposta come stringa e poi elaborarla
        string responseContentAsstring = await response.Content.ReadAsStringAsync();
        Product prodotto = JsonSerializer.Deserialize<Product>(responseContentAsstring);
        Console.WriteLine("Prodotto creato come stringa\n" + responseContentAsstring);
        // return URI of the created resource.
        Console.WriteLine("Headers location: " + response.Headers.Location);
        return response.Headers.Location;
    }

    /// <summary>
    /// Effettua una GET sul server per recuperare un oggetto di cui è specificato l'id nella
    rotta
    /// </summary>
    /// <param name="path"></param>
    /// <returns></returns>
    static async Task<Product> GetProductAsync(string path)
    {
        Product product = null;
        HttpResponseMessage response = await client.GetAsync(path);
        if (response.IsSuccessStatusCode)
        {
            product = await response.Content.ReadFromJsonAsync<Product>();
        }
        return product;
    }

    /// <summary>
    /// Effettua una GET sul server per richiedere tutti i prodotti.
    /// In questo caso si utilizza il metodo GetStreamAsync di HttpClient in abbinamento al
    DeserializeAsync
    /// di JsonSerializer
    /// </summary>
    /// <param name="path"></param>
    /// <returns></returns>
    static async Task<List<Product>> GetAllProductsAsync(string path)
    {
        var streamTask = client.GetStreamAsync(path);
        var products = await JsonSerializer.DeserializeAsync<List<Product>>(await streamTask);
        return products;
    }

    static async Task<Product> PutProductAsync(Product product)
    {
        HttpResponseMessage response = await client.PutAsJsonAsync(
            $"products/{product.Id}", product);
        response.EnsureSuccessStatusCode();
        //Console.WriteLine("Aggiornamento effettuato");
        // Deserialize the updated product from the response body.
    }
}

```

```

        product = await response.Content.ReadFromJsonAsync<Product>();
        //Console.WriteLine("Prodotto aggiornato sul server:\n" +
JsonSerializer.Serialize(product));
        return product;
    }

    static async Task<HttpStatusCode> DeleteProductAsync(int id)
{
    HttpResponseMessage response = await client.DeleteAsync(
        $"products/{id}");
    return response.StatusCode;
}
#endregion REST_Methods_Products

#region REST_Methods_Companies
static async Task<Company> GetCompanyAsync(string path)
{
    return await client.GetFromJsonAsync<Company>(path);
}
static async Task<List<Company>> GetAllCompaniesAsync(string path)
{
    var streamTask = client.GetStreamAsync(path);
    var companies = await JsonSerializer.DeserializeAsync<List<Company>>(await streamTask);
    return companies;
}
static async Task<List<Company>> GetAllCompaniesAsync2(string path)
{
    var response = await client.GetAsync(path);
    var companies = await response.Content.ReadAsStringAsync();
    return JsonSerializer.Deserialize<List<Company>>(companies);
}
static async Task<List<Company>> GetAllCompaniesAsync3(string path)
{
    return await client.GetFromJsonAsync<List<Company>>(path);
}
#endregion REST_Methods_Companies
}

}

```

## 9.3. Esempi di servizi REST

### 9.3.1. Weather API

<https://openweathermap.org>

<https://openweathermap.org/api>

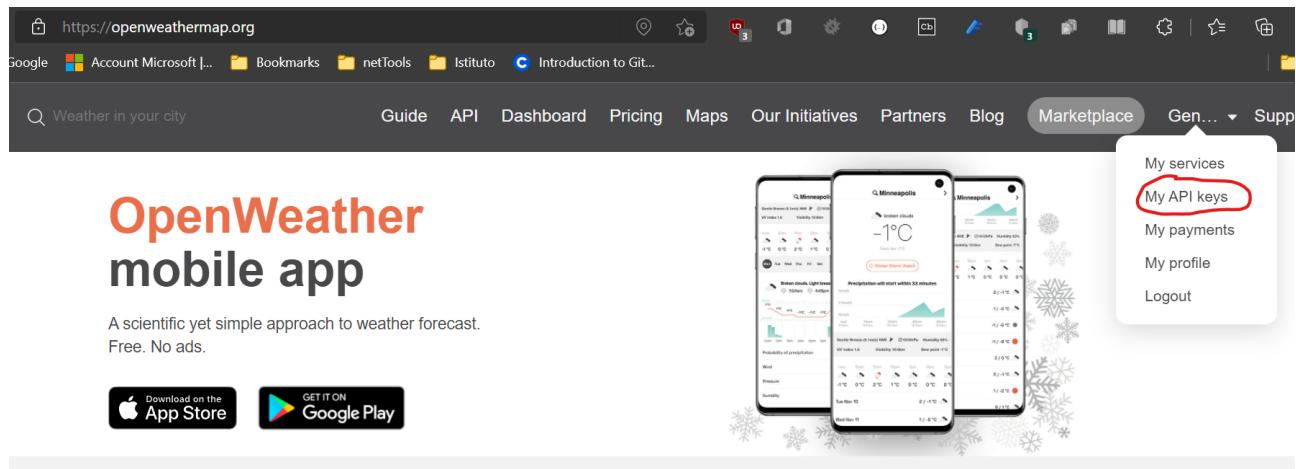
<https://openweathermap.org/current>

<https://openweathermap.org/api/hourly-forecast>

#### 9.3.1.1. Un esempio di client console per il servizio OpenWeatherMap

Per utilizzare le API di OpenWeatherMap occorre creare un account gratuito al link: [https://home.openweathermap.org/users/sign\\_up](https://home.openweathermap.org/users/sign_up). Dopo aver creato l'account occorre creare una API Key da associare alle richieste che verranno effettuate dall'applicazione client che scriveremo.

Per la creazione di una chiave basta collegarsi al link: [https://home.openweathermap.org/api\\_keys](https://home.openweathermap.org/api_keys), come evidenziato nella figura seguente:



Si possono creare anche più chiavi, basta inserire il nome che si vuole assegnare alla chiave e cliccare sul bottone "Generate":

A screenshot of the 'Create key' section in the OpenWeatherMap API keys management interface. It shows a table with columns for Key, Name, Status, Actions, and a 'Create key' button. There are two existing keys: one named 'ff' (Default, Active) and another named '4' (MagicWeather, Active). The 'Create key' button has a red arrow pointing to it, and the entire button area is circled in red. A message box at the top says: 'You can generate as many API keys as needed for your subscription. We accumulate the total load from all of them.'

Per poter scrivere un client per le API di OpenWeatherMap occorre prima di tutto creare il modello dei dati restituiti dalle API. In questo esempio verranno illustrati due casi particolari: **il recupero dei dati meteo correnti (il tempo atmosferico corrente)** e **il recupero delle previsioni meteo a 7 giorni**.

#### 9.3.1.1.1. URL Encoding

Negli esempi seguenti verranno scritte URL contenenti stringhe di testo. È importante osservare che, affinché le URL siano valide, devono contenere solo caratteri ASCII e non devono contenere spazi. Nel caso in cui in una stringa siano presenti spazi oppure caratteri non esprimibili con l'ASCII standard, si dovrà ricorrere ad un URL Encoding, come previsto dalle tabelle riportate qui:

[https://www.w3schools.com/tags/ref\\_urlencode.asp](https://www.w3schools.com/tags/ref_urlencode.asp)

In C# esiste il metodo `HttpUtility.UrlEncode()` per effettuare l'URL encoding di una stringa

### 9.3.1.1.2. Recupero dei dati relativi al tempo atmosferico corrente

Dalla pagina della documentazione delle API: <https://openweathermap.org/current> scopriamo che per poter effettuare una richiesta GET per ottenere il tempo atmosferico corrente di una data località, occorre eseguire una GET verso l'endpoint remoto:

`api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}`

#### Parameters

lat required Geographical coordinates (latitude, longitude). If you need the geocoder to automatic convert city names and zip-codes to geo coordinates and the other way around, please use our [Geocoding API](#).

appid required Your unique API key (you can always find it on your account page under the ["API key" tab](#))

mode optional Response format. Possible values are xml and html. If you don't use the mode parameter format is JSON by default. [Learn more](#)

<https://openweathermap.org/current#format>

units optional Units of measurement. standard, metric and imperial units are available. If you do not use the units parameter, standard units will be applied by default. [Learn more](#)

<https://openweathermap.org/weather-data>

lang optional You can use this parameter to get the output in your language. [Learn more](#)  
<https://openweathermap.org/current#multi>

#### 9.3.1.1.2.1. OpenWeatherMap Geocoding service

Le API di OpenWeatherMap forniscono anche un servizio di Geocoding diretto e inverso, documentato a questo link: <https://openweathermap.org/api/geocoding-api>. Infatti, sebbene sia possibile effettuare una richiesta specificando direttamente il nome di una località, questo tipo di query è deprecato.

Per ottenere il geocoding osserviamo che i dati restituiti dall'API:

`http://api.openweathermap.org/geo/1.0/direct?q={city name},{state code},{country code}&limit={limit}&appid={API key}`

sono del tipo seguente:

<https://api.openweathermap.org/geo/1.0/direct?q=Monticello%20Brianza,it&limit=5&appid={API Key}>

[  
{

```

        "name": "Monticello Brianza",
        "lat": 45.7087797,
        "lon": 9.3141406,
        "country": "IT",
        "state": "Lombardy"
    },
]

```

Se si lascia limit al valore massimo (5) e si inserisce una località presente in più posti, si ottengono più risultati. Ad esempio, con la richiesta:

<https://api.openweathermap.org/geo/1.0/direct?q=Monticello,it&limit=5&appid={API Key}>

```
[
{
    "name": "Monticello",
    "lat": 42.7962244,
    "lon": 12.4798518,
    "country": "IT",
    "state": "Umbria"
},
{
    "name": "Monticello",
    "lat": 43.5191175,
    "lon": 11.6458579,
    "country": "IT",
    "state": "Tuscany"
},
{
    "name": "Monticello",
    "lat": 45.3963973,
    "lon": 11.5989684,
    "country": "IT",
    "state": "Veneto"
},
{
    "name": "Monticello",
    "lat": 45.7346613,
    "lon": 9.3920896,
    "country": "IT",
    "state": "Lombardy"
},
{
    "name": "Monticello",
    "lat": 45.3758865,
    "lon": 8.5968126,
    "country": "IT",
    "state": "Piemont"
}
]
```

]

Per ottenere il modello dei dati possiamo procedere come già visto in precedenza, sfruttando la funzione di Visual Studio che permette di creare un Model direttamente dai dati in formato JSON: nella cartella **CurrentWeatherModel** creiamo la classe **Location**:

//File: Location.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace OpenWeatherMapClient.CurrentWeatherModel
{
    public class Location
    {
        public string Name { get; set; }
        public float Lat { get; set; }
        public float Lon { get; set; }
        public string Country { get; set; }
        public string State { get; set; }
    }
}
```

### **9.3.1.1.2.2. Recupero dei dati relativi alle condizioni meteo correnti**

Per il modello dei dati restituiti dalla API

<https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&units={units}&appid={OpenWeatherMapKey}>

<https://api.openweathermap.org/data/2.5/weather?lat=45.70878&lon=9.31414&units=metric&appid={API Key}>

Osserviamo che, provando a fare una GET per “Monticello Brianza”, otteniamo una risposta in formato JSON come la seguente:

```
{
  "coord": {
    "lon": 9.3141,
    "lat": 45.7088
  },
  "weather": [
    {
      "id": 803,
      "main": "Clouds",
      "description": "broken clouds",
      "icon": "04d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 9.47,
    "feels_like": 9.47,
    "temp_min": 6.98,
```

```

    "temp_max": 10.81,
    "pressure": 1029,
    "humidity": 36
  },
  "visibility": 10000,
  "wind": {
    "speed": 0.45,
    "deg": 94
  },
  "clouds": {
    "all": 70
  },
  "dt": 1644674549,
  "sys": {
    "type": 2,
    "id": 2019808,
    "country": "IT",
    "sunrise": 1644647399,
    "sunset": 1644684249
  },
  "timezone": 3600,
  "id": 6534829,
  "name": "Monticello Brianza",
  "cod": 200
}

```

Con il solito procedimento ricaviamo il modello dei dati:

```
//File: CurrentWeatherForecasts.cs

using System.Text.Json.Serialization;

namespace OpenWeatherMapClient.CurrentWeatherModel
{

  public class CurrentWeatherForecasts
  {
    public Coord Coord { get; set; }
    public Weather[] Weather { get; set; }

    [JsonPropertyName("_base")]
    public string Base { get; set; }
    public Main Main { get; set; }
    public int Visibility { get; set; }
    public Wind Wind { get; set; }
    public Clouds Clouds { get; set; }
    public int Dt { get; set; }
    public Sys Sys { get; set; }
    public int Timezone { get; set; }
    public int Id { get; set; }
    public string Name { get; set; }
    public int Cod { get; set; }
  }

  public class Coord
  {
    public float Lon { get; set; }
  }
}
```

```

        public float Lat { get; set; }

    }

public class Main
{
    public float Temp { get; set; }

    [JsonPropertyName("feels_like")]
    public float FeelsLike { get; set; }

    [JsonPropertyName("temp_min")]
    public float TempMin { get; set; }

    [JsonPropertyName("temp_max")]
    public float TempMax { get; set; }
    public int Pressure { get; set; }
    public int Humidity { get; set; }

    [JsonPropertyName("sea_level")]
    public int SeaLevel { get; set; }

    [JsonPropertyName("grnd_level")]

    public int GrndLevel { get; set; }
}

public class Wind
{
    public float Speed { get; set; }
    public int Deg { get; set; }
    public float Gust { get; set; }
}

public class Clouds
{
    public int All { get; set; }
}

public class Sys
{
    public int Type { get; set; }
    public int Id { get; set; }
    public string Country { get; set; }
    public int Sunrise { get; set; }
    public int Sunset { get; set; }
}

public class Weather
{
    public int Id { get; set; }
    public string Main { get; set; }
    public string Description { get; set; }
    public string Icon { get; set; }
}

}

```

A questo punto possiamo scrivere il codice che permette di effettuare una chiamata con l'API delle condizioni meteo correnti:

```

//File: Program.cs
using HttpProxyControl;
using OpenWeatherMapClient.CurrentWeatherModel;
//using OpenWeatherMapClient.OneCallForecastModel;

```

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Http;
using System.Net.Http.Json;
using System.Text.Json;
using System.Text.Json.Serialization;
using System.Threading.Tasks;
using System.Web;

namespace OpenWeatherMapClient
{
    public class OpenWeatherMapStore
    {
        [JsonPropertyName("api_key")]
        public string APIKeyValue { get; set; }

    }

    class Program
    {

        static OpenWeatherMapStore openWeatherMapStore = GetDataFromStore();
        static readonly string openWeatherMapKey = openWeatherMapStore.APIKeyValue;
        static HttpClient client;

        static OpenWeatherMapStore GetDataFromStore()
        {
            //il file è nella cartella che contiene la soluzione corrente
            //il file contiene un oggetto JSON del tipo:
            //{
            //    "api_key": "api_key_value"
            //}
            string keyStorePath = "../../../../../MyOpenWeatherMapStore.json";
            string store = File.ReadAllText(keyStorePath);
            OpenWeatherMapStore openWeatherMapStore = JsonSerializer.Deserialize<OpenWeatherMapStore>(store);
            return openWeatherMapStore;
        }

        static async Task Main(string[] args)
        {
            HttpProxyHelper.HttpClientProxySetup(out client);
            await EsempioDatiMeteoCorrenti();
            //await EsempioPrevisioniMeteo();
        }

        static async Task EsempioDatiMeteoCorrenti()
        {
            Console.WriteLine("\n*****");
            Console.WriteLine("Motodo: EsempioDatiMeteoCorrenti");
            Console.WriteLine("*****\n");
            //https://openweathermap.org/api
            //https://openweathermap.org/current
            //esempio: api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}
            //esempio: api.openweathermap.org/data/2.5/weather?q={city name},{state code}&appid={API key}
            //Importante:
            //l'utilizzo del built-in geocoding è deprecato,
            //occorre invocare una API di geocoding specifica per ottenere le coordinate gps di una località
            //Quindi occorre prima conoscere le coordinate di una località mediante l'API call:
            // https://openweathermap.org/api/geocoding-api
            // api.openweathermap.org/geo/1.0/direct?q={city name},{state code},{country
            code}&limit={limit}&appid={API key}
            //lo state code è usato solo per gli USA
            //Una volta ottenute le coordinate gps delle località è possibile invocare l'API per ottenere i
            dati meteo per quella località
            //esempio: api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}
            string city = "Monticello Brianza";
            string cityName = HttpUtility.UrlEncode(city);
            string countryCode = "it";
            int limit = 5;
            string geocodingUrl =
                $"https://api.openweathermap.org/geo/1.0/direct?q={cityName},{countryCode}&limit={limit}&appid={openWeatherMa
            pKey}";
        }
    }
}

```

```
try
{
    var responseGeocoding = await client.GetAsync($"{geocodingUrl}");
    if (responseGeocoding.IsSuccessStatusCode)
    {
        List<Location> geocodingResult = await
responseGeocoding.Content.ReadFromJsonAsync<List<Location>>();
        string units = "metric";
        string lang = "it";
        float lat = geocodingResult[0].Lat;
        float lon = geocodingResult[0].Lon;
        string addressUrl =
"https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&units={units}&lang={lang}&appid={openWe
atherMapKey}";
        var response = await client.GetAsync($"{addressUrl}");
        if (response.IsSuccessStatusCode)
        {
            //https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-
how-to?pivots=dotnet-6-0#httpclient-and-httpcontent-extension-methods
            //HttpClient già usa le impostazioni web defaults come JsonSerializerOptions.
            CurrentWeatherForecasts weatherForecasts = await
response.Content.ReadFromJsonAsync<CurrentWeatherForecasts>();
            //per stampare a console usiamo i web defaults
            JsonSerializerOptions options = new JsonSerializerOptions(JsonSerializerDefaults.Web)
{ WriteIndented = true };
            Console.WriteLine("Dati ricevuti dall'endpoint remoto:\n" +
JsonSerializer.Serialize(weatherForecasts, options));
            Console.WriteLine($"Tempo attuale di {city}:
{weatherForecasts.Weather[0].Description}");
            Console.WriteLine($"Temperatura attuale di {city}: {weatherForecasts.Main.Temp}\t
temperatura percepita: {weatherForecasts.MainFeelsLike}");
        }
    }
    catch (Exception ex)
    {
        if (ex is HttpRequestException || ex is ArgumentException)
        {
            Console.WriteLine(ex.Message + "\nIl recupero dei dati dal server non è riuscito");
        }
    }
}
```

#### **9.3.1.1.3. Previsioni meteo a 7 giorni e previsioni dettagliate a 48 ore**

Sfruttando l'API descritta al link <https://openweathermap.org/api/one-call-api> è possibile ottenere le seguenti informazioni dettagliate:

- Current weather
  - Minute forecast for 1 hour
  - Hourly forecast for 48 hours
  - Daily forecast for 7 days
  - National weather alerts
  - Historical weather data for the previous 5 days

Si tratta di tantissime informazioni e per questo motivo l'API permette di filtrare quali dati ricevere e quali no. L'endpoint remoto è:

```
https://api.openweathermap.org/data/2.5/onecall?lat={lat}&lon={lon}&exclude={part}&appid={API key}
```

## Parameters

---

lat, lon required Geographical coordinates (latitude, longitude)

appid required Your unique API key (you can always find it on your account page under the "["API key" tab](#))

exclude optional By using this parameter you can exclude some parts of the weather data from the API response. It should be a comma-delimited list (without spaces).

Available values:

- current
- minutely
- hourly
- daily
- alerts

units optional Units of measurement. standard, metric and imperial units are available. If you do not use the units parameter, standard units will be applied by default. [Learn more](#)

lang optional You can use the lang parameter to get the output in your language. [Learn more](#)

Nell'esempio sviluppato di seguito useremo la seguente API:

<https://api.openweathermap.org/data/2.5/onecall?lat={lat}&lon={lon}&exclude=minutely&units={units}&lang={lang}&appid={openWeatherMapKey}>

Per la deserializzazione dei dati procediamo con il solito metodo: effettuiamo una get sull'endpoint remoto nel browser, copiamo la risposta e da questa creiamo in automatico le classi del model, con la funzione di Visual Studio "Paste JSON as Classes". Si ossevi che questo procedimento non è infallibile, poiché i valori che sono restituiti da una GET potrebbero avere un valore che non consente una modellizzazione corretta. Ad esempio, se un parametro assume un valore intero, Visual Studio lo tratterà come int, ma potrebbe succedere che in realtà il valore di quel parametro possa assumere dei valori frazionari in future chiamate. Occorre, quindi, analizzare bene la documentazione per accertarsi che i dati siano stati modellati correttamente.

Facendo una GET all'endpoint remoto:

<https://api.openweathermap.org/data/2.5/onecall?lat=45.70878&lon=9.31414&exclude=minutely&units=metric&lang=it&appid={API Key}>

```

    "lat": 45.7088,
    "lon": 9.3141,
    "timezone": "Europe/Rome",
    "timezone_offset": 3600,
    "current": {
        "dt": 1644737739,
        "sunrise": 1644733709,
        "sunset": 1644770738,
        "temp": 1.33,
        "feels_like": 1.33,
        "pressure": 1029,
        "humidity": 70,
        "dew_point": -3.13,
        "uvi": 0.45,
        "clouds": 72,
        "visibility": 10000,
        "wind_speed": 0.13,
        "wind_deg": 119,
        "wind_gust": 0.51,
        "weather": [
            {
                "id": 803,
                "main": "Clouds",
                "description": "nubi sparse",
                "icon": "04d"
            }
        ],
        "hourly": [ ... ], // 48 items
        "daily": [ ... ] // 8 items
    }
}

```

si ottiene un risultato come quello della figura a fianco. Come si vede dalla risposta, ci sono diverse informazioni che riguardano il tempo espresso come UTC (tempo coordinato universale). Questo tempo è espresso mediante Unix Epoch TimeStamp, ossia come numero di secondi a partire dal primo gennaio 1970: [https://en.wikipedia.org/wiki/Unix\\_time](https://en.wikipedia.org/wiki/Unix_time)

Per convertire il tempo espresso come Unix Epoch Time in un DateTime locale, come fa, ad esempio, il tool <https://www.epochconverter.com/> si può implementare una semplice funzione in C# come la seguente:

```

/// <summary>
/// https://stackoverflow.com/questions/249760/how-can-i-convert-a-unix-timestamp-to-datetime-and-vice-versa
/// https://stackoverflow.com/a/250400
/// https://www.epochconverter.com/
/// </summary>
/// <param name="unixTimeStamp"></param>
/// <returns></returns>
static DateTime UnixTimeStampToDateTimne(double unixTimeStamp)
{
    // Unix timestamp is seconds past epoch
    DateTime dateTimne = new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc);
    dateTimne = dateTimne.AddSeconds(unixTimeStamp).ToLocalTime();
    return dateTimne;
}

```

Il modello dei dati per i dati restituiti dall'API One Call, escludendo i valori minuto per minuto, è inserito nella cartella **OneCallForecastsModel**:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.Json.Serialization;
using System.Threading.Tasks;

namespace OpenWeatherMapClient.OneCallForecastModel
{
    //https://openweathermap.org/api/one-call-api#hist_parameter
    public class Forecast
    {
        public double Lat { get; set; }

```

```

        public double Lon { get; set; }
        public string Timezone { get; set; }

        [JsonPropertyName("timezone_offset")]
        public uint TimezoneOffset { get; set; }
        public Current Current { get; set; }
        public Hourly[] Hourly { get; set; }
        public Daily[] Daily { get; set; }
    }

    public class Current
    {
        public uint Dt { get; set; }
        public uint Sunrise { get; set; }
        public uint Sunset { get; set; }
        public float Temp { get; set; }

        [JsonPropertyName("feels_like")]
        public float FeelsLike { get; set; }
        public float Pressure { get; set; }
        public float Humidity { get; set; }

        [JsonPropertyName("dew_point")]
        public float DewPoint { get; set; }
        public float Uvi { get; set; }
        public float Clouds { get; set; }
        public float Visibility { get; set; }

        [JsonPropertyName("wind_speed")]
        public float WindSpeed { get; set; }

        [JsonPropertyName("wind_deg")]
        public float WindDeg { get; set; }

        [JsonPropertyName("wind_gust")]
        public float WindGust { get; set; }
        public Weather[] Weather { get; set; }
    }

    public class Weather
    {
        public int Id { get; set; }
        public string Main { get; set; }
        public string Description { get; set; }
        public string Icon { get; set; }
    }

    public class Hourly
    {
        public uint Dt { get; set; }
        public float Temp { get; set; }

        [JsonPropertyName("feels_like")]
        public float FeelsLike { get; set; }
        public float Pressure { get; set; }
        public float Humidity { get; set; }

        [JsonPropertyName("dew_point")]
        public float DewPoint { get; set; }
        public float Uvi { get; set; }
        public float Clouds { get; set; }
        public float Visibility { get; set; }

        [JsonPropertyName("wind_speed")]
    }

```

```

        public float WindSpeed { get; set; }

        [JsonPropertyName("wind_deg")]
        public float WindDeg { get; set; }

        [JsonPropertyName("wind_gust")]
        public float WindGust { get; set; }
        public Weather[] Weather { get; set; }
        public float Pop { get; set; }
    }

    public class Daily
    {
        public uint Dt { get; set; }
        public uint Sunrise { get; set; }
        public uint Sunset { get; set; }
        public uint Moonrise { get; set; }
        public uint Moonset { get; set; }

        [JsonPropertyName("moon_phase")]
        public float MoonPhase { get; set; }
        public Temp Temp { get; set; }

        [JsonPropertyName("feels_like")]
        public FeelsLike FeelsLike { get; set; }
        public float Pressure { get; set; }
        public float Humidity { get; set; }

        [JsonPropertyName("dew_point")]
        public float DewPoint { get; set; }

        [JsonPropertyName("wind_speed")]
        public float WindSpeed { get; set; }

        [JsonPropertyName("wind_deg")]
        public float WindDeg { get; set; }

        [JsonPropertyName("wind_gust")]
        public float WindGust { get; set; }
        public Weather[] Weather { get; set; }
        public float Clouds { get; set; }
        public float Pop { get; set; }
        public float Uvi { get; set; }
        public float Rain { get; set; }
        public float Snow { get; set; }
    }

    public class Temp
    {
        public float Day { get; set; }
        public float Min { get; set; }
        public float Max { get; set; }
        public float Night { get; set; }
        public float Eve { get; set; }
        public float Morn { get; set; }
    }

    public class FeelsLike
    {
        public float Day { get; set; }
        public float Night { get; set; }
        public float Eve { get; set; }
        public float Morn { get; set; }
    }

```

```

    }
}

}

```

Con il modello creato, possiamo ottenere le previsioni meteo dettagliate, come descritto nel seguente metodo:

```

static async Task EsempioPrevisioniMeteo()
{
    Console.WriteLine("\n*****");
    Console.WriteLine("Metodo: EsempioPrevisioniMeteo");
    Console.WriteLine("*****\n");
    //https://openweathermap.org/api
    //https://openweathermap.org/api/one-call-api
    //esempio: api.openweathermap.org/data/2.5/onecall?lat={lat}&lon={lon}&exclude={part}&appid={API
key}
    //esempio: api.openweathermap.org/data/2.5/onecall?lat=33.44&lon=-
94.04&exclude=hourly,daily&appid={API key}
    //coordinate GPS di Monticello Brianza
    //https://www.coordinate-gps.it/
    string city = "Monticello Brianza";
    string cityName = HttpUtility.UrlEncode(city);
    string countryCode = "it";
    int limit = 5;
    string geocodingUrl =
 $"https://api.openweathermap.org/geo/1.0/direct?q={cityName},{countryCode}&limit={limit}&appid={openWeatherMa
pKey}";
    try
    {
        var responseGeocoding = await client.GetAsync($"{geocodingUrl}");
        if (responseGeocoding.IsSuccessStatusCode)
        {
            List<Location> geocodingResult = await
responseGeocoding.Content.ReadFromJsonAsync<List<Location>>();
            float lat = geocodingResult[0].Lat;
            float lon = geocodingResult[0].Lon;
            string units = "metric";
            string lang = "it";
            string addressUrl =
 $"https://api.openweathermap.org/data/2.5/onecall?lat={lat}&lon={lon}&exclude=minutely&units={units}&lang={la
ng}&appid={openWeatherMapKey}";
            var response = await client.GetAsync($"{addressUrl}");

            if (response.IsSuccessStatusCode)
            {
                //https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-
how-to?pivots=dotnet-6-0#httpclient-and-httpcontent-extension-methods
                //HttpClient già usa le impostazioni web defaults come JsonSerializerOptions.
                Forecast forecast = await response.Content.ReadFromJsonAsync<Forecast>();
                //per stampare a console usiamo i web defaults
                JsonSerializerOptions options = new JsonSerializerOptions(JsonSerializerDefaults.Web)
{ WriteIndented = true };
                Console.WriteLine("Dati ricevuti dall'endpoint remoto:\n" +
JsonSerializer.Serialize(forecast, options));

                //stampa condizioni meteo attuali
                Console.WriteLine($"Previsioni meteo di {city}");
                Console.WriteLine("Condizioni atmosferiche correnti:");
                Console.WriteLine($"Data e ora {UnixTimeStampToDateTIme(forecast.Current.Dt)}\t" +
                    $"Temperatura: {forecast.Current.Temp}°C\tUmidità:
{forecast.Current.Humidity}\t" +
                    $"Pressione: {forecast.Current.Pressure}hPa\tCondizione meteo attuale:
{forecast.Current.Weather[0].Description}");

                //stampa previsioni meteo giornaliere (daily)
                Console.WriteLine($"Previsioni meteo giornaliere a 7 giorni di {city}");
                Array.ForEach(forecast.Daily, (d) => {
                    Console.WriteLine($"Data: {UnixTimeStampToDateTIme(d.Dt).ToShortDateString()}" +
                        $" Il sole sorge alle ore:
{UnixTimeStampToDateTIme(d.Sunrise).ToShortTimeString()}" +
                        $" e tramonta alle ore
{UnixTimeStampToDateTIme(d.Sunset).ToShortTimeString()}");
                    Console.WriteLine($"Temperatura del giorno: {d.Temp.Day}°C\tUmidità:
{d.Humidity}\t" +
                        $"Umidità: {d.Umidity}\t" +
                        $"Pressione: {d.Pressure}hPa\tCondizione meteo attuale:
{d.Weather[0].Description}");
                });
            }
        }
    }
}

```

```

        $"Pressione atmosferica:{d.Pressure}hPa\t Previsione meteo: ");
        foreach (var weather in d.Weather)
        {
            Console.WriteLine($" {weather.Description}, ");
        }
        Console.WriteLine("\n");
    });

    //stampa previsioni meteo, ora per ora (hourly)
    Console.WriteLine($"Previsioni meteo di {city}, ora per ora nelle prossime 48 ore");
    Array.ForEach(forecast.Hourly, (h) => {
        Console.WriteLine($"Data e ora: {UnixTimeStampToDate(h.Dt)}");
        Console.WriteLine($" \tTemperatura del giorno: {h.Temp}°C \tUmidità: {h.Humidity}% \t" +
            $"Pressione atmosferica: {h.Pressure}hPa\t Previsione meteo: ");
        foreach (var weather in h.Weather)
        {
            Console.WriteLine($" {weather.Description}, ");
        }
        Console.WriteLine("\n");
    });
}

}
catch (Exception ex)
{
    if (ex is HttpRequestException || ex is ArgumentException)
    {
        Console.WriteLine(ex.Message + "\nIl recupero dei dati dal server non è riuscito");
    }
}
}

```

### 9.3.2. Bing Maps API

Bing Maps API: <https://www.microsoft.com/en-us/maps/choose-your-bing-maps-api>

Bing Maps REST Services: <https://docs.microsoft.com/en-us/bingmaps/rest-services/>

Getting Started with Bing Maps REST Services:

<https://docs.microsoft.com/en-us/bingmaps/rest-services/getting-started-with-the-bing-maps-rest-services>

<https://docs.microsoft.com/en-us/bingmaps/getting-started/bing-maps-dev-center-help/getting-a-bing-maps-key>

Portale per gestire le API Keys di Bing Maps: <https://www.bingmapsportal.com/> (richiede un account Microsoft)

Esempi di Bing Maps Geocoding:

<https://docs.microsoft.com/en-us/bingmaps/rest-services/locations/find-a-location-by-address>

<https://docs.microsoft.com/en-us/bingmaps/rest-services/locations/find-a-location-by-query>

<https://docs.microsoft.com/en-us/bingmaps/rest-services/common-response-description>

<https://docs.microsoft.com/en-us/bingmaps/rest-services/locations/location-data>

#### 9.3.2.1. Un esempio di client console per Bing Maps API

Dopo aver creato un account e una chiave per il servizio di mappe di Bing Maps, in maniera analoga a quanto fatto per OpenWeather Map, procediamo alla scrittura di un client console che utilizza alcune API del servizio Bing Maps. In particolare useremo il servizio di geocoding by address e geocoding

by query per poi scaricare una mappa statica centrata nel punto scelto.

Facendo una query del tipo:

[https://dev.virtualearth.net/REST/v1/Locations/IT/23876/Monticello+Brianza+\(Lc\)/Via+dei+Mille%2c+27?includeNeighborhood=1&include=ciso2&maxResults=5&key={API Key}](https://dev.virtualearth.net/REST/v1/Locations/IT/23876/Monticello+Brianza+(Lc)/Via+dei+Mille%2c+27?includeNeighborhood=1&include=ciso2&maxResults=5&key={API Key})

otteniamo una risposta del tipo:

```
{
  "authenticationResultCode": "ValidCredentials",
  "brandLogoUri": "http://dev.virtualearth.net/Branding/logo_powered_by.png",
  "copyright": "Copyright © 2022 Microsoft and its suppliers. All rights reserved. This API cannot be accessed and the content and any results may not be used, reproduced or transmitted in any manner without express written permission from Microsoft Corporation.",
  "resourceSets": [
    {
      "estimatedTotal": 1,
      "resources": [
        {
          "__type": "Location:http://schemas.microsoft.com/search/local/ws/rest/v1",
          "bbox": [45.70312, 9.30391, 45.70479, 9.30639],
          "name": "Via dei Mille, 23876 Monticello Brianza",
          "point": {
            "type": "Point",
            "coordinates": [45.70401, 9.30493]
          },
          "address": {
            "addressLine": "Via dei Mille",
            "adminDistrict": "Lombardy",
            "adminDistrict2": "Lecco",
            "countryRegion": "Italy",
            "formattedAddress": "Via dei Mille, 23876 Monticello Brianza",
            "locality": "Monticello Brianza",
            "postalCode": "23876",
            "countryRegionIso2": "IT"
          },
          "confidence": "High",
          "entityType": "RoadBlock",
          "geocodePoints": [
            {
              "type": "Point",
              "coordinates": [45.70401, 9.30493],
              "calculationMethod": "None",
              "usageTypes": ["Display"]
            }
          ],
          "matchCodes": ["Good"]
        }
      ]
    },
    "statusCode": 200,
    "statusDescription": "OK",
    "traceId": "ea0b191e628d47b093e77942982dabd6|DU00002754|0.0.0.1|Ref A: 59FBE76E04044E959D358CC719DE980F Ref B: DB3EDGE3009 Ref C: 2022-02-13T08:16:24Z"
  }
}
```

Procedendo con il solito modo per creare il modello dei dati, necessario per la deserializzazione, abbiamo, nella cartella **Model**:

```
//File: Location.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.Json.Serialization;
using System.Threading.Tasks;

namespace BingMapsAPIClient.Model
```

```

{
    public class Location
    {
        public string AuthenticationResultCode { get; set; }
        public string BrandLogoUri { get; set; }
        public string Copyright { get; set; }
        public ResourceSet[] ResourceSets { get; set; }
        public int StatusCode { get; set; }
        public string StatusDescription { get; set; }
        public string TraceId { get; set; }
    }

    public class ResourceSet
    {
        public int EstimatedTotal { get; set; }
        public Resource[] Resources { get; set; }
    }

    public class Resource
    {
        [JsonPropertyName("_type")]
        public string __Type { get; set; }
        public float[] Bbox { get; set; }
        public string Name { get; set; }
        public Point Point { get; set; }
        public Address Address { get; set; }
        public string Confidence { get; set; }
        public string EntityType { get; set; }
        public GeocodePoint[] GeocodePoints { get; set; }
        public string[] MatchCodes { get; set; }
    }

    public class Point
    {
        public string Type { get; set; }
        public double[] Coordinates { get; set; }
    }

    public class Address
    {
        public string AddressLine { get; set; }
        public string AdminDistrict { get; set; }
        public string AdminDistrict2 { get; set; }
        public string CountryRegion { get; set; }
        public string FormattedAddress { get; set; }
        public string Locality { get; set; }
        public string PostalCode { get; set; }
        public string CountryRegionIso2 { get; set; }
    }

    public class GeocodePoint
    {
        public string Type { get; set; }
        public double[] Coordinates { get; set; }
        public string CalculationMethod { get; set; }
        public string[] UsageTypes { get; set; }
    }
}

```

Con questo modello possiamo effettuare la deserializzazione dei dati, come mostrato nel seguente

esempio:

```
using System;
using System.Diagnostics;
using System.IO;
using System.Net.Http;
using System.Net.Http.Json;
using System.Text.Json;
using System.Text.Json.Serialization;
using System.Threading.Tasks;
using System.Web;
using BingMapsAPIClient.Model;
using HttpProxyControl;
namespace BingMapsAPIClient
{
    public class BingMapsStore
    {
        [JsonPropertyName("api_key")]
        public string APIKeyValue { get; set; }

    }
    class Program
    {
        static BingMapsStore bingMapsStore = GetDataFromStore();
        static readonly string bingMapsAPIKey = bingMapsStore.APIKeyValue;
        static HttpClient client;

        static BingMapsStore GetDataFromStore()
        {
            //il file è nella cartella che contiene la soluzione corrente
            //il file contiene un oggetto JSON del tipo:
            //{
            //    "api_key": "api_key_value"
            //}
            string keyStorePath = "../../../../../MyBingMapsStore.json";
            string store = File.ReadAllText(keyStorePath);
            BingMapsStore bingMapsStore = JsonSerializer.Deserialize<BingMapsStore>(store);
            return bingMapsStore;
        }

        static async Task EsempioDiFindLocationByAddress()
        {
            Console.WriteLine("Motodo: EsempioDiFindLocationByAddress");
            //https://docs.microsoft.com/en-us/bingmaps/rest-services/locations/find-a-location-by-address
            //esempio:
            https://dev.virtualearth.net/REST/v1/Locations/FR/{postalCode}/{locality}/{addressLine}?includeNeighborhood={includeNeighborhood}&include={includeValue}&maxResults={maxResults}&key={bingMapsAPIKey}
            //https://docs.microsoft.com/en-us/bingmaps/rest-services/locations/find-a-location-by-address#api-parameters
            string postalCode = "23876";
            string locality = HttpUtility.UrlEncode("Monticello Brianza (Lc)");
            string addressLine = HttpUtility.UrlEncode("Via dei Mille, 27");
            int includeNeighborhood = 1;
            string includeValue = "ciso2";
            int maxResults = 5;
            string addressUrl =
                $"https://dev.virtualearth.net/REST/v1/Locations/IT/{postalCode}/{locality}/{addressLine}?includeNeighborhood={includeNeighborhood}&include={includeValue}&maxResults={maxResults}&key={bingMapsAPIKey}";
            try
            {
                HttpResponseMessage response = await client.GetAsync(addressUrl);
                Location data = await response.Content.ReadFromJsonAsync<Location>();
                //per stampare a console usiamo i web defaults
                JsonSerializerOptions options = new JsonSerializerOptions(JsonSerializerDefaults.Web) { WriteIndented = true };
                Console.WriteLine($"Dati recuperati dall'endpoint:\n {JsonSerializer.Serialize(data, options)}");
                Point point = data.ResourceSets[0].Resources[0].Point;
                Console.WriteLine($"Coordinate del punto: lat = {point.Coordinates[0]}, lon = {point.Coordinates[1]}");
            }
        }
    }
}
```

```

        catch (Exception ex)
    {
        if (ex is HttpRequestException || ex is ArgumentException)
        {
            Console.WriteLine(ex.Message + "\nIl recupero dei dati dal server non è riuscito");
        }
    }
}

static async Task<Point> FindLocationByAddress(string countryCode, string postalCode, string locality, string addressLine)
{
    //https://docs.microsoft.com/en-us/bingmaps/rest-services/locations/find-a-location-by-address
    //esempio:
    https://dev.virtualearth.net/REST/v1/Locations/FR/{postalCode}/{locality}/{addressLine}?includeNeighborhood={includeNeighborhood}&include={includeValue}&maxResults={maxResults}&key={bingMapsAPIKey}
    //https://docs.microsoft.com/en-us/bingmaps/rest-services/locations/find-a-location-by-
    address#api-parameters

    locality = HttpUtility.UrlEncode(locality);
    addressLine = HttpUtility.UrlEncode(addressLine);
    int includeNeighborhood = 1;
    string includeValue = "ciso2";
    int maxResults = 5;
    string addressUrl =
    $"https://dev.virtualearth.net/REST/v1/Locations/{countryCode}/{postalCode}/{locality}/{addressLine}?includeN
eighborhood={includeNeighborhood}&include={includeValue}&maxResults={maxResults}&key={bingMapsAPIKey}";
    Point point = null;
    try
    {
        HttpResponseMessage response = await client.GetAsync(addressUrl);
        Location data = await response.Content.ReadFromJsonAsync<Location>();
        point = data.ResourceSets[0].Resources[0].Point;
    }
    catch (Exception ex)
    {
        if (ex is HttpRequestException || ex is ArgumentException)
        {
            Debug.WriteLine(ex.Message + "\nIl recupero dei dati dal server non è riuscito");
        }
    }
    return point;
}

static async Task EsempioDiFindLocationByQuery()
{
    Console.WriteLine("Motodo: EsempioDiFindLocationByQuery");
    //https://docs.microsoft.com/en-us/bingmaps/rest-services/locations/find-a-location-by-query
    //esempio:
    https://dev.virtualearth.net/REST/v1/Locations/{locationQuery}?includeNeighborhood={includeNeighborhood}&maxR
esults={maxResults}&include={includeValue}&key={BingMapsAPIKey}
    //https://docs.microsoft.com/en-us/bingmaps/rest-services/locations/find-a-location-by-query#api-
parameters
    int includeNeighborhood = 1;
    string includeValue = "queryParse,ciso2";
    int maxResults = 5;
    string locationQuery = HttpUtility.UrlEncode("Via dei Mille 27, 23876 Monticello Brianza (Lc),
Italia");
    string addressUrl =
    $"https://dev.virtualearth.net/REST/v1/Locations/{locationQuery}?includeNeighborhood={includeNeighborhood}&ma
xResults={maxResults}&include={includeValue}&key={bingMapsAPIKey}";

    try
    {
        HttpResponseMessage response = await client.GetAsync(addressUrl);
        Location data = await response.Content.ReadFromJsonAsync<Location>();
        Point point = data.ResourceSets[0].Resources[0].Point;
        //per stampare a console usiamo i web defaults
        JsonSerializerOptions options = new JsonSerializerOptions(JsonSerializerDefaults.Web) {
            WriteIndented = true };

```

```

        Console.WriteLine($"Dati recuperati dall'endpoint:\n {JsonSerializer.Serialize(data,
options)}");
        Console.WriteLine($"Coordinate del punto: lat = {point.Coordinates[0]}, lon =
{point.Coordinates[1]}");
    }
    catch (Exception ex)
    {
        if (ex is HttpRequestException || ex is ArgumentException)
        {
            Console.WriteLine(ex.Message + "\nIl recupero dei dati dal server non è riuscito");
        }
    }
}

static async Task<Point> FindLocationByQuery(string queryString)
{
    //https://docs.microsoft.com/en-us/bingmaps/rest-services/locations/find-a-location-by-query
    //esempio:
    https://dev.virtualearth.net/REST/v1/Locations/{locationQuery}?includeNeighborhood={includeNeighborhood}&maxR
esults={maxResults}&include={includeValue}&key={BingMapsAPIKey}
    //https://docs.microsoft.com/en-us/bingmaps/rest-services/locations/find-a-location-by-query#api-
parameters
    int includeNeighborhood = 1;
    string includeValue = "queryParse,ciso2";
    int maxResults = 5;
    string locationQuery = HttpUtility.UrlEncode(queryString);
    string addressUrl =
 $"https://dev.virtualearth.net/REST/v1/Locations/{locationQuery}?includeNeighborhood={includeNeighborhood}&m
axResults={maxResults}&include={includeValue}&key={bingMapsAPIKey}";
    Point point = null;
    try
    {
        HttpResponseMessage response = await client.GetAsync(addressUrl);
        Location data = await response.Content.ReadFromJsonAsync<Location>();
        point = data.ResourceSets[0].Resources[0].Point;
    }
    catch (Exception ex)
    {
        if (ex is HttpRequestException || ex is ArgumentException)
        {
            Console.WriteLine(ex.Message + "\nIl recupero dei dati dal server non è riuscito");
        }
    }
    return point;
}

/// <summary>
/// Scrive un file nel percorso di destinazione a partire da uno stream di input
/// </summary>
/// <param name="filePathDestination"></param>
/// <param name="inputStream"></param>
/// <returns></returns>
static async Task WriteBinaryAsync(string filePathDestination, Stream inputStream, int
writeBufferSize = 4096)
{
    using FileStream outputStream = new FileStream(filePathDestination,
        FileMode.Create, FileAccess.Write, FileShare.None,
        bufferSize: writeBufferSize, useAsync: true);
    await inputStream.CopyToAsync(outputStream);
}
static async Task ScaricaMappaDaBingMaps(string mapUrl, string pathToFile)
{
    try
    {
        HttpResponseMessage response = await client.GetAsync(mapUrl);
        response.EnsureSuccessStatusCode();
        using Stream streamToReadFrom = await response.Content.ReadAsStreamAsync();
        //copio in modalità async il file
        await WriteBinaryAsync(pathToFile, streamToReadFrom);
    }
}

```

```

        catch (HttpRequestException e)
    {
        Console.WriteLine("\nException Caught!");
        Console.WriteLine("Message :{0} ", e.Message);
    }
}

static async Task ScaricaMappaStatica(string center, string pathToFile)
{
    //https://docs.microsoft.com/en-us/bingmaps/rest-services/imagery/get-a-static-map
    string mapCenter = HttpUtility.UrlEncode(center);
    int zoomLevel = 17;
    string mapSize = HttpUtility.UrlEncode("1024, 1024");
    string mapUrl =
 $"https://dev.virtualearth.net/REST/v1/Imagery/Map/Aerial/{mapCenter}/{zoomLevel}?mapSize={mapSize}&key={bing
MapsAPIKey}";

    await ScaricaMappaDaBingMaps(mapUrl, pathToFile);
}
static async Task EsempioDiScaricamentoMappaStaticaByAddress()
{
    //scarichiamo una mappa che ha come centro il punto definito mediante l'indirizzo dato
    Point punto = await FindLocationByAddress("IT", "23876", "Monticello Brianza (Lc)", "Via dei
Mille, 27");
    Console.WriteLine($"Le coordinate del punto corrispondente all'indirizzo fornito sono (lat, lon):
{punto.Coordinates[0]}, {punto.Coordinates[1]}");
    string fileName = "mappaByAddress.jpg";
    Console.WriteLine($"Scarico una mappa statica centrata sul punto. Controlla sul desktop
l'immagine {fileName}");
    string pathToFile = Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory) +
Path.DirectorySeparatorChar + fileName;
    await ScaricaMappaStatica($"{punto.Coordinates[0]},{punto.Coordinates[1]}", pathToFile);
}

static async Task EsempioDiScaricamentoMappaStaticaByQuery()
{
    string query = "Via dei Mille 27, 23876 Monticello Brianza (Lc), Italia";
    Point punto = await FindLocationByQuery(query);
    Console.WriteLine($"Le coordinate del punto corrispondente all'indirizzo fornito sono (lat, lon):
{punto.Coordinates[0]}, {punto.Coordinates[1]}");
    string fileName = "mappaByQuery.jpg";
    Console.WriteLine($"Scarico una mappa statica centrata sul punto. Controlla sul desktop
l'immagine {fileName}");
    string pathToFile = Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory) +
Path.DirectorySeparatorChar + fileName;
    await ScaricaMappaStatica($"{punto.Coordinates[0]},{punto.Coordinates[1]}", pathToFile);
}
static async Task Main(string[] args)
{
    HttpProxyHelper.HttpClientProxySetup(out client);

    await EsempioDiFindLocationByAddress();

    await EsempioDiFindLocationByQuery();

    await EsempioDiScaricamentoMappaStaticaByAddress();

    await EsempioDiScaricamentoMappaStaticaByQuery();

}
}
}

```

### 9.3.2.2. Altri esempi d'uso delle REST Bing Maps API

#### 9.3.2.2.1. Find a Location by Point (inverse geocoding)

<https://docs.microsoft.com/en-us/bingmaps/rest-services/locations/find-a-location-by-point>

Use the following URL template to get the location information associated with latitude and longitude coordinates.

When you make a request by using the following URL template, the response returns one or more Location resources that contain location information associated with the latitude and longitude coordinate values that you specify. Location information can be as specific as an address or more general such as the country or region. You can specify the type of location information you want to receive by setting the includeEntityTypes parameter in the URL. For example, you can specify to only receive information about the neighborhood that corresponds to the coordinates. For more information about the Location resource, see [Location Data](#). You can also view the example URL and response values in the Examples section.

URL template:

<http://dev.virtualearth.net/REST/v1/Locations/{point}?includeEntityTypes={entityTypes}&includeNeighborhood={includeNeighborhood}&include={includeValue}&key={BingMapsKey}>

Esempio:

<http://dev.virtualearth.net/REST/v1/Locations/45.70401,9.30493?includeEntityTypes=Address&includeNeighborhood=1&include=ciso2&key={BingMapsKey}>

### 9.3.2.2.2. Location Recognition

<https://docs.microsoft.com/en-us/bingmaps/rest-services/locations/location-recognition>

Given a pair of location coordinates (latitude, longitude), the Location Recognition API returns a list of entities ranked by their proximity to that location. The URL response contains three components: local business entities near that location (e.g. restaurants, hotels, office buildings, transit stations, etc.), natural points of interest near that location (e.g. beaches, valleys, etc.), and a reverse geocoded address of that location. See [Location Recognition Entity Types](#) for a list of supported entity types. You can specify the type of entities you want to receive by setting the includeEntityTypes parameter in the URL.

URL template:

[https://dev.virtualearth.net/REST/v1/LocationRecog/{point}?radius={search\\_radius}&top={number\\_of\\_results}&datetime={visit\\_date\\_time}&distanceunit={dist\\_unit}&verboseplacenames={true\\_or\\_false}&includeEntityTypes={list\\_of\\_entity\\_types}&key={BingMapsKey}](https://dev.virtualearth.net/REST/v1/LocationRecog/{point}?radius={search_radius}&top={number_of_results}&datetime={visit_date_time}&distanceunit={dist_unit}&verboseplacenames={true_or_false}&includeEntityTypes={list_of_entity_types}&key={BingMapsKey})

Esempio:

<https://dev.virtualearth.net/REST/v1/LocationRecog/45.70401,9.30493?radius=0.5&top=15&datetime=2022-02-22%2018:50:42Z&distanceunit=km&verboseplacenames=true&includeEntityTypes=businessAndPOI,naturalPOI,address&includeNeighborhood=1&include=ciso2&key={BingMapsKey}>

Si noti come sia possibile mettere optionalmente il datetime. Dalla documentazione si legge: **Optional.** Date and time at which the location is visited, in UTC format. When this parameter is specified, entities closed at that day and/or time are ranked lower than entities which are open

### 9.3.2.2.3. Bing Maps Imagery API

<https://docs.microsoft.com/en-us/bingmaps/rest-services/imagery>

<https://docs.microsoft.com/en-us/bingmaps/rest-services/imagery/get-a-static-map>

### 9.3.2.2.4. Bing Maps Routes API

<https://docs.microsoft.com/en-us/bingmaps/rest-services/routes>

Use the Routes API to create a route that includes two or more locations and to create routes from

major roads. You can create driving or walking routes. Driving routes can include traffic information. You can also overlay routes on map imagery.

For information about how to overlay routes on a static map using the Bing Maps REST Services Imagery API, see [Get a Static Map](#).

#### **9.3.2.2.4.1. Calculate a Route**

<https://docs.microsoft.com/en-us/bingmaps/rest-services/routes/calculate-a-route>

Use the following URL templates to get a walking, driving or transit route by specifying a series of waypoints. A waypoint is a specified geographical location defined by longitude and latitude that is used for navigational purposes. The route includes information such as route instructions, travel duration, travel distance or transit information. You can also request a set of route points.

To overlay a route on a static map by using the Imagery API, see [Get a Static Map](#).

URL template:

<http://dev.virtualearth.net/REST/v1/Routes?wayPoint.1={wayPoint1}&viaWaypoint.2={viaWaypoint2}&waypoint.3={waypoint3}&wayPoint.n={waypointN}&heading={heading}&optimize={optimize}&avoid={avoid}&distanceBeforeFirstTurn={distanceBeforeFirstTurn}&routeAttributes={routeAttributes}&timeType={timeType}&dateTime={dateTime}&maxSolutions={maxSolutions}&tolerances={tolerances}&distanceUnit={distanceUnit}&key={BingMapsKey}>

Esempio: per andare da Via dei Mille 27, Monticello Brianza a Piazza del Colosseo, Roma:

<https://dev.virtualearth.net/REST/v1/Routes?wp.1=45.70401,9.30493&wp.2=41.8910161588578,1249324840744635&optimize=time&tt=departure&dt=2022-02-22%2019:35:00&distanceUnit=km&c=it&ra=regionTravelSummary&key={BingMapsKey}>

Oppure

<https://dev.virtualearth.net/REST/v1/Routes?wp.1=Via%20dei%20Mille%2027,%20Monticello%20Brianza&wp.2=Piazza%20del%20Colosseo,%20Roma&optimize=time&tt=departure&dt=2022-02-22%2019:35:00&distanceUnit=km&c=it&ra=regionTravelSummary&key={BingMapsKey}>

#### **9.3.3. Here Rest API**

<https://developer.here.com/develop/rest-apis>

[https://developer.here.com/documentation/geocoding-search-api/dev\\_guide/topics/quick-start.html](https://developer.here.com/documentation/geocoding-search-api/dev_guide/topics/quick-start.html)

Lo studio delle Here Rest API è lasciato per esercizio.

#### **9.3.4. Mediawiki API**

La piattaforma Mediawiki, alla base del sito Wikipedia, offre diverse tipologie di API.

##### **9.3.4.1. MediaWiki Action API**

<https://www.mediawiki.org/wiki/API:Tutorial>

###### **9.3.4.1.1. Esempio: Ricerca fulltext in Wikipedia**

[https://www.mediawiki.org/wiki/API:Search#GET\\_request](https://www.mediawiki.org/wiki/API:Search#GET_request)

Ad esempio, vogliamo ricercare le pagine di Wikipedia che contengono argomenti relativi alla seguente frase: “lingua parlata dagli eschimesi”.

La query da effettuare è (imponendo, ad esempio, il limite di 5 risultati):

<https://it.wikipedia.org/w/api.php?action=query&list=search&srsearch=lingua%20parlata%20dagli%20eschimesi&utf8=&format=json&srlimit=5>

Il risultato riportato di seguito permette di ottenere un elenco di pagine riferite all'argomento in ordine decrescente di rilevanza:

```
{  
    "batchcomplete": "",  
    "continue": {  
        "sroffset": 5,  
        "continue": "-||"  
    },  
    "query": {  
        "searchinfo": {  
            "totalhits": 27,  
            "suggestion": "lingue parlata dagli eschimesi",  
            "suggestionsnippet": "<em>lingue parlata</em> dagli eschimesi"  
        },  
        "search": [  
            {  
                "ns": 0,  
                "title": "Lingua inuktitut",  
                "pageid": 604485,  
                "size": 6049,  
                "wordcount": 338,  
                "snippet": "delle <span class=\"searchmatch\">lingue</span> <span  
class=\"searchmatch\">eschimesi</span>, a loro volta parte della Famiglia linguistica delle <span  
class=\"searchmatch\">Lingue</span> eschimo-aleutine. Secondo Ethnologue e lo standard ISO 639, la  
<span class=\"searchmatch\">lingua</span> inuktitut",  
                "timestamp": "2022-02-04T08:54:53Z"  
            },  
            {  
                "ns": 0,  
                "title": "Lingue eschimo-aleutine",  
                "pageid": 1096718,  
                "size": 132001,  
                "wordcount": 748,  
                "snippet": "Vengono chiamate escaleutine, <span class=\"searchmatch\">eschimesi</span>,  
eskimo-aleutine o macro-<span class=\"searchmatch\">eschimesi</span>; questa famiglia è composta dalle  
<span class=\"searchmatch\">lingue</span> <span class=\"searchmatch\">eschimesi</span> (conosciute  
come Inuit) nel",  
                "timestamp": "2022-01-17T17:42:46Z"  
            },  
            {  
                "ns": 0,  
                "title": "Lingua groenlandese",  
                "pageid": 330971,  
                "size": 9437,  
                "wordcount": 972,  
            }  
        ]  
    }  
}
```

```

"snippet": "denominato <span class=\"searchmatch\">eschimese</span> di Groenlandia o inuktitut groenlandese) è una <span class=\"searchmatch\">lingua</span> appartenente alla famiglia delle <span class=\"searchmatch\">lingue</span> eschimo-aleutine. <span class=\"searchmatch\">Parlato</span> in Groenlandia",
    "timestamp": "2020-12-26T18:53:31Z"
},
{
    "ns": 0,
    "title": "Lingua inuit",
    "pageid": 422231,
    "size": 29874,
    "wordcount": 3536,
    "snippet": "Per <span class=\"searchmatch\">lingua</span> inuit si intende un continuum di <span class=\"searchmatch\">lingue</span> appartenenti alla famiglia delle <span class=\"searchmatch\">lingue</span> eschimo-aleutine tradizionalmente <span class=\"searchmatch\">parlata</span> in tutta l'Artide",
    "timestamp": "2021-07-17T02:51:21Z"
},
{
    "ns": 0,
    "title": "Lingua protoindoeuropea",
    "pageid": 127459,
    "size": 77162,
    "wordcount": 9264,
    "snippet": "<span class=\"searchmatch\">parlata</span> circa settemila anni fa e chiamata per convenzione proto-indoeuropeo. L'indagine sistematica fra le documentazioni più arcaiche delle <span class=\"searchmatch\">lingue</span> indoeuropee",
    "timestamp": "2022-02-11T01:03:09Z"
}
]
}
}

```

### 9.3.4.1.2. Esempio: Estrazione del summary di una pagina

Una volta ottenute le pagine rilevanti possiamo effettuare una ulteriore ricerca su esse per estrarre, ad esempio, il **summary della pagina più pertinente**. Infatti, leggendo la documentazione ufficiale di MediaWiki e alcune discussioni su StackOverflow, si deduce che:

Documentazione: <https://en.wikipedia.org/w/api.php?action=help&modules=query%2Bextracts>

<https://stackoverflow.com/a/28401782>

<https://stackoverflow.com/questions/8555320/is-there-a-wikipedia-api-just-for-retrieve-the-content-summary>

Per ottenere il **summary** di una pagina da Wikipedia, basta effettuare la query:

<https://it.wikipedia.org/w/api.php?format=json&action=query&prop=extracts&exintro&explaintext&exsectionformat=plain&redirects=1&titles={titolo della pagina}>

Ad esempio, prendendo come titolo della pagina quello del primo risultato della query relativa alla lingua parlata dagli eschimesi, otteniamo:

<https://it.wikipedia.org/w/api.php?format=json&action=query&prop=extracts&exintro&explaintext&exsectionformat=plain&redirects=1&titles=Lingua%20inuktitut>

```
{  
    "batchcomplete": "",  
    "query": {  
        "pages": {  
            "604485": {  
                "pageid": 604485,  
                "ns": 0,  
                "title": "Lingua inuktitut",  
                "extract": "L'inuktitut (ᐃᓄᒃᑎᑐᑦ inuktitut) è una lingua eschimo-aleutina parlata dal popolo  
Inuit in Canada."  
            }  
        }  
    }  
}
```

#### 9.3.4.1.3. Analisi del JSON DOM con la classe JsonDocument

L'esempio precedente, relativo all'oggetto JSON restituito dall'action API di Wikipedia mette in luce una questione molto importante: non sempre è possibile effettuare la deserializzazione di un oggetto JSON, mediante un modello precostituito, come è stato fatto negli esempi di OpenWeatherMap, di Bing Maps e di tanti altri casi presi in considerazione. Nell'oggetto JSON restituito dall'action API:

<https://it.wikipedia.org/w/api.php?format=json&action=query&prop=extracts&exintro&explaintext&exsectionformat=plain&redirects=1&titles={titolo della pagina}>

l'oggetto "pages" contiene al suo interno un campo che corrisponde all'id della pagina restituita e che quindi cambia da pagina a pagina. Ad esempio, il summary della pagina di Wikipedia relativa a Dante Alighieri è:

[https://it.wikipedia.org/w/api.php?format=json&action=query&prop=extracts&exintro&explaintext&exsectionformat=plain&redirects=1&titles=Dante\\_Alighieri](https://it.wikipedia.org/w/api.php?format=json&action=query&prop=extracts&exintro&explaintext&exsectionformat=plain&redirects=1&titles=Dante_Alighieri)

```
{  
    "batchcomplete": "",  
    "query": {  
        "normalized": [  
            {  
                "from": "Dante_Alighieri",  
                "to": "Dante Alighieri"  
            }  
        ],  
        "pages": {  
            "1476868": {  
                "pageid": 1476868,  
                "ns": 0,  
                "title": "Dante Alighieri",  
                "extract": "summary di Dante Alighieri..."  
            }  
        }  
    }  
}
```

In questo caso non si può fare una de-serializzazione, perché il Model cambia a seconda della pagina richiesta (la parte in giallo cambia). Per trattare questo caso occorre effettuare un'analisi del JSON DOM (Document Object Model), come descritto nella documentazione ufficiale di .NET: <https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-use-dom-utf8jsonreader-utf8jsonwriter?pivots=dotnet-5-0#json-dom-choices>

### Working with a DOM is an alternative to deserialization with [JsonSerializer](#):

- When you don't have a type to deserialize into.
- When the JSON you receive doesn't have a fixed schema and must be inspected to know what it contains.

System.Text.Json provides two ways to build a JSON DOM:

- [JsonDocument](#) provides the ability to build a read-only DOM by using Utf8JsonReader. The JSON elements that compose the payload can be accessed via the [JsonElement](#) type. The JsonElement type provides array and object enumerators along with APIs to convert JSON text to common .NET types. JsonDocument exposes a [RootElement](#) property. For more information, see [Use JsonDocument](#) later in this article.
- Starting in .NET 6, [JsonNode](#) and the classes that derive from it in the [System.Text.Json.Nodes](#) namespace provide the ability to create a mutable DOM. For more information, see the [.NET 6 version of this article](#).

Fondamentalmente si sono due oggetti che si possono utilizzare per la gestione del JSON DOM: JsonDocument (disponibile dalla versione .NET 5) e JsonNode (disponibile dalla versione .NET 6). JsonNode offre funzionalità aggiuntive rispetto a JsonDocument, poiché consente di modificare dinamicamente il DOM. Tuttavia, se bisogna soltanto analizzare il DOM, senza doverlo modificare dinamicamente, si può utilizzare tranquillamente JsonDocument.

Nell'esempio seguente viene mostrato come utilizzare la classe JsonDocument per estrarre il campo "extract" dall'oggetto JSON restituito dalle action API di mediawiki quando si richiede il summary di una pagina:

```
using System;
using System.Net.Http;
using System.Text;
using System.Text.Json;
using System.Threading.Tasks;
using HttpProxyControl;
namespace JSONDocumentWikiSummaryDemo
{
    class Program
    {
        static async Task Main(string[] args)
        {
            string wikiUrl =
"https://it.wikipedia.org/w/api.php?format=json&action=query&prop=extracts&exintro&explaintext&exsectionformat=plain&redirects=1&titles=Dante_Albighieri";

            HttpProxyHelper.HttpClientProxySetup(out HttpClient client);
            //ottengo la risposta da Wikipedia come stringa
            string wikiSummaryJSON = await client.GetStringAsync(wikiUrl);
            string summary = ExtractSummaryFromJSON(wikiSummaryJSON);
            Console.OutputEncoding = Encoding.Unicode;
            Console.WriteLine(summary);
        }
        /// <summary>
        /// Estrae il campo summary dal JSON restituito dall'Action API di Wikipedia
        /// </summary>
        /// <param name="wikiSummary">il summary in formato JSON</param>
```

```
    /// <returns>La stringa corrispondente al contenuto del campo extract, stringa vuota  
    se non riesce a fare l'estrazione</returns>  
    private static string ExtractSummaryFromJSON(string wikiSummary)  
    {  
        using (JsonDocument document = JsonDocument.Parse(wikiSummary))  
        {  
            JsonElement root = document.RootElement;  
            JsonElement query = root.GetProperty("query");  
            JsonElement pages = query.GetProperty("pages");  
            //per prendere il primo elemento dentro pages, creo un enumeratore delle  
            properties  
            JsonElement.ObjectEnumerator enumerator = pages.EnumerateObject();  
            //quando si crea un enumeratore su una collection, si deve farlo avanzare di  
            una posizione per portarlo sul primo elemento della collection  
            //il primo elemento corrisponde all'id della pagina all'interno dell'oggetto  
            pages  
            if (enumerator.MoveNext())  
            {  
                //accedo all'elemento  
                JsonElement targetJsonElem = enumerator.Current.Value;  
                if (targetJsonElem.TryGetProperty("extract", out JsonElement extract))  
                {  
                    return extract.GetString();  
                }  
            }  
            return string.Empty;  
        }  
    }  
}
```

//Output:

*Dante Alighieri, o Alighiero, battezzato Durante di Alighiero degli Alighieri e anche noto con il solo nome Dante, della famiglia Alighieri (Firenze, tra il 14 maggio e il 13 giugno 1265 – Ravenna, notte tra il 13 e il 14 settembre 1321), è stato un poeta, scrittore e politico italiano.*

*Il nome "Dante", secondo la testimonianza di Jacopo Alighieri, è un ipocoristico di Durante; nei documenti era seguito dal patronimico Alagherii o dal gentilizio de Alagherii, mentre la variante "Alighieri" si affermò solo con l'avvento di Boccaccio.*

*È considerato il padre della lingua italiana; la sua fama è dovuta alla paternità della Comedia, divenuta celebre come Divina Commedia e universalmente considerata la più grande opera scritta in lingua italiana e uno dei maggiori capolavori della letteratura mondiale. Espressione della cultura medievale, filtrata attraverso la lirica del Dolce stil novo, la Commedia è anche veicolo allegorico della salvezza umana, che si concreta nel toccare i drammi dei dannati, le pene purgatoriali e le glorie celesti, permettendo a Dante di offrire al lettore uno spaccato di morale ed etica.*

*Importante linguista, teorico politico e filosofo, Dante spaziò all'interno dello scibile umano, segnando profondamente la letteratura italiana dei secoli successivi e la stessa cultura occidentale, tanto da essere soprannominato il "Sommo Poeta" o, per antonomasia, il "Poeta". Dante, le cui spoglie si trovano presso la tomba a Ravenna costruita nel 1780 da Camillo Morigia, è diventato uno dei simboli dell'Italia nel mondo, grazie al nome del principale ente della diffusione della lingua italiana, la Società Dante Alighieri, mentre gli studi critici e filologici sono mantenuti vivi dalla Società dantesca.*

### 9.3.4.2. MediaWiki REST API

[https://www.mediawiki.org/wiki/API:REST\\_API/Reference](https://www.mediawiki.org/wiki/API:REST_API/Reference)

### **9.3.4.2.1. Esempio: trovare le pagine che trattano un determinato argomento**

La documentazione di riferimento è quella relativa alla ricerca di una pagina:

[https://www.mediawiki.org/wiki/API:REST API/Reference#Search\\_pages](https://www.mediawiki.org/wiki/API:REST_API/Reference#Search_pages)

Leggendo la documentazione si scopre che per ricercare la prima pagina di Wikipedia che tratta un

determinato argomento, si può utilizzare il seguente url:

[https://it.wikipedia.org/w/rest.php/v1/search/page?q={parole da cercare}&limit={limit\\_value}](https://it.wikipedia.org/w/rest.php/v1/search/page?q={parole da cercare}&limit={limit_value})

*Searches wiki page titles and contents for the provided search terms, and returns matching pages.*

Ad esempio, trovare le pagine relative alla ricerca “lingua parlata dagli eschimesi” (imponendo il limite di 5 risultati):

<https://it.wikipedia.org/w/rest.php/v1/search/page?q=lingua%20parlata%20dagli%20eschimesi&limit=5>

```
{  
  "pages": [  
    {  
      "id": 604485,  
      "key": "Lingua_inuktitut",  
      "title": "Lingua inuktitut",  
      "excerpt": "delle <span class=\"searchmatch\">lingue</span> <span  
class=\"searchmatch\">eschimesi</span>, a loro volta parte della Famiglia linguistica delle <span  
class=\"searchmatch\">Lingue</span> eschimo-aleutine. Secondo Ethnologue e lo standard ISO 639, la  
<span class=\"searchmatch\">lingua</span> inuktitut",  
      "matched_title": null,  
      "description": "lingua eschimo-aleutina",  
      "thumbnail": null  
    },  
    {  
      "id": 1096718,  
      "key": "Lingue_eschimo-aleutine",  
      "title": "Lingue eschimo-aleutine",  
      "excerpt": "Vengono chiamate escaleutine, <span class=\"searchmatch\">eschimesi</span>, eskimo-  
aleutine o macro-<span class=\"searchmatch\">eschimesi</span>; questa famiglia è composta dalle <span  
class=\"searchmatch\">lingue</span> <span class=\"searchmatch\">eschimesi</span> (conosciute come  
Inuit) nel",  
      "matched_title": null,  
      "description": "famiglia linguistica nativa della Groenlandia, Canada del nord, Alaska e parte  
della Siberia",  
      "thumbnail": {  
        "mimetype": "image/png",  
        "size": null,  
        "width": 200,  
        "height": 181,  
        "duration": null,  
        "url": "//upload.wikimedia.org/wikipedia/commons/thumb/4/4a/Eskimo-Aleut_langs.png/200px-  
Eskimo-Aleut_langs.png"  
      }  
    },  
    {  
      "id": 330971,  
      "key": "Lingua_groenlandese",  
      "title": "Lingua groenlandese",  
      "excerpt": "denominato <span class=\"searchmatch\">eschimese</span> di Groenlandia o inuktitut  
groenlandese) è una <span class=\"searchmatch\">lingua</span> appartenente alla famiglia delle <span
```

```

class="searchmatch">lingue</span> eschimo-aleutine. <span class="searchmatch">Parlato</span> in
Groenlandia",
    "matched_title": null,
    "description": "lingua appartenente alla famiglia delle lingue eschimo-aleutine",
    "thumbnail": {
        "mimetype": "image/png",
        "size": null,
        "width": 200,
        "height": 165,
        "duration": null,
        "url": "//upload.wikimedia.org/wikipedia/commons/thumb/c/c4/Idioma_groenland%C3%A9s.png/200px-
Idioma_groenland%C3%A9s.png"
    }
},
{
    "id": 422231,
    "key": "Lingua_inuit",
    "title": "Lingua inuit",
    "excerpt": "Per <span class="searchmatch">lingua</span> inuit si intende un continuum di <span
class="searchmatch">lingue</span> appartenenti alla famiglia delle <span
class="searchmatch">lingue</span> eschimo-aleutine tradizionalmente <span
class="searchmatch">parlata</span> in tutta l'Artide",
    "matched_title": null,
    "description": "lingua",
    "thumbnail": {
        "mimetype": "image/svg+xml",
        "size": 28705,
        "width": 200,
        "height": 179,
        "duration": null,
        "url": "//upload.wikimedia.org/wikipedia/commons/thumb/8/88/Inuktitut_dialect_map.svg/200px-
Inuktitut_dialect_map.svg.png"
    }
},
{
    "id": 127459,
    "key": "Lingua_protoindoeuropea",
    "title": "Lingua protoindoeuropea",
    "excerpt": "<span class="searchmatch">parlata</span> circa settemila anni fa e chiamata per
convenzione proto-indoeuropeo. L'indagine sistematica fra le documentazioni più arcaiche delle <span
class="searchmatch">lingue</span> indoeuropee",
    "matched_title": null,
    "description": "protolingua che costituisce l'origine comune delle lingue indoeuropee",
    "thumbnail": null
}
]
}

```

A seguito della prima ricerca si può effettuare una seconda ricerca per ottenere il summary del risultato più pertinente, come già mostrato nell'esempio della Action API:

Documentazione: <https://www.mediawiki.org/wiki/Extension:TextExtracts#API>

Esempio:

[https://it.wikipedia.org/w/api.php?format=json&action=query&prop=extracts&exintro&explaintext&exsectionformat=plain&redirects=1&titles=Lingua\\_inuktitud](https://it.wikipedia.org/w/api.php?format=json&action=query&prop=extracts&exintro&explaintext&exsectionformat=plain&redirects=1&titles=Lingua_inuktitud)

Il risultato è lo stesso dell'esempio precedente:

```
{  
    "batchcomplete": "",  
    "query": {  
        "normalized": [  
            {  
                "from": "Lingua_inuktitud",  
                "to": "Lingua inuktitud"  
            }  
        ],  
        "pages": {  
            "604485": {  
                "pageid": 604485,  
                "ns": 0,  
                "title": "Lingua inuktitud",  
                "extract": "L'inuktitud (Δນuktitud) è una lingua eschimo-aleutina parlata dal popolo  
Inuit in Canada."  
            }  
        }  
    }  
}
```

### 9.3.4.2.2. Esempio: trovare il summary di una pagina contenente le parole da cercare

Ad esempio: ricercare la pagina di Wikipedia che parla di Dante Alighieri:

<https://it.wikipedia.org/w/rest.php/v1/search/page?q=dante%20alighieri&limit=1>

Nota: impostando limit=1 prendiamo solo la prima pagina, la più pertinente.

Si ottiene:

```
{  
    "pages": [  
        {  
            "id": 1476868,  
            "key": "Dante_Alighieri",  
            "title": "Dante Alighieri",  
            "excerpt": "Dante Alighieri, o <span class=\"searchmatch\">Alighiero</span>, battezzato Durante  
di <span class=\"searchmatch\">Alighiero</span> degli <span class=\"searchmatch\">Alighieri</span> e  
anche noto con il solo nome <span class=\"searchmatch\">Dante</span>, della famiglia <span  
class=\"searchmatch\">Alighieri</span> (Firenze, tra il",  
            "matched_title": null,  
            "description": "poeta, scrittore e politico italiano, considerato il padre della lingua italiana  
(1265-1321)",  
            "thumbnail": {  
                "mimetype": "image/jpeg",  
                "size": null,  
            }  
        }  
    ]  
}
```

```

        "width": 131,
        "height": 200,
        "duration": null,
        "url": "//upload.wikimedia.org/wikipedia/commons/thumb/6/6f/Portrait_de_Dante.jpg/131px-
Portrait_de_Dante.jpg"
    }
}
]
}

```

Prendendo la chiave (campo `"key": "Dante_Alighieri"`) dal risultato precedente e invocando la Action API per ottenere il summary abbiamo:

[https://it.wikipedia.org/w/api.php?format=json&action=query&prop=extracts&exintro&explaintext&exsectionformat=plain&redirects=1&titles=Dante\\_Alighieri](https://it.wikipedia.org/w/api.php?format=json&action=query&prop=extracts&exintro&explaintext&exsectionformat=plain&redirects=1&titles=Dante_Alighieri)

Il cui risultato è:

```

{
  "batchcomplete": "",
  "query": {
    "normalized": [
      {
        "from": "Dante_Alighieri",
        "to": "Dante Alighieri"
      }
    ],
    "pages": {
      "1476868": {
        "pageid": 1476868,
        "ns": 0,
        "title": "Dante Alighieri",
        "extract": "Dante Alighieri, o Alighiero, battezzato Durante di Alighiero degli Alighieri e anche noto con il solo nome Dante, della famiglia Alighieri (Firenze, tra il 21 maggio e il 21 giugno 1265 – Ravenna, notte tra il 13 e il 14 settembre 1321), è stato un poeta, scrittore e politico italiano.\nIl nome \"Dante\", secondo la testimonianza di Jacopo Alighieri, è un ipocoristico di Durante; nei documenti era seguito dal patronimico Alagherii o dal gentilizio de Alagheriis, mentre la variante \"Alighieri\" si affermò solo con l'avvento di Boccaccio. \nÈ considerato il padre della lingua italiana; la sua fama è dovuta alla paternità della Commedia, divenuta celebre come Divina Commedia e universalmente considerata la più grande opera scritta in lingua italiana e uno dei maggiori capolavori della letteratura mondiale. Espressione della cultura medievale, filtrata attraverso la lirica del Dolce stil novo, la Commedia è anche veicolo allegorico della salvezza umana, che si concreta nel toccare i drammi dei dannati, le pene purgatoriali e le glorie celesti, permettendo a Dante di offrire al lettore uno spaccato di morale ed etica. \nImportante linguista, teorico politico e filosofo, Dante spaziò all'interno dello scibile umano, segnando profondamente la letteratura italiana dei secoli successivi e la stessa cultura occidentale, tanto da essere soprannominato il \"Sommo Poeta\" o, per antonomasia, il \"Poeta\". Dante, le cui spoglie si trovano presso la tomba a Ravenna costruita nel 1780 da Camillo Morigia, è diventato uno dei simboli dell'Italia nel mondo, grazie al nome del principale ente della diffusione della lingua italiana, la Società Dante Alighieri, mentre gli studi critici e filologici sono mantenuti vivi dalla Società dantesca."
      }
    }
  }
}

```

}

Da questo risultato si può leggere l'estratto della pagina relativa a Dante Alighieri, prendendo il valore del campo "extract".

### 9.3.4.2.3. Esempio: trovare le sezioni di una pagina di Wikipedia

La documentazione di riferimento è quella relativa al parsing:

[https://www.mediawiki.org/wiki/API:Parsing\\_wikitext](https://www.mediawiki.org/wiki/API:Parsing_wikitext)

Supponiamo di voler trovare la parte della pagina di wikipedia che parla delle opere di Giovanni Boccaccio. Procediamo nel seguente modo:

Step 1: otteniamo la pagina relativa a una query string:

[https://it.wikipedia.org/w/rest.php/v1/search/page?q={query\\_string}&limit=1](https://it.wikipedia.org/w/rest.php/v1/search/page?q={query_string}&limit=1)

Scopriamo che la chiave (page\_key) è "key": "Giovanni\_Boccaccio"

Step 2: otteniamo le sezioni della pagina selezionata allo step precedente

Con la chiave ottenuta effettuiamo una Action API call per estrarre le informazioni che ci interessano. In particolare, seguendo quanto riportato in:

<https://stackoverflow.com/a/59511898>

<https://stackoverflow.com/questions/59499885/how-to-get-a-text-of-a-specific-section-via-wikipedia-api>

Scopriamo che possiamo avere l'elenco delle sezioni di una pagina di Wikipedia con la Action API:

[https://it.wikipedia.org/w/api.php?action=parse&format=json&page={page\\_key}&prop=sections&disabletoc=1](https://it.wikipedia.org/w/api.php?action=parse&format=json&page={page_key}&prop=sections&disabletoc=1)

Step 3: otteniamo il contenuto di una specifica sezione

Possiamo ottenere il contenuto di una specifica sezione di una pagina di Wikipedia con la Action API:

[https://it.wikipedia.org/w/api.php?action=parse&format=json&page={page\\_key}&prop=wikitext&section={section\\_number}&disabletoc=1](https://it.wikipedia.org/w/api.php?action=parse&format=json&page={page_key}&prop=wikitext&section={section_number}&disabletoc=1)

Ad esempio, per accedere alla sezione delle *opere del periodo napolitano* della pagina relativa a Giovanni Boccaccio osserviamo che le sezioni sono tutte numerate con un indice. Di seguito si riporta una parte del risultato della query precedente:

```
{  
  "parse": {  
    "title": "Giovanni Boccaccio",  
    "pageid": 2952346,  
    "sections": [  
      {  
        "toclevel": 1,  
        "level": "2",  
        "line": "Biografia",  
        "number": "1",  
        "index": "1",  
        "fromtitle": "Giovanni_Boccaccio",  
        "byteoffset": 4950,
```

```

    "anchor": "Biografia"
},
{
  "toclevel": 2,
  "level": "3",
  "line": "L'infanzia fiorentina (1313-1327)",
  "number": "1.1",
  "index": "2",
  "fromtitle": "Giovanni_Boccaccio",
  "byteoffset": 4966,
  "anchor": "L'infanzia_fiorentina_(1313-1327)"
},
{
  "toclevel": 2,
  "level": "3",
  "line": "L'adolescenza napoletana (1327-1340)",
  "number": "1.2",
  "index": "3",
  "fromtitle": "Giovanni_Boccaccio",
  "byteoffset": 7829,
  "anchor": "L'adolescenza_napoletana_(1327-1340)"
},
{
  "toclevel": 3,
  "level": "4",
  "line": "Un ambiente cosmopolita: la formazione da autodidatta",
  "number": "1.2.1",
  "index": "4",
  "fromtitle": "Giovanni_Boccaccio",
  "byteoffset": 7874,
  "anchor": "Un_ambiente_cosmopolita:_la_formazione_da_autodidatta"
},
{
  "toclevel": 3,
  "level": "4",
  "line": "Fiammetta",
  "number": "1.2.2",
  "index": "5",
  "fromtitle": "Giovanni_Boccaccio",
  "byteoffset": 11403,
  "anchor": "Fiammetta"
},
{
  "toclevel": 2,
  "level": "3",
  "line": "L'inizio del secondo periodo fiorentino (1340-1350)",
  "number": "1.3",
  "index": "6",
  "fromtitle": "Giovanni_Boccaccio",
  "byteoffset": 12775,

```

```

        "anchor": "L'inizio_del_secondo_periodo_fiorentino_(1340-1350)"
    },
    {
        "toclevel": 1,
        "level": "2",
        "line": "Opere",
        "number": "2",
        "index": "23",
        "fromtitle": "Giovanni_Boccaccio",
        "byteoffset": 48677,
        "anchor": "Opere"
    },
    {
        "toclevel": 2,
        "level": "3",
        "line": "Opere del periodo napoletano",
        "number": "2.1",
        "index": "24",
        "fromtitle": "Giovanni_Boccaccio",
        "byteoffset": 48866,
        "anchor": "Opere_del_periodo_napoletano"
    },
    {
        "toclevel": 3,
        "level": "4",
        "line": "<i>La Caccia di Diana</i> (1333-1334)",
        "number": "2.1.1",
        "index": "25",
        "fromtitle": "Giovanni_Boccaccio",
        "byteoffset": 49290,
        "anchor": "La_Caccia_di_Diana_(1333-1334)"
    }
}

]
}
}

```

Per ottenere la sezione relativa alle “*opere del periodo napoletano*”, possiamamo effettuare la richiesta:

[https://it.wikipedia.org/w/api.php?action=parse&format=json&page=Giovanni\\_Boccaccio&prop=wikitext&section=24&disabletoc=1](https://it.wikipedia.org/w/api.php?action=parse&format=json&page=Giovanni_Boccaccio&prop=wikitext&section=24&disabletoc=1)

Il risultato è:

```
{
    "parse": {
        "title": "Giovanni Boccaccio",
        "pageid": 2952346,
        "wikitext": {

```

"\*": "== Opere del periodo napoletano ==\n{{vedi anche|Opere della giovinezza di Giovanni Boccaccio}}\n\nTra le sue prime opere del periodo napoletano vengono ricordate: ''[[Caccia di Diana]]'' (1334 circa)<ref>{{Cita|Branca 1977|p. 41}}.</ref>, ''[[Filostrato (Boccaccio)|Filostrato]]'' (1335), il ''[[Filocolo]]'' (1336-38)<ref>{{Cita|Branca 1977|p. 44}}.</ref>, ''[[Teseida]]'' (1339-41)<ref>{{Cita|Branca 1977|p. 49}}.</ref>.\n==== ''La Caccia di Diana'' (1333-1334) ===\n{{vedi anche|Caccia di Diana}}\nLa ''[[Caccia di Diana]]'' è un poemetto di 18 [[Canto (metrica)|canti]] in [[Terzina dantesca|terzine dantesche]] che celebra in chiave mitologica alcune gentildonne napoletane. Le ninfe, seguaci della casta [[Diana]], si ribellano alla dea e offrono le loro prede di caccia a [[Venere (divinità)|Venere]], che trasforma gli animali in bellissimi uomini. Tra questi vi è anche il giovane Boccaccio che, grazie all'amore, diviene un uomo pieno di virtù: il poemetto propone, dunque, la concezione cortese e [[Dolce stil novo|stilnovistica]] dell'amore che ingentilisce e nobilita l'essere umano<ref>{{Cita web|autore = Rachele Jesurum|url = http://www.oilproject.org/lezione/boccaccio-caccia-di-diana-analisi-e-commento-4334.html|titolo = Boccaccio, \"La Caccia di Diana\"}: analisi e commento|accesso = 23 giugno 2015|editore = Oilproject}</ref>.\n==== Il ''Filostrato'' (1335) ===\n{{vedi anche|Filostrato (Boccaccio)}}\n[[File:Boccaccio Altonensis 1.jpg|miniatura|Codice riportante un passo del ''Filocolo''. 'Codex Christianei', conservato nella Bibliotheca Gymnasii Altonani ([[Amburgo]])]]\nIl ''[[Filostrato (Boccaccio)|Filostrato]]'' (che alla lettera dovrebbe significare nel [[lingua greca|greco]] approssimativo del Boccaccio «vinto d'amore») è un poemetto scritto in [[ottava rima|ottave]] che narra la tragica storia di [[Troilo]], figlio del [[re di Troia]] [[Priamo]], che si era innamorato della principessa greca [[Criseide]]. La donna, in seguito a uno scambio di prigionieri, torna al campo greco, e dimentica Troilo. Quando Criseide in seguito s'innamora di Diomede, Troilo si dispera e va incontro alla morte per mano di [[Achille]]. Nell'opera l'autore si confronta in maniera diretta con la precedente tradizione dei «cantari», fissando i parametri per un nuovo tipo di ottava essenziale per tutta la letteratura italiana fino al [[XVII secolo|Seicento]]<ref>{{Cita|Branca 1977|pp. 42-43}}.</ref>. Il linguaggio adottato è difficile, altolocato, spedito, a differenza di quello presente nel ''Filocolo'', in cui è molto sovrabbondante<ref>Per il ''Filostrato'' in generale, si veda: {{Cita web|autore = Rachele Jesurum|url = http://www.oilproject.org/lezione/filostrato-boccaccio-trama-4331.html|titolo = Boccaccio, \"Il Filostrato\"}: riassunto e commento|accesso = 23 giugno 2015|editore = Oilproject}</ref>.\n==== Il ''Filocolo'' (1336-1339) ===\n{{vedi anche|Filocolo}}\nIl ''[[Filocolo]]'' (secondo un'etimologia approssimativa «fatica d'amore») è un romanzo in prosa: rappresenta una svolta rispetto ai romanzi delle origini scritti in versi. La storia ha come protagonisti Florio, figlio di un re saraceno, e Biancifiore (o Biancofiore), una schiava cristiana abbandonata da bambina. I due fanciulli crescono assieme e da grandi, in seguito alla lettura del libro di [[Publio Ovidio Nasone|Ovidio]] ''[[Ars amatoria|Ars Amandi]]'' s'innamorano, come era successo per [[Paolo e Francesca]] dopo avere letto ''[[Ginevra (ciclo arturiano)|Ginevra]]'' e ''[[Lancillotto]]''. Tuttavia il padre di Florio decide di separarli vendendo Biancifiore a dei mercanti. Florio decide quindi di andarla a cercare e dopo mille peripezie (da qui il titolo ''Filocolo'') la rincontra. Infine, il giovane si converte al [[cristianesimo]] e sposa la fanciulla<ref>{{Cita web|autore = Rachele Jesurum|url = http://www.oilproject.org/lezione/boccaccio-filocolo-trama-4361.html|titolo = Boccaccio, \"Il Filocolo\"}: riassunto e commento|accesso = 23 giugno 2015|editore = Oilproject}</ref>.\n==== ''Teseida delle nozze d'Emilia'' (1339-1340) ===\n{{vedi anche|Teseida}}\n[[File:Emilia in the rose-garden (Teseida).jpg|thumb|upright=1.5|'Emilia nel roseto', manoscritto francese del 1460 ca.|alt=]]\nIl ''[[Teseida]]'' è un poema epico in ottave in cui si rievocano le gesta di [[Teseo]] che combatte contro [[Tebe (città greca antica)|Tebe]] e le [[Amazzoni]]. L'opera costituisce il primo caso in assoluto nella storia letteraria in lingua italiana di poema epico in volgare e già si manifesta la tendenza di Boccaccio a isolare nuclei narrativi sentimentali, cosicché il vero centro della narrazione finisce per essere l'amore dei prigionieri tebani [[Arcita]] e [[Palemone (divinità)|Palemone]], molto amici, per Emilia, regina delle Amazzoni e cognata di Teseo; il duello

```

fra i due innamorati si conclude con la morte di Arcita e le nozze tra Palemone ed Emilia<ref>Per il
''Teseida'' in generale, si veda: {{Cita web|autore = Rachele Jesurum|url =
http://www.oilproject.org/lezione/boccaccio-teseida-introduzione-e-commento-dell-opera-
4392.html|titolo = Boccaccio, "Teseida": introduzione e commento dell'opera|accesso = 23 giugno
2015|editore = Oilproject}}</ref>."
}

}
}

Il formato del testo è wikitext. Se invece si inserisce come parametro della query prop=text si ottiene il risultato sotto forma di HTML.
https://it.wikipedia.org/w/api.php?action=parse&format=json&page=Giovanni\_Boccaccio&prop=txt&section=24&disabletoc=1

```

#### 9.3.4.2.4. Da Wikitext a testo leggibile

Il formato wikitext è un linguaggio di markup, con una sua sintassi e parole chiave. I dettagli del linguaggio sono specificati qui: <https://en.wikipedia.org/wiki/Help:Wikitext>

Per trasformare il formato wikitext in un formato di testo leggibile senza i tag occorre effettuare un parsing del wikitext. In altre parole occorre interpretare i tag e la struttura del documento per estrarre solo le informazioni che interessano (il PlainText). Per fare questo esistono diversi software di parsing già disponibili per diversi linguaggi di programmazione. Al link: [https://www.mediawiki.org/wiki/Alternative\\_parsers](https://www.mediawiki.org/wiki/Alternative_parsers) c'è un elenco completo.

Per il linguaggio C# esiste il progetto: <https://github.com/CXuesong/MwParserFromScratch>

Per utilizzare MWParserFromScratch occorre installare, tramite NuGet la libreria **CXuesong.MW.MwParserFromScratch**. Successivamente si può utilizzare, come mostrato di seguito, il parser.

Nel seguente esempio è stato creato il progetto di tipo libreria **WikitextExtensions**:

//file Program.cs

```

using MwParserFromScratch;
using MwParserFromScratch.Nodes;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace WikitextExtensions
{
    public static class WikitextHelper
    {

        /// <summary>
        /// Converte un testo in formato wikitext in testo leggibile
        /// </summary>
        /// <param name="wikitext">Il wikitext in input</param>
        /// <returns>La stringa corrispondente al testo leggibile</returns>
        public static string WikiTextToReadableText(this string wikitext)
        {
            //https://github.com/CXuesong/MwParserFromScratch
            var parser = new WikitextParser();

```

```

var ast = parser.Parse(wikitext);
StringBuilder sb = new StringBuilder();
foreach (var line in ast.Lines)
{
    if (line.GetType() == typeof(Paragraph) &&
line.ToPlainText(NodePlainTextOptions.RemoveRefTags).StartsWith("[["))
        continue;//non inserisco i paragrafi delle illustrazioni
    var children = line.EnumChildren();
    foreach (var child in children)
    {
        string childText = child.ToPlainText(NodePlainTextOptions.RemoveRefTags);
        if (child.GetType() == typeof(Template))//in questo caso si effettua un parsing
manuale, ad esempio per gestire: https://www.mediawiki.org/wiki/Help:Magic\_words#Formatting
        {
            string elemento = child.ToString();
            List<string> paroleChiave = new List<string>() { "vedi" };
            string result = paroleChiave.FirstOrDefault(_ => elemento.Contains(_, StringComparison.CurrentCultureIgnoreCase));
            if (result == default)//il child non contiene le parole chiave
            {
                //posso inserire il contenuto
                int startIndex = elemento.IndexOfAny(new char[] { ' ', ':' });//l'ordine
conta
                startIndex = (startIndex == -1) ? elemento.LastIndexOf('{') :
startIndex;//se non trovo il punto di inizio del contenuto assumo che ci siano due {{}
                int stopIndex = elemento.IndexOfAny(new char[] { '|', '}' });//l'ordine
conta
                int lunghezzaContenuto = stopIndex - startIndex - 1;
                string contenuto = elemento.Substring(startIndex + 1, lunghezzaContenuto);
                sb.Append(contenuto);
            }
            else if (child.GetType() == typeof(WikiLink))//quando è un WikiLink non stampo i
riferimenti a file, thumb e alt, etc.
            {
                List<string> paroleChiave = new List<string>() { "file", "thumb", "alt" };
                string result = paroleChiave.FirstOrDefault(_ => childText.Contains(_, StringComparison.CurrentCultureIgnoreCase));
                if (result == default)//il childText non contiene le parole chiave
                {
                    sb.Append(childText);
                }
            }
            else //child.GetType() != WikiLink && child.GetType() != Template
            {
                sb.Append(childText);
            }
        }
        if (line.GetType() == typeof(Heading) || line.GetType() == typeof(Paragraph))//quando
sono alla fine di un'intestazione oppure di un paragrafo decido se andare a capo
    }
}

```

```

    {
        if
        (!string.IsNullOrWhiteSpace(line.ToPlainText(NodePlainTextOptions.RemoveRefTags)))
            {//controllo che ci sia un contenuto nella riga corrente per andare a capo
                sb.Append('\n');
            }
    }
}

return sb.ToString();
}

/// <summary>
/// Converte un testo in formato wikitext in testo leggibile.
/// Per ogni linea nel wikitext viene restituita una stringa risultato.
/// A differenza del metodo WikiTextToReadableText, questo restituisce un array di stringhe
/// </summary>
/// <param name="wikitext">Il wikitext in input</param>
/// <returns>Un array di stringhe, ciascuna con testo leggibile. Ogni stringa restituirà
//corrisponde a una linea all'interno del AST (Abstract Syntax Tree) del wikitext parser</returns>
public static string[] WikiTextToReadableTextArray(this string wikitext)
{
    //https://github.com/CXuesong/MwParserFromScratch
    var parser = new WikitextParser();
    var ast = parser.Parse(wikitext);
    StringBuilder sb = new StringBuilder();
    List<string> readableLines = new List<string>();
    foreach (var line in ast.Lines)
    {
        sb.Clear(); //ripulisco lo StringBuilder per la linea in corso
        if (line.GetType() == typeof(Paragraph) &&
line.ToPlainText(NodePlainTextOptions.RemoveRefTags).StartsWith("[["))
            continue; //non inserisco i paragrafi delle illustrazioni
        var children = line.EnumChildren();
        foreach (var child in children)
        {
            string childText = child.ToPlainText(NodePlainTextOptions.RemoveRefTags);
            if (string.IsNullOrWhiteSpace(childText) || childText.Equals("\n")) //non inserisco
elementi vuoti o con solo a capo
            {
                continue;
            }
            else if (child.GetType() == typeof(Template)) //in questo caso si effettua un
parsing manuale, ad esempio per gestire: https://www.mediawiki.org/wiki/Help:Magic_words#Formatting
            {
                string elemento = child.ToString();
                List<string> paroleChiave = new List<string>() { "vedi" };
                string result = paroleChiave.FirstOrDefault(_ => elemento.Contains(_,
 StringComparison.CurrentCultureIgnoreCase));
                if (result == default) //il child non contiene le parole chiave
                {
                    //posso inserire il contenuto
                }
            }
        }
    }
}

```

```

        int startIndex = elemento.IndexOfAny(new char[] { ' ', ':' });//l'ordine
conta
        startIndex = (startIndex == -1) ? elemento.LastIndexOf('{') :
startIndex;//se non trovo il punto di inizio del contenuto assumo che ci siano due {{{
        int stopIndex = elemento.IndexOfAny(new char[] { '|', '}' });//l'ordine
conta
        int lunghezzaContenuto = stopIndex - startIndex - 1;
        string contenuto = elemento.Substring(startIndex + 1, lunghezzaContenuto);
        sb.Append(contenuto);
    }
}
else if (child.GetType() == typeof(WikiLink))//quando è un WikiLink non stampo i
riferimenti a file, thumb e alt, etc.
{
    List<string> paroleChiave = new List<string>() { "file", "thumb", "alt" };
    string result = paroleChiave.FirstOrDefault(_ => childText.Contains(_, StringComparison.CurrentCultureIgnoreCase));
    if (result == default)//il childText non contiene le parole chiave
    {
        sb.Append(childText);
    }
}
else //child.GetType() != WikiLink && child.GetType() != Template
{
    sb.Append(childText);
}
}
if (sb.Length > 0)
{
    readableLines.Add(sb.ToString());
}
}

}
return readableLines.ToArray();
}
/// <summary>
/// Converte un testo in formato wikitext in testo leggibile. Vengono rimossi gli spazi
corrispondenti a new line
/// </summary>
/// <param name="wikitext">Il wikitext in input</param>
/// <returns>La stringa corrispondente al testo leggibile</returns>
public static string WikiTextToReadableTextNoSpace(this string wikitext)
{
    //https://github.com/CXuesong/MwParserFromScratch
    var parser = new WikitextParser();
    var ast = parser.Parse(wikitext);
    StringBuilder sb = new StringBuilder();
    foreach (var line in ast.Lines)
    {

```

```

        if (line.GetType() == typeof(Paragraph) &&
line.ToPlainText(NodePlainTextOptions.RemoveRefTags).StartsWith("[["))
            continue;//non inserisco i paragrafi delle illustrazioni
        var children = line.EnumChildren();
        foreach (var child in children)
        {
            string childText = child.ToPlainText(NodePlainTextOptions.RemoveRefTags);
            if (string.IsNullOrWhiteSpace(childText) || childText.Equals("\n"))
            {
                continue;
            }
            else if (child.GetType() == typeof(Template))//in questo caso si effettua un
parsing manuale, ad esempio per gestire: https://www.mediawiki.org/wiki/Help:Magic_words#Formatting
            {
                string elemento = child.ToString();
                List<string> paroleChiave = new List<string>() { "vedi" };
                string result = paroleChiave.FirstOrDefault(_ => elemento.Contains(_, StringComparison.CurrentCultureIgnoreCase));
                if (result == default)//il child non contiene le parole chiave
                {
                    //posso inserire il contenuto
                    int startIndex = elemento.IndexOfAny(new char[] { ' ', ':' })//l'ordine
conta
                    startIndex = (startIndex == -1) ? elemento.LastIndexOf('{') :
startIndex;//se non trovo il punto di inizio del contenuto assumo che ci siano due {{{
                    int stopIndex = elemento.IndexOfAny(new char[] { '|', '}' })//l'ordine
conta
                    int lunghezzaContenuto = stopIndex - startIndex - 1;
                    string contenuto = elemento.Substring(startIndex + 1, lunghezzaContenuto);
                    sb.Append(contenuto);
                }
            }
            else if (child.GetType() == typeof(WikiLink))//quando è un WikiLink non stampo i
riferimenti a file, thumb e alt, etc.
            {
                List<string> paroleChiave = new List<string>() { "file", "thumb", "alt" };
                string result = paroleChiave.FirstOrDefault(_ => childText.Contains(_, StringComparison.CurrentCultureIgnoreCase));
                if (result == default)//il childText non contiene le parole chiave
                {
                    sb.Append(childText);
                }
            }
            else //child.GetType() != WikiLink && child.GetType() != Template
            {
                sb.Append(childText);
            }
        }
    }
}

```

```

        //if (line.GetType() == typeof(Heading) || line.GetType() ==
typeof(Paragraph))//quando sono alla fine di un'intestazione oppure di un paragrafo decido se andare a
capo
        //{
        //    if
(!string.IsNullOrWhiteSpace(line.ToPlainText(NodePlainTextOptions.RemoveRefTags)))
        //        {//controllo che ci sia un contenuto nella riga corrente per andare a capo
        //            sb.Append('\n');
        //        }
        //}
    }
    return sb.ToString();
}
/// <summary>
/// Effettua lo split di una stringa di testo in un array di stringhe, usando il punto come
separatore.
/// Il punto non è rimosso dal risultato
/// </summary>
/// <param name="text">Testo che si vuole suddividere in più stringhe</param>
/// <returns>Array di stringhe ottenuto facendo lo split sul punto</returns>
public static string[] SplitOnPeriod(this string text)
{
    //return text.Split(separators, StringSplitOptions.RemoveEmptyEntries); //questo
funziona, ma toglie i punti

    List<String> afterSplit = new List<string>(); //lista contenente le frasi separate da .
    int start = 0; //indice di inizio frase
    for (int counter = 0; counter < text.Length; counter++)
    {
        if (text[counter] == '.') //se trovo un punto
        {
            int lineLength = counter - start + 1;
            afterSplit.Add(text.Substring(start, lineLength));
            start = counter + 1; //aggiorno start
        }
    }
    if (start < text.Length) //se, uscito dal ciclo, start è < di text.Length vuol dire che
l'ultimo pezzo del testo non è stato incluso, perché non c'era il punto finale
    {
        afterSplit.Add(text.Substring(start));
    }
    return afterSplit.ToArray();
}
}

//file di progetto:
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
</PropertyGroup>

```

```

<ItemGroup>
  <PackageReference Include="CXuesong.MW.MwParserFromScratch" Version="0.2.1" />
</ItemGroup>

</Project>

```

Per testare la libreria è stato creato il progetto **WikitextParsingDemo**:

```

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <ProjectReference Include="..\WikitextExtensions\WikitextExtensions.csproj" />
  </ItemGroup>

</Project>

```

//File Program.cs

```

using System;
using WikitextExtensions;

namespace WikitextParsingDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            string wikitext = "==== Opere del periodo napoletano ===\n{{vedi anche|Opere della giovinezza di Giovanni Boccaccio}}\n\nTra le sue prime opere del periodo napoletano vengono ricordate: ''[[Caccia di Diana]]'' (1334 circa)<ref>{{Cita|Branca 1977|p. 41}}.</ref>, ''[[Filostrato (Boccaccio)|Filostrato]]'' (1335), il ''[[Filocolo]]'' (1336-38)<ref>{{Cita|Branca 1977|p. 44}}.</ref>, ''[[Teseida]]'' (1339-41)<ref>{{Cita|Branca 1977|p. 49}}.</ref>.\n\n==== 'La Caccia di Diana' (1333-1334) ====\n{{vedi anche|Caccia di Diana}}\n\nLa ''[[Caccia di Diana]]'' è un poemetto di 18 [[Canto (metrica)|canti]] in [[Terzina dantesca|terzine dantesche]] che celebra in chiave mitologica alcune gentildonne napoletane. Le ninfe, seguaci della casta [[Diana]], si ribellano alla dea e offrono le loro prede di caccia a [[Venere (divinità)|Venere]], che trasforma gli animali in bellissimi uomini. Tra questi vi è anche il giovane Boccaccio che, grazie all'amore, diviene un uomo pieno di virtù: il poemetto propone, dunque, la concezione cortese e [[Dolce stil novo|stilnovistica]] dell'amore che ingentilisce e nobilita l'essere umano<ref>{{Cita web|autore = Rachele Jesurum|url = http://www.oilproject.org/lezione/boccaccio-caccia-di-diana-analisi-e-commento-4334.html|titolo = Boccaccio, \"La Caccia di Diana\": analisi e commento|accesso = 23 giugno 2015|editore = Oilproject}}</ref>.\n\nIl ''Filostrato'' (1335) ====\n{{vedi anche|Filostrato (Boccaccio)}}\n[[File:Boccaccio Altonensis 1.jpg|miniatura|Codice riportante un passo del 'Filocolo'. 'Codex Christianei', conservato nella Bibliotheca Gymnasii Altonani ([[Amburgo]])]]\n\nIl ''[[Filostrato (Boccaccio)|Filostrato]]'' (che alla lettera dovrebbe significare nel [[lingua greca|greco]] approssimativo del Boccaccio «vinto d'amore») è un poemetto scritto in [[ottava rima|ottave]] che narra la tragica storia di [[Troilo]], figlio del [[re di Troia]] [[Priamo]], che si era innamorato della principessa greca [[Criseide]]. La donna, in seguito a uno scambio di prigionieri, torna al campo greco, e dimentica Troilo. Quando Criseide in seguito s'innamora di Diomede, Troilo si dispera e va incontro alla morte per mano di [[Achille]]. Nell'opera

```

l'autore si confronta in maniera diretta con la precedente tradizione dei «cantari», fissando i parametri per un nuovo tipo di ottava essenziale per tutta la letteratura italiana fino al [[XVII secolo|Seicento]]<ref>{{Cita|Branca 1977|pp. 42-43}}.</ref>. Il linguaggio adottato è difficile, altolocato, spedito, a differenza di quello presente nel "Filocolo", in cui è molto sovrabbondante<ref>Per il "Filostrato" in generale, si veda: {{Cita web|autore = Rachele Jesurum|url = http://www.oilproject.org/lezione/filostrato-boccaccio-trama-4331.html|titolo = Boccaccio, "Il Filostrato": riassunto e commento|accesso = 23 giugno 2015|editore = Oilproject}}</ref>. \n\n==== Il "Filocolo" (1336-1339) ====\n{{vedi anche|Filocolo}}\n\nIl "[[Filocolo]]" (secondo un'etimologia approssimativa «fatica d'amore») è un romanzo in prosa: rappresenta una svolta rispetto ai romanzi delle origini scritti in versi. La storia ha come protagonisti Florio, figlio di un re saraceno, e Biancifiore (o Biancofiore), una schiava cristiana abbandonata da bambina. I due fanciulli crescono assieme e da grandi, in seguito alla lettura del libro di [[Publio Ovidio Nasone|Ovidio]] "[[Ars amatoria|Ars Amandi]]" s'innamorano, come era successo per [[Paolo e Francesca]] dopo avere letto "[[Ginevra (ciclo arturiano)|Ginevra]]" e "[[Lancillotto]]". Tuttavia il padre di Florio decide di separarli vendendo Biancifiore a dei mercanti. Florio decide quindi di andarla a cercare e dopo mille peripezie (da qui il titolo "Filocolo") la rincontra. Infine, il giovane si converte al [[cristianesimo]] e sposa la fanciulla<ref>{{Cita web|autore = Rachele Jesurum|url = http://www.oilproject.org/lezione/boccaccio-filocolo-trama-4361.html|titolo = Boccaccio, "Il Filocolo": riassunto e commento|accesso = 23 giugno 2015|editore = Oilproject}}</ref>. \n\n==== "Teseida delle nozze d'Emilia" (1339-1340) ====\n{{vedi anche|Teseida}}\n[[File:Emilia in the rose garden (Teseida).jpg|thumb|upright=1.5|'"Emilia nel roseto"', manoscritto francese del 1460 ca.|alt=]]\n\nIl "[[Teseida]]" è un poema epico in ottave in cui si rievocano le gesta di [[Teseo]] che combatte contro [[Tebe (città greca antica)|Tebe]] e le [[Amazzoni]]. L'opera costituisce il primo caso in assoluto nella storia letteraria in lingua italiana di poema epico in volgare e già si manifesta la tendenza di Boccaccio a isolare nuclei narrativi sentimentali, cosicché il vero centro della narrazione finisce per essere l'amore dei prigionieri tebani [[Arcita]] e [[Palemone (divinità)|Palemone]], molto amici, per Emilia, regina delle Amazzoni e cognata di Teseo; il duello fra i due innamorati si conclude con la morte di Arcita e le nozze tra Palemone ed Emilia<ref>Per il "Teseida" in generale, si veda: {{Cita web|autore = Rachele Jesurum|url = http://www.oilproject.org/lezione/boccaccio-teseida-introduzione-e-commento-dell-opera-4392.html|titolo = Boccaccio, "Teseida": introduzione e commento dell'opera|accesso = 23 giugno 2015|editore = Oilproject}}</ref>. ";

```

    Console.WriteLine("*****OUTPUT IN FORMATO WIKITEXT*****");
    Console.WriteLine(wikitext);
    Console.WriteLine("\n*****OUTPUT LEGGIBILE*****");
    //string[] readableText = WikiTextToReadableTextArray(wikitext);
    string[] readableText = wikitext.WikiTextToReadableTextNoSpace().SplitOnPeriod();
    //TextSplitOnPeriod(WikiTextToReadableTextNoSpace(wikitext));
    foreach (var item in readableText)
    {
        Console.WriteLine(item);
    }
}
}
  
```

Il risultato leggibile è il seguente:

Opere del periodo napoletano Tra le sue prime opere del periodo napoletano vengono ricordate: Caccia di Diana (1334 circa), Filostrato (1335), il Filocolo (1336-38), Teseida (1339-41).

**La Caccia di Diana (1333-1334)** La Caccia di Diana è un poemetto di 18 canti in terzine dantesche che celebra in chiave mitologica alcune gentildonne napoletane.

Le ninfe, seguaci della casta Diana, si ribellano alla dea e offrono le loro prede di caccia a Venere, che trasforma gli animali in bellissimi uomini.

Tra questi vi è anche il giovane Boccaccio che, grazie all'amore, diviene un uomo pieno di virtù: il poemetto propone, dunque, la concezione cortese e stilnovistica dell'amore che ingentilisce e nobilita l'essere umano.

**Il Filostrato (1335)** Il Filostrato (che alla lettera dovrebbe significare nel greco approssimativo del Boccaccio «vinto d'amore») è un poemetto scritto in ottave che narra la tragica storia di Troilo, figlio del re di Troia Priamo, che si era innamorato della principessa greca Criseide.

La donna, in seguito a uno scambio di prigionieri, torna al campo greco, e dimentica Troilo.

Quando Criseide in seguito s'innamora di Diomede, Troilo si dispera e va incontro alla morte per mano di Achille.

Nell'opera l'autore si confronta in maniera diretta con la precedente tradizione dei «cantari», fissando i parametri per un nuovo tipo di ottava essenziale per tutta la letteratura italiana fino al Seicento.

Il linguaggio adottato è difficile, altolocato, spedito, a differenza di quello presente nel Filocolo, in cui è molto sovrabbondante.

**Il Filocolo (1336-1339)** Il Filocolo (secondo un'etimologia approssimativa «fatica d'amore») è un romanzo in prosa: rappresenta una svolta rispetto ai romanzi delle origini scritti in versi.

La storia ha come protagonisti Florio, figlio di un re saraceno, e Biancifiore (o Biancofiore), una schiava cristiana abbandonata da bambina.

I due fanciulli crescono assieme e da grandi, in seguito alla lettura del libro di Ovidio *Ars Amandi* s'innamorano, come era successo per Paolo e Francesca dopo avere letto *Ginevra e Lancillotto*.

Tuttavia il padre di Florio decide di separarli vendendo Biancifiore a dei mercanti.

Florio decide quindi di andarla a cercare e dopo mille peripezie (da qui il titolo Filocolo) la rincontra.

Infine, il giovane si converte al cristianesimo e sposa la fanciulla.

**Teseida delle nozze d'Emilia (1339-1340)** Il Teseida è un poema epico in ottave in cui si rievocano le gesta di Teseo che combatte contro Tebe e le Amazzoni.

L'opera costituisce il primo caso in assoluto nella storia letteraria in lingua italiana di poema epico in volgare e già si manifesta la tendenza di Boccaccio a isolare nuclei narrativi sentimentali, cosicché il vero centro della narrazione finisce per essere l'amore dei prigionieri tebani Arcita e Palemone, molto amici, per Emilia, regina delle Amazzoni e cognata di Teseo; il duello fra i due innamorati si conclude con la morte di Arcita e le nozze tra Palemone ed Emilia.

## 10. Microsoft Azure

<https://azure.microsoft.com/it-it/>

[https://it.wikipedia.org/wiki/Microsoft\\_Azure\\_\(piattaforma\)](https://it.wikipedia.org/wiki/Microsoft_Azure_(piattaforma))

Microsoft Azure è la [piattaforma cloud](#) pubblica di [Microsoft](#), che offre servizi di [cloud computing](#).

Tramite Azure vengono erogati servizi appartenenti a diverse categorie<sup>[2]</sup> quali: risorse di elaborazione, archiviazione e memorizzazione dati, trasmissione dati e interconnessione di reti, analisi, intelligence, [apprendimento automatico](#) sicurezza e gestione delle identità, monitoraggio e gestione, nonché servizi per lo sviluppo di applicazioni.

Il numero e il tipo dei servizi erogati vengono modificati da Microsoft con cadenza periodica.

I servizi messi a disposizione da Microsoft Azure possono essere classificati in tre aree, a seconda della modalità di erogazione adottata: [Infrastructure as a Service](#) (IaaS), [Platform as a Service](#) (PaaS) e infine [Software as a Service](#) (SaaS). Fornisce anche servizi di [mBaaS](#) (*mobile Backend as a Service*).

### 10.1. Sottoscrizione Studenti per Azure

<https://azure.microsoft.com/it-it/free/students/>

Utilizzare l'account Microsoft della scuola e seguire la procedura di iscrizione gratuita.

Una volta iscritti si ha un credito virtuale di \$100, spendibile in servizi a pagamento e l'accesso a una lunga serie di servizi gratuiti, tra cui, in particolare:

- **LUIS (Language Understanding)**
- **Riconoscimento vocale**
- **Sintesi vocale**
- **Molto altro...**

### 10.2. Azure Cognitive Services

#### 10.2.1. Speech to text

##### 10.2.1.1. What is speech-to-text?

<https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/speech-to-text>

##### 10.2.1.2. Get started with speech-to-text

<https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/get-started-speech-to-text?tabs=windowsinstall&pivots=programming-language-csharp>

#### Prerequisiti

- Azure subscription - [Create one for free](#)
- [Create a Speech resource](#) in the Azure portal to get your key and endpoint. You can use the free pricing tier (F0) to try the service, and upgrade later to a paid tier for production.

Alla fine si ottiene un Servizio come il seguente:

- Get the subscription key and regional endpoint. After your Speech resource is deployed, select **Go to resource** to view and manage keys. For more information about subscription keys and other Cognitive Services resources, see [Get the keys for your resource](#).

Creiamo un progetto console di Visual Studio, installiamo il pacchetto **Microsoft.CognitiveServices.Speech** e scriviamo il seguente codice di esempio, seguendo gli esempi descritti qui:

<https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-recognize-speech?tabs=windowsinstall&pivots=programming-language-csharp>

//file: Program.cs

```
using System;
using System.IO;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using HttpProxyControl;
using System.Text.Json.Serialization;
using System.Text.Json;
//https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-recognize-speech?tabs=windowsinstall&pivots=programming-language-csharp
namespace SpeechRecognitionDemo1
{
    public class AzureSpeechServiceStore
    {
        [JsonPropertyName("api_key")]
        public string APIKeyValue { get; set; }

        [JsonPropertyName("region_location")]
    }
}
```

- Se non è già stato creato un gruppo di risorse, occorre preliminarmente crearne uno. Un gruppo di risorse è un contenitore di risorse collegate su Azure.
- Definire la regione nella quale opererà il lo speech service (West Europe)
- dare un nome allo speech service
- selezionare come Pricing tier, il piano F0 gratuito

```

    public string RegionLocation { get; set; }

    [JsonPropertyName("endpoint")]
    public string EndPoint { get; set; }
}
class Program
{
    static AzureSpeechServiceStore azureSpeechServiceStore = GetDataFromStore();
    static readonly string azureSpeechServiceKey = azureSpeechServiceStore.APIKeyValue;
    static readonly string azureSpeechServiceRegion =
        azureSpeechServiceStore.RegionLocation;

    static AzureSpeechServiceStore GetDataFromStore()
    {
        //il file è nella cartella che contiene la soluzione corrente
        //il file contiene un oggetto JSON del tipo:
        //{
        //    "api_key": "api_key_value",
        //    "region_location": "region_location_value",
        //    "endpoint": "endpoint_value"
        //}
        string keyStorePath = "../../../../../MyAzureRoboVoiceStore.json";
        string store = File.ReadAllText(keyStorePath);
        AzureSpeechServiceStore azureSpeechServiceStore =
JsonSerializer.Deserialize<AzureSpeechServiceStore>(store);
        return azureSpeechServiceStore;
    }
    async static Task FromFile(SpeechConfig speechConfig)
    {
        using var audioConfig = AudioConfig.FromWavFileInput("PathToFile.wav");
        using var recognizer = new SpeechRecognizer(speechConfig, audioConfig);
        var result = await recognizer.RecognizeOnceAsync();
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
    }
    async static Task FromStream(SpeechConfig speechConfig)
    {
        var reader = new BinaryReader(File.OpenRead("PathToFile.wav"));
        using var audioInputStream = AudioInputStream.CreatePushStream();
        using var audioConfig = AudioConfig.FromStreamInput(audioInputStream);
        using var recognizer = new SpeechRecognizer(speechConfig, audioConfig);
        byte[] readBytes;
        do
        {
            readBytes = reader.ReadBytes(1024);
            audioInputStream.Write(readBytes, readBytes.Length);
        } while (readBytes.Length > 0);
        var result = await recognizer.RecognizeOnceAsync();
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
    }
    async static Task FromMic(SpeechConfig speechConfig)
    {
        using var audioConfig = AudioConfig.FromDefaultMicrophoneInput();
        using var recognizer = new SpeechRecognizer(speechConfig, audioConfig);
        Console.WriteLine("Speak into your microphone.");
        var result = await recognizer.RecognizeOnceAsync();
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
    }
    async static Task ContinuousRecognition(SpeechConfig speechConfig)
    {
        using var audioConfig = AudioConfig.FromDefaultMicrophoneInput();
        using var recognizer = new SpeechRecognizer(speechConfig, audioConfig);
        var stopRecognition = new TaskCompletionSource<int>();
        recognizer.Recognizing += (s, e) =>

```

Il codice precedente implementa diverse funzionalità:

- Speech to text di una sola frase prendendo come sorgente il microfono di default.

If you want to use a *specific* audio input device, you need to specify the device ID in AudioConfig. Learn [how to get the device ID](#) for your audio input device.

```
audioConfig = AudioConfig.FromMicrophoneInput("<device id>");
```

In C#, you can use the [NAudio](#) library to access the CoreAudio API and enumerate devices as follows:

```
using System;

using NAudio.CoreAudioApi;

namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            var enumerator = new MMDeviceEnumerator();
            foreach (var endpoint in enumerator.EnumerateAudioEndPoints(DataFlow.Capture,
DeviceState.Active))
            {
                Console.WriteLine("{0} ({1})", endpoint.FriendlyName, endpoint.ID);
            }
        }
    }
}
```

A sample device ID is {0.0.1.00000000}.{5f23ab69-6181-4f4a-81a4-45414013aac8}.

Speech to text continuo, prendendo come sorgente audio il microfono di default

- Speech to text da file

<https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-recognize-speech?tabs=windowsinstall&pivots=programming-language-csharp#recognize-speech-from-a-file>

- Speech to text da stream

<https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-recognize-speech?tabs=windowsinstall&pivots=programming-language-csharp#recognize-speech-from-an-in-memory-stream>

## 10.2.2. Text to speech

<https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/text-to-speech>

### 10.2.2.1. Get started with text to speech

<https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/get-started-text-to-speech>

I prerequisiti sono gli stessi dell'esempio relativo allo speech to text. Nel seguente esempio verrà usato lo stesso speech service utilizzato per il progetto relativo allo speech to text. Quindi non ci sarà bisogno di creare uno specifico endpoint su Azure, ma verrà usato proprio quello precedentemente creato per l'esempio relativo allo speech to text.

The text-to-speech feature in the Azure Speech service supports more than 270 voices and more than

110 languages and variants. You can get the [full list](#) or try them in a [text-to-speech demo](#).

Creiamo un progetto console di Visual Studio, installiamo il pacchetto **Microsoft.CognitiveServices.Speech** e scriviamo il seguente codice di esempio, seguendo gli esempi descritti qui:

<https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/get-started-text-to-speech>

//File: Program.cs

```
using System;
using System.IO;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using HttpProxyControl;
using System.Text.Json.Serialization;
using System.Text.Json;
using Microsoft.CognitiveServices.Speech.Audio;

namespace TextToSpeechDemo1
{
    public class AzureSpeechServiceStore
    {
        [JsonPropertyName("api_key")]
        public string APIKeyValue { get; set; }

        [JsonPropertyName("region_location")]
        public string RegionLocation { get; set; }

        [JsonPropertyName("endpoint")]
        public string EndPoint { get; set; }
    }

    class Program
    {
        static AzureSpeechServiceStore azureSpeechServiceStore = GetDataFromStore();
        static readonly string azureSpeechServiceKey = azureSpeechServiceStore.APIKeyValue;
        static readonly string azureSpeechServiceRegion = azureSpeechServiceStore.RegionLocation;

        static AzureSpeechServiceStore GetDataFromStore()
        {
            //il file è nella cartella che contiene la soluzione corrente
            //il file contiene un oggetto JSON del tipo:
            // {
            //   "api_key": "api_key_value",
            //   "region_location": "region_location_value",
            //   "endpoint": "endpoint_value"
            //}
            string keyStorePath = "../../../../../MyAzureRoboVoiceStore.json";
            string store = File.ReadAllText(keyStorePath);
            AzureSpeechServiceStore azureSpeechServiceStore =
JsonSerializer.Deserialize<AzureSpeechServiceStore>(store);
            return azureSpeechServiceStore;
        }

        async static Task Main(string[] args)
        {
            string testo = @"Il Filocolo (secondo un'etimologia approssimativa «fatica d'amore») è un romanzo in prosa: rappresenta una svolta rispetto ai romanzi delle origini scritti in versi. La storia ha come protagonisti Florio, figlio di un re saraceno, e Biancifiore (o Biancofiore), una schiava cristiana abbandonata da bambina. I due fanciulli crescono assieme e da grandi, in seguito alla lettura del libro di Ovidio Ars Amandi s'innamorano, come era successo per Paolo e Francesca dopo avere letto Ginevra e Lancillotto. Tuttavia il padre di Florio decide di separarli vendendo Biancifiore a dei mercanti. Florio decide quindi di andarla a cercare e dopo mille peripezie (da qui il titolo Filocolo) la rincontra. Infine, il giovane si converte al cristianesimo e sposa la fanciulla.";
            await EsempioTextToSpeechAudioOutput(testo);
            //await EsempioTextToSpeechFileOutput(testo, "Filocolo.wav");
        }
    }
}
```

```

        //await EsempioTextToSpeechStreamOutput(testo, "Filocolo2.wav");
    }

    private static async Task EsempioTextToSpeechAudioOutput(string text)
    {
        //https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/get-
        started-text-to-speech?tabs=script%2Cbrowserjs%2Cwindowsinstall&pivots=programming-language-
        csharp#synthesize-to-speaker-output
        var config = SpeechConfig.FromSubscription(azureSpeechServiceKey,
        azureSpeechServiceRegion);
        //imposta automaticamente il proxy se presente
        ProxyParams? proxyParams = HttpProxyHelper.GetHttpClientProxyParams();
        if (proxyParams.HasValue)
        {
            config.SetProxy(proxyParams.Value.ProxyAddress, proxyParams.Value.ProxyPort);
        }
        // Note: if only language is set, the default voice of that language is chosen.
        config.SpeechSynthesisLanguage = "it-IT"; // For example, "de-DE"
        // The voice setting will overwrite the language setting.
        // The voice setting will not overwrite the voice element in input SSML.
        //https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/language-
        support#prebuilt-neural-voices
        config.SpeechSynthesisVoiceName = "it-IT-ElsaNeural";
        using var synthesizer = new SpeechSynthesizer(config);
        //await synthesizer.SpeakTextAsync("A simple test to write to a file.");
        await synthesizer.SpeakTextAsync(text);
    }

    private static async Task EsempioTextToSpeechFileOutput(string text, string
    outputFileName)
    {
        //https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/get-
        started-text-to-speech?tabs=script%2Cbrowserjs%2Cwindowsinstall&pivots=programming-language-
        csharp#synthesize-speech-to-a-file
        var config = SpeechConfig.FromSubscription(azureSpeechServiceKey,
        azureSpeechServiceRegion);
        //imposta automaticamente il proxy se presente
        ProxyParams? proxyParams = HttpProxyHelper.GetHttpClientProxyParams();
        if (proxyParams.HasValue)
        {
            config.SetProxy(proxyParams.Value.ProxyAddress, proxyParams.Value.ProxyPort);
        }
        // Note: if only language is set, the default voice of that language is chosen.
        config.SpeechSynthesisLanguage = "it-IT"; // For example, "de-DE"
        // The voice setting will overwrite the language setting.
        // The voice setting will not overwrite the voice element in input SSML.
        //https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/language-
        support#prebuilt-neural-voices
        config.SpeechSynthesisVoiceName = "it-IT-ElsaNeural";
        using var audioConfig = AudioConfig.FromWavFileOutput(outputFileName);
        using var synthesizer = new SpeechSynthesizer(config, audioConfig);
        //await synthesizer.SpeakTextAsync("A simple test to write to a file.");
        await synthesizer.SpeakTextAsync(text);
    }

    static async Task EsempioTextToSpeechStreamOutput(string text, string outputFileName)
    {
        //https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/get-
        started-text-to-speech?tabs=script%2Cbrowserjs%2Cwindowsinstall&pivots=programming-language-
        csharp#get-a-result-as-an-in-memory-stream
        var config = SpeechConfig.FromSubscription(azureSpeechServiceKey,
        azureSpeechServiceRegion);
        //imposta automaticamente il proxy se presente
        ProxyParams? proxyParams = HttpProxyHelper.GetHttpClientProxyParams();
        if (proxyParams.HasValue)
        {
            config.SetProxy(proxyParams.Value.ProxyAddress, proxyParams.Value.ProxyPort);
        }
    }

```

```
        }
        // Note: if only language is set, the default voice of that language is chosen.
        config.SpeechSynthesisLanguage = "it-IT"; // For example, "de-DE"
        // The voice setting will overwrite the language setting.
        // The voice setting will not overwrite the voice element in input SSML.
        //https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/language-support#prebuilt-neural-voices
        config.SpeechSynthesisVoiceName = "it-IT-ElsaNeural";

    config.SetSpeechSynthesisOutputFormat(SpeechSynthesisOutputFormat.Riff24Khz16BitMonoPcm);
        using var synthesizer = new SpeechSynthesizer(config, null);
        var result = await synthesizer.SpeakTextAsync(text);
        using var stream = AudioDataStream.FromResult(result);
        await stream.SaveToWaveFileAsync(outputFileName);
    }
}
```

### 10.2.3. Intent Recognition

<https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/intent-recognition>

### **10.2.3.1. Pattern matching**

<https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-use-simple-language-pattern-matching?pivots=programming-language-csharp>

The SDK provides an embedded pattern matcher that you can use to recognize intents in a very strict way. This is useful for when you need a quick offline solution. This works especially well when the user is going to be trained in some way or can be expected to use specific phrases to trigger intents. For example: "Go to floor seven", or "Turn on the lamp" etc. It is recommended to start here and if it no longer meets your needs, switch to using LUIS or a combination of the two.

## When should you use this?

Use this sample code if:

- You are only interested in matching very strictly what the user said. These patterns match more aggressively than LUIS.
  - You do not have access to a [LUIS](#) app, but still want intents. This can be helpful since it is embedded within the SDK.
  - You cannot or do not want to create a [LUIS](#) app but you still want some voice-commanding capability.

If you do not have access to a [LUIS](#) app, but still want intents, this can be helpful since it is embedded within the SDK.

## Prerequisites

Be sure you have the following items before you begin this guide:

- A [Cognitive Services Azure resource](#) or a [Unified Speech resource](#)
  - [Visual Studio 2019](#) (any edition).

A pattern is a phrase that includes an Entity somewhere within it. An Entity is defined by wrapping a word in curly brackets. For example:

Take me to the {floorName}

This defines an Entity with the ID "floorName" which is case-sensitive.

All other special characters and punctuation will be ignored.

Intents will be added using calls to the IntentRecognizer->AddIntent() API.

Creiamo un progetto Console di Visual Studio e installiamo il pacchetto **Microsoft.CognitiveServices.Speech**

<https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-use-simple-language-pattern-matching?pivots=programming-language-csharp>

<https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-use-custom-entity-pattern-matching?pivots=programming-language-csharp>

```
using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Serialization;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Intent;
using HttpProxyControl;

namespace PatternMatchingRecognitionDemo1
{
    public class AzureSpeechServiceStore
    {
        [JsonPropertyName("api_key")]
        public string APIKeyValue { get; set; }

        [JsonPropertyName("region_location")]
        public string RegionLocation { get; set; }

        [JsonPropertyName("endpoint")]
        public string EndPoint { get; set; }
    }
    class Program
    {
        //https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-use-simple-language-pattern-matching?pivots=programming-language-csharp
        //https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-use-custom-entity-pattern-matching?pivots=programming-language-csharp
        static AzureSpeechServiceStore azureSpeechServiceStore = GetDataFromStore();
        static readonly string azureSpeechServiceKey = azureSpeechServiceStore.APIKeyValue;
        static readonly string azureSpeechServiceRegion = azureSpeechServiceStore.RegionLocation;

        static AzureSpeechServiceStore GetDataFromStore()
        {
            //il file è nella cartella che contiene la soluzione corrente
            //il file contiene un oggetto JSON del tipo:
            // {
            //   "api_key": "api_key_value",
            //   "region_location": "region_location_value",
            //   "endpoint": "endpoint_value"
            //}
            string keyStorePath = "../../../../../MyAzureRoboVoiceStore.json";
            string store = File.ReadAllText(keyStorePath);
            AzureSpeechServiceStore azureSpeechServiceStore =
JsonSerializer.Deserialize<AzureSpeechServiceStore>(store);
            return azureSpeechServiceStore;
        }
        static void Main(string[] args)
        {
            IntentPatternMatchingWithMicrophoneAsync().Wait();
        }
        private static async Task IntentPatternMatchingWithMicrophoneAsync()
        {
```

```

var config = SpeechConfig.FromSubscription(azureSpeechServiceKey,
azureSpeechServiceRegion);
//imposta automaticamente il proxy se presente
ProxyParams? proxyParams = HttpProxyHelper.GetHttpClientProxyParams();
if (proxyParams.HasValue)
{
    config.SetProxy(proxyParams.Value.ProxyAddress, proxyParams.Value.ProxyPort);
}
config.SpeechRecognitionLanguage = "it-IT"; // For example, "de-DE"
using (var intentRecognizer = new IntentRecognizer(config))
{
    intentRecognizer.AddIntent("Portami al piano {floorName}.", "ChangeFloors");
    intentRecognizer.AddIntent("Vai al piano {floorName}.", "ChangeFloors");
    intentRecognizer.AddIntent("{action} la porta.", "OpenCloseDoor");

    Console.WriteLine("Say something...");

    var result = await intentRecognizer.RecognizeOnceAsync();

    switch (result.Reason)
    {
        case ResultReason.RecognizedSpeech:
            Console.WriteLine($"RECOGNIZED: Text= {result.Text}");
            Console.WriteLine($"      Intent not recognized.");
            break;
        case ResultReason.RecognizedIntent:
            Console.WriteLine($"RECOGNIZED: Text= {result.Text}");
            Console.WriteLine($"      Intent Id= {result.IntentId}.");
            var entities = result.Entities;
            if (entities.TryGetValue("floorName", out string floorNameValue))
            {
                Console.WriteLine($"      FloorName= {floorNameValue}");
            }

            if (entities.TryGetValue("action", out string actionValue))
            {
                Console.WriteLine($"      Action= {actionValue}");
            }

            break;
        case ResultReason.NoMatch:
            Console.WriteLine($"NOMATCH: Speech could not be recognized.");
            var noMatch = NoMatchDetails.FromResult(result);
            switch (noMatch.Reason)
            {
                case NoMatchReason.NotRecognized:
                    Console.WriteLine($"NOMATCH: Speech was detected, but not
recognized.");
                    break;
                case NoMatchReason.InitialSilenceTimeout:
                    Console.WriteLine($"NOMATCH: The start of the audio stream
contains only silence, and the service timed out waiting for speech.");
                    break;
                case NoMatchReason.InitialBabbleTimeout:
                    Console.WriteLine($"NOMATCH: The start of the audio stream
contains only noise, and the service timed out waiting for speech.");
                    break;
                case NoMatchReason.KeywordNotRecognized:
                    Console.WriteLine($"NOMATCH: Keyword not recognized");
                    break;
            }
            break;
        case ResultReason.Canceled:
            var cancellation = CancellationDetails.FromResult(result);
            Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

            if (cancellation.Reason == CancellationReason.Error)

```

```
        {
            Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
            Console.WriteLine($"CANCELED:");
            ErrorDetails = {cancellation.ErrorDetails}");
            Console.WriteLine($"CANCELED: Did you update the subscription
info?");
        }
        break;
    default:
        break;
}
}
}
}
```

### **10.2.3.2. LUIS**

<https://www.luis.ai/>

The Microsoft LUIS service is available as a complete AI intent service that works well when your domain of possible intents is large and you are not really sure what the user will say. It supports many complex scenarios, intents, and entities.

### **10.2.3.2.1.What is LUIS**

<https://docs.microsoft.com/en-us/azure/cognitive-services/luis/what-is-luis>

### **10.2.3.2.2. How to create and manage LUIS resources**

<https://docs.microsoft.com/en-us/azure/cognitive-services/LUIS/luis-how-to-azure-subscription?tabs=portal>

<https://docs.microsoft.com/en-us/azure/cognitive-services/LUIS/luis-how-to-azure-subscription?tabs=portal>

### **10.2.3.2.3. Get Started Creating App**

<https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-get-started-create-app>

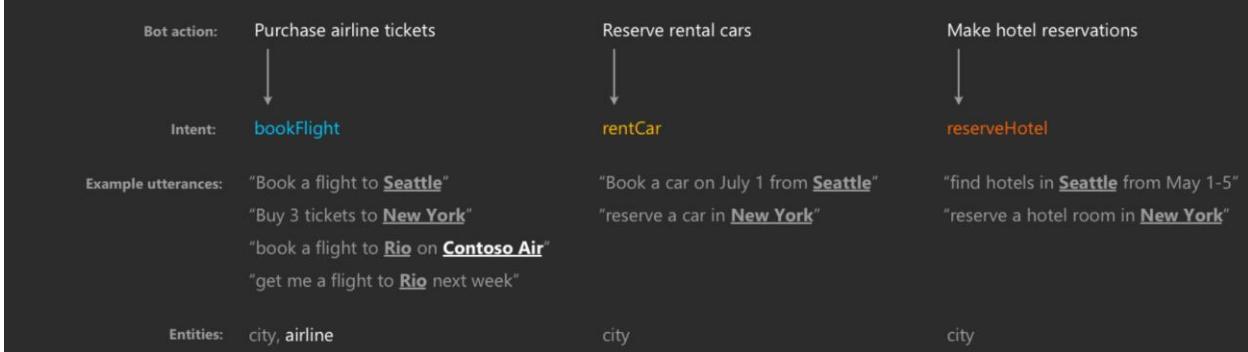
Seguendo il tutorial della guida Microsoft creiamo un'applicazione di comprensione del linguaggio che riconosce gli intent dell'utente a partire dalle frasi pronunciate. Per prima cosa occorre creare un gruppo di risorse e, al suo interno, una risorsa di tipo Language Understanding. I passaggi sono riportati nella guida Microsoft. Successivamente possiamo creare un'app di riconoscimento del linguaggio sul portale [www.luis.ai](http://www.luis.ai)

Per far in modo che l'app funzioni, occorre creare un modello di classificazione basato sui concetti di:

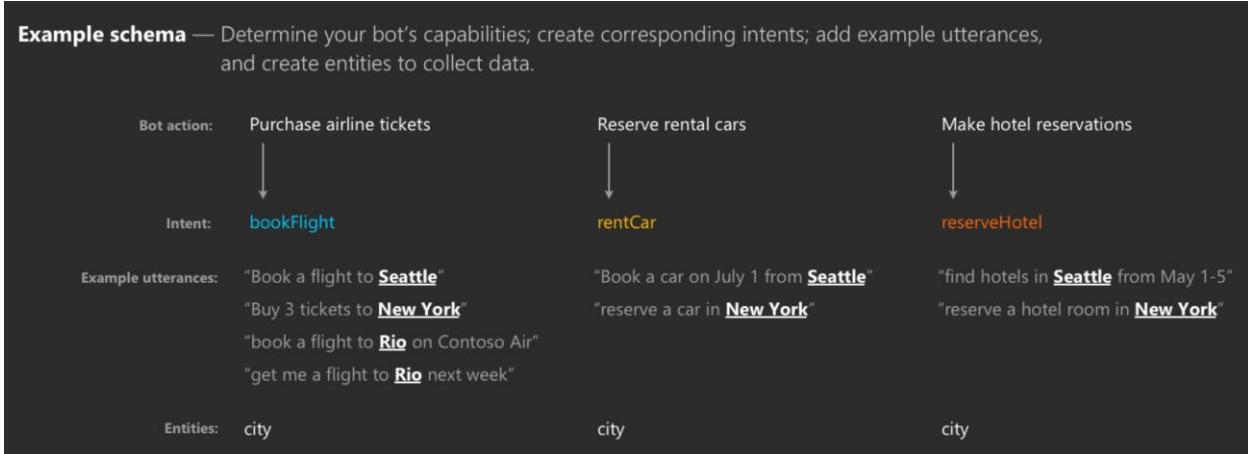
- **Intent**: l'intenzione dell'utente quando pronuncia una frase, ossia cosa vuole fare
  - **Utterance**: espressione di riconoscimento, ossia una frase anticipata associata a un determinato intent
  - **Entity**: categoria che racchiude i termini riconosciuti nell'utterance.

Ad esempio:

**Example schema** — Determine your bot's capabilities; create corresponding intents; add example utterances, and create entities to collect data. Create additional entities for each type of information collected.



**Example schema** — Determine your bot's capabilities; create corresponding intents; add example utterances, and create entities to collect data.



Utilizziamo i prebuilt domains per creare un'applicazione in grado di riconoscere frasi relative a previsioni meteo e a indicazioni geografiche.

Una volta compreso il meccanismo si potrà creare un'applicazione in grado di riconoscere qualsiasi tipo di intent. Nell'esempio seguente è stato caricato l'intero dominio Weather e una parte del

## dominio Places.

Cognitive Services Language Understanding - westeurope

Help us improve LUIS. Give us feedback!

My LUIS / MyIntelligentResponder v0.1

App Assets

**Intents**

- Entities
- Prebuilt Domains
- Improve app performance
- Review endpoint utterances
- Features
- Patterns

**Intents**

+ Create + Add prebuilt domain intent Rename Delete

Name ↑	Examples	Features
None	25	+ Add feature
Places.GetAddress	92	+ Add feature
Weather.ChangeTemperatureUnit	22	+ Add feature
Weather.CheckWeatherTime	61	+ Add feature
Weather.CheckWeatherValue	111	+ Add feature
Weather.GetWeatherAdvisory	36	+ Add feature
Weather.QueryWeather	75	+ Add feature

My LUIS / MyIntelligentResponder v0.1

App Assets

Intents

**Entities**

- Entities
- Prebuilt Domains
- Improve app performance
- Review endpoint utterances
- Features
- Patterns

**Entities**

+ Create + Add prebuilt entity + Add prebuilt domain entity Rename

Name ↑	Type
datetimeV2	Prebuilt
dimension	Prebuilt
Places.AbsoluteLocation	Machine learned
Places.Distance	Machine learned
Places.Nearby	Machine learned
Places.PlaceName	Machine learned
Places.PlaceType	Machine learned
temperature	Prebuilt
Weather.AdditionalWeatherCondition	List
Weather.Historic	Machine learned

App Assets

Intents

Entities

Prebuilt Domains

Improve app performance

Review endpoint utterances

Features

Patterns

Vediamo qualche esempio di utterance relativo all'intent **Weather.CheckWeatherTime**:

quando pioverà a middleton , wisconsin  
Weather.W... Places.AbsoluteLocation

quando pioverà a seattle  
Weather.W... Places.Abs...

quando pioverà di più domani  
Weather.W... Weather... datetime...

quando pioverà domani  
Weather.W... datetime...

quando pioverà la prossima volta  
Weather.W...

quando pioverà oggi  
Weather.W... dated...

quando pioverà questa settimana  
Weather.W... datetimeV2

quando pioverà questo fine settimana  
Weather.W... datetimeV2

quando sono i mesi migliori per visitare l ' australia  
Weather.Historic Weather.Suit... Places.Absolute...

un buon periodo per andare in canada  
Weather.Historic Weather.Suit... Places.A...

## Nel caso dell'intent Weather.CheckWeatherValue:

quali sono le previsioni meteo per milano il 25 febbraio  
Weather.Weath... datetimeV2

che tempo fa a monticello brianza  
Weath... Places.AbsoluteLocation

velocità e direzione del vento a satellite beach ( fl )  
Weather.Ad... Weather.Addit... Weath... Places.AbsoluteLocation

quali sono le previsioni meteo di domani per la lombardia  
Weather.Weath... datetime... Places.Absolute...

che umidità abbiamo oggi a monticello brianza  
Weather.W... dated... Places.AbsoluteLocation

che umidità abbiamo oggi  
Weather.W... dated...

quali sono le previsioni per oggi  
Weather.Weath... dated...

la temperatura di tokio  
Weather.Weather... Places....

che tempo fa a tokio oggi  
Weath... Places... dated...

quale sarà la temperatura domani a monticello brianza  
Weather.Weather... datetime... Places.AbsoluteLocation

## Nel caso dell'intent Places.GetAddress:

trova qualche buon caffè nelle vicinanze  
 Places.PlaceType      Places.Nearby

trova qualche ristorante buono ad acapulco  
 Places.PlaceType      Places.Absol...

trova saloni per unghie nella mia zona  
 Places.PlaceType      Places.Nearby

trova un barbiere vicino alla mia posizione  
 Places.Place...

trova un distributore vicino a me  
 Places.PlaceType      Places.Nearby

trova un home depot che prende prenotazioni  
 Places.PlaceName

trova un negozi di scarpe in questa zona  
 Places.PlaceType      Places.Nearby

trova un seven eleven vicino a me  
 Places.PlaceName      Places.Nearby

trovare qualche hotel economico nelle vicinanze  
 Places....      Places.Nearby

zuppa di pho vietnamita nelle vicinanze  
 Places.Nearby

Etc.

### 10.2.3.2.4. Get started with intent recognition

<https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/get-started-intent-recognition?pivots=programming-language-csharp>

Il seguente esempio mostra come creare un'applicazione client che sfrutta le potenzialità dell'Intelligenza Artificiale (AI) dell'applicazione LUIS creata su Azure. Il progetto utilizza un modello chiamato LanguageUnderstandingServiceResponse, per deserializzare la risposta dell'endpoint LUIS.

//File: LanguageUnderstandingServiceResponse.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.Json.Serialization;
using System.Threading.Tasks;

namespace IntentRecognitionWeatherDemo1.Model
{

    public class LanguageUnderstandingServiceResponse
    {
        public string Query { get; set; }
        public TopScoringIntent TopScoringIntent { get; set; }
        public EntityModel[] Entities { get; set; }
    }

    public class TopScoringIntent
    {
```

```

        public string Intent { get; set; }
        public float Score { get; set; }
    }

    public class EntityModel
    {
        public string Entity { get; set; }
        public string Type { get; set; }
        public int StartIndex { get; set; }
        public int EndIndex { get; set; }
        public float Score { get; set; }
        public Resolution Resolution { get; set; }
    }

    public class Resolution
    {
        public ValueModel[] Values { get; set; }
    }

    public class ValueModel
    {
        public string Timex { get; set; }
        public string Type { get; set; }
        public string Value { get; set; }
    }
}

```

//File: Program.cs

```

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Serialization;
using System.Threading.Tasks;
using IntentRecognitionWeatherDemo1.Model;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Intent;
using HttpProxyControl;

namespace IntentRecognitionWeatherDemo1
{

    public class AzureIntentRecognitionStore
    {
        [JsonPropertyName("luis_app_name")]
        public string LuisAppName { get; set; }

        [JsonPropertyName("luis_app_id")]
        public string LuisAppId { get; set; }

        [JsonPropertyName("luis_culture")]
        public string LuisCulture { get; set; }

        [JsonPropertyName("azure_resource")]
    }
}

```

```

    public AzureResource AzureResource { get; set; }
}

public class AzureResource
{
    [JsonPropertyName("resource_name")]
    public string ResourceName { get; set; }

    [JsonPropertyName("api_key")]
    public string ApiKey { get; set; }

    [JsonPropertyName("region_location")]
    public string RegionLocation { get; set; }

    [JsonPropertyName("endpoint")]
    public string Endpoint { get; set; }
}

class Program
{
    static AzureIntentRecognitionStore azureIntentRecognitionStore = GetDataFromStore();
    static readonly string azureIntentServiceKey =
        azureIntentRecognitionStore.AzureResource.ApiKey;
    static readonly string azureIntentServiceRegion =
        azureIntentRecognitionStore.AzureResource.RegionLocation;
    static readonly string luisIntentServiceAppId = azureIntentRecognitionStore.LuisAppId;
    static readonly string luisIntentServiceCulture = azureIntentRecognitionStore.LuisCulture;

    static AzureIntentRecognitionStore GetDataFromStore()
    {
        //il file è nella cartella che contiene la soluzione corrente
        //il file contiene un oggetto JSON del tipo:
        //{
        //    "luis_app_name": "value",
        //    "luis_app_id": "value",
        //    "luis_culture": "it-IT",
        //    "azure_resource": {
        //        "resource_name": "value",
        //        "api_key": "value",
        //        "region_location": "westeurope",
        //        "endpoint": "https://westeurope.api.cognitive.microsoft.com/"
        //    }
        //}
        string keyStorePath = "../../../../../MyAzureWeatherResponderStore.json";
        string store = File.ReadAllText(keyStorePath);
        AzureIntentRecognitionStore azureSpeechServiceStore =
            JsonSerializer.Deserialize<AzureIntentRecognitionStore>(store);
        return azureSpeechServiceStore;
    }
}

```

```

//https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-recognize-
intents-from-speech-csharp#continuous-recognition-from-a-file
public static async Task ContinuousRecognitionIntentFromFileAsync(SpeechConfig config,
LanguageUnderstandingModel model, string inputFile)
{
    // Creates an instance of a speech config with specified subscription key
    // and service region. Note that in contrast to other services supported by
    // the Cognitive Services Speech SDK, the Language Understanding service
    // requires a specific subscription key from https://www.luis.ai/.
    // The Language Understanding service calls the required key 'endpoint key'.
    // Once you've obtained it, replace with below with your own Language Understanding
subscription key
    // and service region (e.g., "westus").
    //var config = SpeechConfig.FromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

    // Creates an intent recognizer using file as audio input.
    // Replace with your own audio file name.
    //https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-
recognize-intents-from-speech-csharp#continuous-recognition-from-a-file
    using (var audioInput = AudioConfig.FromWavFileInput(inputFile))
    {
        using (var recognizer = new IntentRecognizer(config, audioInput))
        {
            // The TaskCompletionSource to stop recognition.
            var stopRecognition = new TaskCompletionSource<int>();

            // Creates a Language Understanding model using the app id, and adds specific
intents from your model
            //var model =
LanguageUnderstandingModel.FromAppId("YourLanguageUnderstandingAppId");
            //recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
            //recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
            //recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-
IntentId-here");
            //https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-
recognize-intents-from-speech-csharp#import-a-luis-model-and-add-intents
            recognizer.AddAllIntents(model);

            // Subscribes to events.
            recognizer.Recognizing += (s, e) =>
            {
                Console.WriteLine($"RECOGNIZING: Text={e.Result.Text}");
            };

            recognizer.Recognized += (s, e) =>
            {
                if (e.Result.Reason == ResultReason.RecognizedIntent)
                {
                    Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
                }
            };
        }
    }
}

```

```

        Console.WriteLine($"    Intent Id: {e.Result.IntentId}.");
        Console.WriteLine($"    Language Understanding JSON:
{e.Result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult)}.");
    }

    else if (e.Result.Reason == ResultReason.RecognizedSpeech)
    {
        Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
        Console.WriteLine($"    Intent not recognized.");
    }

    else if (e.Result.Reason == ResultReason.NoMatch)
    {
        Console.WriteLine($"NOMATCH: Speech could not be recognized.");
    }
}

recognizer.Canceled += (s, e) =>
{
    Console.WriteLine($"CANCELED: Reason={e.Reason}");

    if (e.Reason == CancellationReason.Error)
    {
        Console.WriteLine($"CANCELED: ErrorCode={e.ErrorCode}");
        Console.WriteLine($"CANCELED: ErrorDetails={e.ErrorDetails}");
        Console.WriteLine($"CANCELED: Did you update the subscription info?");
    }
}

stopRecognition.TrySetResult(0);
};

recognizer.SessionStarted += (s, e) =>
{
    Console.WriteLine("\n    Session started event.");
};

recognizer.SessionStopped += (s, e) =>
{
    Console.WriteLine("\n    Session stopped event.");
    Console.WriteLine("\nStop recognition.");
    stopRecognition.TrySetResult(0);
};

// Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop
recognition.

Console.WriteLine("Say something...");
//why we use ConfigureAwait(false)
//https://docs.microsoft.com/en-us/dotnet/fundamentals/code-analysis/quality-rules/ca2007
await recognizer.StartContinuousRecognitionAsync().ConfigureAwait(false);

// Waits for completion.

```

```

        // Use Task.WaitAny to keep the task rooted.
        Task.WaitAny(new[] { stopRecognition.Task });

        // Stops recognition.
        await recognizer.StopContinuousRecognitionAsync().ConfigureAwait(false);
    }
}

public static async Task ContinuousRecognitionIntentAsync(SpeechConfig config,
LanguageUnderstandingModel model)
{
    // Creates an instance of a speech config with specified subscription key
    // and service region. Note that in contrast to other services supported by
    // the Cognitive Services Speech SDK, the Language Understanding service
    // requires a specific subscription key from https://www.luis.ai/.
    // The Language Understanding service calls the required key 'endpoint key'.
    // Once you've obtained it, replace with below with your own Language Understanding
subscription key
    // and service region (e.g., "westus").
    //var config = SpeechConfig.FromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

    // Creates an intent recognizer using file as audio input.
    // Replace with your own audio file name.
    //https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-
recognize-intents-from-speech-csharp#continuous-recognition-from-a-file
    using (var audioInput = AudioConfig.FromDefaultMicrophoneInput())
    {
        using (var recognizer = new IntentRecognizer(config, audioInput))
        {
            // The TaskCompletionSource to stop recognition.
            var stopRecognition = new TaskCompletionSource<int>();

            // Creates a Language Understanding model using the app id, and adds specific
intents from your model
            //var model =
LanguageUnderstandingModel.FromAppId("YourLanguageUnderstandingAppId");
            //recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
            //recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
            //recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-
IntentId-here");
            //https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-
recognize-intents-from-speech-csharp#import-a-luis-model-and-add-intents
            recognizer.AddAllIntents(model);

            // Subscribes to events.
            recognizer.Recognizing += (s, e) =>
            {
                Console.WriteLine($"RECOGNIZING: Text={e.Result.Text}");
            };
        }
    }
}

```

```

recognizer.Recognized += (s, e) =>
{
    if (e.Result.Reason == ResultReason.RecognizedIntent)
    {
        Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
        Console.WriteLine($"      Intent Id: {e.Result.IntentId}.");
        string languageUnderstandingJSON =
e.Result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult);
        Console.WriteLine($"      Language Understanding JSON:
{languageUnderstandingJSON}.");
        JsonSerializerOptions options = new
JsonSerializerOptions(JsonSerializerDefaults.Web) { WriteIndented = true };
        LanguageUnderstandingServiceResponse resultObject =
JsonSerializer.Deserialize<LanguageUnderstandingServiceResponse>(languageUnderstandingJSON, options);
        Console.WriteLine("*****ANALISI DEL RISULTATO DA C*****");
        Console.WriteLine($"Intent principale:
{resultObject.TopScoringIntent.Intent}");
        Console.WriteLine($"Entities trovate:");
        foreach (var entity in resultObject.Entities)
        {
            Console.WriteLine($"entity: {entity.Entity}, tipo: {entity.Type} ");
            if (entity.Resolution != null)
            {
                foreach (var value in entity.Resolution.Values)
                {
                    Console.WriteLine($"\\tvalue: {value.Value}, type:
{value.Type}, timex: {value.Timex}");
                }
            }
        }
        Console.WriteLine("*****");
    }
    else if (e.Result.Reason == ResultReason.RecognizedSpeech)
    {
        Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
        Console.WriteLine($"      Intent not recognized.");
    }
    else if (e.Result.Reason == ResultReason.NoMatch)
    {
        Console.WriteLine($"NOMATCH: Speech could not be recognized.");
    }
};

recognizer.Canceled += (s, e) =>
{
    Console.WriteLine($"CANCELED: Reason={e.Reason}");

    if (e.Reason == CancellationReason.Error)
    {

```

```

        Console.WriteLine($"CANCELED: ErrorCode={e.ErrorCode}");
        Console.WriteLine($"CANCELED: ErrorDetails={e.ErrorDetails}");
        Console.WriteLine($"CANCELED: Did you update the subscription info?");
    }

    stopRecognition.TrySetResult(0);
};

recognizer.SessionStarted += (s, e) =>
{
    Console.WriteLine("\n      Session started event.");
};

recognizer.SessionStopped += (s, e) =>
{
    Console.WriteLine("\n      Session stopped event.");
    Console.WriteLine("\nStop recognition.");
    stopRecognition.TrySetResult(0);
};

// Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop
recognition.
Console.WriteLine("Say something...");
//why we use ConfigureAwait(false)
//https://docs.microsoft.com/en-us/dotnet/fundamentals/code-analysis/quality-rules/ca2007
await recognizer.StartContinuousRecognitionAsync().ConfigureAwait(false);

// Waits for completion.
// Use Task.WaitAny to keep the task rooted.
Task.WaitAny(new[] { stopRecognition.Task });

// Stops recognition.
await recognizer.StopContinuousRecognitionAsync().ConfigureAwait(false);
}

}

}

public static async Task RecognizeIntentAsync(SpeechConfig config, LanguageUnderstandingModel
model)
{
    // Creates an instance of a speech config with specified subscription key
    // and service region. Note that in contrast to other services supported by
    // the Cognitive Services Speech SDK, the Language Understanding service
    // requires a specific subscription key from https://www.luis.ai/.
    // The Language Understanding service calls the required key 'endpoint key'.
    // Once you've obtained it, replace with below with your own Language Understanding
subscription key
    // and service region (e.g., "westus").
    // The default language is "en-us".
}

```

```

// Creates an intent recognizer using microphone as audio input.
using (var recognizer = new IntentRecognizer(config))
{
    //recognizer.AddIntent(model, "Weather.ChangeTemperatureUnit", "id1");
    //recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
    //recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-
here");

        // To add all of the possible intents from a LUIS model to the recognizer, uncomment
the line below:
        // recognizer.AddAllIntents(model);
        //https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/how-to-
recognize-intents-from-speech-csharp#import-a-luis-model-and-add-intents
        recognizer.AddAllIntents(model);

        // Starts recognizing.
        Console.WriteLine("Say something...");

        // Starts intent recognition, and returns after a single utterance is recognized. The
end of a
        // single utterance is determined by listening for silence at the end or until a
maximum of 15
        // seconds of audio is processed. The task returns the recognition text as result.
        // Note: Since RecognizeOnceAsync() returns only a single utterance, it is suitable
only for single
        // shot recognition like command or query.
        // For long-running multi-utterance recognition, use StartContinuousRecognitionAsync()
instead.
        var result = await recognizer.RecognizeOnceAsync();

        // Checks result.
        switch (result.Reason)
        {
            case ResultReason.RecognizedIntent:
                Console.WriteLine($"RECOGNIZED: Text={result.Text}");
                Console.WriteLine($"      Intent Id: {result.IntentId}");
                var json =
result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult);
                Console.WriteLine($"      Language Understanding JSON: {json}");
                break;
            case ResultReason.RecognizedSpeech:
                Console.WriteLine($"RECOGNIZED: Text={result.Text}");
                Console.WriteLine($"      Intent not recognized.");
                break;
            case ResultReason.NoMatch:
                Console.WriteLine($"NOMATCH: Speech could not be recognized.");
                break;
            case ResultReason.Canceled:
                var cancellation = CancellationDetails.FromResult(result);
                Console.WriteLine($"CANCELED: Reason={cancellation Reason}");
        }
}

```

```

        if (cancellation.Reason == CancellationReason.Error)
        {
            Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
            Console.WriteLine($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
            Console.WriteLine($"CANCELED: Did you update the subscription info?");
        }
        break;
    }
}

static async Task Main()
{
    SpeechConfig config = SpeechConfig.FromSubscription(azureIntentServiceKey,
    azureIntentServiceRegion);
    config.SpeechRecognitionLanguage = luisIntentServiceCulture;
    ProxyParams? proxyParams = HttpProxyHelper.GetHttpClientProxyParams();
    if (proxyParams.HasValue)
    {
        config.SetProxy(proxyParams.Value.ProxyAddress, proxyParams.Value.ProxyPort);
    }
    // Creates a Language Understanding model using the app id, and adds specific intents from
    your model
    LanguageUnderstandingModel model =
    LanguageUnderstandingModel.FromAppId(luisIntentServiceAppId);
    await RecognizeIntentAsync(config, model);
    //await ContinuousRecognitionIntentAsync(config, model);
    Console.WriteLine("Please press <Return> to continue.");
    Console.ReadLine();
}
}
}

```

## 10.3. Il progetto Alice Neural

Creare l'assistente vocale "Alice Neural" che abbia le seguenti caratteristiche:

Quando l'utente dice "Ok Alice", l'applicazione si mette in ascolto per Intent Recognition, sfruttando le API di Azure Cognitive Services (Language Understanding).

Quando l'utente dice "stop", l'applicazione blocca l'attività in corso (ad esempio lettura di una pagina da Wikipedia, o la lettura delle condizioni meteo, etc.) e si riporta nelle condizioni iniziali di ascolto.

Le attività che l'assistente è in grado di riconoscere sono:

- a) Intent relativi alle condizioni meteo:

**Intent: Weather.CheckWeatherValue**

## Esempi:

quali sono le <u>previsioni</u> meteo per milano <u>il 25 febbraio</u> Weather.Weath... datetimeV2	che <u>tempo</u> farà <u>domani</u> a <u>monticello brianza</u> Weath... datetime... Places.AbsoluteLocation
che <u>tempo</u> fa a <u>monticello brianza</u> Weath... Places.AbsoluteLocation	le <u>previsioni</u> meteo per <u>domani</u> Weather.Weath... datetime...
velocità e direzione del vento a <u>satellite beach ( fl )</u> Weather.Ad... Weather.Addit... Weath... Places.AbsoluteLocation	le <u>previsioni</u> meteo per <u>oggi</u> Weather.Weath... dateti...
quali sono le <u>previsioni</u> meteo di <u>domani</u> per la <u>lombardia</u> Weather.Weath... datetime... Places.Absolut...	come è il <u>tempo</u> <u>oggi</u> Weath... dateti...
che <u>umidità</u> abbiamo <u>oggi</u> a <u>monticello brianza</u> Weather.W... dateti... Places.AbsoluteLocation	che <u>umidità</u> avremo <u>domani</u> Weather.W... datetime...
che <u>umidità</u> abbiamo <u>oggi</u> dateti...	che <u>umidità</u> ci sarà <u>domani</u> Weather.W... datetime...
quali sono le <u>previsioni</u> per <u>oggi</u> Weather.Weath... dateti...	qual è l' <u>umidità</u> <u>oggi</u> Weather.W... dateti...
la <u>temperatura</u> di <u>tokio</u> Weather.Weather... Places....	che tasso di <u>umidità</u> c' è <u>oggi</u> Weather.W... dateti...
che <u>tempo</u> fa a <u>tokio</u> <u>oggi</u> Weath... Places.... dateti...	che <u>temperatura</u> ci sarà a <u>stoccolma</u> <u>domani</u> Weather.Weather... datetime...
quale sarà la <u>temperatura</u> <u>domani</u> a <u>monticello brianza</u> Weather.Weather... datetime... Places.AbsoluteLocation	che temperatura c' è a <u>stoccolma</u>

Entity: Weather.WeatherCondition; possibili valori: e.g. tempo, temperatura, umidità, pressione, previsioni, previsione, sole, pioggia, neve, piove, etc.

Entity: Places.AbsoluteLocation; possibili valori: e.g. Milano, Roma, Napoli, Monticello Brianza, etc.

Entity: DateTimeV2; possibili valori: oggi, domani, dopodomani, 23 febbraio. I valori di DateTime sono restituiti da Azure in un formato standard yyyy-mm-dd

**Action:** l'applicazione recupera i dati delle previsioni meteo relative al Place.AbsoluteLocation e al DateTimeV2, usando le API di OpenWeatherMap. Nel caso in cui non sia specificata la data si assume la data di oggi. Nel caso in cui non sia specificato il luogo assumere il luogo corrente, ossia un luogo predefinito configurabile dalle impostazioni dell'applicazione.

Una volta ottenuti i dati delle previsioni meteo per quel giorno/ora, questi vengono processati e mediante la funzione di TextToSpeech di Azure viene fatta una lettura a voce.

**Intent:** Weather.CheckWeatherTime

## Esempi:

quando pioverà a middleton , wisconsin Weather.W... Places.AbsoluteLocation	quando dovrebbe piovere qui a escondido , california Weather.W... Places.AbsoluteLocation
quando pioverà a seattle Weather.W... Places.Abs...	quando è prevista nebbia a mesa Weather... Place...
quando pioverà di più domani Weather.W... Weather... datetime...	quando inizierà a piovere a tampa Weather.W... Places....
quando pioverà domani Weather.W... datetime...	quando inizierà a piovere nella città di san fernando Weather.W... Places.AbsoluteLoca...
quando pioverà la prossima volta Weather.W...	quando nevicherà domani nel comune di san dimas Weather.West... datetime... Places.Absolute...
quando pioverà oggi Weather.W... dateti...	quando nevicherà domenica nel municipio di berwyn Weather.West... datetimeV2 Places.A...
quando pioverà questa settimana Weather.W... datetimeV2	quando nevicherà la prossima volta Weather.West...
quando pioverà questo fine settimana Weather.W... datetimeV2	quando nevicherà oggi Weather.West... dateti...
quando sono i mesi migliori per visitare l ' australia Weather.Historic Weather.Suit... Places.Absolut...	quando pioverà a hyderabad , india Weather.W... Places.Absolute...
un buon periodo per andare in canada Weather.Historic Weather.Suit... Places.A...	quando pioverà a hyderabad , india Weather.W... Places.AbsoluteLocation

Entity: Weather.WeatherCondition; possibili valori: e.g. tempo, temperatura, umidità, pressione, previsioni, previsione, sole, pioggia, neve, piove, etc.

Entity: Places.AbsoluteLocation; possibili valori: e.g. Milano, Roma, Napoli, Monticello Brianza, etc.

Entity: DateTimeV2; possibili valori: oggi, domani, dopodomani, 23 febbraio. I valori di DateTime sono restituiti da Azure in un formato standard yyyy-mm—dd

Action: l'applicazione recupera i dati delle previsioni meteo relative al Place.AbsoluteLocation e al DateTimeV2, usando le API di OpenWeatherMap. Una volta ottenuti i dati delle previsioni meteo per quel giorno/ora, questi vengono processati. Nel caso in cui non sia specificata la data si assume la data di oggi. Nel caso in cui non sia specificato il luogo assumere il luogo corrente, ossia un luogo predefinito configurabile dalle impostazioni dell'applicazione.

In questo caso l'applicazione dovrebbe cercare di capire quale tipo di WheatherCondition è richiesta e usando le previsioni a 48 ore oppure a 7 giorni provare a capire quando si verificano le condizioni richieste, oppure dovrebbe rispondere che le condizioni non si verificheranno. Ad esempio, se la domanda è quando pioverà? l'applicazione dovrebbe recuperare i dati a 48 ore o 7 giorni e verificare se trova un periodo nel quale pioverà...a quel punto dovrebbe rispondere quando dovrebbe iniziare a piovere. L'applicazione client che riceve il risultato dell'intent recognition dovrebbe essere in grado di processare le variazioni linguistiche associate a un determinato valore di un'entity. Ad esempio, i termini "pioggia", "piove", "pioverà" dovrebbero essere trattati allo stesso modo e normalizzati rispetto al valore restituito da OpenWeatherMap...

Mediante e la funzione di TextToSpeech di Azure viene fatta una lettura a voce.

Intent: Weather.QueryWeather

Esempi:

ci sarà pioggia a napoli domani	sarà soleggiato oggi
Weather.W... Places.A... datetime...	Weather.Weath... dateti...
sarà nuvoloso domani	pioverà all ' una lunedì
Weather.We... datetime...	Weather.W...
ci sarà vento oggi	pioverà a west columbia stasera
Weath... dateti...	Weather.W... Places.AbsoluteLocati... datetimeV2
ci sarà pioggia domani a napoli	sarà soleggiato a saint austell domenica
Weather.W... datetime... Places.A...	Weather.Weath... Places.AbsoluteLocati... datetimeV2
sarà nuvoloso domani a pescara	sarà abbastanza caldo per nuotare
Weather.We... datetime... Places.Abs...	Weath... Weather.S...
ci sarà pioggia domani	pioverà in gambia
Weather.W... datetime...	Weather.W... Places.A...
ci sarà il sole domani	pioverà tra cinque giorni
Weat... datetime...	Weather.W... datetimeV2
ci sarà il sole domani a monticello brianza	piove per tutto il giorno
Weat... datetime... Places.AbsoluteLocation	Weath... datetimeV2
piove domani a napoli	pioverà alle cinque domani
datetime... Places.A...	Weather.W... datetimeV2
pioverà stasera	pioverà a twickenham oggi
Weather.W... datetimeV2	Weather.W... dateti...

Entity: Weather.WeatherCondition; possibili valori: e.g. tempo, temperatura, umidità, pressione, previsioni, previsione, sole, pioggia, neve, piove, etc.

Entity: Places.AbsoluteLocation; possibili valori: e.g. Milano, Roma, Napoli, Monticello Brianza, etc.

Entity: DateTimeV2; possibili valori: oggi, domani, dopodomani, 23 febbraio. I valori di DateTime sono restituiti da Azure in un formato standard yyyy-mm—dd

Action: l'applicazione recupera i dati delle previsioni meteo relative al Place.AbsoluteLocation e al DateTimeV2 usando le API di OpenWeatherMap. Una volta ottenuti i dati delle previsioni meteo per quel giorno/ora, questi vengono processati. Nel caso in cui non sia specificata la data si assume la data di oggi. Nel caso in cui non sia specificato il luogo assumere il luogo corrente, ossia un luogo predefinito configurabile dalle impostazioni dell'applicazione.

In questo caso **l'applicazione dovrebbe cercare di capire quale tipo di wheather condition è richiesta e usando le previsioni a 48 ore oppure a 7 giorni provare a capire se si verificano le condizioni richieste, oppure dovrebbe rispondere che le condizioni non si verificheranno. Ad esempio, se la domanda è se pioverà? l'applicazione dovrebbe recuperare i dati a 48 ore o 7 giorni e vedere se trova un periodo nel quale pioverà...a quel punto dovrebbe rispondere se pioverà oppure no (sì pioverà..., oppure no) e indicare le previsioni meteo per la data/ora e il luogo indicato.** L'applicazione client che riceve il risultato dell'intent recognition dovrebbe essere in grado di processare le variazioni linguistiche associate a un determinato valore di un'entity. Ad esempio, i termini "pioggia", "piove", "pioverà" dovrebbero essere trattati allo stesso modo e normalizzati rispetto al valore restituito da OpenWeatherMap...

Mediante e la funzione di TextToSpeech di Azure viene fatta una lettura a voce.

- b) Intent relativi alla ricerca di luoghi, attività. etc.:

## Intent: Places.FindPlaces

Esempi:

- trova qualche buon negozio di caffè nelle vicinanze  
Places.PlaceType      Places.Nearby
- trova qualche ristorante buono ad acapulco  
Places.PlaceType      Places.AbsoluteLocation
- trova saloni per unghie nella mia zona  
Places.PlaceType      Places.Nearby
- trova un barbiere vicino alla mia posizione  
Places.Place...
- trova un distributore vicino a me  
Places.PlaceType      Places.Nearby
- trova un home depot che prende prenotazioni  
Places.PlaceName
- trova un negozio di scarpe in questa zona  
Places.PlaceType      Places.Nearby
- trova un seven eleven vicino a me  
Places.PlaceName      Places.Nearby
- trovare qualche hotel economico nelle vicinanze  
Places....      Places.Nearby
- zuppa di pho vietnamita nelle vicinanze  
Places.Nearby

Entity: Places.PlaceName; possibili valori: itt tech, chiesa battista di live oak, Microsoft store, studio Hudson, clinica veterinaria paw, Colosseo, Torre di Pisa

Entity: Places.AbsoluteLocation; possibili valori: Los Angeles, Belgio, Monticello Brianza

Entity: Places.Nearby; possibili valori: più vicino, nelle vicinanze, in questa zona

Entity: Places.PlaceType; possibili valori: riparazioni auto, università, aeroporto, hotel, negozio, ristorante

Entity: Places.PlaceDistance; possibili valori: raggio di 5 chilometri, raggio di 500 metri, etc.

Entity: DateTimeV2; possibili valori: oggi, domani, dopodomani, 23 febbraio. I valori di DateTime sono restituiti da Azure in un formato standard yyyy-mm-dd

**Action:** l'applicazione recupera i dati richiesti utilizzando le Bing Maps API. In particolare, può essere utile sfruttare la funzionalità di Location Recognition vista a lezione per trovare esercizi commerciali e punti di interesse intorno ad una certa posizione.

Mediante e la funzione di TextToSpeech di Azure viene fatta una lettura a voce.

## Intent: Places.GetDistance

Esempi:

quanto è distante monticello brianza  
Places.AbsoluteLocation

quanto è distante il grand canyon da qui  
Places.AbsoluteLoca...

a quante miglia è hastings da qui  
dimension Places.Absol...

distanza da qui a londra  
Places.A...

quanto è distante l ' ufficio del dottor spooner  
Places.PlaceType

quanto è distante bonner in montana  
Places.AbsoluteLocation

quanto è distante bay 66  
Places.Pla...

quanto è distante da wichita in kansas  
Places.AbsoluteLocation

quanto è distante la metropolitana più vicina  
Places.PlaceName Places.Nearby

quanto dista lufborough da casa mia  
Places.Absolute...

**Entity:** Places.PlaceName; possibili valori: itt tech, chiesa battista di live oak, Microsoft store, studio Hudson, clinica veterinaria paw, Colosseo, Torre di Pisa

**Entity:** Places.AbsoluteLocation; possibili valori: Los Angeles, Belgio, Monticello Brianza

**Action:** l'applicazione recupera i dati richiesti utilizzando le Bing Maps API. In particolare, può essere utile sfruttare la funzionalità di Routing vista a lezione per calcolare la distanza da un punto ad un altro.

c) Intent relativi alle ricerche su Wikipedia:

**Intent:** **Wiki.Search**

Esempi:

come funziona l' impollinazione  
Wiki.MainSearchItem

cosa sono i neuroni  
Wiki.Main...

che cos' è la funzione esponenziale  
Wiki.MainSearchItem

qual è la funzione del fegato  
Wiki.Subtitle... Wiki.Mai...

cerca la funzione esponenziale su wikipedia  
Wiki.MainSearchItem

che cosa è il pancreas  
Wiki.MainSe...

cerca dante alighieri su wikipedia  
Wiki.MainSearchItem

la funzione logaritmo  
Wiki.MainSearchItem

chi era dante alighieri  
Wiki.MainSearchItem

che cos' è una funzione sinusoidale  
Wiki.MainSearchItem

trova dante su wikipedia  
Wiki.M...

che cosa si intende per programmazione ad oggetti  
Wiki.MainSearchItem

trova i delfini su wikipedia  
Wiki.Main...

che cosa è la dieresi  
Wiki.Main...

che cosa è la gazza ladra  
Wiki.MainSearchIt...

come funziona uno scaldabagno elettrico  
Wiki.MainSearchItem

che cosa è il moto armonico  
Wiki.MainSearchItem

come funziona un motore elettrico  
Wiki.MainSearchItem

cosa sono le sinapsi  
Wiki.Main...

che cosa è un alternatore  
Wiki.MainSearchIt...

come funziona un transistor  
Wiki.MainSear...

chi ha vinto euro 2020  
Wiki.MainSear... dati...

come funziona una barca a vela  
Wiki.MainSearchItem

chi è cristiano ronaldo  
Wiki.MainSearchItem

come funziona un motore a scoppio  
Wiki.MainSearchItem

che cosa è squid game  
Wiki.MainSearch...

i capoluoghi di provincia italiani  
Wiki.MainSearchItem

chi è elon musk  
Wiki.MainSear...

quali sono le regioni d' italia  
Wiki.MainSearchItem

chi è la regina elisabetta  
Wiki.MainSearchItem

le principali attrazioni dell' italia  
Wiki.SubitemSe... Wiki.Mai...

quali sono le fonti di energia della russia  
Wiki.SubitemSearch Wiki.Mai...

quali sono le risorse naturali della francia  
Wiki.SubitemSearch Wiki.Mai...

l' economia della russia  
Wiki.Subite... Wiki.Mai...

chi è donald trump  
Wiki.MainSearchItem

cosa ha fatto mozart  
Wiki.Mai...

chi è joe biden  
Wiki.MainSear...

chi era mozart  
Wiki.Mai...

mostrami le informazioni sugli usa  
Wik...

parlami della vita di mozart  
Wiki.... Wiki.Mai...

quali sono le opere composte da mozart Wiki.Su... Wiki.Main...	che cosa è lo zibaldone Wiki.MainSe...
a cosa serve il pistone Wiki.Main...	lo zibaldone Wiki.MainSe...
a cosa serve il pistone del motore a scoppio Wiki.Subit... Wiki.MainSearchitem	la vita di giacomo leopardi Wiki... Wiki.MainSearchitem
come funziona un motore asincrono Wiki.MainSearchitem	le opere di giacomo leopardi Wiki.Su... Wiki.MainSearchitem
qual è il principio di funzionamento del motore asincrono Wiki.SubitemSearch	trova il ciclo dell ' acqua Wiki.MainSearchitem
come funziona il motore asincrono Wiki.MainSearchitem	cosa ha scritto giacomo leopardi Wiki.MainSearchitem
qual è il principio di funzionamento di una centrale nucleare Wiki.SubitemSearch	cosa dice la teoria della relatività ristretta Wiki.MainSearchitem
il principio di funzionamento di una centrale nucleare Wiki.SubitemSearch	cosa dice la teoria della relatività generale Wiki.MainSearchitem
che cosa è una centrale nucleare Wiki.MainSearchitem	cosa dice la teoria della relatività generale di albert einstein Wiki.SubitemS... Wiki.MainSearchitem
centrale nucleare Wiki.MainSearchitem	parlami della biografia di albert einstein Wiki.SubitemS... Wiki.MainSearchitem

etc.

Creare almeno un centinaio di esempi per Wiki.Search

Entity: Wiki.MainSearchItem. Rappresenta la chiave di ricerca principale su wikipedia

Entity: Wiki.SubItemSearch. Rappresenta un sotto argomento dell'argomento principale di ricerca.

Quando è intercettato un intent di tipo Wiki l'applicazione effettua una ricerca su Wikipedia con le REST API o le Action API viste a lezione. In particolare, se è presente solo il Wiki.MainSearchItem viene ricercata la pagina più pertinente e di questa viene recuperato il summary. Se, oltre al Wiki.MainSearchItem, sono presenti anche uno o più Wiki.SubItemSearch, l'applicazione recupera la pagina relativa al Wiki.MainSearchItem e su questa effettua una scansione delle sezioni per vedere se esiste una sezione che contiene i termini del Wiki.SubItemSearch. In caso affermativo, l'applicazione recupera la sezione specifica dalla pagina di wikipedia in formato wikitext e lo converte in plain text come mostrato a lezione. Se non trova il Wiki.SubItemSearch restituisce il summary della pagina relativa al Wiki.MainItemSearch.

Una volta ottenuto il testo richiesto da wikipedia (summary o sezione di pagina), l'applicazione effettua una lettura a voce, mediante e la funzione di TextToSpeech di Azure.

## Valutazione

Sufficienza: riconoscimento e gestione degli intent Weather. Funzionalità di base corrette.

Discreto - Buono: quanto previsto per la sufficienza e, in aggiunta, riconoscimento e gestione degli intent Places.

Ottimo- Eccellente: quanto previsto per il livello Discreto-Buono e, in aggiunta, riconoscimento e gestione degli intent Wiki. Gestione di casi particolari, etc.

Come punto di partenza per il progetto è possibile utilizzare il seguente esempio:

### 10.3.1. Punto di partenza per Alice Neural

//file di esempio del progetto

```

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>net5.0</TargetFramework>
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="Microsoft.CognitiveServices.Speech" Version="1.20.0" />
</ItemGroup>

<ItemGroup>
  <ProjectReference Include="..\WikitextExtensions\WikitextExtensions.csproj" />
</ItemGroup>

<ItemGroup>
  <Reference Include="HttpProxyControl">
    <HintPath>..\HttpProxyControl\bin\Debug\net5.0\HttpProxyControl.dll</HintPath>
  </Reference>
</ItemGroup>

</Project>

```

//File Program.cs

```

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Serialization;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Intent;
using HttpProxyControl;
using WikitextExtensions;

namespace OkStopEnabledRecognizingLoop
{
  public class AzureSpeechServiceStore
  {
    [JsonPropertyName("api_key")]
    public string APIKeyValue { get; set; }

    [JsonPropertyName("region_location")]
    public string RegionLocation { get; set; }

    [JsonPropertyName("endpoint")]
    public string EndPoint { get; set; }
  }

  public class AzureIntentRecognitionStore
  {
    [JsonPropertyName("luis_app_name")]
    public string LuisAppName { get; set; }
  }
}

```

```

[JsonPropertyName("luis_app_id")]
public string LuisAppId { get; set; }

[JsonPropertyName("luis_culture")]
public string LuisCulture { get; set; }

[JsonPropertyName("azure_resource")]
public AzureResource AzureResource { get; set; }
}

public class AzureResource
{
    [JsonPropertyName("resource_name")]
    public string ResourceName { get; set; }

    [JsonPropertyName("api_key")]
    public string ApiKey { get; set; }

    [JsonPropertyName("region_location")]
    public string RegionLocation { get; set; }

    [JsonPropertyName("endpoint")]
    public string Endpoint { get; set; }
}
class Program
{
    //parametri di configurazione per il servizio Speech di Azure
    static AzureSpeechServiceStore azureSpeechServiceStore = GetSpeechDataFromStore();
    static readonly string azureSpeechServiceKey = azureSpeechServiceStore.APIKeyValue;
    static readonly string azureSpeechServiceRegion = azureSpeechServiceStore.RegionLocation;

    //parametri di configurazione del servizio Intent Recognition di Azure
    static AzureIntentRecognitionStore azureIntentRecognitionStore = GetIntentDataFromStore();
    static readonly string azureIntentServiceKey =
        azureIntentRecognitionStore.AzureResource.ApiKey;
    static readonly string azureIntentServiceRegion =
        azureIntentRecognitionStore.AzureResource.RegionLocation;
    static readonly string luisIntentServiceAppId = azureIntentRecognitionStore.LuisAppId;
    static readonly string luisIntentServiceCulture = azureIntentRecognitionStore.LuisCulture;
    static bool currentIntentStopped = false;
    static AzureSpeechServiceStore GetSpeechDataFromStore()
    {
        //il file è nella cartella che contiene la soluzione corrente
        //il file contiene un oggetto JSON del tipo:
        // {
        //     "api_key": "api_key_value",
        //     "region_location": "region_location_value",
        //     "endpoint": "endpoint_value"
        //}
        string keyStorePath = "../../../../../MyAzureRoboVoiceStore.json";
    }
}

```

```

        string store = File.ReadAllText(keyStorePath);
        AzureSpeechServiceStore azureSpeechServiceStore =
JsonSerializer.Deserialize<AzureSpeechServiceStore>(store);
        return azureSpeechServiceStore;
    }
    static AzureIntentRecognitionStore GetIntentDataFromStore()
    {
        //il file è nella cartella che contiene la soluzione corrente
        //il file contiene un oggetto JSON del tipo:
        //{
        //    "luis_app_name": "value",
        //    "luis_app_id": "value",
        //    "luis_culture": "it-IT",
        //    "azure_resource": {
        //        "resource_name": "value",
        //        "api_key": "value",
        //        "region_location": "westeurope",
        //        "endpoint": "https://westeurope.api.cognitive.microsoft.com/"
        //}
        //}
        string keyStorePath = "../../../../../MyAzureWeatherResponderStore.json";
        string store = File.ReadAllText(keyStorePath);
        AzureIntentRecognitionStore azureSpeechServiceStore =
JsonSerializer.Deserialize<AzureIntentRecognitionStore>(store);
        return azureSpeechServiceStore;
    }

    static void Main(string[] args)
    {
        //configurazione per il servizio Speech e il Pattern Matching
        var config = SpeechConfig.FromSubscription(azureSpeechServiceKey,
azureSpeechServiceRegion);
        //imposta automaticamente il proxy se presente
        ProxyParams? proxyParams = HttpProxyHelper.GetHttpClientProxyParams();
        if (proxyParams.HasValue)
        {
            config.SetProxy(proxyParams.Value.ProxyAddress, proxyParams.Value.ProxyPort);
        }
        config.SpeechRecognitionLanguage = "it-IT";
        config.SpeechSynthesisLanguage = "it-IT";
        config.SpeechSynthesisVoiceName = "it-IT-ElsaNeural";

        //avvia il pattern matching continuo
        IntentPatternMatchingWithMicrophoneAsync(config).Wait();
    }
    private static async Task IntentPatternMatchingWithMicrophoneAsync(SpeechConfig config)
    {
        using (var intentRecognizer = new IntentRecognizer(config))
        {

```

```

        const string fraseOK = "ok Alice";
        const string fraseStop = "stop";
        //configurazione degli intent per il Pattern Matching
        // "ok Alice" ascolta per far partire una ricerca; "stop" per fermare lo speech in
corso
        //lo speech in corso è fermato sia quando viene riconosciuto l'intent di stop,
        //sia quando viene riconosciuta la frase "stop" nel testo pronunciato
        //Infatti, quando c'è il text to speech attivo, potrebbe succedere che il pattern
matching non
        //riesca a riconoscere l'intent di stop. In questo caso interviene la funzione di
speech to text in loop
        //continuo che trascrive tutto quello che viene detto
        intentRecognizer.AddIntent(fraseOK, "Ok");
        intentRecognizer.AddIntent(fraseStop, "Stop");

        var stopRecognition = new TaskCompletionSource<int>();
        using var synthesizer = new SpeechSynthesizer(config);

        //configurazione per il servizio LUIS
        SpeechConfig speechConfigForLuis =
SpeechConfig.FromSubscription(azureIntentServiceKey, azureIntentServiceRegion);
        speechConfigForLuis.SpeechRecognitionLanguage = luisIntentServiceCulture;
        ProxyParams? proxyParams = HttpProxyHelper.GetHttpClientProxyParams();
        if (proxyParams.HasValue)
        {
            speechConfigForLuis.SetProxy(proxyParams.Value.ProxyAddress,
proxyParams.Value.ProxyPort);
        }
        LanguageUnderstandingModel model =
LanguageUnderstandingModel.FromAppId(luisIntentServiceAppId);
        //gestione eventi
        intentRecognizer.Recognized += async (s, e) =>
        {
            switch (e.Result.Reason)
            {
                case ResultReason.RecognizedSpeech:
                    Console.WriteLine($"PATTERN MATCHING - RECOGNIZED: Text=
{e.Result.Text}");
                    //Console.WriteLine($"      Intent not recognized.");
                    if (ContainsTokens(e.Result.Text, fraseOK))
                    {
                        await HandleOkCommand(synthesizer, speechConfigForLuis, model);
                    }
                    else if (ContainsTokens(e.Result.Text, fraseStop))
                    {
                        currentIntentStopped = true;//segnalo che c'è stato uno stop
dell'intent
                        Console.WriteLine("Stopping current speaking...");
                        await synthesizer.StopSpeakingAsync();
                    }
            }
        };
    }
}

```

```

        break;
    case ResultReason.RecognizedIntent:
    {
        Console.WriteLine($"PATTERN MATCHING - RECOGNIZED: Text={e.Result.Text}");
        Console.WriteLine($"      Intent Id= {e.Result.IntentId}");
        if (e.Result.IntentId == "Ok")
        {
            await HandleOkCommand(synthesizer, speechConfigForLuis, model);
        }
        else if (e.Result.IntentId == "Stop")
        {
            currentIntentStopped = true;//segnalo che c'è stato uno stop
dell'intent
            Console.WriteLine("Stopping current speaking...");
            await synthesizer.StopSpeakingAsync();
        }
    }

        break;
    case ResultReason.NoMatch:
        Console.WriteLine($"NOMATCH: Speech could not be recognized.");
        var noMatch = NoMatchDetails.FromResult(e.Result);
        switch (noMatch.Reason)
        {
            case NoMatchReason.NotRecognized:
                Console.WriteLine($"PATTERN MATCHING - NOMATCH: Speech was
detected, but not recognized.");
                break;
            case NoMatchReason.InitialSilenceTimeout:
                Console.WriteLine($"PATTERN MATCHING - NOMATCH: The start of the
audio stream contains only silence, and the service timed out waiting for speech.");
                break;
            case NoMatchReason.InitialBabbleTimeout:
                Console.WriteLine($"PATTERN MATCHING - NOMATCH: The start of the
audio stream contains only noise, and the service timed out waiting for speech.");
                break;
            case NoMatchReason.KeywordNotRecognized:
                Console.WriteLine($"PATTERN MATCHING - NOMATCH: Keyword not
recognized");
                break;
        }
        break;

    default:
        break;
    }
};

intentRecognizer.Canceled += (s, e) =>
{

```

```

        Console.WriteLine($"PATTERN MATCHING - CANCELED: Reason={e.Reason}");

        if (e.Reason == CancellationReason.Error)
        {
            Console.WriteLine($"PATTERN MATCHING - CANCELED: ErrorCode={e.ErrorCode}");
            Console.WriteLine($"PATTERN MATCHING - CANCELED:
ErrorDetails={e.ErrorDetails}");
            Console.WriteLine($"PATTERN MATCHING - CANCELED: Did you update the speech key
and location/region info?");
        }

        stopRecognition.TrySetResult(0);
    };
    intentRecognizer.SessionStopped += (s, e) =>
    {
        Console.WriteLine("\n      Session stopped event.");
        stopRecognition.TrySetResult(0);
    };
    Console.WriteLine($"In ascolto. Di \"\{fraseOK}\\" per impartire un ordine, oppure
\"{fraseStop}\\" per fermare l'azione in corso");
    await intentRecognizer.StartContinuousRecognitionAsync();
    // Waits for completion. Use Task.WaitAny to keep the task rooted.
    Task.WaitAny(new[] { stopRecognition.Task });
}

private static async Task HandleOkCommand(SpeechSynthesizer synthesizer, SpeechConfig
speechConfigForLuis, LanguageUnderstandingModel model)
{
    currentIntentStopped = false;//segnalo che un nuovo intent è partito
    await synthesizer.SpeakTextAsync("Sono in ascolto");
    //avvia l'intent recognition su Azure
    string jsonResult = await RecognizeIntentAsync(speechConfigForLuis, model);
    if (jsonResult != null)
    {
        //process jsonResult
        //deserializzo il json
        //determino l'action da fare, eventualmente effettuando una richiesta GET su un
        endpoint remoto scelto in base al topScoringIntent
        //ottengo il risultato dall'endpoit remoto
        //effettuo un text to speech per descrivere il risultato
    }
    else
    {
        //non è stato riconosciuto l'intent
    }
    //il testo che viene letto tramite le istruzioni seguenti serve solo a simulare la
    situazione in cui
}

```

```

    //l'assistente sta leggendo qualcosa e contemporaneamente l'utente vuole impartire un
    nuovo comando
        //in questo caso deve fermare lo speech (con "stop") e poi ripartire con "ok Alice"...
        string wikitext = @"Il Filocolo (secondo un'etimologia approssimativa «fatica d'amore») è
    un romanzo in prosa: rappresenta una svolta rispetto ai romanzi delle origini scritti in versi. La
    storia ha come protagonisti Florio, figlio di un re saraceno, e Biancifiore (o Biancofiore), una
    schiava cristiana abbandonata da bambina. I due fanciulli crescono assieme e da grandi, in seguito
    alla lettura del libro di Ovidio Ars Amandi s'innamorano, come era successo per Paolo e Francesca dopo
    avere letto Ginevra e Lancillotto. Tuttavia il padre di Florio decide di separarli vendendo
    Biancifiore a dei mercanti. Florio decide quindi di andarla a cercare e dopo mille peripezie (da qui
    il titolo Filocolo) la rincontra. Infine, il giovane si converte al cristianesimo e sposa la
    fanciulla.";
        string[] readableText = wikitext.WikiTextToReadableTextNoSpace().SplitOnPeriod();
//TextSplitOnPeriod(WikiTextToReadableTextNoSpace(wikitext));
        int i = 0;
        while (!currentIntentStopped && i < readableText.Length)
        {
            await synthesizer.SpeakTextAsync(readableText[i]);
            i++;
        }
    }

    /// <summary>
    /// Riconosce l'intent associato a una frase pronunciata dall'utente.
    /// </summary>
    /// <param name="config"></param>
    /// <param name="model"></param>
    /// <returns>Nel caso di intent riconosciuto restituisce il Language Understanding JSON.
    /// Nel caso di intent non riconosciuto restituisce null</returns>
    public static async Task<string> RecognizeIntentAsync(SpeechConfig config,
LanguageUnderstandingModel model)
{
    string jsonResult = null;
    using (var recognizer = new IntentRecognizer(config))
    {
        recognizer.AddAllIntents(model);
        // Starts recognizing.
        Console.WriteLine("Say something...");

        var result = await recognizer.RecognizeOnceAsync();

        // Checks result.
        switch (result.Reason)
        {
            case ResultReason.RecognizedIntent:
                Console.WriteLine($"RECOGNIZED: Text={result.Text}");
                Console.WriteLine($"      Intent Id: {result.IntentId}.");

```

```

        jsonResult =
result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult);
    Console.WriteLine($"      Language Understanding JSON: {jsonResult}.");

    break;
case ResultReason.RecognizedSpeech:
    Console.WriteLine($"RECOGNIZED: Text={result.Text}");
    Console.WriteLine($"      Intent not recognized.");
    break;
case ResultReason.NoMatch:
    Console.WriteLine($"NOMATCH: Speech could not be recognized.");
    break;
case ResultReason.Canceled:
    var cancellation = CancellationDetails.FromResult(result);
    Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

    if (cancellation.Reason == CancellationReason.Error)
    {
        Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
        Console.WriteLine($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
        Console.WriteLine($"CANCELED: Did you update the subscription info?");
    }
    break;
}
return jsonResult;
}

/// <summary>
/// Restituisce true se phrase contiene tutti i token presenti in stringWithTokens
/// I token sono le parole prive dei segni di punteggiatura ',', ' ', ';', '.', '?', '!'
/// </summary>
/// <param name="phrase">stringa di testo in cui ricercare i token</param>
/// <param name="stringWithTokens">stringa contenente i token</param>
/// <returns>true se phrase contiene tutti i token, false altrimenti</returns>
public static bool ContainsTokens(string phrase, string stringWithTokens)
{
    char[] separators = { ',', ' ', ';' , '.', '?', '!' };
    string[] tokens = stringWithTokens.Split(separators,
StringSplitOptions.RemoveEmptyEntries);
    Console.WriteLine($"ContainsTokens-> phrase: {phrase}");
    Console.WriteLine("ContainsTokens -> tokens: ");
    foreach (var item in tokens)
    {
        Console.Write(item + " ");
    }
    Console.WriteLine();
    foreach (var token in tokens)
    {
        if (!phrase.Contains(token, StringComparison.CurrentCultureIgnoreCase))
        {

```

```
        return false;
    }
}
return true;
}
}
```

## 11. HttpListener

[#remarks](https://docs.microsoft.com/en-us/dotnet/api/system.net.http.listener?view=netcore-3.1)

Fornisce un listener semplice del protocollo HTTP controllato a livello di codice. La classe non può essere ereditata.

La classe `HttpListener` permette di creare applicazioni server che rispondono a richieste HTTP.

Utilizzando la classe [HttpListener](#), è possibile creare un semplice listener del protocollo HTTP che risponde alle richieste HTTP. Il listener è attivo per la durata dell'oggetto [HttpListener](#) e viene eseguito all'interno dell'applicazione con le relative autorizzazioni.

Per utilizzare [HttpListener](#), creare una nuova istanza della classe utilizzando il costruttore di [HttpListener](#) e utilizzare la proprietà [Prefixes](#) per accedere alla raccolta che include le stringhe che specificano i prefissi di Uniform Resource Identifier (URI) che devono essere elaborati dall'[HttpListener](#).

Una stringa di prefisso URI è costituita da uno schema (http o HTTPS), da un host, da una porta facoltativa e da un percorso facoltativo. Un esempio di stringa di prefisso completa è <http://www.contoso.com:8080/customerData/>. I prefissi devono terminare con una barra ("/"). L'oggetto [HttpListener](#) con il prefisso che corrisponde maggiormente a un URI richiesto risponde alla richiesta. Più oggetti [HttpListener](#) non possono aggiungere lo stesso prefisso. viene generata un'eccezione [Win32Exception](#) se una [HttpListener](#) aggiunge un prefisso già in uso.

Quando si specifica una porta, l'elemento host può essere sostituito con "\*" per indicare che l'[HttpListener](#) accetta le richieste inviate alla porta se l'URI richiesto non corrisponde ad alcun altro prefisso. Per ricevere, ad esempio, tutte le richieste inviate alla porta 8080 quando l'URI richiesto non è gestito da alcun [HttpListener](#), il prefisso è `http://*: 8080/`. Analogamente, per specificare che il [HttpListener](#) accetta tutte le richieste inviate a una porta, sostituire l'elemento host con il carattere "+". Ad esempio: `https://+:8080`. I caratteri "\*" e "+" possono essere presenti nei prefissi che includono i percorsi.

A partire da .NET Core 2,0 o .NET Framework 4,6 in Windows 10, i sottodomini con caratteri jolly sono supportati nei prefissi URI gestiti da un oggetto [HttpListener](#). Per specificare un sottodominio con caratteri jolly, usare il carattere "\*" come parte del nome host in un prefisso URI. Ad esempio, `http://*.foo.com/`. Passare come argomento al metodo [Add](#). Funziona a partire da .NET Core 2,0 o .NET Framework 4,6 in Windows 10; nelle versioni precedenti, viene generato un [HttpListenerException](#).

Per iniziare l'ascolto delle richieste provenienti dai client, aggiungere i prefissi URI alla raccolta e chiamare il metodo [Start](#). [HttpListener](#) offre modelli sincroni e asincroni per l'elaborazione delle richieste client.

Se si crea un [HttpListener](#) usando HTTPS, è necessario selezionare un certificato del server per il

listener. In caso contrario, una query [HttpWebRequest](#) di questo [HttpListener](#) avrà esito negativo con una chiusura imprevista della connessione.

## Warning

Top-level wildcard bindings (*http://\*:8080/* and *http://+:8080*) should **not** be used. Top-level wildcard bindings can open up your app to security vulnerabilities. This applies to both strong and weak wildcards. Use explicit host names rather than wildcards. Subdomain wildcard binding (for example, \*.mysub.com) doesn't have this security risk if you control the entire parent domain (as opposed to \*.com, which is vulnerable). See [rfc7230 section-5.4](#) for more information.

## Nota

È possibile configurare i certificati del server e altre opzioni del listener usando la shell di rete (netsh. exe). Per ulteriori informazioni, vedere la pagina relativa alla [Shell di rete \(netsh\)](#). Il file eseguibile ha iniziato a distribuire con Windows Server 2008 e Windows Vista.

## Nota

Se si specificano più schemi di autenticazione per la [HttpListener](#), il listener rileverà i client nell'ordine seguente: Negotiate, NTLM, Digest e quindi Basic.

## HTTP.sys

La classe [HttpListener](#) si basa su HTTP.sys, ovvero sul listener in modalità kernel che gestisce tutto il traffico HTTP per Windows. HTTP.sys fornisce la gestione della connessione, la limitazione della larghezza di banda e la registrazione del server Web. Utilizzare lo strumento [HttpCfg. exe](#) per aggiungere certificati SSL.

### 11.1.1. Links utili per HttpListener

<https://inneka.com/programming/c/multi-threaded-httplistener-with-await-async-and-tasks/>

<https://stackoverflow.com/questions/28273345/how-to-process-multiple-connections-simultaneously-with-httplistener>

<https://stackoverflow.com/questions/10017564/url-mapping-with-c-sharp-httplistener>

<https://docs.microsoft.com/it-it/dotnet/csharp/tutorials/console-webapiclient>

### 11.2. Esempi di un Server che accetta una sola richiesta e stampa una pagina HTML

#### //Esempio N1

```
using System;
using System.IO;
using System.Net;
using System.Text;

namespace HTTPListenerEsempioBase
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] prefixes = { "http://localhost:8080/primo/" };
            if (!HttpListener.IsSupported)
```

```

{
    Console.WriteLine("HttpListener is not supported.");
    return;
}
// URI prefixes are required,
// for example "http://contoso.com:8080/index/".
if (prefixes == null || prefixes.Length == 0)
    throw new ArgumentException("prefixes");

// Create a listener.
HttpListener listener = new HttpListener();
// Add the prefixes.
foreach (string s in prefixes)
{
    listener.Prefixes.Add(s);
}
listener.Start();
Console.WriteLine("Listening...");
// Note: The GetContext method blocks while waiting for a request.
HttpListenerContext context = listener.GetContext();
HttpListenerRequest request = context.Request;
// Obtain a response object.
HttpListenerResponse response = context.Response;
// Construct a response.
string responseString = "<HTML><BODY> Hello world!</BODY></HTML>";
byte[] buffer = System.Text.Encoding.UTF8.GetBytes(responseString);
// Get a response stream and write the response to it.
response.ContentEncoding = Encoding.UTF8;
response.ContentLength64 = buffer.Length;
response.ContentType = "text/html; charset=utf-8";

System.IO.Stream output = response.OutputStream;
output.Write(buffer, 0, buffer.Length);
// You must close the output stream.
output.Close();
// l'istruzione Console.Read(); serve a non chiudere subito il listener prima
ancora che il client abbia ricevuto la risposta
Console.WriteLine("Il listener ha terminato la sua attività. Premi invio nella
console per chiudere il programma");
Console.ReadLine();
listener.Stop();
}
}
}

//Esempio N2
using System;
using System.IO;
using System.Net;
using System.Text;

namespace HttpListenerDemo
{
    class Program
    {
        /*
         * Semplice utilizzo della classe HttpListener per inviare una stringa di dati al
client.
         * See https://docs.microsoft.com/it-
it/dotnet/api/system.net.httplistener?view=netcore-3.1
        */
    }
}

```

```

static void Main(string[] args)
{
    // Create a listener.
    HttpListener listener = new HttpListener();

    // Add the prefixes. Indicano gli indirizzi su cui rimanere in ascolto
    listener.Prefixes.Add("http://localhost:8080/");
    // .Start() fa partire il server, ma non rimane in attesa che un client si
    // connetta. Non è bloccante
    listener.Start();
    Console.WriteLine("Listening at http://localhost:8080 ...");

    //questa versione dell'HTTPListener è single thread, quindi può gestire una
    //richiesta alla volta
    while (true)
    {
        // GetContext() mette in attesa il thread finché una client non si connette.
        // appena un client si connette GetContext() restituisce un oggetto che
        // rappresenta la connessione con UN SINGOLO client
        // ctx è l'oggetto che effettivamente verrà utilizzato per gestire il client
        var ctx = listener.GetContext();

        /** Gestione del client**/

        // l'oggetto Request contiene i dati inviati dal client, mentre l'oggetto
        Response contiene i dati inviati dal server
        // Request rappresenta i dati che vengono inviati dal client (Header http,
        body http, url richiesto ...)
        var req = ctx.Request;
        //Response è l'oggetto che rappresenta la risposta da inviare al client
        var res = ctx.Response;
        Console.WriteLine($"{req.HttpMethod} - {req.Url}");
        //dati della query string
        //https://docs.microsoft.com/en-
        us/dotnet/api/system.net.httplistenerrequest.querystring?view=net-6.0
        //https://codersblock.com/blog/getting-query-string-parameters-safely-and-
        concisely-in-c/
        //come accedere alle variabili della query string
        var keys = req.QueryString.AllKeys;
        foreach (var key in keys)
        {
            Console.WriteLine($"key = {key} - value {req.QueryString[key]}");
        }

        /* Per inviare dei dati al client c'è bisogno di usare l'oggetto Response
         * In particolare bisogna scrivere nel body http della risposta, settare gli
         header della risposta e poi inviare.
         */
        // OutputStream rappresenta lo Stream per scrivere nel body della risposta.
        Analogamente req.InputStream è lo Stream per leggere il body della richiesta
        Stream output = res.OutputStream;

        // Procedimento analogo alla scrittura su un FileStream
        //IMPORTANTE l'utilizzo dell'encoding
        byte[] buffer = Encoding.UTF8.GetBytes("Ciao client");

        // Importante settare gli headers della risposta, perché alcuni browser non
        // accettano la richiesta se non sono presenti
        res.ContentLength64 = buffer.Length;
        res.ContentEncoding = Encoding.UTF8;
        res.ContentType = "text/html; charset=utf-8";
        // Scrivo nel body http
        output.Write(buffer, 0, buffer.Length);

```

```
// invio effettivo dei dati
res.Close();
/** Fine gestione client*/

}
//questo codice seguente non verrà mai eseguito per la presenza del
while(true){}
precedente
//Console.Read();
//listener.Stop();

}
}
```

### **11.3. Classe `HttpListenerContext`**

<https://docs.microsoft.com/en-us/dotnet/api/system.net.http.listenercontext?view=netcore-3.1>

## 11.4. Classe `HttpListenerRequest`

<https://docs.microsoft.com/en-us/dotnet/api/system.net.httplistenerrequest?view=netcore-3.1>

## 11.5. Classe `HttpListenerResponse`

## 11.6. Certificati SSL per HttpListener

<https://stackoverflow.com/questions/11403333/httplistener-with-https-support>

<https://stackoverflow.com/questions/4004/how-do-i-add-ssl-to-a-net-application-that-uses-httplistener-it-will-not-be>

## 12. Evoluzione del linguaggio C#

C# 8: <https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-8> (a partire da .NET Core 3.1)

C# 9: <https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-9> (apartire da .NET 5)

C# 10: <https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-10> (a partire da .NET 6)

In questa sezione verranno mostrati solo alcuni aspetti tra tutti quelli introdotti con le diverse versioni del linguaggio e di .NET. Per i riferimenti completi, si rimanda alla documentazione ufficiale.

### 12.1.C# 8

In C#8 sono state introdotte (tra le tante altre) le seguenti caratteristiche:

### 12.1.1. Using declarations

<https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-8#using-declarations>

A **using declaration** is a variable declaration preceded by the using keyword. It tells the compiler that the variable being declared should be disposed at the end of the enclosing scope.

## 12.1.2. Nullable reference types

<https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-8#nullable-reference-types>

<https://docs.microsoft.com/en-us/dotnet/csharp/nullable-references>

Prior to C# 8.0, all reference types were nullable. *Nullable reference types* refers to a group of features introduced in C# 8.0 that you can use to minimize the likelihood that your code causes the runtime to throw [System.NullReferenceException](#). *Nullable reference types* includes three features that help you avoid these exceptions, including the ability to explicitly mark a reference type as *nullable*:

- Improved static flow analysis that determines if a variable may be null before dereferencing it.
- Attributes that annotate APIs so that the flow analysis determines *null-state*.
- Variable annotations that developers use to explicitly declare the intended *null-state* for a variable.

Null-state analysis and variable annotations are disabled by default for existing projects—meaning that all reference types continue to be nullable. Starting in .NET 6, they're enabled by default for new projects. For information about enabling these features by declaring a *nullable annotation context*, see [Nullable contexts](#).

### 12.1.3. Null-coalescing assignment

<https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-8#null-coalescing-assignment>

## 12.2.C# 9

In C# 9 sono state introdotte (tra le tante altre) le seguenti caratteristiche:

### 12.2.1. Record types

<https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-9#record-types>

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/record>

You use the record keyword to define a reference type that provides built-in functionality for encapsulating data. You can create record types with immutable properties by using positional parameters or standard property syntax.

### 12.2.2. Top level statements

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/top-level-statements>

Starting in C# 9, you don't have to explicitly include a Main method in a console application project. Instead, you can use the *top-level statements* feature to minimize the code you have to write. In this case, the compiler generates a class and Main method entry point for the application.

#### 12.2.2.1. Only one top-level file

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/top-level-statements#only-one-top-level-file>

An application must have only one entry point. A project can have only one file with top-level statements. Putting top-level statements in more than one file in a project results in the following compiler error:

CS8802 Only one compilation unit can have top-level statements.

#### 12.2.2.2. No other entry points

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/top-level-statements#no-other-entry-points>

You can write a Main method explicitly, but it can't function as an entry point. The compiler issues the following warning:

CS7022 The entry point of the program is global code; ignoring 'Main()' entry point.

In a project with top-level statements, you can't use the [-main](#) compiler option to select the entry point, even if the project has one or more Main methods.

### 12.2.2.3. *using directives*

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/top-level-statements#using-directives>

If you include using directives, they must come first in the file, as in this example:

```
using System.Text;
```

```
StringBuilder builder = new();
builder.AppendLine("Hello");
builder.AppendLine("World!");

Console.WriteLine(builder.ToString());
```

### 12.2.2.4. *Global namespace*

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/top-level-statements#global-namespace>

Top-level statements are implicitly in the global namespace.

### 12.2.2.5. *Namespaces and type definitions*

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/top-level-statements#namespaces-and-type-definitions>

A file with top-level statements can also contain namespaces and type definitions, but they must come after the top-level statements. For example:

```
MyClass.TestMethod();
MyNamespace.MyClass.MyMethod();

public class MyClass
{
    public static void TestMethod()
    {
        Console.WriteLine("Hello World!");
    }
}

namespace MyNamespace
{
    class MyClass
    {
        public static void MyMethod()
        {
            Console.WriteLine("Hello World from MyNamespace.MyClass.MyMethod!");
        }
    }
}
```

```
}
```

### 12.2.2.6. args

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/top-level-statements#args>

Top-level statements can reference the args variable to access any command-line arguments that were entered. The args variable is never null but its Length is zero if no command-line arguments were provided. For example:

```
if (args.Length > 0)
{
    foreach (var arg in args)
    {
        Console.WriteLine($"Argument={arg}");
    }
}
else
{
    Console.WriteLine("No arguments");
}
```

### 12.2.2.7. await

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/top-level-statements#await>

You can call an async method by using await. For example:

```
Console.WriteLine("Hello ");
await Task.Delay(5000);
Console.WriteLine("World!");
```

### 12.2.2.8. Exit code for the process

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/top-level-statements#exit-code-for-the-process>

To return an int value when the application ends, use the return statement as you would in a Main method that returns an int. For example:

```
string? s = Console.ReadLine();

int returnValue = int.Parse(s ?? "-1");
return returnValue;
```

### 12.2.2.9. Implicit entry point method

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/top-level-statements#implicit-entry-point-method>

The compiler generates a method to serve as the program entry point for a project with top-level statements. The name of this method isn't actually Main, it's an implementation detail that your code can't reference directly. The signature of the method depends on whether the top-level statements contain the await keyword or the return statement. The following table shows what the method signature would look like, using the method name Main in the table for convenience.

Top-level code contains	Implicit Main signature
await and return	static async Task<int> Main(string[] args)
await	static async Task Main(string[] args)
return	static int Main(string[] args)
No await or return	static void Main(string[] args)

### 12.2.2.10. New General Structure of a C# Program

```
// A skeleton of a C# program
using System;

// Your program starts here:
Console.WriteLine("Hello world!");

namespace YourNamespace
{
    class YourClass
    {

    }

    struct YourStruct
    {

    }

    interface IYourInterface
    {

    }

    delegate int YourDelegate();

    enum YourEnum
    {

    }

    namespace YourNestedNamespace
    {
        struct YourStruct
        {

        }
    }
}
```

### 12.2.3. Struttura del Main (solo per approfondimento)

#### 12.2.3.1. *Main() and command-line arguments*

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/main-command-line>

Starting in C# 9, you can omit the Main method, and write C# statements as if they were in the Main method, as in the following example:

```

using System.Text;

StringBuilder builder = new();
builder.AppendLine("Hello");
builder.AppendLine("World!");

Console.WriteLine(builder.ToString());

```

### 12.2.3.1.1. Caratteristiche del Main

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/main-command-line#overview>

### 12.2.3.1.2. Valori di ritorno del Main

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/main-command-line#main-return-values>

### 12.2.3.1.3. Command-Line Arguments

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/main-command-line#command-line-arguments>

## 12.3. C# 10

### 12.3.1. Record Structs

<https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-10#record-structs>

You can declare value type records using the [record struct or readonly record struct declarations](#). You can now clarify that a record is a reference type with the record class declaration.

### 12.3.2. Global and implicit using

<https://devblogs.microsoft.com/dotnet/welcome-to-csharp-10/>

**using** directives simplify how you work with namespaces. C# 10 includes a new **global using** directive and *implicit usings* to reduce the number of usings you need to specify at the top of each file.

#### Global using directives

<https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-10#global-using-directives>

If the keyword **global** appears prior to a **using** directive, that using applies to the entire project:

```
global using System;
```

You can use any feature of **using** within a **global using** directive. For example, adding **static** imports a type and makes the type's members and nested types available throughout your project. If you use an alias in your using directive, that alias will also affect your entire project:

```
global using static System.Console;
global using Env = System.Environment;
```

You can put global usings in any **.cs** file, including **Program.cs** or a specifically named file like **globalusings.cs**. The scope of global usings is the current compilation, which generally corresponds to the current project.

For more information, see [global using directives](#).

### Implicit usings

The Implicit usings feature automatically adds common `global using` directives for the type of project you are building. To enable implicit usings set the `ImplicitUsings` property in your `.csproj` file:

```
<PropertyGroup>
    <!-- Other properties like OutputType and TargetFramework -->
    <ImplicitUsings>enable</ImplicitUsings>
</PropertyGroup>
```

Implicit usings are enabled in the new .NET 6 templates. Read more about the changes to the .NET 6 templates at this [blog post](#).

The specific set of `global using` directives included depend on the type of application you are building. For example, implicit usings for a console application or a class library are different than those for an ASP.NET application.

For more information, see this [implicit usings](#) article.

<https://docs.microsoft.com/en-us/dotnet/core/project-sdk/overview#implicit-using-directives>

### Combining using features

Traditional `using` directives at the top of your files, global `using` directives, and implicit usings work well together. Implicit usings let you include the .NET namespaces appropriate to the kind of project you're building with a single line in your project file. `global using` directives let you include additional namespaces to make them available throughout your project. The `using` directives at the top of your code files let you include namespaces used by just a few files in your project.

Regardless of how they are defined, extra `using` directives increase the possibility of ambiguity in name resolution. If you encounter this, consider adding an alias or reducing the number of namespaces you are importing. For example, you can replace `global using` directives with explicit `using` directives at the top of a subset of files.

If you need to remove namespaces that have been included via implicit usings, you can specify them in your project file:

```
<ItemGroup>
    <Using Remove="System.Threading.Tasks" />
</ItemGroup>
```

You can also add namespaces that behave as though they were `global using` directives, you can add `Using` items to your project file, for example:

```
<ItemGroup>
    <Using Include="System.IO.Pipes" />
</ItemGroup>
```

### 12.3.3. File-scoped namespaces

<https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-10#file-scoped-namespace-declaration>

Many files contain code for a single namespace. Starting in C# 10, you can include a namespace as a statement, followed by a semi-colon and without the curly brackets:

```
namespace MyCompany.MyNamespace;  
  
class MyClass // Note: no indentation  
{ ... }
```

This simplifies the code and removes a level of nesting. Only one file-scoped namespace declaration is allowed, and it must come before any types are declared.

For more information about file-scoped namespaces, see the [namespace keyword](#) article.

## 13. Sviluppo di RESTful API con Minimal API .NET

### 13.1. Getting started with ASP.NET Core Minimal API

<https://docs.microsoft.com/en-us/learn/patterns/aspnet-core-minimal-api/>

Building a web API is a common task. You want to be able to serve some data and know how an application, or service, consumes it. How you build the API might differ vastly between tech stacks. As part of building an API, you know there are parts like data storage, security, versioning, and documentation. Getting all these parts to work can be a complex undertaking.

#### 13.1.1. What is minimal API?

<https://docs.microsoft.com/en-us/learn/modules/build-web-api-minimal-api/2-what-is-minimal-api/>

Building an API can be complex because it needs to support many features like routing, reading and writing to data storage, and authentication. To save time, you begin with the .NET frameworks, which deliver many of the features you need. But those frameworks might require considerable setup before you have a basic API up and running. With minimal API for .NET 6, that's not the case. You can get started with just a few lines.

To get started with minimal API, the main requirement is to use at least .NET 6. Then, you need a text editor, such as Visual Studio or Visual Studio Code, or any other text editor of your choice. Finally, you can use either a Windows, macOS, or Linux operating system.

With Minimal API

- **No *Startup.cs*:** all the tasks happen in *Program.cs*, including setting up routes and configuring dependency injections, security, etc.
- **Top-level statements:** Because minimal API uses .NET 6, you can use top-level statements. The minimal API uses this technique to bring down the number of lines you need to type. To create an API, you use only these lines:

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();
app.MapGet("/", () => "Hello World!");
app.Run();
```

There's no using statement, Main() method, or class. There are only four lines of code.

With minimal API, you add the route right away on the app instance:

```
app.MapGet("/todos", await (TodoDb db) => db.Todos.ToListAsync());
app.MapPost("/todos", await (Todo todo) => {});
app.MapPut("/todos", (Todo todo) => {});
app.MapDelete("/todos/{id}", (int id) => {});
```

The same functionality is still there. You still can configure a database, add authentication and much more.

#### 13.1.2. Minimal API .NET a partire da un'App console

E' possibile creare una minimal API partendo direttamente da un'applicazione console.

Da Visual Studio 2022, scegliamo di creare un nuovo progetto console, utilizzando come target framework .NET6.

```
//File Program.cs
// See https://aka.ms/new-console-template for more information
```

```
Console.WriteLine("Hello, World!");
```

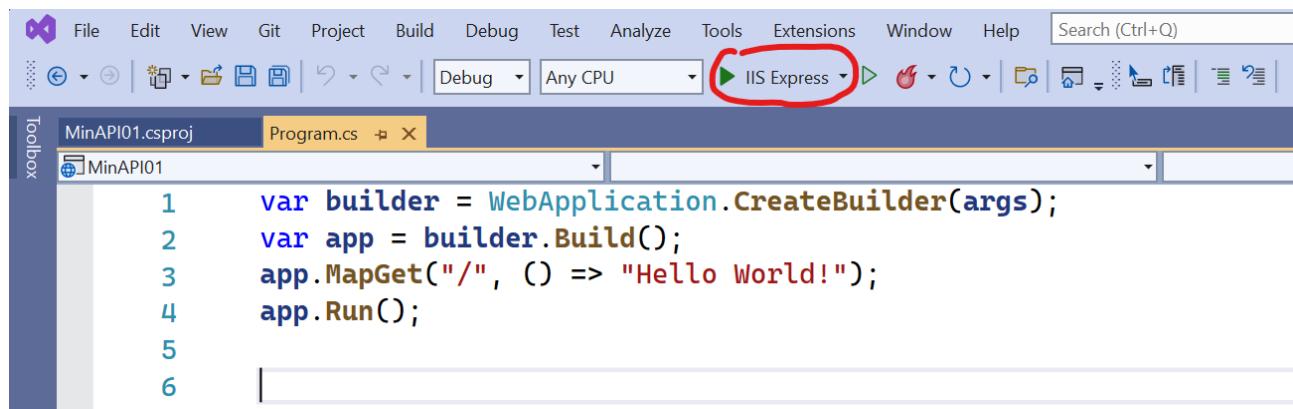
Modifichiamo le impostazioni del progetto console come segue:

```
<Project Sdk="Microsoft.NET.Sdk.Web">  
  
  <PropertyGroup>  
    <OutputType>Exe</OutputType>  
    <TargetFramework>net6.0</TargetFramework>  
    <ImplicitUsings>enable</ImplicitUsings>  
    <Nullable>enable</Nullable>  
  </PropertyGroup>  
  
</Project>
```

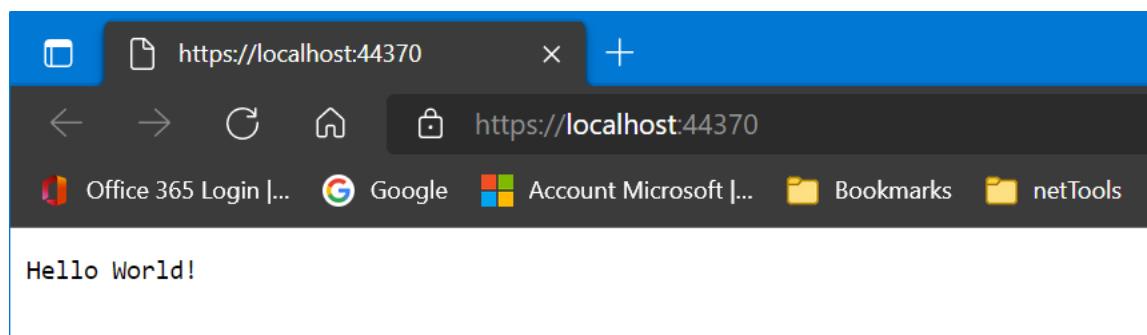
Salviamo il File .csproj e ricarichiamo il progetto. Scriviamo il seguente codice nel file Program.cs

```
var builder = WebApplication.CreateBuilder(args);  
var app = builder.Build();  
app.MapGet("/", () => "Hello World!");  
app.Run();
```

Lanciamo l'applicazione.



Otteniamo:



Abbiamo scritto la nostra prima applicazione server in localhost! Tutte le volte che viene eseguita una GET all'URL <https://localhost:44370/> viene restituita la risposta: Hello World!

Se provassimo a modificare il codice della nostra applicazione come segue:

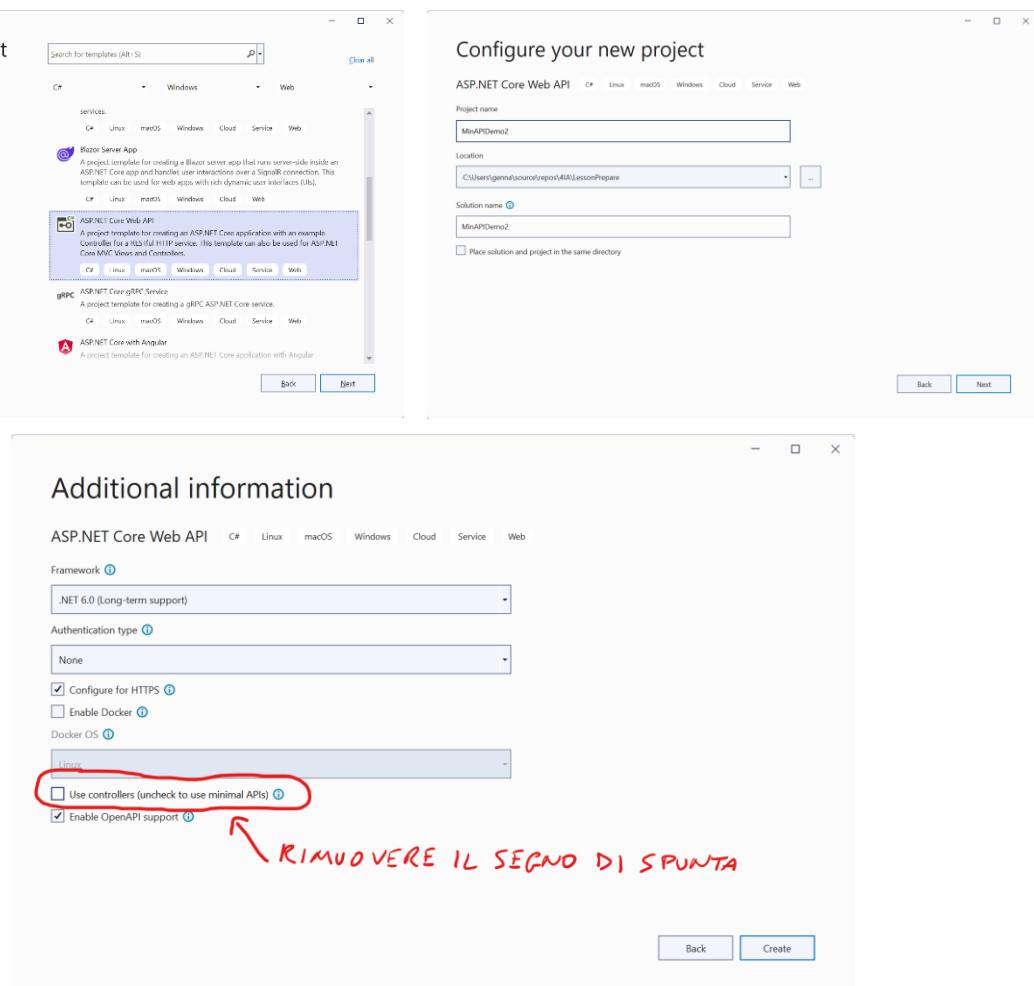
```
var builder = WebApplication.CreateBuilder(args);  
var app = builder.Build();  
app.MapGet("/", () => new {Message = "Hello World!", Version=1}); //un oggetto di  
anonymous type https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/types/anonymous-types  
app.Run();
```

Oterremo la risposta serializzata automaticamente come JSON:

```
{"message": "Hello World!", "version": 1}
```

### 13.1.3. Minimal API .NET a partire da un template ASP.NET Core Web API

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/min-web-api>



Otteniamo un template di progetto come il seguente:

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at
// https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
```

```

var summaries = new[]
{
    "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot",
    "Sweltering", "Scorching"
};

app.MapGet("/weatherforecast", () =>
{
    var forecast = Enumerable.Range(1, 5).Select(index =>
        new WeatherForecast
        (
            DateTime.Now.AddDays(index),
            Random.Shared.Next(-20, 55),
            summaries[Random.Shared.Next(summaries.Length)]
        )
        .ToArray();
    return forecast;
})
.WithName("GetWeatherForecast");

app.Run();
}

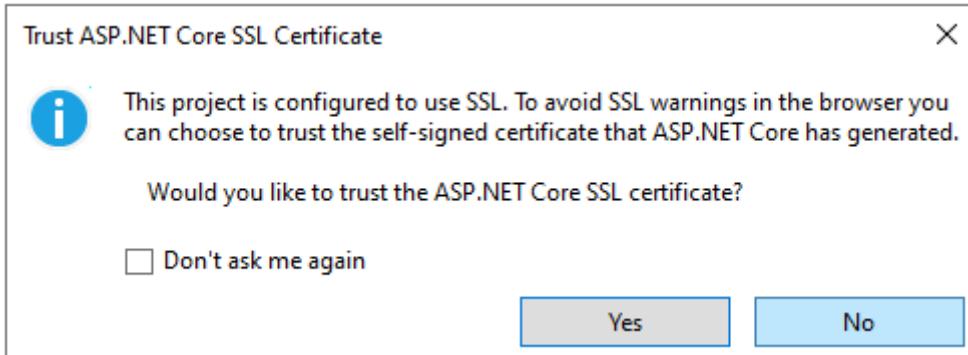
internal record WeatherForecast(DateTime Date, int TemperatureC, string? Summary)
{
    public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
}

```

### 13.1.3.1. Esecuzione dell'App

<https://localhost:7091/swagger/index.html>

Potrebbe succedere che Visual Studio presenti i seguenti warnings:



Select Yes if you trust the IIS Express SSL certificate.

The following dialog is displayed:

### Security Warning



You are about to install a certificate from a certification authority (CA) claiming to represent:

localhost

Windows cannot validate that the certificate is actually from "localhost". You should confirm its origin by contacting "localhost". The following number will assist you in this process:

Thumbprint (sha1): 35AAFB1F 70EC5F89 C7180FBD 61AFE491  
94580874

Warning:

If you install this root certificate, Windows will automatically trust any certificate issued by this CA. Installing a certificate with an unconfirmed thumbprint is a security risk. If you click "Yes" you acknowledge this risk.

Do you want to install this certificate?

Select **Yes** if you agree to trust the development certificate.

The screenshot shows the Swagger UI interface for a .NET API named "MinAPIDemo".

**Endpoint:** /weatherforecast (GET)

**Parameters:** No parameters

**Responses:**

- Code:** 200 **Description:** Success **Links:** No links

**Media type:** application/json (selected)

**Example Value:** [ { "date": "2022-03-06T15:37:19.451Z", "temperatureC": 0, "summary": "string", "temperatureF": 0 } ]

**Schemas:**

```
WeatherForecast:
  date: string($date-time)
  temperatureC: integer($int32)
  summary: string
  nullable: true
  temperatureF: integer($int32)
  readonly: true
```

Si può effettuare una GET sull’endpoint della nostra Web API, tramite l’interfaccia web di Swagger, oppure direttamente tramite la barra degli indirizzi del browser, oppure, in alternativa, utilizzando un software come Postman, o curl.

Per effettuare una GET dall’interfaccia web di Swagger basta cliccare sul pulsante “Try it out” e poi su “Execute”. Si ottiene una risposta del tipo seguente:

The screenshot shows the Swagger UI interface for a service named "MinAPIDemo". The main title bar says "MinAPIDemo 1.0 OAS3". Below it, the URL is https://localhost:7091/swagger/v1/swagger.json. A dropdown menu "Select a definition" is set to "MinAPIDemo v1".

The main content area displays the "MinAPIDemo" section. Under "GET /weatherforecast", there are tabs for "Parameters" (which shows "No parameters") and "Responses".

- Responses:**
  - curl:** curl -X 'GET' \ 'https://localhost:7091/weatherforecast' \ -H 'accept: application/json'
  - Request URL:** https://localhost:7091/weatherforecast
  - Server response:**

Code	Details
200	<b>Response body</b> <pre>[   {     "date": "2022-03-07T16:41:35.1445909+01:00",     "temperatureC": -6,     "summary": "Chilly",     "temperatureF": 22   },   {     "date": "2022-03-08T16:41:35.1447348+01:00",     "temperatureC": 35,     "summary": "Mild",     "temperatureF": 94   },   {     "date": "2022-03-09T16:41:35.1447375+01:00",     "temperatureC": -9,     "summary": "Bracing",     "temperatureF": 16   },   {     "date": "2022-03-10T16:41:35.1447378+01:00",     "temperatureC": 26,     "summary": "Mild",     "temperatureF": 78   },   {     "date": "2022-03-11T16:41:35.1447380+01:00",     "temperatureC": 42,     "summary": null   } ]</pre> <div style="text-align: right;"> <a href="#">Copy</a> <a href="#">Download</a> </div> <b>Response headers</b> <pre>content-type: application/json; charset=utf-8 date: Sun, 06 Mar 2022 15:41:34 GMT server: Kestrel</pre>

**Responses:**

Code	Description	Links
200	Success	No links

**Media type:** application/json

Controls Accept header.

[Example Value](#) | [Schema](#)

Effettuando la GET dalla barra di ricerca del browser si ha:

GET: <https://localhost:7091/weatherforecast>

Risposta:

```
[  
{
  "date": "2022-03-07T16:47:11.2226576+01:00",
  "temperatureC": -6,
  "summary": "Chilly",
  "temperatureF": 22
},  
{
  "date": "2022-03-08T16:47:11.2226757+01:00",
  "temperatureC": 42,
```

```

"summary": "Balmy",
"temperatureF": 107
},
{
"date": "2022-03-09T16:47:11.2226776+01:00",
"temperatureC": 9,
"summary": "Scorching",
"temperatureF": 48
},
{
"date": "2022-03-10T16:47:11.2226779+01:00",
"temperatureC": 8,
"summary": "Balmy",
"temperatureF": 46
},
{
"date": "2022-03-11T16:47:11.2226782+01:00",
"temperatureC": 47,
"summary": "Balmy",
"temperatureF": 116
}
]

```

Usando Postman:

The screenshot shows the Postman web interface with a collection named 'MinAPIDemoTest' containing a single item 'First GET'. The request URL is 'https://localhost:7091/weatherforecast'. In the 'Headers' tab, there is a highlighted 'Accept' header with the value 'application/json'. The 'Body' tab displays a JSON array with 32 elements, each representing a weather forecast entry. The JSON structure is as follows:

```

[{"date": "2022-03-07T16:55:55.4618125+01:00", "temperatureC": 5, "summary": "Hot", "temperatureF": 40}, {"date": "2022-03-08T16:55:55.4618353+01:00", "temperatureC": 40, "summary": "Warm", "temperatureF": 103}, {"date": "2022-03-09T16:55:55.4618357+01:00", "temperatureC": 21, "summary": "Balmy", "temperatureF": 69}, {"date": "2022-03-10T16:55:55.461836+01:00", "temperatureC": 15, "summary": "Chilly", "temperatureF": 58}, {"date": "2022-03-11T16:55:55.461837+01:00", "temperatureC": 19, "summary": "Mild", "temperatureF": 66}]

```

### 13.1.3.2. Concetto di Certificato Digitale e di HTTPS

Si noti che l'applicazione server utilizza un certificato autoprodotto di development per l'https, quindi

si ottiene un warning da Postman sul fatto che il certificato del server (la nostra applicazione) non è stato verificato:

The screenshot shows the Postman interface with the 'Network' tab selected. At the top, it displays a green '200 OK' status, 16 ms response time, and 653 B size. A red arrow points to the status bar. Below the status bar, there are two sections: 'Agent' and 'Desktop Agent'. The 'Agent' section contains 'Local Address' and 'Remote Address' both set to '::1'. The 'Desktop Agent' section contains 'TLS Protocol' set to 'TLSv1.3', 'Cipher Name' set to 'TLS\_AES\_256\_GCM\_SHA384', 'Certificate CN' set to 'localhost', 'Issuer CN' set to 'localhost', and 'Valid Until' set to 'Aug 24 14:34:30 2022 GMT'. At the bottom, a red warning box contains the text '⚠️ Unable to verify the first certificate'.

Un certificato digitale è un documento digitale che permette ad un client di verificare l'autenticità del server. Per essere valido un certificato digitale deve essere firmato digitalmente da una Trusted CA (Certification Authority) riconosciuta. A livello mondiale esistono diverse Trusted CA authorities che possono firmare certificati digitali:

[https://it.wikipedia.org/wiki/Certificate\\_authority](https://it.wikipedia.org/wiki/Certificate_authority)

<https://www.ssl.com/faqs/what-is-a-certificate-authority/>

Si noti anche che il server è in ascolto anche su un'altra porta con il protocollo http (non sicuro). Infatti, effettuando una GET all'indirizzo: <http://localhost:5091/weatherforecast> si ottiene ancora una risposta dal server mediante un redirect: il client viene rediretto all'URL <https://localhost:7091/weatherforecast>

Per fermare l'applicazione server ci sono diverse opzioni che dipendono dal modo in cui è stata avviata: Ctrl +c nella console dell'app, oppure chiusura della console dell'app, oppure stop debugging da Visual Studio.

### 13.1.4. Chi esegue l'applicazione server?

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers>

An ASP.NET Core app runs with an in-process HTTP server implementation. The server implementation listens for HTTP requests and surfaces them to the app as a set of [request features](#) composed into an [HttpContext](#).

In Windows:

ASP.NET Core ships with the following:

- [Kestrel server](#) is the default, cross-platform HTTP server implementation. Kestrel provides the best performance and memory utilization, but it doesn't have some of the advanced features in HTTP.sys. For more information, see [Kestrel vs. HTTP.sys](#) in the next section.
- IIS HTTP Server is an [in-process server](#) for IIS.

- [HTTP.sys server](#) is a Windows-only HTTP server based on the [HTTP.sys kernel driver and HTTP Server API](#).

When using [IIS](#) or [IIS Express](#), the app either runs:

- In the same process as the IIS worker process (the [in-process hosting model](#)) with the IIS HTTP Server. *In-process* is the recommended configuration.
- In a process separate from the IIS worker process (the [out-of-process hosting model](#)) with the [Kestrel server](#).

In Linux:

ASP.NET Core ships with [Kestrel server](#), which is the default, cross-platform HTTP server.

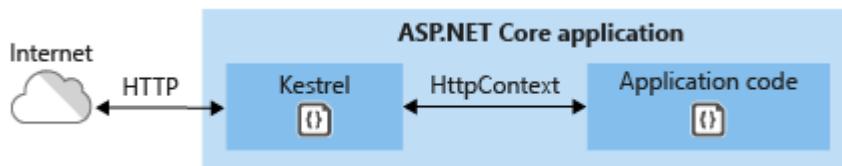
#### 13.1.4.1. Kestrel

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/?view=aspnetcore-6.0&tabs=linux#kestrel>

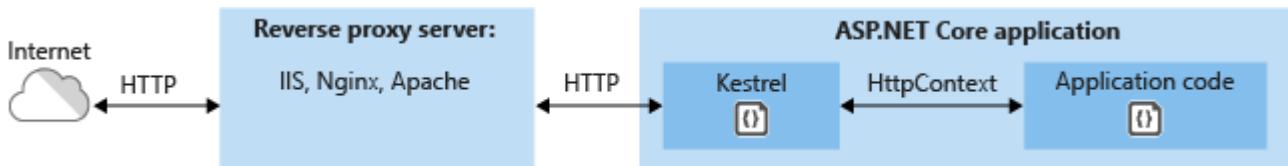
[Kestrel server](#) is the default, cross-platform HTTP server implementation. Kestrel provides the best performance and memory utilization, but it doesn't have some of the advanced features in HTTP.sys. For more information, see [Kestrel vs. HTTP.sys](#) in this document.

Use [Kestrel](#):

- By itself as an edge server processing requests directly from a network, including the Internet.



- With a *reverse proxy server*, such as [Internet Information Services \(IIS\)](#), [Nginx](#), or [Apache](#). A reverse proxy server receives HTTP requests from the Internet and forwards them to Kestrel.



Either hosting configuration—with or without a reverse proxy server—is supported.

For Kestrel configuration guidance and information on when to use Kestrel in a reverse proxy configuration, see [Kestrel web server implementation in ASP.NET Core](#).

- **Nginx with Kestrel**

For information on how to use Nginx on Linux as a reverse proxy server for Kestrel, see [Host ASP.NET Core on Linux with Nginx](#).

- **Apache with Kestrel**

For information on how to use Apache on Linux as a reverse proxy server for Kestrel, see [Host ASP.NET Core on Linux with Apache](#).

#### 13.1.4.2. IIS Express

<https://docs.microsoft.com/en-us/iis/extensions/introduction-to-iis-express/iis-express-overview>

### 13.1.5. La configurazione di lancio dell'App

A seconda delle opzioni disponibili sul proprio ambiente di sviluppo si possono avere diverse configurazioni di lancio possibili. Ad esempio, il file `launchSettings.json` del progetto MinAPIDemo è il seguente:

```
{  
    "$schema": "https://json.schemastore.org/launchsettings.json",  
    "iisSettings": {  
        "windowsAuthentication": false,  
        "anonymousAuthentication": true,  
        "iisExpress": {  
            "applicationUrl": "http://localhost:20739",  
            "sslPort": 44330  
        }  
    },  
    "profiles": {  
        "MinAPIDemo": {  
            "commandName": "Project",  
            "launchBrowser": true,  
            "launchUrl": "swagger",  
            "environmentVariables": {  
                "ASPNETCORE_ENVIRONMENT": "Development"  
            },  
            "applicationUrl": "https://localhost:7091;http://localhost:5091",  
            "dotnetRunMessages": true  
        },  
        "IIS Express": {  
            "commandName": "IISExpress",  
            "launchBrowser": true,  
            "launchUrl": "swagger",  
            "environmentVariables": {  
                "ASPNETCORE_ENVIRONMENT": "Development"  
            }  
        },  
        "WSL": {  
            "commandName": "WSL2",  
            "launchBrowser": true,  
            "launchUrl": "https://localhost:7091/swagger",  
            "environmentVariables": {  
                "ASPNETCORE_ENVIRONMENT": "Development",  
                "ASPNETCORE_URLS": "https://localhost:7091;http://localhost:5091"  
            },  
            "distributionName": ""  
        }  
    }  
}
```

Ad esempio, cambiando i numeri di porta del launchURL, oppure dell'applicationURL è possibile cambiare il numero di porta su cui l'applicazione server è in ascolto.

### 13.1.6. Come funziona una minimal API?

First you create a builder. From the builder, you construct an application instance app:

```
var builder = WebApplication.CreateBuilder(args);  
var app = builder.Build();
```

The builder has a `Services` property. By using the `Services` property, you can add features like CORS, Entity Framework, or Swagger. Here's an example:

```
builder.Services.AddCors(options => {});
```

In the `Services` property, you tell the API that there's a capability to use. Conversely, the app instance is used to actually use it. So you can use the app instance to set up routing:

```
app.MapGet("/", () => "Hello World!");
```

You can also use app instance to add middleware. Here's an example of how you would use a capability like CORS:

```
app.UseCors("some unique string");
```

Middleware is usually code that intercepts the request and carries out checks like checking for authentication or ensuring the client is allowed to perform this operation according to CORS. After the middleware is done executing, the actual request is carried out. Data is either read or written to the store and a response is sent back to the calling client.

Finally, app.Run() starts your API and makes it listen for requests from the client.

To run your code, you start your project, like any .NET Core project with dotnet run. By default, that means you have a project running on `http://localhost:{PORT}`, where PORT is a value between 5000 and 5300.

### 13.1.6.1.1.Add documentation with Swagger

Documentation is something you want for your API. You want it for yourself, your colleagues, and any third-party developers who might want to use your API. It's key to keep the documentation in sync with your API as it changes. A good approach is to describe your API in a standardized way and ensure it's self-documenting. By *self-documenting*, we mean that if the code changes, the documentation changes with it.

Swagger implements the Open API specification. This format describes your routes but also what data they accept and produce. Swagger UI is a collection of tools that render the Open API specification as a website and let you interact with your API via the website.

To use Swagger and the Swagger UI in your API, you do two things:

- **Install a package.** To install Swagger, you specify to install a package called Swashbuckle:  
`dotnet add package Swashbuckle.AspNetCore`
- **Configure it.** After the package is installed, you configure it via your code. You add a few different entries:

Add a namespace. You need this namespace when you later call SwaggerDoc() and provide the header information for your API.

```
using Microsoft.OpenApi.Models;

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo { Title = "Todo API", Description = "Keep track
of your tasks", Version = "v1" });
});
```

- Add UseSwagger() and UseSwaggerUI(). These two code lines tell the API project to use Swagger and also where to find the specification file `swagger.json`.

```
app.UseSwagger();
app.UseSwaggerUI(c =>
{
    c.SwaggerEndpoint("/swagger/v1/swagger.json", "Todo API V1");
});
```

### 13.1.7. How to add routes and use other advanced commands

<https://docs.microsoft.com/en-us/learn/modules/build-web-api-minimal-api/4-advanced-commands>

How do you build out your app and deal with things such as router parameters, posted bodies, and returning more advanced data than a literal string?

To support so-called RESTful APIs, you need to support the use of HTTP verbs and attach those verbs to different routes. Different verbs have different meanings. Respect the meanings of the HTTP verbs.

HTTP verb	Description
GET	Returns data
POST	Sends data that creates a resource
PUT	Sends data that updates a resource
DELETE	Removes a resource

**Note**

A resource is a piece of data. For example, it can be a product, a user, or an order. It's something you're likely to operate on and for which you want to manage the lifecycle.

#### 13.1.7.1. *HTTP verbs in minimal API*

<https://docs.microsoft.com/en-us/learn/modules/build-web-api-minimal-api/4-advanced-commands>

When a client initiates a request, it does so toward a server endpoint. Imagine a request made toward GET http://localhost:3000/products. The server verifies the request to see what HTTP verb is used. It also needs to know where it's going, which is indicated by "/products." The server then attempts to resolve the request by producing a response.

A minimal API handles routes and HTTP verbs by offering convenience methods. You can map a request by HTTP verb and route, with the route being "/products," in a request to localhost:3000/products. Here's an example of such a convenience method:

```
app.MapGet("/products", () => data);
```

This code should be read in the following way: if the client uses the GET HTTP verb toward the route "/products", then respond with data.

#### GET: Fetch a resource

Two major cases are good to know about when it comes to routing with GET requests:

- **Just the route:** You've seen this route already. For example:

```
app.MapGet("/products", () => data);
```

- **Use a route parameter:** A route parameter is used to find a specific resource. If "/products" means a list of all products, "/products/1" means a specific record. The unique identifier has the value "1." To handle such a request, you use a wildcard to match it. You use "{ID}" to capture the "1" in the preceding example. You can also map the captured value to a parameter:

```
app.MapGet("/products/{id}", (int id) => data.SingleOrDefault(product => product.Id == id));
```

In this code, the id parameter has captured the route parameter the client sent, which is the "1" in "/products/1" or the "11." In this request, it's "/products/1". Then id is used to find a specific record.

## POST: Create a resource

You often want to create a resource, too. For creation, you use the POST HTTP verb. The method to use is called MapPost():

```
app.MapPost("/products", (Product product) => /* */);
```

Note how **product** is sent into the lambda that handles the request. Imagine the following JSON being the body that's sent by the client and that's been serialized by the framework. Imagine if the client has sent the following JSON as its body:

```
{  
    "Name": "New product",  
    "Description": "a description"  
}
```

Then this JSON can map these fields to an object instance of the same shape. Here's a class, Product, that matches the described posted body:

```
public record Product(int Id, string Name);
```

## PUT: Update a resource

The verb PUT means to update a resource. The framework has the method MapPut() for this reason. The MapPut() method is similar semantically to MapPost(). The idea is that you, as a client, should send a posted body with a resource that contains changes. You want these changes applied to an existing resource on the server. Here's how you use MapPut():

```
app.MapPut("/products", (Product product) => /* Update the data store using  
the `product` instance */);
```

## DELETE: Remove a resource

To support the HTTP verb DELETE, use MapDelete(). The idea is for the client to send across a unique identifier that helps the server identify which record to delete. Here's a typical use of this method:

```
app.MapDelete("/products/{id}", (int id) => /* Remove the record whose unique  
identifier matches `id` */);
```

### Return a response

By default, when you respond with certain types, the framework recognizes that these types should be serialized as JSON. Here are some cases:

```
app.MapGet("/products", () => products);  
app.MapGet("/products/{id}", (int id) => products.SingleOrDefault(product =>  
    product.Id == id));  
app.MapGet("/product", () => new { id = 1 });
```

For these cases, you get a JSON response that looks like this example:

```
[{  
    "id": 1,  
    "name": "a product"
```

```

}, {
  "id": 2,
  "name": "another product"
}]

[{
  "id": 1,
  "name": "a product"
}]

{
  "id": 1,
}

```

### 13.1.8. Un semplice esempio di Minimal API: progetto PizzaStore

<https://docs.microsoft.com/en-us/learn/modules/build-web-api-minimal-api/3-exercise-create-minimal-api>

<https://docs.microsoft.com/en-us/learn/modules/build-web-api-minimal-api/5-exercise-advanced-commands>

Scriviamo un esempio di server, sviluppato mediante minimal API, che permette di effettuare alcune operazioni fornamentali come GET, POST, PUT e DELETE, utilizzando una collection di oggetti di tipo Record come repository dei dati.

Creiamo un nuovo progetto, chiamato **PizzaStore**, utilizzando il template, già visto in precedenza, relativo a ASP.NET Core Web API, togliendo la spunta nel wizard di Visual Studio relativamente all'uso dei controllers:

Il file di configurazione del progetto (.csproj) è configurato come segue:

```

<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.2.3" />
  </ItemGroup>

</Project>

```

Aggiungiamo un nuovo file al progetto, chiamato Db.cs con il seguente codice:

```

//file Db.cs

namespace PizzaStore.DB;

public record Pizza
{
  public int Id {get; set;}
  public string? Name { get; set; }
}

```

```

public class PizzaDB
{
    private static List<Pizza> _pizzas = new List<Pizza>()
    {
        new Pizza{ Id=1, Name="Montemagno, Pizza shaped like a great mountain" },
        new Pizza{ Id=2, Name="The Galloway, Pizza shaped like a submarine, silent but
deadly" },
        new Pizza{ Id=3, Name="The Noring, Pizza shaped like a Viking helmet, where's the
mead" }
    };

    public static List<Pizza> GetPizzas()
    {
        return _pizzas;
    }

    public static Pizza? GetPizza(int id)
    {
        return _pizzas.SingleOrDefault(pizza => pizza.Id == id);
    }

    public static Pizza CreatePizza(Pizza pizza)
    {
        _pizzas.Add(pizza);
        return pizza;
    }

    public static Pizza UpdatePizza(Pizza update)
    {
        _pizzas = _pizzas.Select(pizza =>
        {
            if (pizza.Id == update.Id)
            {
                pizza.Name = update.Name;
            }
            return pizza;
        }).ToList();
        return update;
    }

    public static void RemovePizza(int id)
    {
        _pizzas = _pizzas.FindAll(pizza => pizza.Id != id).ToList();
    }
}

```

Modifichiamo il file Program.cs come segue:

```

//file: Program.cs

using Microsoft.OpenApi.Models;
using PizzaStore.DB;
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at
// https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo { Title = "Pizza Store API", Description =
    "Making the Pizzas you love", Version = "v1" });
});
var app = builder.Build();

```

```

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "Pizza Store API V1");
    });
}

app.UseHttpsRedirection();

//var summaries = new[]
//{
//    "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot",
//"Sweltering", "Scorching"
//};

//app.MapGet("/weatherforecast", () =>
//{
//    var forecast = Enumerable.Range(1, 5).Select(index =>
//        new WeatherForecast
//        (
//            DateTime.Now.AddDays(index),
//            Random.Shared.Next(-20, 55),
//            summaries[Random.Shared.Next(summaries.Length)])
//        )
//        .ToArray();
//    return forecast;
//})
//.WithName("GetWeatherForecast");
//app.MapGet("/", () => "Hello World!");
app.MapGet("/pizzas/{id}", (int id) => PizzaDB.GetPizza(id));
app.MapGet("/pizzas", () => PizzaDB.GetPizzas());
app.MapPost("/pizzas", (Pizza pizza) => PizzaDB.CreatePizza(pizza));
app.MapPut("/pizzas", (Pizza pizza) => PizzaDB.UpdatePizza(pizza));
app.MapDelete("/pizzas/{id}", (int id) => PizzaDB.RemovePizza(id));

app.Run();

//internal record WeatherForecast(DateTime Date, int TemperatureC, string? Summary)
//{
//    public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
//}

```

Lanciando il programma possiamo notare che:

The screenshot shows the Swagger UI interface for the "Pizza Store API V1". At the top, there's a navigation bar with the title "Swagger" and "Select a definition" dropdown set to "Pizza Store API V1". The main content area is titled "Pizza Store API v1 OAS3". Below the title, it says "Making the Pizzas you love". The "PizzaStore" section contains several API endpoints:

- GET /pizzas/{id}** (blue button)
- DELETE /pizzas/{id}** (red button)
- GET /pizzas** (blue button)
- POST /pizzas** (green button)
- PUT /pizzas** (orange button)

Below the endpoints is a "Schemas" section showing the definition for the "Pizza" entity:

```

Pizza <-
  {
    id: integer($int32),
    name: string,
    nullable: true
  }
  
```

Il file launchSettings.json è del tipo (si noti che i numeri di porta potrebbero cambiare):

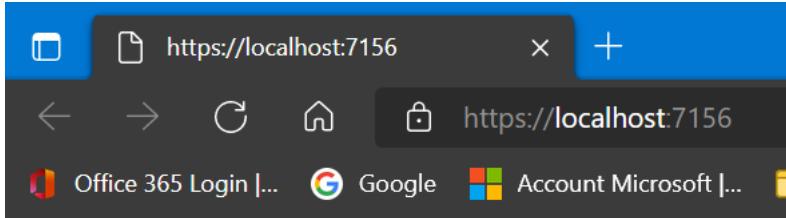
```
{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:34755",
      "sslPort": 44351
    }
  },
  "profiles": {
    "PizzaStore": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "swagger",
      "applicationUrl": "https://localhost:7156;http://localhost:5156",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "swagger",
    }
  }
}
```

```

    "environmentVariables": {
      "ASPNETCORE_ENVIRONMENT": "Development"
    }
  }
}

```

Notiamo che commentando la riga `"launchUrl": "swagger"`, nel file launchSettings.json e aggiungendo la rotta `app.MapGet("/", () => "Hello World!");` nel file Program.cs e facendo ripartire l'applicazione otterremo una risposta nella quale non sarà più caricata la pagina di Swagger, ma direttamente la risposta dell'app in corrispondenza di `/`.



Hello World!

### 13.1.9. Esempio guidato di Minimal API: TodoAPI

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/min-web-api>

In questo esempio guidato vedremo come creare un progetto di Minimal API che fa uso di EntityFramework; useremo Postman per testare le funzionalità dell'applicazione server e applicheremo il concetto di Data Transfer Object (DTO) per elaborare il Model dei dati.

In questo tutorial verranno create le seguenti API:

This tutorial creates the following API:

API	Description	Request body	Response body
GET /	Browser test, "Hello World"	None	Hello World!
GET /todoitems	Get all to-do items	None	Array of to-do items
GET /todoitems/complete	Get completed to-do items	None	Array of to-do items
GET /todoitems/{id}	Get an item by ID	None	To-do item
POST /todoitems	Add a new item	To-do item	To-do item
PUT /todoitems/{id}	Update an existing item	To-do item	None
DELETE /todoitems/{id}	Delete an item	None	None

Creiamo un nuovo progetto di tipo ASP.NET Core Web API:

- In the **Additional information** dialog:
  - Select **.NET 6.0 (Long-term support)**

- Remove **Use controllers** (**uncheck to use minimal APIs**)
- Select **Create**

Visual Studio crea il solito progetto con un template di partenza che genera delle previsioni meteo con valori casuali.

### 13.1.9.1. Prima versione

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at
// https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

var summaries = new[]
{
    "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot",
    "Sweltering", "Scorching"
};

app.MapGet("/weatherforecast", () =>
{
    var forecast = Enumerable.Range(1, 5).Select(index =>
        new WeatherForecast
    (
        DateTime.Now.AddDays(index),
        Random.Shared.Next(-20, 55),
        summaries[Random.Shared.Next(summaries.Length)]
    ))
        .ToArray();
    return forecast;
})
.WithName("GetWeatherForecast");

app.Run();

internal record WeatherForecast(DateTime Date, int TemperatureC, string? Summary)
{
    public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
}
```

Si ottiene l'output già visto nei precedenti esempi:

<https://localhost:7100/swagger/index.html>

The screenshot shows the Swagger UI interface for a TodoApi. At the top, it says "TodoApi 1.0 OAS3". Below that, there's a link to "https://localhost:7100/swagger/v1/swagger.json". The main area is titled "TodoApi" and shows a "GET /weatherforecast" operation. Under "Parameters", it says "No parameters". There are "Execute" and "Clear" buttons. Under "Responses", there's a "curl" command, a "Request URL" (https://localhost:7100/weatherforecast), and a "Server response" section. The "Server response" section shows a code 200 status with a JSON response body containing an array of weather forecast objects.

Se si va sulla rotta <https://localhost:7100/weatherforecast> (che è stata specificata mediante il metodo `app.MapGet("/weatherforecast", () => { //action da eseguire })`) si ottiene la risposta in formato JSON:

```
[
  {
    "date": "2022-03-16T13:37:37.7832285+01:00",
    "temperatureC": 18,
    "summary": "Sweltering",
    "temperatureF": 64
  },
  {
    "date": "2022-03-17T13:37:37.7832519+01:00",
    "temperatureC": 27,
    "summary": "Sweltering",
    "temperatureF": 80
  },
  {
    "date": "2022-03-18T13:37:37.7832544+01:00",
    "temperatureC": 1,
    "summary": "Balmy",
    "temperatureF": 33
  },
  {
    "date": "2022-03-19T13:37:37.783255+01:00",
    "temperatureC": -20,
    "summary": "Cool",
    "temperatureF": -3
  }
]
```

```

},
{
  "date": "2022-03-20T13:37:37.7832556+01:00",
  "temperatureC": -5,
  "summary": "Hot",
  "temperatureF": 24
}
]

```

### 13.1.9.2. Seconda versione: analizziamo Swagger

Modifichiamo l'applicazione generata dal template di Visual Studio, rimuovendo la parte relativa alle previsioni meteo e alla rotta predefinita /weatherforecast:

```

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at
// https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.MapGet("/", () => "Hello World!").WithName("HelloWorld");
app.Run();

```

Al link: <https://localhost:7100/swagger/index.html> viene mostrata la solita pagina di documentazione di Swagger, mentre all'indirizzo: <https://localhost:7100> viene restituita direttamente la risposta: Hello World!

Per fare in modo che all'avvio dell'applicazione server non sia eseguita la pagina della documentazione di Swagger basta commentare la parte relativa al launchUrl nel file **launchSettings.json**:

```

{
  "$schema": "https://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:22556",
      "sslPort": 44333
    }
  },
  "profiles": {
    "TodoApi": {
      "commandName": "Project",
      "launchBrowser": true,
      // "launchUrl": "swagger",
      "environmentVariables": {
        ...
      }
    }
  }
}

```

```

        "ASPNETCORE_ENVIRONMENT": "Development"
    },
    "applicationUrl": "https://localhost:7100;http://localhost:5100",
    "dotnetRunMessages": true
},
"IIS Express": {
    "commandName": "IISExpress",
    "launchBrowser": true,
    //"launchUrl": "swagger",
    "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
    }
},
"WSL": {
    "commandName": "WSL2",
    "launchBrowser": true,
    //"launchUrl": "https://localhost:7100/swagger",
    "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development",
        "ASPNETCORE_URLS": "https://localhost:7100;http://localhost:5100"
    },
    "distributionName": ""
}
}
}

```

Per la gestione dettagliata di Swagger si possono consultare i link:

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle>

<https://docs.microsoft.com/en-us/samples/aspnet/aspnetcore.docs/getstarted-swashbuckle-aspnetcore>

Per le minimal API è possibile, ad esempio specificare (ai fini della documentazione di Swagger) il tipo di dato restituito mettendo un'annotazione direttamente nella lambda che rappresenta l'action da eseguire. Ad esempio:

```
app.MapGet("/", [Produces("text/plain")] () => "Hello World!").WithName("HelloWorld");
```

Questo non è sempre necessario perché il framework, in genere è in grado di dedurre automaticamente il tipo. Nel caso in cui però volessimo forzare un tipo diverso per la documentazione di Swagger potremmo specificarlo direttamente come annotazione della lambda, ad esempio:

```
app.MapGet("/", [Produces("application/json")] () => "Hello
World!").WithName("HelloWorld");
```

E' anche possibile documentare il codice http restituito, utilizzando annotazioni specifiche. Ad esempio:

```
app.MapGet("/", 
[Produces("text/plain")][ProducesResponseType(StatusCodes.Status201Created)][ProducesResponseType(StatusCodes.Status400BadRequest)] () => "Hello
World!").WithName("HelloWorld");
```

In questo caso, anche se l'action dovrebbe, di norma, restituire 200 come codice, dal momento che stampa una semplice stringa, Swagger mostrerà i codici 201 e 400.

### 13.1.9.3. Terza versione: utilizziamo EntityFramework

Aggiungiamo, tramite NuGet, i pacchetti:

**Microsoft.EntityFrameworkCore.InMemory**

## Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore

Il database che useremo è **InMemory** che è un database non persistente e che salva gli oggetti in memoria. In questo esempio serve solo a mostrare l'utilizzo di EF Core. Nei prossimi esempi vedremo come utilizzare EF Core in abbinamento a database reali come SQLite e MySQL/MariaDB.

Il codice completo di questa versione è:

//file Program.cs:

```
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddDbContext<TodoDb>(opt => opt.UseInMemoryDatabase("TodoList"));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.MapGet("/", () => "Hello World!");

app.MapGet("/todoitems", async (TodoDb db) =>
    await db.Todos.ToListAsync());

app.MapGet("/todoitems/complete", async (TodoDb db) =>
    await db.Todos.Where(t => t.IsComplete).ToListAsync());

app.MapGet("/todoitems/{id}", async (int id, TodoDb db) =>
    await db.Todos.FindAsync(id)
        is Todo todo
            ? Results.Ok(todo)
            : Results.NotFound());

app.MapPost("/todoitems", async (Todo todo, TodoDb db) =>
{
    db.Todos.Add(todo);
    await db.SaveChangesAsync();

    return Results.Created($"/todoitems/{todo.Id}", todo);
});

app.MapPut("/todoitems/{id}", async (int id, Todo inputTodo, TodoDb db) =>
{
    var todo = await db.Todos.FindAsync(id);

    if (todo is null) return Results.NotFound();

    todo.Name = inputTodo.Name;
    todo.IsComplete = inputTodo.IsComplete;
})
```

```

    await db.SaveChangesAsync();
    return Results.NoContent();
});

app.MapDelete("/todoitems/{id}", async (int id, TodoDb db) =>
{
    if (await db.Todos.FindAsync(id) is Todo todo)
    {
        db.Todos.Remove(todo);
        await db.SaveChangesAsync();
        return Results.Ok(todo);
    }

    return Results.NotFound();
});

app.Run();

class Todo
{
    public int Id { get; set; }
    public string? Name { get; set; }
    public bool IsComplete { get; set; }
}

class TodoDb : DbContext
{
    public TodoDb(DbContextOptions<TodoDb> options)
        : base(options) { }

    public DbSet<Todo> Todos => Set<Todo>();
}

```

Otteniamo:

```

Todo
{
    id: integer($int32)
    name: string
    nullable: true
    isComplete: boolean
}

```

### 13.1.9.3.1. Testing dell'applicazione con Swagger

Per testare l'applicazione possiamo usare direttamente l'interfaccia di Swagger. Ad esempio, eseguiamo la post di alcuni oggetti nel database in memoria. In questo caso è indifferente mettere un id specifico oppure non metterlo affatto, poiché l'id verrà generato automaticamente dal database.

```

{
    "name": "Studiare informatica",
    "isComplete": false
}

```

Risposta:

Request URL  
<https://localhost:7100/todoitems>

Server response

Code	Details	Links
201 Undocumented	<b>Response body</b> <pre>{   "id": 1,   "name": "Studiare informatica",   "isComplete": false }</pre>	
	<b>Response headers</b> <pre>content-type: application/json; charset=utf-8 date: Wed, 16 Mar 2022 07:46:06 GMT location: /todoitems/1 server: Kestrel</pre>	
	<b>Responses</b>	
Code	Description	Links
200	Success	No links

Mettendo qualche altro oggetto nel database, possiamo poi verificare che la GET sull'intera collection restituisce tutti gli oggetti come previsto.

Curl  

```
curl -X 'GET' \
'https://localhost:7100/todoitems' \
-H 'accept: application/json'
```

Request URL  
<https://localhost:7100/todoitems>

Server response

Code	Details	Links
200	<b>Response body</b> <pre>[   {     "id": 1,     "name": "Studiare informatica",     "isComplete": false   },   {     "id": 2,     "name": "Studiare matematica",     "isComplete": false   },   {     "id": 3,     "name": "sviluppare il progetto Alice Neural",     "isComplete": true   } ]</pre>	
	<b>Response headers</b> <pre>content-type: application/json; charset=utf-8 date: Wed, 16 Mar 2022 07:48:47 GMT server: Kestrel</pre>	
	<b>Responses</b>	
Code	Description	Links
200	Success	No links

Media type

Content-Type header

E' possibile filtrare solo gli oggetti con il campo isComplete a true:

Curl

```
curl -X 'GET' \
'https://localhost:7100/todoitems/complete' \
-H 'accept: application/json'
```

Request URL

<https://localhost:7100/todoitems/complete>

Server response

Code	Details	Links
200	<p>Response body</p> <pre>[   {     "id": 3,     "name": "sviluppare il progetto Alice Neural",     "isComplete": true   } ]</pre> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Wed, 16 Mar 2022 07:50:17 GMT server: Kestrel</pre>	

Responses

Code	Description	Links
200	Success	No links

Etc.

Come si può vedere dalla finestra di Debug, ogni volta che si esegue un'operazione sul database viene generato un messaggio di log:

Select Microsoft Visual Studio Debug Console

```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7100
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5100
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\genna\source\repos\4IA\LessonPrepare\MinAPI\MinAPIExplorer\TodoApi\TodoApi\
info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 6.0.1 initialized 'TodoDb' using provider 'Microsoft.EntityFrameworkCore.InMemory:6.0.1' with options: StoreName=TodoList
info: Microsoft.EntityFrameworkCore.Update[30100]
      Saved 1 entities to in-memory store.
info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 6.0.1 initialized 'TodoDb' using provider 'Microsoft.EntityFrameworkCore.InMemory:6.0.1' with options: StoreName=TodoList
info: Microsoft.EntityFrameworkCore.Update[30100]
      Saved 1 entities to in-memory store.
info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 6.0.1 initialized 'TodoDb' using provider 'Microsoft.EntityFrameworkCore.InMemory:6.0.1' with options: StoreName=TodoList
info: Microsoft.EntityFrameworkCore.Update[30100]
      Saved 1 entities to in-memory store.
info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 6.0.1 initialized 'TodoDb' using provider 'Microsoft.EntityFrameworkCore.InMemory:6.0.1' with options: StoreName=TodoList
info: Microsoft.EntityFrameworkCore.Update[30100]
      Saved 1 entities to in-memory store.
C:\Users\genna\source\repos\4IA\LessonPrepare\MinAPI\MinAPIExplorer\TodoApi\TodoApi\bin\Debug\net6.0\TodoApi.exe (process)
```

### 13.1.9.3.2. Testing dell'applicazione con Postman

E' possibile testare l'applicazione con Postman, seguendo l'esercitazione guidata al link:

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/min-web-api?view=aspnetcore-6.0&tabs=visual-studio#install-postman-to-test-the-app>

### 13.1.9.3.3.Return Values

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/min-web-api?view=aspnetcore-6.0&tabs=visual-studio#return-values>

ASP.NET Core automatically serializes the object to [JSON](#) and writes the JSON into the body of the response message. The response code for this return type is [200 OK](#), assuming there are no unhandled exceptions. Unhandled exceptions are translated into 5xx errors.

The return types can represent a wide range of HTTP status codes. For example, GET /todoitems/{id} can return two different status values:

- If no item matches the requested ID, the method returns a [404 status NotFound](#) error code.
- Otherwise, the method returns 200 with a JSON response body. Returning item results in an HTTP 200 response.

### 13.1.9.4. Quarta versione: uso di Data Transfer Object (DTO)

Currently the sample app exposes the entire Todo object. Production apps typically limit the data that's input and returned using a subset of the model. There are multiple reasons behind this and [security is a major one](#). The subset of a model is usually referred to as a Data Transfer Object (DTO), input model, or view model. **DTO** is used in this article.

A DTO may be used to:

- Prevent over-posting.
- Hide properties that clients are not supposed to view.
- Omit some properties in order to reduce payload size.
- Flatten object graphs that contain nested objects. Flattened object graphs can be more convenient for clients.

To demonstrate the DTO approach, update the Todo class to include a secret field:

```
class Todo
{
    public int Id { get; set; }
    public string? Name { get; set; }
    public bool IsComplete { get; set; }
    public string? Secret { get; set; }
}
```

The secret field needs to be hidden from this app, but an administrative app could choose to expose it.

Verify you can post and get the secret field.

Create a DTO model:

```
public class TodoItemDTO
{
    public int Id { get; set; }
    public string? Name { get; set; }
    public bool IsComplete { get; set; }

    public TodoItemDTO() { }
    public TodoItemDTO(Todo todoItem) =>
```

```
        (Id, Name, IsComplete) = (todoItem.Id, todoItem.Name, todoItem.IsComplete);
    }
```

Il codice completo di questa versione è:

//File Program.cs

```
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddDbContext<TodoDb>(opt => opt.UseInMemoryDatabase("TodoList"));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.MapGet("/", () => "Hello World!");

app.MapGet("/todoitems", async (TodoDb db) =>
    await db.Todos.Select(x => new TodoItemDTO(x)).ToListAsync());

app.MapGet("/todoitems/{id}", async (int id, TodoDb db) =>
    await db.Todos.FindAsync(id)
        is Todo todo
            ? Results.Ok(new TodoItemDTO(todo))
            : Results.NotFound());

app.MapPost("/todoitems", async (TodoItemDTO todoItemDTO, TodoDb db) =>
{
    var todoItem = new Todo
    {
        IsComplete = todoItemDTO.IsComplete,
        Name = todoItemDTO.Name
    };

    db.Todos.Add(todoItem);
    await db.SaveChangesAsync();

    return Results.Created($"/todoitems/{todoItem.Id}", new TodoItemDTO(todoItem));
});

app.MapPut("/todoitems/{id}", async (int id, TodoItemDTO todoItemDTO, TodoDb db) =>
{
    var todo = await db.Todos.FindAsync(id);

    if (todo is null) return Results.NotFound();

    todo.Name = todoItemDTO.Name;
    todo.IsComplete = todoItemDTO.IsComplete;
}
```

```

        await db.SaveChangesAsync();

        return Results.NoContent();
    });

app.MapDelete("/todoitems/{id}", async (int id, TodoDb db) =>
{
    if (await db.Todos.FindAsync(id) is Todo todo)
    {
        db.Todos.Remove(todo);
        await db.SaveChangesAsync();
        return Results.Ok(new TodoItemDTO(todo));
    }

    return Results.NotFound();
});

app.Run();
}

public class Todo
{
    public int Id { get; set; }
    public string? Name { get; set; }
    public bool IsComplete { get; set; }
    public string? Secret { get; set; }
}

public class TodoItemDTO
{
    public int Id { get; set; }
    public string? Name { get; set; }
    public bool IsComplete { get; set; }

    public TodoItemDTO() { }
    public TodoItemDTO(Todo todoItem) =>
    (Id, Name, IsComplete) = (todoItem.Id, todoItem.Name, todoItem.IsComplete);
}

class TodoDb : DbContext
{
    public TodoDb(DbContextOptions<TodoDb> options)
        : base(options) { }

    public DbSet<Todo> Todos => Set<Todo>();
}

```

### 13.1.10. Esempio guidato di Minimal API: PizzaStoreSQLite

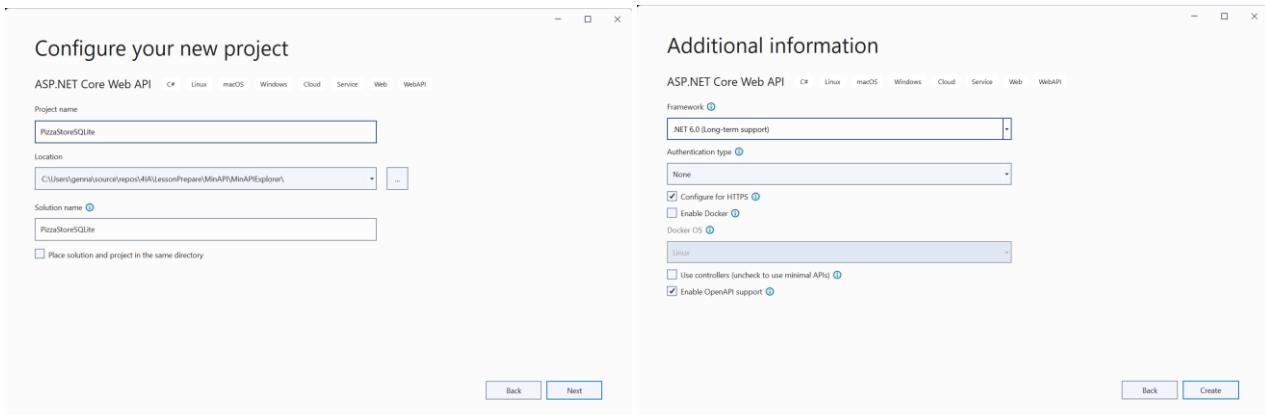
<https://docs.microsoft.com/en-us/learn/modules/build-web-api-minimal-database/1-introduction>

<https://docs.microsoft.com/en-us/learn/modules/build-web-api-minimal-database/2-what-is-entity-framework-core>

<https://docs.microsoft.com/en-us/learn/modules/build-web-api-minimal-database/3-exercise-add-entity-framework-core>

<https://docs.microsoft.com/en-us/learn/modules/build-web-api-minimal-database/5-exercise-use-sqlite-database>

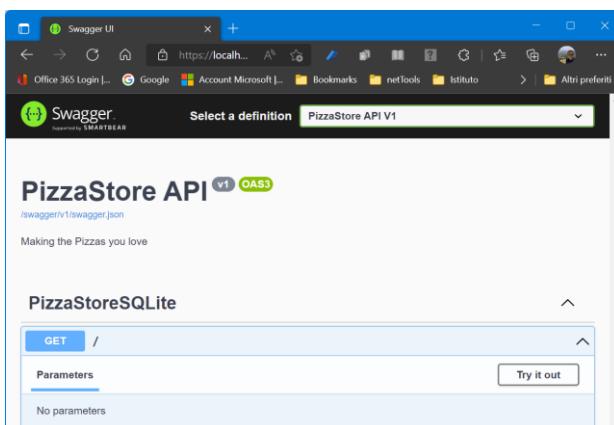
Creiamo un nuovo progetto ASP.NET Core Web API, chiamato PizzaStoreSQLite, togliendo la spunta dall'opzione “Use controllers”:



Modifichiamo il template di default che presenta il solito esempio relativo alle previsioni meteo, con il seguente codice:

```
using Microsoft.OpenApi.Models;
var builder = WebApplication.CreateBuilder(args);
// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "PizzaStore API",
        Description = "Making the Pizzas you love",
        Version = "v1"
    });
});
var app = builder.Build();
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "PizzaStore API V1");
    });
}
app.UseHttpsRedirection();
app.MapGet("/", () => "Hello World!");
app.Run();
```

Ottieniamo:



Creiamo il file Pizza.cs nella cartella Models, con il seguente contenuto:

```
//file Pizza.cs

namespace PizzaStoreSQLite.Models
{
    public class Pizza
    {
        public int Id { get; set; }
        public string? Name { get; set; }
        public string? Description { get; set; }
    }
}
```

### Aggiungiamo EF Core InMemory:

Per usare la versione InMemory si può operare come già visto nel precedente esempio PizzaStore:

- Aggiungere **Microsoft.EntityFrameworkCore.InMemory** da NuGet
- Aggiungere using **Microsoft.EntityFrameworkCore**; in testa al file Program.cs
- Aggiungere la classe PizzaDb nella cartella Data:

```
//file PizzaDb.cs
```

```
using Microsoft.EntityFrameworkCore;
using PizzaStoreSQLite.Models;
namespace PizzaStoreSQLite.Data
{
    class PizzaDb : DbContext
    {
        public PizzaDb(DbContextOptions options) : base(options) { }
        public DbSet<Pizza> Pizzas => Set<Pizza>();
    }
}
```

//Scrivere il file Program.cs come segue:

```
using Microsoft.OpenApi.Models;
using Microsoft.EntityFrameworkCore;
using PizzaStoreSQLite.Data;
using PizzaStoreSQLite.Models;

var builder = WebApplication.CreateBuilder(args);
// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddDbContext<PizzaDb>(options =>
options.UseInMemoryDatabase("items"));
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "PizzaStore API",
        Description = "Making the Pizzas you love",
        Version = "v1"
    });
});
var app = builder.Build();
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
```

```

app.UseSwaggerUI(c =>
{
    c.SwaggerEndpoint("/swagger/v1/swagger.json", "PizzaStore API V1");
});
app.UseHttpsRedirection();
app.MapGet("/", () => "Hello World!");
app.MapGet("/pizzas", async (PizzaDb db) => await db.Pizzas.ToListAsync());
app.MapPost("/pizza", async (PizzaDb db, Pizza pizza) =>
{
    await db.Pizzas.AddAsync(pizza);
    await db.SaveChangesAsync();
    return Results.Created($"/pizza/{pizza.Id}", pizza);
});
app.MapGet("/pizza/{id}", async (PizzaDb db, int id) => await
db.Pizzas.FindAsync(id));
app.MapPut("/pizza/{id}", async (PizzaDb db, Pizza updatepizza, int id) =>
{
    var pizza = await db.Pizzas.FindAsync(id);
    if (pizza is null) return Results.NotFound();
    pizza.Name = updatepizza.Name;
    pizza.Description = updatepizza.Description;
    await db.SaveChangesAsync();
    return Results.NoContent();
});
app.MapDelete("/pizza/{id}", async (PizzaDb db, int id) =>
{
    var pizza = await db.Pizzas.FindAsync(id);
    if (pizza is null)
    {
        return Results.NotFound();
    }
    db.Pizzas.Remove(pizza);
    await db.SaveChangesAsync();
    return Results.Ok();
});
app.Run();

```

### Testiamo l'applicazione mediante Swagger

L'applicazione creata utilizza come Data Provider di EntityFramework un database InMemory che non è persistente: quando si chiude l'app si perdono i dati. Per rendere i dati persistenti, bisogna cambiare provider ed usarene uno come, ad esempio, SQLite.

### Modifichiamo l'applicazione usando SQLite come Data Provider di EntityFramework:

Installiamo tramite Nuget:

- Microsoft.EntityFrameworkCore.Sqlite
- Microsoft.EntityFrameworkCore.Tools
- Microsoft.EntityFrameworkCore.Design
- Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore

Nel file Program.cs modifichiamo il codice con le istruzioni evidenziate di seguito:

//file Program.cs

```

using Microsoft.OpenApi.Models;
using Microsoft.EntityFrameworkCore;
using PizzaStoreSQLite.Data;
using PizzaStoreSQLite.Models;

```

```

var builder = WebApplication.CreateBuilder(args);
var connectionString = builder.Configuration.GetConnectionString("Pizzas") ?? "Data
Source=DefaultPizzas.db";
// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
//builder.Services.AddDbContext<PizzaDb>(options =>
options.UseInMemoryDatabase("items"));
//builder.Services.AddSqlite<PizzaDb>(connectionString);
builder.Services.AddDbContext<PizzaDb>(options =>
options.UseSqlite(connectionString));
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "PizzaStore API",
        Description = "Making the Pizzas you love",
        Version = "v1"
    });
});
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
var app = builder.Build();
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "PizzaStore API V1");
    });
}
app.UseHttpsRedirection();
app.MapGet("/", () => "Hello World!");
app.MapGet("/pizzas", async (PizzaDb db) => await db.Pizzas.ToListAsync());
app.MapPost("/pizza", async (PizzaDb db, Pizza pizza) =>
{
    await db.Pizzas.AddAsync(pizza);
    await db.SaveChangesAsync();
    return Results.Created($"/pizza/{pizza.Id}", pizza);
});
app.MapGet("/pizza/{id}", async (PizzaDb db, int id) => await
db.Pizzas.FindAsync(id));
app.MapPut("/pizza/{id}", async (PizzaDb db, Pizza updatePizza, int id) =>
{
    var pizza = await db.Pizzas.FindAsync(id);
    if (pizza is null) return Results.NotFound();
    pizza.Name = updatePizza.Name;
    pizza.Description = updatePizza.Description;
    await db.SaveChangesAsync();
    return Results.NoContent();
});
app.MapDelete("/pizza/{id}", async (PizzaDb db, int id) =>
{
    var pizza = await db.Pizzas.FindAsync(id);
    if (pizza is null)
    {
        return Results.NotFound();
    }
    db.Pizzas.Remove(pizza);
    await db.SaveChangesAsync();
    return Results.Ok();
});
app.Run();

```

```

//file appsettings.json

{
    "ConnectionStrings": {
        "Pizzas": "Data Source=Pizzas.db"
    },
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft.AspNetCore": "Warning"
        }
    },
    "AllowedHosts": "*"
}

//file PizzaDb.cs

using Microsoft.EntityFrameworkCore;
using PizzaStoreSQLite.Models;
namespace PizzaStoreSQLite.Data
{
    class PizzaDb : DbContext
    {
        public PizzaDb(DbContextOptions options) : base(options) { }
        public DbSet<Pizza> Pizzas => Set<Pizza>();
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Pizza>().HasData(
                new Pizza { Id = 1, Name = "Montemagno", Description= "Pizza shaped like a great mountain" },
                new Pizza { Id = 2, Name = "The Galloway", Description= "Pizza shaped like a submarine, silent but deadly" },
                new Pizza { Id = 3, Name = "The Noring", Description= "Pizza shaped like a Viking helmet, where's the mead" });
        }
    }
}

```

Nota: la modifica al file PizzaDb.cs non era strettamente necessaria: la parte che è stata aggiunta serve solo a popolare il database nel momento in cui il database è creato.

Salviamo tutti i file.

Effettuiamo la Migration del database con i comandi della Packet Manager Console di Visual Studio. Selezioniamo come default project della Packet Manager Console il progetto PizzaStoreSQLite ed eseguiamo i seguenti comandi:

```

Add-Migration InitialCreate
Update-Database

```

```

Package Manager Console
Package source: All Default project: PizzaStoreSQLite
Done.
PM> Add-Migration InitialCreate
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 6.0.3 initialized 'PizzaDb' using provider 'Microsoft.EntityFrameworkCore.Sqlite:6.0.3' with options: None
To undo this action, use Remove-Migration.
PM> Update-Database
Build started...
Build succeeded.
144 %

```

Testiamo l'applicazione.

### PizzaStoreSQLite

**GET** /

**GET** /pizzas

Parameters

No parameters

**Execute** **Clear**

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7203/pizzas' \
  -H 'accept: application/json'
```

Request URL

```
https://localhost:7203/pizzas
```

Server response

Code	Details	Links
200	<p>Response body</p> <pre>[   {     "id": 1,     "name": "Montemagno",     "description": "Pizza shaped like a great mountain"   },   {     "id": 2,     "name": "The Galloway",     "description": "Pizza shaped like a submarine, silent but deadly"   },   {     "id": 3,     "name": "The Morning",     "description": "Pizza shaped like a Viking helmet, where's the mead"   } ]</pre> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Fri, 25 Mar 2022 07:45:11 GMT server: Kestrel</pre>	
Responses		
Code	Description	Links
200	Success	No links

## 13.2. DBMS relazionali client/server

In questo corso impareremo ad usare un DBMS (Data Base Management System) relazionale, basato su architettura client/server. Un DBMS è un software che si occupa della gestione dei dati in file speciali chiamati database. L'accesso ai dati presenti nei database può essere fatto solo interagendo con il software del DBMS. Un'applicazione, scritta in qualsiasi linguaggio, che voglia accedere ai dati memorizzati su un database, deve comportarsi come un'applicazione client e interfacciarsi con il software del DBMS, che si comporta da server.

Un DBMS non si occupa soltanto della gestione dei dati su memoria di massa, ma anche della gestione degli accessi e dei permessi sui dati gestiti.

Un DBMS basato su architettura client/server è caratterizzato dall'esistenza di un processo server in ascolto su una specifica porta di un determinato dominio. Ad esempio, il DBMS MariaDb che useremo negli esempi, quando è in esecuzione, rimane in ascolto su una porta di default (3306), e accetta connessioni entranti contenenti richieste di accesso, solo da utenti autorizzati. Se la richiesta di accesso è autorizzata, il DBMS effettua le operazioni richieste sui dati (lettura, scrittura, modifica, cancellazione) e restituisce il risultato dell'elaborazione al processo client.

Un DBMS, in genere è in grado di gestire più database. Ogni database può essere visto come un'insieme di dati collegati tra di loro.

**Un database relazionale** è caratterizzato dal fatto che i dati sono memorizzati su tabelle a schema fisso, collegate tra loro mediante il concetto di chiave esterna (foreign key). In ogni tabella ci sono uno o più attributi che fungono da chiave primaria (primary key). In una tabella non possono esistere due righe che hanno lo stesso valore della chiave primaria, mentre è possibile che più righe della stessa tabella presentino lo stesso valore della chiave esterna. Il concetto di chiave esterna è assimilabile a quello di puntatore (o reference) che punta, in genere, ad una chiave primaria di un'altra tabella. Questi concetti sono stati già introdotti nella prima parte del corso, quando è stato presentato il framework Entity Core e il database SQLite. In questa parte del corso si presenterà un esempio di database relazionale molto utilizzato nelle applicazioni web, chiamato MariaDb.

La teoria dei database in generale e dei database relazionali in particolare verrà ripresa e approfondita nel corso dell'anno prossimo, ma in questo contesto appare importante incominciare ad usare alcuni esempi di DBMS relazionali per poterli rapidamente integrare nello sviluppo di REST API.

### 13.2.1. Il database MySQL/MariaDb

<https://it.wikipedia.org/wiki/MariaDB>

MariaDb è un fork del progetto MySQL, che negli ultimi anni, soprattutto in ambito open source, si è guadagnato molto spazio nello sviluppo di codice Open Source. Il database MySQL è attualmente gestito da Oracle, mentre il software Maria DB è gestito dalla Maria DB Foundation.

Gran parte delle caratteristiche di MariaDB e MySQL sono sovrapponibili, nel senso che il set di istruzioni e le caratteristiche supportate dai due software sono molto simili.

#### 13.2.1.1. Installazione del software MariaDb

Esistono diversi modi per installare il software MariaDb e di seguito illustreremo uno dei più semplici.

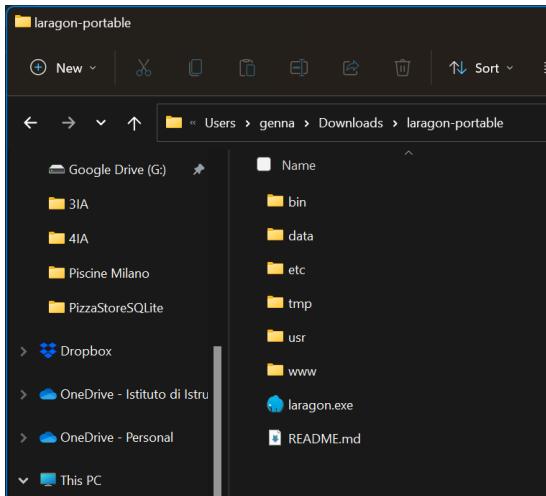
Step. 1: installazione del pacchetto Laragon: <https://laragon.org/download/index.html>

Per chi utilizza il proprio PC personale è possibile effettuare un'installazione full, mentre per chi non

ha i permessi di amministratore è possibile installare la versione portable.

L'installazione della versione portable è molto semplice: basta scaricare il file compresso e scompattarlo in una cartella su cui si hanno i diritti di lettura/scrittura/esecuzione.

Ad esempio:

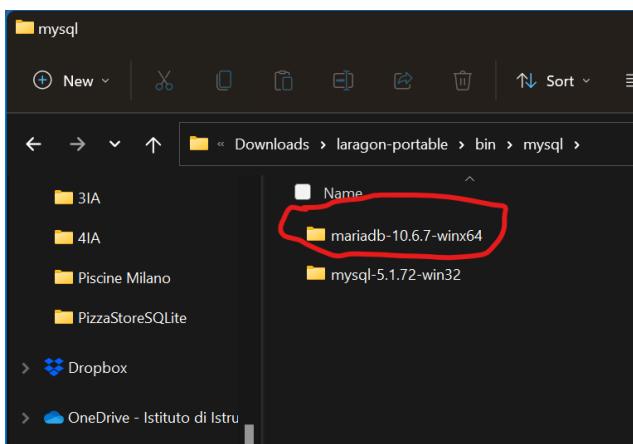


Il pacchetto Laragon permette di avere un'interfaccia grafica di semplice utilizzo per la gestione del DBMS MariaDB, tuttavia, nella sua configurazione di default, Laragon viene distribuito con MySQL 5.1. Per passare alla versione di MariaDB 10.6.7 che useremo negli esempi, basta scaricare il file compresso (.zip) del pacchetto completo di MariaDB dal link:

<https://mariadb.com/kb/en/portable-mariadb/>

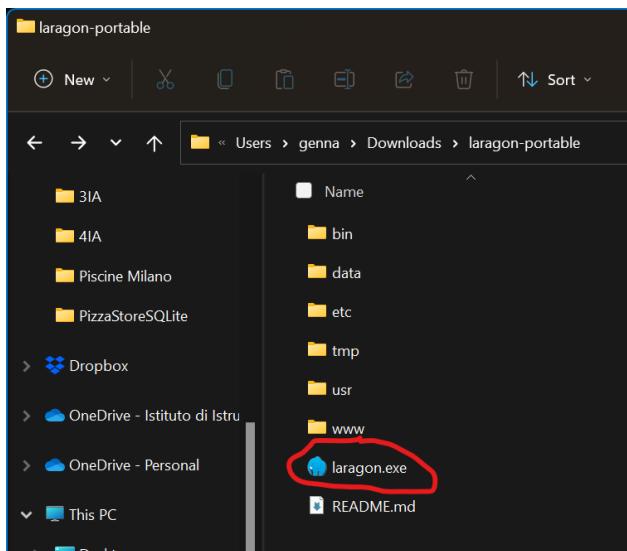
[https://mariadb.org/download/?t=mariadb&p=mariadb&r=10.6.7&os=windows&cpu=x86\\_64&pkg=zip&m=mva](https://mariadb.org/download/?t=mariadb&p=mariadb&r=10.6.7&os=windows&cpu=x86_64&pkg=zip&m=mva)

Il file zip va scompattato nella cartella: **laragon-portable\bin\mysql**, se si utilizza la versione portable, oppure nella cartella **laragon\bin\mysql** se si utilizza la versione completa.



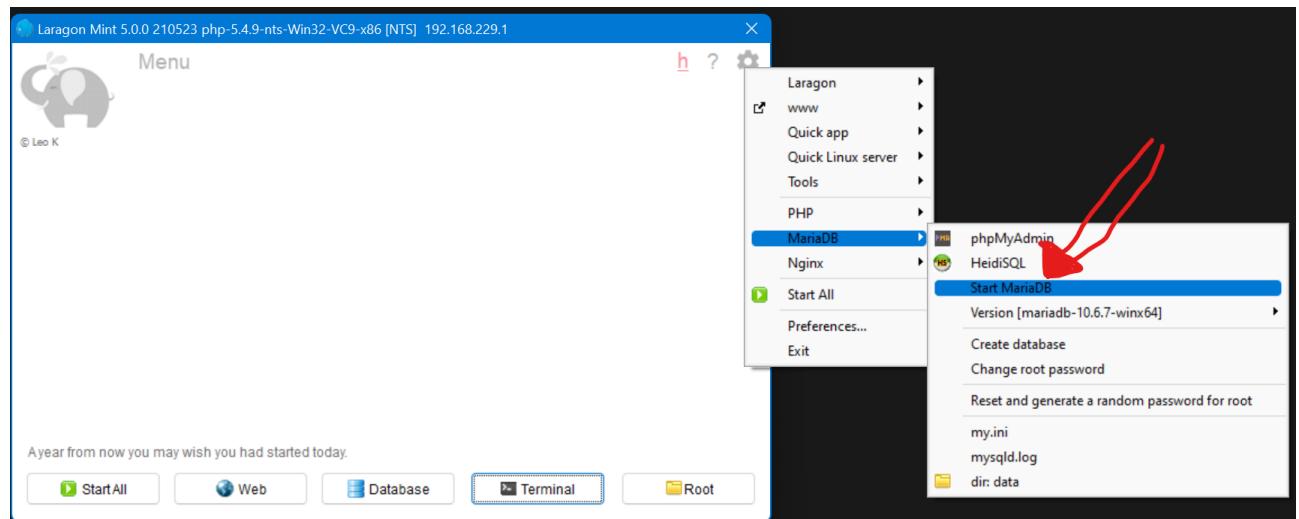
Per utilizzare MariaDB da Laragon occorre:

- lasciare il pannello grafico di Laragon, facendo click sul file laragon.exe (versione portable), oppure richiamando il programma Laragon dalla barra di ricerca di Windows (versione installata sul PC)

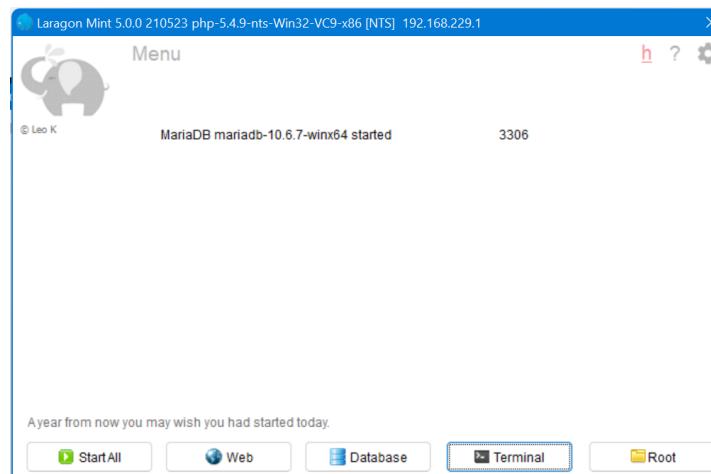


### 13.2.1.2. Avvio di MariaDB

Dal pannello di controllo di Laragon avviare MariaDB come mostrato in figura:

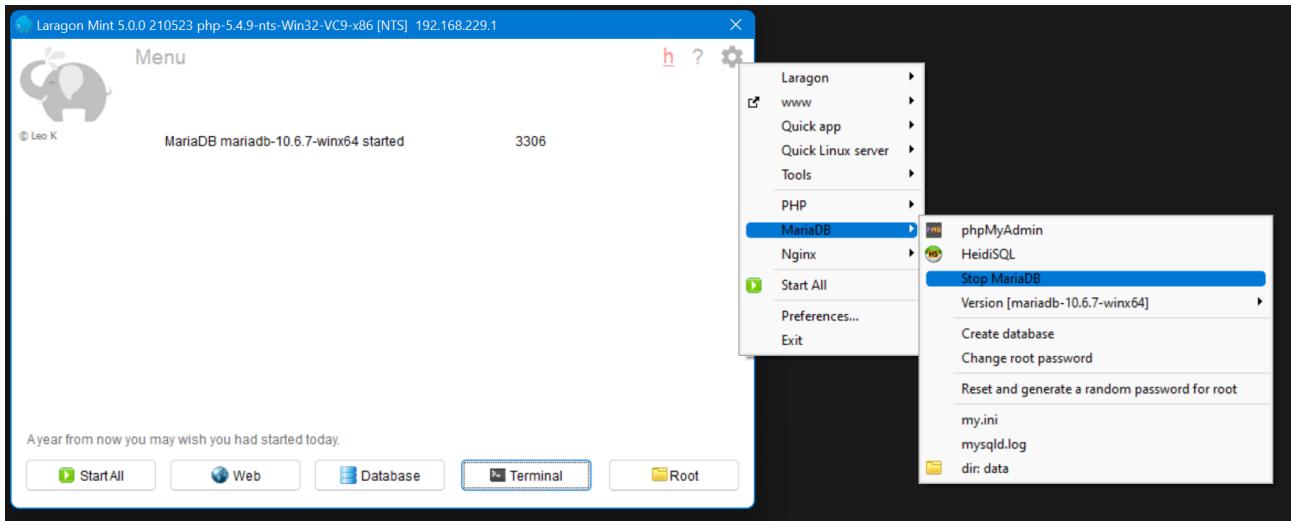


Si dovrebbe vedere il server di MariaDB in ascolto sulla porta 3306



### 13.2.1.3. Stop di MariaDB

Per fermare MariaDb:



#### 13.2.1.4. MySQL Monitor (interfaccia console)

Per interagire con il server è possibile usare un client testuale che si chiama mysql monitor (programma mysql.exe) che si può lanciare direttamente dal pulsante “Terminal” di Laragon:

Dal terminale scrivere il comando:

```
mysql -u root -p
```

Premere invio. Quando il programma chiede la password inserire solo invio. Dovrebbe comparire una schermata come la seguente:

The terminal window shows the command 'mysql -u root -p' entered. The response is as follows:

```
C:\laragon\www>mysql -u root -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 10.6.7-MariaDB-log mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Se il percorso di mysql.exe non è nelle variabili d’ambiente basta aprire la console nella directory dove si trova il programma mysql.exe. Ad esempio nel caso di laragon portable con la versione di MariaDB installata:

```
laragon-portable\bin\mysql\mariadb-10.6.7-winx64\bin
```

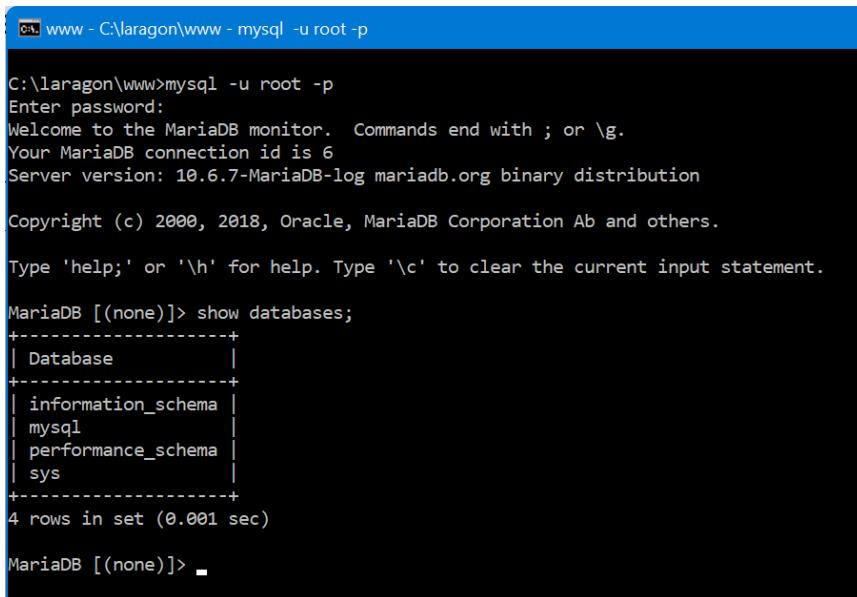
Inizialmente il DBMS è configurato per essere utilizzato da un’utente che si chiama root, senza password. L’utente root è l’utente che ha i privilegi più alti all’interno del DBMS e quindi è molto importante impostare una password per tale utente. Vedremo in seguito che è buona norma creare degli account con minori privilegi che abbiano i permessi minimi richiesti dall’applicazione che utilizza il DBMS.

Per vedere quali database sono gestiti dal BDMS si può scrivere:

```
show databases;
```

(attenzione al ; finale)

Ogni istruzione SQL scritta nel terminale deve avere il ; finale. Nel caso in cui si voglia interrompere la scrittura dell'istruzione corrente, si può scrivere \c e premere invio.



```
C:\www - C:\laragon\www - mysql -u root -p
C:\laragon\www>mysql -u root -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 6
Server version: 10.6.7-MariaDB-log mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.001 sec)

MariaDB [(none)]> -
```

### 13.2.1.5. HeidiSQL (interfaccia grafica)

Se si utilizza la versione portable di Laragon conviene aggiornare il pacchetto HeidiSQL, scaricando l'ultima versione dal sito dello sviluppatore. Infatti, al momento, la versione portable è meno aggiornata di quella full e potrebbero esserci problemi di compatibilità tra le ultime versioni di MariaDb e la versione di HeidiSQL, fornita nella versione portable di Laragon.

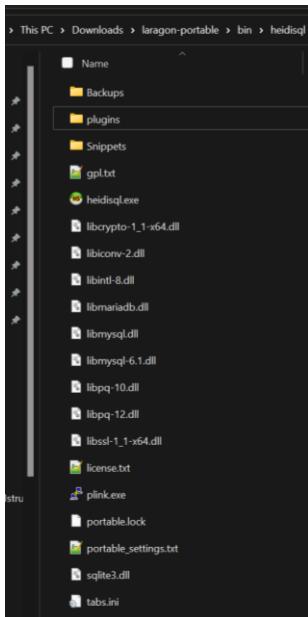
Per l'aggiornamento di HeidiSQL:

Fermare MariaDb

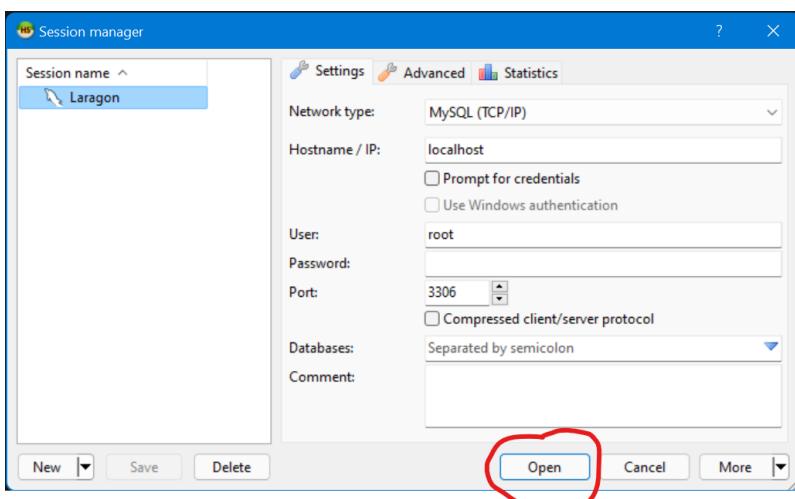
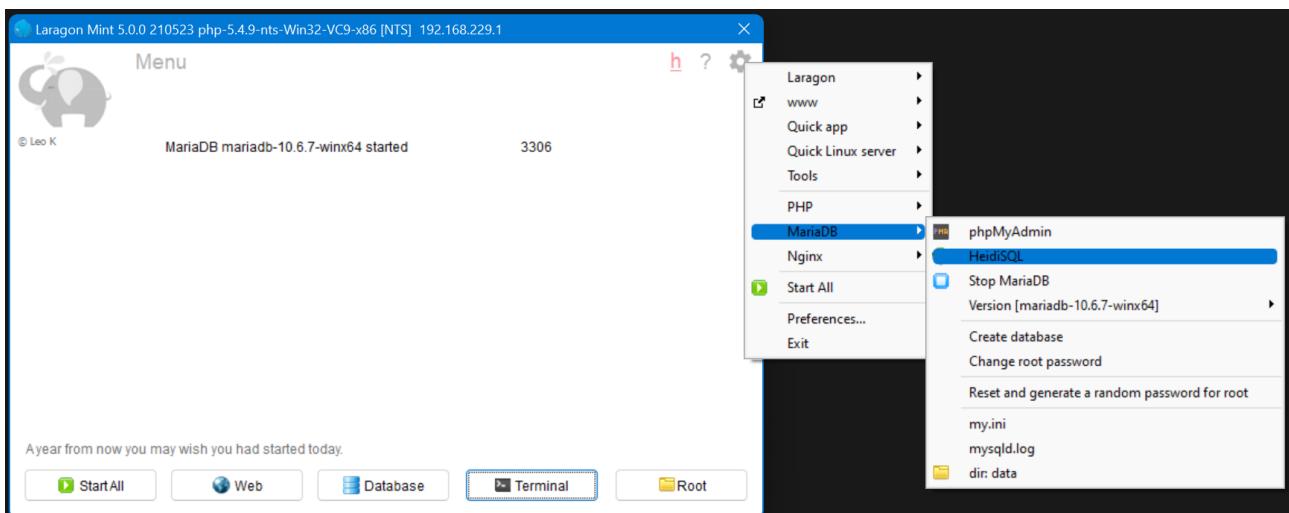
Aprire la cartella: **laragon-portable\bin\heidisql** e cancellare tutto il suo contenuto.

Scaricare la cartella compressa dell'ultima versione di HeidiSQL da

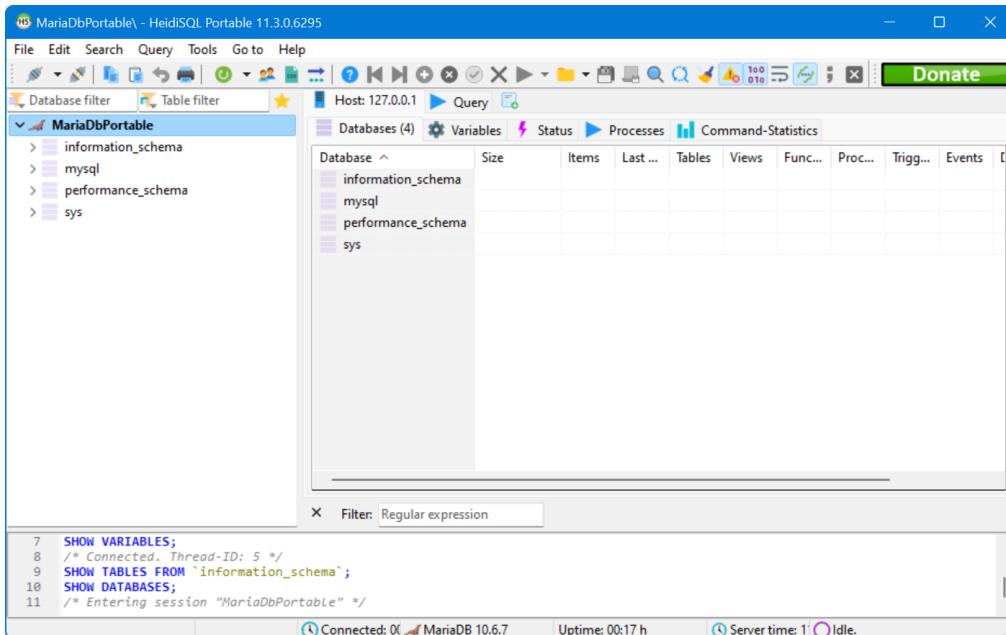
<https://www.heidisql.com/download.php?download=portable> (versione 64 bit) e copiare tutto il suo contenuto nella cartella **laragon-portable\bin\heidisql** in modo da avere:



Avviare HeidiSQL e creare una nuova connessione:



Per aprire l'interfaccia grafica basta cliccare sul pulsante open e utilizzare le credenziali di default dell'utente root per la sessione preimpostata chiamata Laragon.

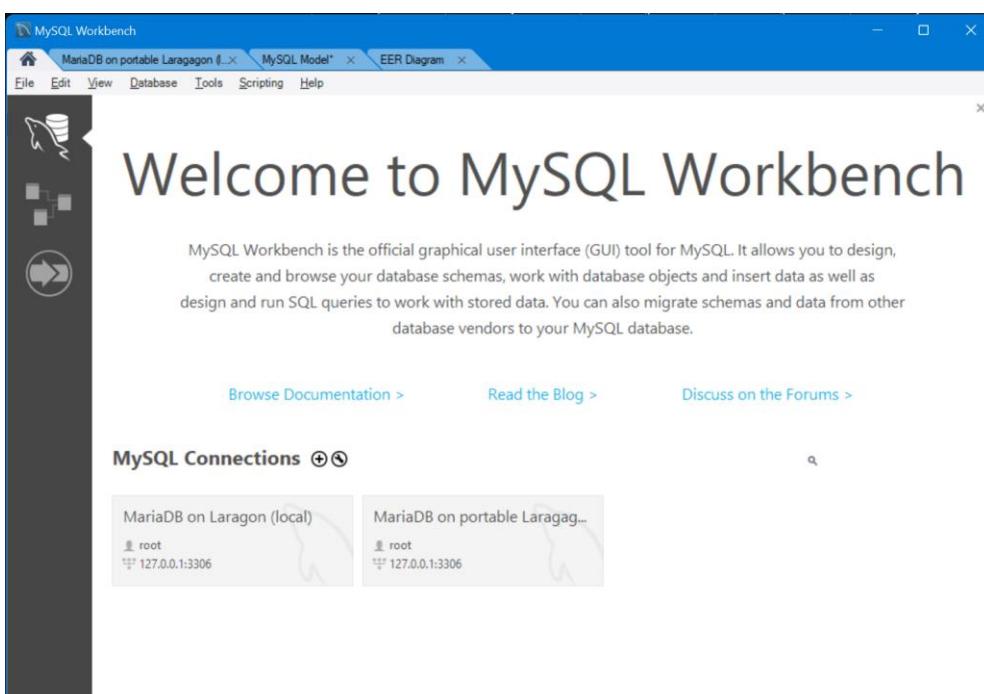


### 13.2.1.6. MySQL Workbench (strumento di sviluppo avanzato)

<https://www.mysql.com/products/workbench/>

<https://dev.mysql.com/downloads/workbench/>

MySQL WorkBench è un altro potente strumento grafico che consente di gestire i database MySQL e MariaDB. Per molti aspetti è simile a HeidiSQL. Rispetto a HeidiSQL permette di effettuare anche la progettazione del database con uno strumento grafico (EER) e permette di fare il forward e il reverse engineering di un database relazionale. Il forward engineering è la creazione del database fisico, con istruzioni SQL, generate automaticamente a partire dal modello grafico. Il reverse engineering è l'operazione opposta, ossia la creazione del modello grafico a partire dal database fisico.



A differenza di HeidiSQL per l'installazione c'è bisogno dei permessi amministrativi sul PC.

### **13.2.1.7. Cambio della password di root**

Dal terminale connettersi al DBMS come root e digitare i seguenti comandi:

<https://mariadb.com/kb/en/alter-user/>

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'root';  
FLUSH PRIVILEGES;
```

In generale è utile selezionare una password sicura e facile da ricordare. Nei prossimi esempi useremo una password standard (“root”) per gli esempi, che non è affatto sicura, ma che è comoda da gestire per gli esempi didattici.

### **13.2.1.8. Dove sono i dati?**

I dati (e le tabelle del database) sono salvate nella cartella **data** di MariaDb. Ad esempio, nel caso della versione portable di Laragon, tutti i database che sono creati all’interno di MariaDb sono salvati nella cartella: **laragon-portable\data\mariadb-10.6**

Se si cancella questa cartella verranno persi tutti i dati memorizzati nel DBMS.

### **13.2.1.9. MariaDb/MySQL con EF Core**

Per utilizzare MariaDb/MySQL con in framework EF Core bisogna installare i pacchetti:

```
Microsoft.EntityFrameworkCore.Design  
Microsoft.EntityFrameworkCore.Tools  
Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore  
Pomelo.EntityFrameworkCore.MySql
```

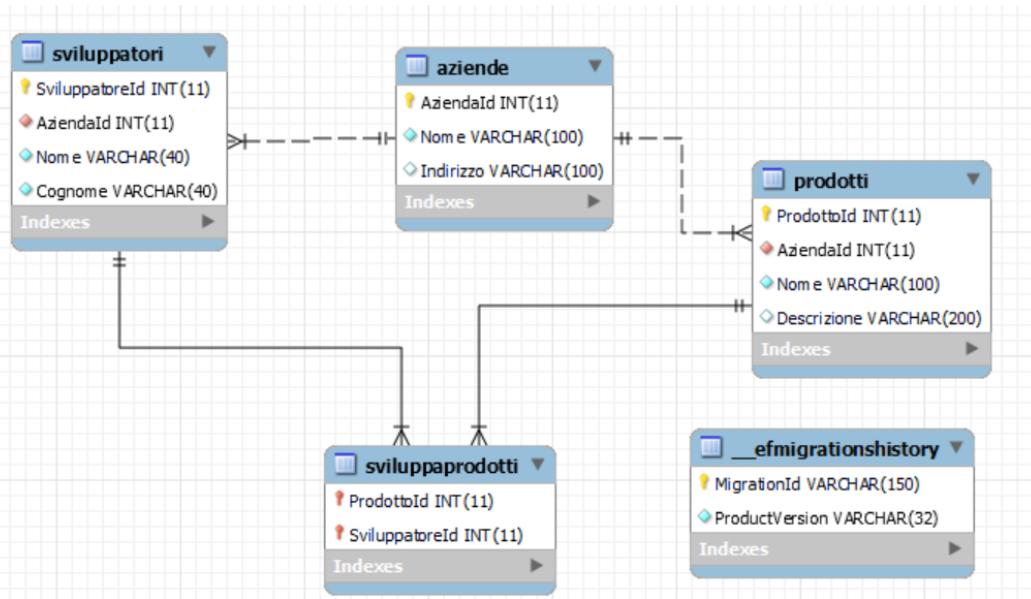
<https://github.com/PomeloFoundation/Pomelo.EntityFrameworkCore.MySql>

## **13.2.2. Un esempio di Minimal API con MariaDB: Azienda API**

Implementare un servizio di API per interfacciarsi ad un database di MariaDb (da creare mediante EF Core) con il seguente schema:

```
AZIENDA (AziendaId, Nome, Indirizzo)  
PRODOTTO (ProdottoId, AziendaId*, Nome, Descrizione)  
SVILUPPAPRODOTTO(ProdottoId*, SviluppatoreId*)  
SVILUPPATORE (SviluppatoreId,AziendaId*, Nome, Cognome)
```

Lo schema del database è graficamente rappresentato nella figura seguente:



Come si vede tra sviluppatori e prodotti c'è una tabella di collegamento che rappresenta un'associazione molti a molti, poiché uno sviluppatore può sviluppare più prodotti e uno stesso prodotto può essere sviluppato da più sviluppatori. Sia la tabella degli sviluppatori che quella dei prodotti hanno una chiave esterna sulla tabella delle aziende. Uno sviluppatore lavora presso una sola azienda, ma un'azienda può avere più sviluppatori. Allo stesso modo un prodotto appartiene ad una sola azienda, ma un'azienda può possedere più prodotti.

Sul database c'è un vincolo non esprimibile mediante la notazione del modello relazionale: uno sviluppatore può sviluppare solo prodotti che appartengono alla stessa azienda a cui afferisce lo sviluppatore. Quest'ultimo vincolo viene garantito dall'applicazione software: quando si inserisce una nuova riga nella tabella sviluppaprodotto si controlla che sia lo sviluppatore che il prodotto abbiano lo stesso valore della chiave esterna sulla tabella aziende.

Il servizio deve prevedere i seguenti endpoints:

//gestione aziende

- **GET /aziende**  
- restituisce la lista delle aziende;
- **POST /aziende**  
- per creare una nuova azienda;
- **GET /aziende/{aziendaId}**  
- restituisce i dati di una specifica azienda;
- **PUT /aziende/{aziendaId}**  
- modifica una specifica azienda;
- **DELETE /aziende/{aziendaId}**  
- elimina l'azienda specificata;

//gestione prodotti

- **GET /aziende/{aziendaId}/prodotti**  
- restituisce la lista dei prodotti dell'azienda specificata;

- **POST /aziende/{aziendaId}/prodotti**  
- per creare un prodotto appartenente all'azienda specificata;
- **GET /prodotti**  
- restituisce tutti i prodotti;
- **GET /prodotti/{prodottoId}**  
- per ottenere il prodotto specificato;
- **PUT /prodotti/{prodottoId}**  
- per modificare il prodotto specificato;
- **DELETE /prodotti/{prodottoId}**  
- per eliminare il prodotto specificato;

//gestione sviluppatori

- **GET /sviluppatori**  
- per ottenere la lista di tutti gli sviluppatori;
- **POST /aziende/{aziendaId}/sviluppatori**  
- per creare uno sviluppatore
- **GET /prodotti/{prodottoId}/sviluppatori**  
- per ottenere la lista degli sviluppatori di un determinato prodotto;
- **GET /aziende/{aziendaId}/sviluppatori**  
- per ottenere la lista degli sviluppatori di una determinata azienda;
- **GET /aziende/{aziendaId}/sviluppatori?prodottoId={prodottoId}**  
- per ottenere la lista degli sviluppatori di una determinata azienda che hanno lavorato a un determinato prodotto. Se non è presente la query string con la chiave prodottoid viene restituita la lista di tutti gli sviluppatori di una determinata azienda.
- **GET /sviluppatori/{sviluppatoreId}**  
- per ottenere i dati di uno sviluppatore
- **PUT /sviluppatori/{sviluppatoreId}**  
- per modificare uno sviluppatore
- **DELETE /sviluppatori/{sviluppatoreId}**  
- per eliminare uno sviluppatore. Elimina anche tutte le associazioni tra lo sviluppatore e i prodotti a cui ha lavorato (progetti)

//gestione progetti

- **POST /sviluppa-prodotto/{sviluppatoreId}/{prodottoId}**  
- per associare uno sviluppatore ad un prodotto. Il prodotto e lo sviluppatore devono esistere. Lo sviluppatore deve appartenere alla stessa azienda a cui appartiene il prodotto.
- **DELETE /sviluppa-prodotto/{sviluppatoreId}/{prodottoId}**  
- per eliminare l'associazione tra uno sviluppatore e un prodotto. Sviluppatore e prodotto devono esistere

### 13.2.2.1. Sviluppo del progetto

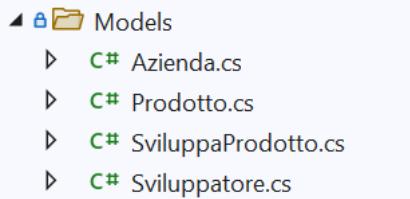
Per prima cosa creiamo un progetto ASP.NET Web API, usando il template di Visual Studio per le Minimal API (senza controllers). Installiamo i pacchetti:

- Swashbuckle.AspNetCore (dovrebbe essere già installato dal template)

- Microsoft.EntityFrameworkCore.Design
- Microsoft.EntityFrameworkCore.Tools
- Pomelo.EntityFrameworkCore.MySql
- Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore

### 13.2.2.1.1.Creazione del Model

Creiamo una cartella **Models** nella quale scriveremo le classi del modello necessario per la creazione del database. Per ogni tabella scriviamo una classe:



//file Azienda.cs

```
using System.ComponentModel.DataAnnotations.Schema;
namespace AziendaAPI.Models
{
    public class Azienda
    {
        public int AziendaId { get; set; }

        [Column(TypeName = "nvarchar(100)")]
        public string Nome { get; set; } = null!;

        [Column(TypeName = "nvarchar(100)")]
        public string? Indirizzo { get; set; }
        public ICollection<Prodotto> Prodotti { get; set; } = new HashSet<Prodotto>();
        public ICollection<Sviluppatore> Sviluppatori { get; set; } = new
        HashSet<Sviluppatore>();
    }
}
```

//file Prodotto.cs

```
using System.ComponentModel.DataAnnotations.Schema;
namespace AziendaAPI.Models
{
    public class Prodotto
    {
        public int ProdottoId { get; set; }
        public int AziendaId { get; set; }
        public Azienda Azienda { get; set; } = null!;

        [Column(TypeName = "nvarchar(100)")]
        public string Nome { get; set; } = null!;

        [Column(TypeName = "nvarchar(200)")]
        public string? Descrizione { get; set; }
        public ICollection<SviluppaProdotto> SviluppaProdotti { get; set; } = null!;
    }
}
```

```

//file Sviluppatore.cs

using System.ComponentModel.DataAnnotations.Schema;
namespace AziendaAPI.Models
{
    public class Sviluppatore
    {
        public int SviluppatoreId { get; set; }
        public int AziendaId { get; set; }
        public Azienda Azienda { get; set; } = null!;

        [Column(TypeName = "nvarchar(40)")]
        public string Nome { get; set; } = null!;

        [Column(TypeName = "nvarchar(40)")]
        public string Cognome { get; set; } = null!;
        public ICollection<SviluppaProdotto> SviluppaProdotti { get; set; } = null!;
    }
}

```

```

//file SviluppaProdotto.cs

namespace AziendaAPI.Models
{
    public class SviluppaProdotto
    {
        public int ProdottoId { get; set; }
        public Prodotto Prodotto { get; set; } = null!;
        public int SviluppatoreId { get; set; }
        public Sviluppatore Sviluppatore { get; set; } = null!;

    }
}

```

Creiamo la cartella **Data** con al suo interno il file AziendaDbContext.cs

```

using AziendaAPI.Models;
using Microsoft.EntityFrameworkCore;

namespace AziendaAPI.Data
{
    public class AziendaDbContext : DbContext
    {
        public AziendaDbContext(DbContextOptions<AziendaDbContext> options) : 
base(options) { }

        public DbSet<Azienda> Aziende => Set<Azienda>();
        public DbSet<Prodotto> Prodotti => Set<Prodotto>();
        public DbSet<Sviluppatore> Sviluppatori => Set<Sviluppatore>();
        public DbSet<SviluppaProdotto> SviluppaProdotti => Set<SviluppaProdotto>();

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            //per la gestione della Molti a molti "SviluppaProdotto"
            modelBuilder.Entity<SviluppaProdotto>()
                .HasKey(sp => new { sp.SviluppatoreId, sp.ProdottoId }); //definizione
di chiave primaria

            modelBuilder.Entity<SviluppaProdotto>()
                .HasOne(sp => sp.Sviluppatore)
                .WithMany(b => b.SviluppaProdotti)
        }
    }
}

```

```

        .HasForeignKey(sp => sp.SviluppatoreId); //definizione di chiave
        esterna su Sviluppatore

        modelBuilder.Entity<SviluppaProdotto>()
            .HasOne(sp => sp.Prodotto)
            .WithMany(c => c.SviluppaProdotti)
            .HasForeignKey(bc => bc.ProdottoId); //definizione di chiave esterna su
        Prodotto

        modelBuilder.Entity<Azienda>().HasData(
            new() {AziendaId=1, Nome = "Microsoft", Indirizzo = "One Microsoft
        Way, Redmond, WA 98052, Stati Uniti" },
            new() {AziendaId=2, Nome = "Google", Indirizzo = "1600 Amphitheatre
        Pkwy, Mountain View, CA 94043, Stati Uniti" },
            new() {AziendaId=3, Nome = "Apple", Indirizzo = "1 Apple Park Way
        Cupertino, California, 95014-0642 United States" }
        );
        modelBuilder.Entity<Prodotto>().HasData(
            new() {ProdottoId=1, Nome="SuperNote", Descrizione="Applicazione per
        la gestione delle Note", AziendaId=1},
            new() {ProdottoId=2, Nome = "My Cinema", Descrizione = "Applicazione
        per la visione di film in streaming", AziendaId=1 },
            new() {ProdottoId=3, Nome = "SuperCad", Descrizione = "Applicazione
        per il cad 3d", AziendaId=2 }
        );
        modelBuilder.Entity<Sviluppatore>().HasData(
            new() { SviluppatoreId = 1, Nome = "Mario", Cognome = "Rossi" ,
        AziendaId=1},
            new() { SviluppatoreId = 2, Nome = "Giulio", Cognome = "Verdi",
        AziendaId=1 },
            new() { SviluppatoreId = 3, Nome = "Leonardo", Cognome = "Bianchi",
        AziendaId=2 }
        );
        modelBuilder.Entity<SviluppaProdotto>().HasData(
            new() { SviluppatoreId = 1, ProdottoId=1 },
            new() { SviluppatoreId = 2, ProdottoId = 1 },
            new() { SviluppatoreId = 3, ProdottoId = 3 }
        );

    }
}
}

```

Si noti che il metodo OnModelCreating contiene la definizione della chiave primaria e delle chiavi esterne per la relazione molti a molti SviluppaProdotto che collega le tabelle Sviluppatori e Prodotti. Inoltre, sono state inseriti alcuni valori per le righe delle tabelle mediante il metodo HasData in modo che quando si effettua la migrazione del database, questo venga automaticamente popolato con alcuni dati iniziali, evitando così di dover inserirli manualmente ogni volta che si ricrea il database.

### 13.2.2.1.2. Collegamento dell'applicazione con il DBMS

Prima di poter effettuare la migrazione occorre:

1. configurare il servizio relativo al DBMS MariaDb (intervenendo sul file appsettings.json e nel codice di Program.cs)
2. assicurarsi che il server di MariaDB sia raggiungibile dalla nostra applicazione di Visual Studio (il server di MariaDb deve essere attivo e raggiungibile da Visual Studio)

Per la configurazione del servizio DBMS si può procedere con la descrizione dei servizi e della

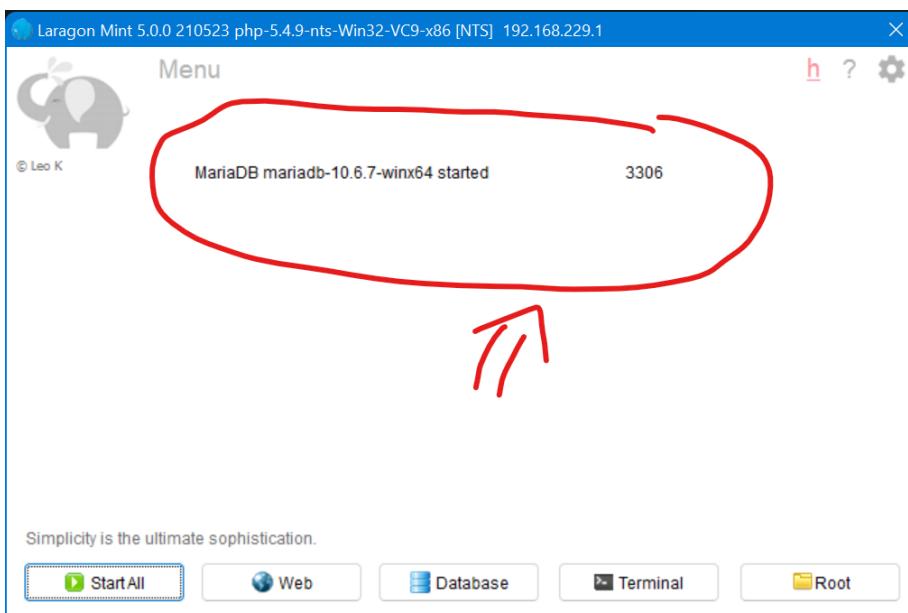
pipeline dell'app nel file Program.cs, in maniera del tutto analoga a quanto visto negli esempi precedenti. Per prima cosa bisogna gestire la connessione con il database MariaDb, introducendo nel file **appsettings.json** la stringa di connessione al database:

```
{  
  "ConnectionStrings": {  
    "AziendaAPIConnection": "Server=localhost;port=3306;Database=aziendaapi;User  
Id=root;Password=root;"  
  },  
  "Logging": {  
    "LogLevel": {  
      "Default": "Information",  
      "Microsoft.AspNetCore": "Warning"  
    }  
  },  
  "AllowedHosts": "*"  
}
```

In questo caso l'accesso è effettuato con le credenziali di root, ma vedremo in seguito che è buona norma creare un account specifico per l'app con permessi limitati.

Il database che sarà creato si chiama **aziendaapi** e sarà gestito dal DBMS MariaDb in ascolto sulla porta 3306 in localhost.

Nota: il DBMS deve essere stato avviato e deve essere in ascolto sulla porta 3306 in localhost:



Si noti che nel caso di un'applicazione reale potrebbe succedere che il database non sia installato in localhost, ma sia su un'altra macchina nel cloud. In questo caso occorre inserire nella stringa di connessione al posto di localhost, l'indirizzo IP, oppure il DNS **fully qualified name** della macchina server che ospita il DBMS e usare come porta, quella sulla quale è in ascolto il DBMS.

Per permettere all'applicazione web di interagire con il database MariaDb occorre associare un servizio all'app che consenta di interfacciarsi con il DBMS. In questo esempio useremo il provider **Pomelo.EntityFrameworkCore.MySql** che permette di usare EF Core sia con MySQL che con MariaDb.

Nel file Program.cs scriveremo il codice (dopo la creazione dell'oggetto **builder** e prima della creazione dell'oggetto **app**):

```
var connectionString =  
builder.Configuration.GetConnectionString("AziendaAPIConnection");  
var serverVersion = ServerVersion.AutoDetect(connectionString);  
builder.Services.AddDbContext<AziendaDbContext>(
```

```

    dbContextOptions => dbContextOptions
        .UseMySql(connectionString, serverVersion)
        // The following three options help with debugging, but should
        // be changed or removed for production.
        .LogTo(Console.WriteLine, LogLevel.Information)
        .EnableSensitiveDataLogging()
        .EnableDetailedErrors()
);

```

Questo codice consente di iniettare nella pipeline dell'app il servizio che ci permetterà di interfacciarsi con MariaDb.

### 13.2.2.1.3. Migration

Effettuiamo la migrazione con i comandi della Packet Manager Console di Visual Studio:

Add-Migration InitialMigrate

Package Manager Console

Package source: All Default project: AziendaAPI

Each package is licensed to you by its owner. NuGet is not responsible for any dependencies.

Package Manager Console Host Version 6.1.0.106

Type 'get-help NuGet' to see all available NuGet commands.

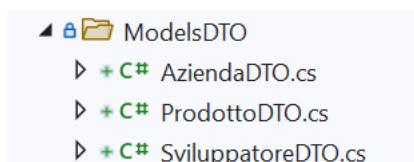
PM> Add-Migration InitialMigrate|

In questo esempio non useremo l'istruzione `Update-Database`, perché vedremo che è possibile far eseguire la migrazione del database direttamente dal codice dell'applicazione.

### 13.2.2.1.4. View Model per i dati (Data Transfer Object – DTO)

Prima di procedere con la scrittura del codice dell'applicazione web scriviamo una View per il modello dei dati che le API esporranno alle applicazioni client. Infatti il modello dei dati utilizzato nella nostra applicazione contiene una serie di componenti che sono funzionali al modo di gestire i dati mediante l'ORM EF Core, ma che non devono essere esposte alle applicazioni client. Ad esempio, le componenti di tipo reference, usate per le chiavi esterne, oppure le collections introdotte per creare le cosiddette navigation properties non dovrebbero comparire negli oggetti restituiti dalle Web API. Tutti questi dettagli devono essere filtrati alle applicazioni client. Per fare questo utilizzeremo il concetto di Data Transfer Object, già illustrato negli esempi precedenti.

Creiamo la cartella `ModelsDTO` con dentro le versioni DTO dei dati del modello:



//file AziendaDTO.cs

```

using AziendaAPI.Models;

namespace AziendaAPI.ModelsDTO
{
    public class AziendaDTO
    {
        public int AziendaId { get; set; }

        public string Nome { get; set; } = null!;

        public string? Indirizzo { get; set; }
        public AziendaDTO() { }
        public AziendaDTO(Azienda azienda) =>
            (AziendaId, Nome, Indirizzo) = (azienda.AziendaId, azienda.Nome,
azienda.Indirizzo);
    }
}

```

//file ProdottoDTO.cs

```

using AziendaAPI.Models;

namespace AziendaAPI.ModelsDTO
{
    public class ProdottoDTO
    {
        public int ProdottoId { get; set; }
        public int AziendaId { get; set; }
        public string Nome { get; set; } = null!;
        public string? Descrizione { get; set; }
        public ProdottoDTO() { }
        public ProdottoDTO(Prodotto prodotto) =>
            (ProdottoId, AziendaId, Nome, Descrizione) = (prodotto.ProdottoId,
prodotto.AziendaId, prodotto.Nome, prodotto.Descrizione);

    }
}

```

//file SviluppatoreDTO.cs

```

using AziendaAPI.Models;

namespace AziendaAPI.ModelsDTO
{
    public class SviluppatoreDTO
    {
        public int SviluppatoreId { get; set; }
        public int AziendaId { get; set; }
        public string Nome { get; set; } = null!;
        public string Cognome { get; set; } = null!;
        public SviluppatoreDTO() { }
        public SviluppatoreDTO(Sviluppatore sviluppatore) =>
            (SviluppatoreId, AziendaId, Nome, Cognome) = (sviluppatore.SviluppatoreId,
sviluppatore.AziendaId, sviluppatore.Nome, sviluppatore.Cognome);
    }
}

```

Si noti che non è stata creata una versione DTO della classe SviluppaProdotto perché le API non restituiscono valori di questa classe, che serve solo per descrivere i collegamenti tra Sviluppatori e Prodotti.

### 13.2.2.1.5. Migrazione con Code First Approach (solo per sviluppo/testing)

Per poter creare il database direttamente dal codice occorre che:

- 1) il DBMS sia in ascolto all'indirizzo e porta specificati nella stringa di connessione
- 2) iniettare un servizio nell'app che effettui la migrazione se richiesto

```
//questa parte va messa dopo aver creato l'app a partire dal builder
//https://stackoverflow.com/questions/42355481/auto-create-database-in-entity-
framework-core
//https://stackoverflow.com/a/55232983
//crea il database se non esiste a partire dalla migrazione (che deve esistere)
//se non è stata fatta la migrazione il database non verrà creato
using (var serviceScope = app.Services.CreateScope())
{
    var context =
serviceScope.ServiceProvider.GetRequiredService<AziendaDbContext>();
    context.Database.Migrate();
    //vedere IN ALTERNATIVA a Migrate anche
    //context.Database.EnsureCreated();
}
```

L'istruzione `context.Database.Migrate()` effettua in automatico la migrazione direttamente quando si manda in esecuzione l'app: se il database non esiste viene creato ed eventualmente anche popolato con i dati che sono stati inseriti nei file di migration. Se il database esiste già non verrà eseguita alcuna azione.

### 13.2.2.1.6. Configurazione di Swagger con il View Model

Per evitare che Swagger riporti come schema dei dati restituiti dagli endpoints delle API i modelli DTO è possibile istruire il servizio in modo tale da togliere il suffisso DTO ai nomi delle classi esposte. In questo modo le applicazioni client vedranno la descrizione dei tipi senza il suffisso DTO:

```
// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at
// https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "Azienda API",
        Description = "Manage your company assets via API",
        Version = "v1"
    });
    //https://stackoverflow.com/questions/40644052/ rename-model-in-swashbuckle-6-
    //swagger-with-asp-net-core-web-api
    //https://stackoverflow.com/a/40684060
    c.CustomSchemaIds(schemaIdStrategy);
    static string schemaIdStrategy(Type currentClass)
    {
        string returnedValue = currentClass.Name;
        if (returnedValue.EndsWith("DTO"))
            returnedValue = returnedValue.Replace("DTO", string.Empty);
        return returnedValue;
    }
});
```

### 13.2.2.1.7. Gestione di tanti endpoints

In questo esempio ci sono molte rotte e per ognuna di queste occorre definire del codice che descriva l'azione da compiere. In questo caso il file Program.cs diventerebbe molto grande con moltissime righe di codice. Per evitare di dover gestire tutto il codice all'interno del file Program.cs useremo una tecnica basata sulla scrittura di metodi di estensione. Questo non è il metodo migliore in assoluto, ma è sicuramente il più semplice. Ad esempio, esistono altri meccanismi più complessi basati sulla reflection, che permettono di ridurre ulteriormente il codice inserito nel file Program.cs, ma richiedono alcune competenze avanzate di programmazione relative alla reflection delle classi.

### 13.2.2.1.8. Il file Program.cs

```
//file Program.cs

using Microsoft.OpenApi.Models;
using AziendaAPI.Data;
using Microsoft.EntityFrameworkCore;
using AziendaAPI.EndPoints;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at
// https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "Azienda API",
        Description = "Manage your company assets via API",
        Version = "v1"
    });
    //https://stackoverflow.com/questions/40644052/rename-model-in-swashbuckle-6-
    //swagger-with-asp-net-core-web-api
    //https://stackoverflow.com/a/40684060
    c.CustomSchemaIds(schemaIdStrategy);
    static string schemaIdStrategy(Type currentClass)
    {
        string returnedValue = currentClass.Name;
        if (returnedValue.EndsWith("DTO"))
            returnedValue = returnedValue.Replace("DTO", string.Empty);
        return returnedValue;
    }
});

var connectionString =
builder.Configuration.GetConnectionString("AziendaAPIConnection");
var serverVersion = ServerVersion.AutoDetect(connectionString);
builder.Services.AddDbContext<AziendaDbContext>(
    dbContextOptions => dbContextOptions
        .UseMySql(connectionString, serverVersion)
        // The following three options help with debugging, but should
        // be changed or removed for production.
        .LogTo(Console.WriteLine, LogLevel.Information)
        .EnableSensitiveDataLogging()
        .EnableDetailedErrors()
);
//builder.Services.add
builder.Services.AddDatabaseDeveloperPageExceptionFilter();

var app = builder.Build();

// Configure the HTTP request pipeline.
```

```

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "Azienda API V1");
    });
}

app.UseHttpsRedirection();
//I metodi seguenti verranno implementati in classi di estensione (descritte più avanti)
app.MapAziendaEndpoints();
app.MapProdottoEndpoints();
app.MapSviluppatoreEndpoints();
app.MapSviluppaProdottoEndpoints();

//https://stackoverflow.com/questions/42355481/auto-create-database-in-entity-framework-core
//https://stackoverflow.com/a/55232983
//crea il database se non esiste a partire dalla migrazione (che deve esistere)
//se non è stata fatta la migrazione il database non verrà creato
using (var serviceScope = app.Services.CreateScope())
{
    var context = serviceScope.ServiceProvider.GetRequiredService<AziendaDbContext>();
    context.Database.Migrate();
    //vedere IN ALTERNATIVA a Migrate anche
    //context.Database.EnsureCreated();
}
//Run dell'applicazione
app.Run();

```

### 13.2.2.1.9. Gestione degli Endpoints (Handlers)

Creiamo la cartella **EndPoints** con al suo interno i file:

```

//file AziendaEndPoints.cs

using AziendaAPI.Data;
using AziendaAPI.Models;
using AziendaAPI.ModelsDTO;
using Microsoft.EntityFrameworkCore;

namespace AziendaAPI.EndPoints
{
    //gestione aziende
    public static class AziendaEndPoints
    {
        public static void MapAziendaEndpoints(this WebApplication app)
        {
            app.MapGet("/aziende", async (AziendaDbContext db) => Results.Ok(await db.Aziende.Select(x => new AziendaDTO(x)).ToListAsync()));

            app.MapPost("/aziende", async (AziendaDbContext db, AziendaDTO aziendaDTO) =>
            {
                var azienda = new Azienda { Nome = aziendaDTO.Nome, Indirizzo = aziendaDTO.Indirizzo };
                await db.Aziende.AddAsync(azienda);
                await db.SaveChangesAsync();
                return Results.Created($"/aziende/{azienda.AziendaId}", new AziendaDTO(azienda));
            });
        }
    }
}

```

```

        app.MapGet("/aziende/{aziendaId}", async (AziendaDbContext db, int aziendaId) =>
            await db.Aziende.FindAsync(aziendaId)
            is Azienda azienda
                ? Results.Ok(new AziendaDTO(azienda))
                : Results.NotFound()
        );

        app.MapPut("/aziende/{aziendaId}", async (AziendaDbContext db, AziendaDTO updateAzienda, int aziendaId) =>
        {
            var azienda = await db.Aziende.FindAsync(aziendaId);
            if (azienda is null) return Results.NotFound();
            azienda.Nome = updateAzienda.Nome;
            azienda.Indirizzo = updateAzienda.Indirizzo;
            await db.SaveChangesAsync();
            return Results.NoContent();
        });
        app.MapDelete("/azienda/{aziendaId}", async (AziendaDbContext db, int aziendaId) =>
        {
            var azienda = await db.Aziende.FindAsync(aziendaId);
            if (azienda is null)
            {
                return Results.NotFound();
            }
            db.Aziende.Remove(azienda);
            await db.SaveChangesAsync();
            return Results.Ok();
        });
    });
}

```

//file ProdottoEndpoints.cs

```

using AziendaAPI.Data;
using AziendaAPI.Models;
using AziendaAPI.ModelsDTO;
using Microsoft.EntityFrameworkCore;

namespace AziendaAPI.EndPoints
{
    //gestione prodotti
    public static class ProdottoEndPoints
    {
        public static void MapProdottoEndpoints(this WebApplication app)
        {

            app.MapGet("/aziende/{aziendaId}/prodotti", async (AziendaDbContext db, int aziendaId) =>
            {
                Azienda? azienda = await db.Aziende.Where(a => a.AziendaId == aziendaId).Include(a => a.Prodotti).FirstOrDefaultAsync();
                if (azienda != null)
                {
                    return Results.Ok(azienda.Prodotti.Select(p => new ProdottoDTO(p)).ToList());
                }
                else
                {
                    return Results.NotFound();
                }
            });
        }
    }
}

```

```

        }
    });

    app.MapPost("/aziende/{aziendaId}/prodotti", async (AziendaDbContext db,
int aziendaId, ProdottoDTO prodottoDTO) =>
{
    Azienda? azienda = await db.Aziende.FindAsync(aziendaId);
    if (azienda != null)
    {
        Prodotto prodotto = new () { Nome = prodottoDTO.Nome, Descrizione =
prodottoDTO.Descrizione, AziendaId=azienda.AziendaId };
        await db.Prodotti.AddAsync(prodotto);
        await db.SaveChangesAsync();
        return
Results.Created($"/aziende/{aziendaId}/prodotti/{prodotto.ProdottoId}", new
ProdottoDTO(prodotto));
    }
    else
    {
        return Results.NotFound();
    }
});

app.MapGet("/prodotti", async (AziendaDbContext db) => Results.Ok(await
db.Prodotti.Select(x => new ProdottoDTO(x)).ToListAsync()));

app.MapGet("/prodotti/{prodottoId}", async (AziendaDbContext db, int
prodottoId) =>
{
    Prodotto? prodotto= await db.Prodotti.Where(p=>p.ProdottoId ==
prodottoId).FirstOrDefaultAsync();
    if(prodotto != null)
    {
        return Results.Ok(new ProdottoDTO(prodotto));
    }
    else
    {
        return Results.NotFound();
    }
});

app.MapPut("/prodotti/{prodottoId}", async (AziendaDbContext db,
ProdottoDTO updateProdotto, int prodottoId) =>
{
    Prodotto? prodotto = await db.Prodotti.FindAsync(prodottoId);
    if (prodotto is null) return Results.NotFound();
    prodotto.Nome = updateProdotto.Nome;
    prodotto.Descrizione = updateProdotto.Descrizione;
    await db.SaveChangesAsync();
    return Results.NoContent();
});

app.MapDelete("/prodotti/{prodottoId}", async (AziendaDbContext db, int
prodottoId) =>
{
    var prodotto = await db.Prodotti.FindAsync(prodottoId);
    if (prodotto is null)
    {
        return Results.NotFound();
    }
    db.Prodotti.Remove(prodotto);
    await db.SaveChangesAsync();
    return Results.Ok();
});

```

```

        });
    }
}

//file SviluppatoreEndPoints.cs

using AziendaAPI.Data;
using AziendaAPI.Models;
using AziendaAPI.ModelsDTO;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace AziendaAPI.EndPoints
{
    public static class SviluppatoreEndPoints
    {
        public static void MapSviluppatoreEndpoints(this WebApplication app)
        {
            app.MapGet("/sviluppatori", async (AziendaDbContext db) =>
Results.Ok(await db.Sviluppatori.Select(x => new SviluppatoreDTO(x)).ToListAsync()));

            app.MapPost("/aziende/{aziendaId}/sviluppatori", async (AziendaDbContext db, int aziendaId, SviluppatoreDTO sviluppatoreDTO) =>
{
                Azienda? azienda = await db.Aziende.FindAsync(aziendaId);
                if (azienda != null)
                {
                    Sviluppatore sviluppatore = new() { Nome = sviluppatoreDTO.Nome,
Cognome = sviluppatoreDTO.Cognome, AziendaId = azienda.AziendaId };
                    await db.Sviluppatori.AddAsync(sviluppatore);
                    await db.SaveChangesAsync();
                    return
Results.Created($"/aziende/{aziendaId}/sviluppatori/{sviluppatore.SviluppatoreId}", new SviluppatoreDTO(sviluppatore));
                }
                else
                {
                    return Results.NotFound();
                }
            });

            app.MapGet("/prodotti/{prodottoId}/sviluppatori", async (AziendaDbContext db, int prodottoId) =>
{
                Prodotto? prodotto = await db.Prodotti.
                    Where(p => p.ProdottoId == prodottoId).
                    Include(p => p.SviluppaProdotti).
                    ThenInclude(s=> s.Sviluppatore).
                    FirstOrDefaultAsync();
                if (prodotto != null)
                {
                    return Results.Ok(prodotto.SviluppaProdotti.Select(x => new SviluppatoreDTO(x.Sviluppatore)).ToList());
                }
                else
                {
                    return Results.NotFound();
                }
            });

            app.MapGet("/aziende/{aziendaId}/sviluppatori",
                async (AziendaDbContext db, int aziendaId, [FromQuery(Name =
"prodottoId")] int? prodottoId) =>

```

```

    {
        if (prodottoId == null)
        {
            Azienda? azienda = await db.Aziende.
                Where(a => a.AziendaId == aziendaId).
                Include(a => a.Sviluppatori).
                FirstOrDefaultAsync();

            if (azienda != null)
            {
                return Results.Ok(azienda.Sviluppatori.Select(x => new
SviluppatoreDTO(x)).ToList());
            }
            else
            {
                return Results.NotFound();
            }
        }
        else //prodottoId!=null
        {
            Prodotto? prodotto = await db.Prodotti.FindAsync(prodottoId);
            if (prodotto == null) { return Results.NotFound(); }
            //effettuo la join tra Sviluppatori e SviluppaProdotti per
ottenere gli sviluppatori che
            //hanno partecipato allo sviluppo di un determinato prodotto
            List<Sviluppatore> listaSviluppatori =
                await db.Sviluppatori.Where(s => s.AziendaId == aziendaId).
                Join(db.SviluppaProdotti.Where(sp => sp.ProdottoId ==
prodottoId),
                    s => s.SviluppatoreId,
                    sp => sp.SviluppatoreId,
                    (s, sp) => s).ToListAsync();
            return Results.Ok(listaSviluppatori.Select(s => new
SviluppatoreDTO(s)));
        }
    });
}

app.MapGet("/sviluppatori/{sviluppatoreId}", async (AziendaDbContext db,
int sviluppatoreId) =>
{
    Sviluppatore? sviluppatore = await
db.Sviluppatori.FindAsync(sviluppatoreId);
    if (sviluppatore != null)
    {
        return Results.Ok(new SviluppatoreDTO(sviluppatore));
    }
    else
    {
        return Results.NotFound();
    }
});

app.MapPut("/sviluppatori/{sviluppatoreId}", async (AziendaDbContext db,
SviluppatoreDTO updateSviluppatore, int sviluppatoreId) =>
{
    Sviluppatore? sviluppatore = await
db.Sviluppatori.FindAsync(sviluppatoreId);
    if (sviluppatore is null) return Results.NotFound();
    sviluppatore.Nome = updateSviluppatore.Nome;
}

```

```
        sviluppatore.Cognome = updateSviluppatore.Cognome;
        await db.SaveChangesAsync();
        return Results.NoContent();
    });

    app.MapDelete("/sviluppatori/{sviluppatoreId}", async (AziendaDbContext
db, int sviluppatoreId) =>
{
    var sviluppatore = await db.Sviluppatori.FindAsync(sviluppatoreId);
    if (sviluppatore == null)
    {
        return Results.NotFound();
    }
    //elimino tutte le associazioni tra lo sviluppatore e i prodotti a cui
ha lavorato
    var righeDaRimuovere =await
db.SviluppaProdotti.Where(sp=>sp.SviluppatoreId==sviluppatoreId).ToListAsync();
    if (righeDaRimuovere!=null && righeDaRimuovere.Count > 0)
    {
        db.SviluppaProdotti.RemoveRange(righeDaRimuovere);
        await db.SaveChangesAsync();
    }
    //elimino lo sviluppatore
    db.Sviluppatori.Remove(sviluppatore);
    await db.SaveChangesAsync();
    return Results.Ok();
});
});
```

```
//file SviluppaProdottoEndPoints.cs

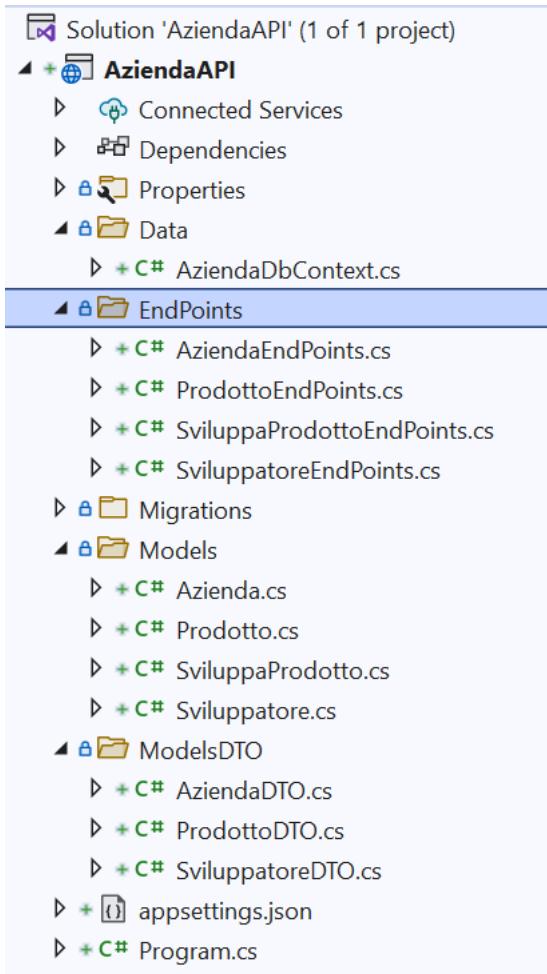
using AziendaAPI.Data;
using AziendaAPI.Models;
using AziendaAPI.ModelsDTO;
using Microsoft.EntityFrameworkCore;

namespace AziendaAPI.EndPoints
{
    public static class SviluppaProdottoEndPoints
    {
        public static void MapSviluppaProdottoEndpoints(this WebApplication app)
        {
            app.MapPost("/sviluppa-prodotto/{sviluppatoreId}/{prodottoId}",
                async (AziendaDbContext db, int sviluppatoreId, int prodottoId) =>
            {
                Prodotto? prodotto = await db.Prodotti.FindAsync(prodottoId);
                Sviluppatore? sviluppatore = await
db.Sviluppatori.FindAsync(sviluppatoreId);

                //controllo che l'associazione non sia già stata creata
                SviluppaProdotto? rigaInTabella =await db.SviluppaProdotti.
                    Where(sp=> sp.SviluppatoreId==sviluppatoreId &&
sp.ProdottoId==prodottoId).
                    FirstOrDefaultAsync();
                //la riga esiste già - nessuna azione da effettuare
                if (rigaInTabella != null)
                {

                    return Results.NoContent();
                }
                //la riga non esiste e ci sono prodotto e sviluppatore nelle
                rispettive tabelle
            });
        }
    }
}
```

La struttura del progetto dovrebbe essere come quella riportata di seguito:



## 13.2.2.1.10. Testing dell'app

The screenshot shows the Swagger UI interface running in a Microsoft Edge browser. The title bar indicates the page is at `https://localhost:7088/swagger/index.html`. The main content area is titled "Azienda API" and "OAS". Below the title, it says "Manage your company assets via API". The interface lists various API endpoints categorized under "AziendaAPI". Each endpoint is shown with its method (e.g., GET, POST, PUT, DELETE), path, and a dropdown arrow to expand the details. The endpoints are color-coded by category: blue for /aziende, green for /aziende/{aziendaId}, orange for /prodotti, red for /sviluppatori, and light blue for /sviluppa-prodotto. The "Schemas" section at the bottom contains a single entry: "Azienda".

Method	Path	Category
GET	/aziende	Azienda
POST	/aziende	Azienda
GET	/aziende/{aziendaId}	Azienda
PUT	/aziende/{aziendaId}	Azienda
DELETE	/azienda/{aziendaId}	Azienda
GET	/aziende/{aziendaId}/prodotti	Azienda
POST	/aziende/{aziendaId}/prodotti	Azienda
GET	/prodotti	Prodotti
GET	/prodotti/{prodottoId}	Prodotti
PUT	/prodotti/{prodottoId}	Prodotti
DELETE	/prodotti/{prodottoId}	Prodotti
GET	/sviluppatori	Sviluppatori
POST	/aziende/{aziendaId}/sviluppatori	Sviluppatori
GET	/aziende/{aziendaId}/sviluppatori	Sviluppatori
GET	/prodotti/{prodottoId}/sviluppatori	Sviluppatori
GET	/sviluppatori/{sviluppatoreId}	Sviluppatori
PUT	/sviluppatori/{sviluppatoreId}	Sviluppatori
DELETE	/sviluppatori/{sviluppatoreId}	Sviluppatori
POST	/sviluppa-prodotto/{sviluppatoreId}/{prodottoId}	Sviluppa Prodotto
DELETE	/sviluppa-prodotto/{sviluppatoreId}/{prodottoId}	Sviluppa Prodotto

**AziendaAPI**

**GET /aziende**

Parameters

No parameters

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7088/aziende' \
  -H 'accept: */*'
```

Request URL

```
https://localhost:7088/aziende
```

Server response

Code Details

200 Response body

```
[{"aziendaId": 1, "name": "Microsoft", "indirizzo": "One Microsoft Way, Redmond, WA 98052, Stati Uniti"}, {"aziendaId": 2, "name": "Google", "indirizzo": "1600 Amphitheatre Pkwy, Mountain View, CA 94043, Stati Uniti"}, {"aziendaId": 3, "name": "Apple", "indirizzo": "1 Apple Park Way Cupertino, California, 95014-0642 United States"}]
```

Response headers

```
content-type: application/json; charset=utf-8
date: Fri, 01 Apr 2022 07:14:58 GMT
server: Kestrel
```

Responses

Code	Description	Links
200	Success	No links

**POST /aziende**

**GET /aziende/{aziendaId}**

Aziendaid	Nome	Indirizzo
1	Microsoft	One Microsoft Way, Redmond, WA 98052, Stati Uniti
2	Google	1600 Amphitheatre Pkwy, Mountain View, CA 94043, Stati Uniti
3	Apple	1 Apple Park Way Cupertino, California, 95014-0642 Stati Uniti

 Below the table, the SQL query history shows various SHOW and SELECT statements for the 'aziendaapi' database."/>


Aziendaid	Nome	Indirizzo
1	Microsoft	One Microsoft Way, Redmond, WA 98052, Stati Uniti
2	Google	1600 Amphitheatre Pkwy, Mountain View, CA 94043, Stati Uniti
3	Apple	1 Apple Park Way Cupertino, California, 95014-0642 Stati Uniti

 Below the table, the SQL query history shows various SHOW and SELECT statements for the 'aziendaapi' database."/>

### 13.2.3. Gestione degli account per MariaDb

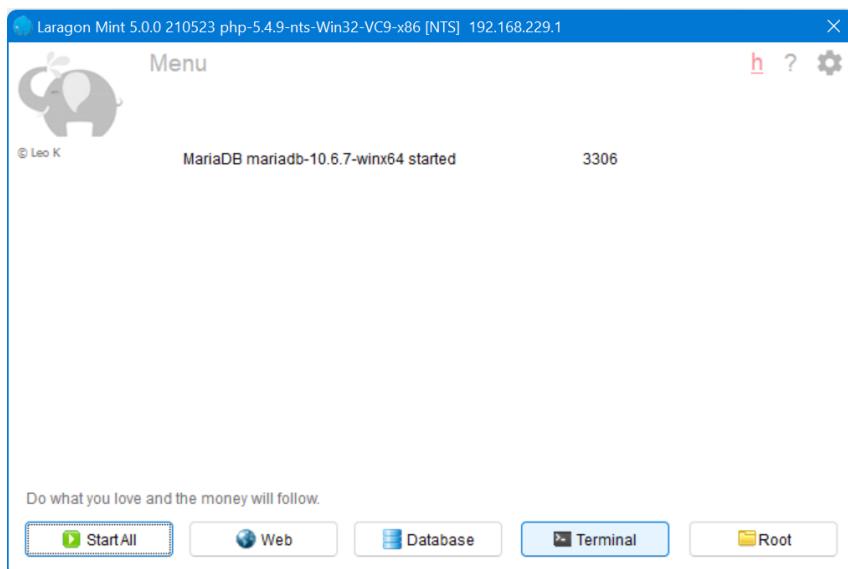
Sebbene sia possibile eseguire tutte le attività su tutti i database gestiti da MariaDB con l’account root, ciò non è opportuno dal punto di vista della sicurezza. Infatti, se un utente malevolo riuscisse a sfruttare una vulnerabilità dell’applicazione con i privilegi di root potrebbe eseguire qualsiasi attività, compresa la cancellazione o modifica di tutti i dati, su tutti i database gestiti dal DBMS. Di solito si creano account limitati per singola applicazione. Ciascun account viene utilizzato per effettuare la

connessione al database dall'applicazione web e ha i **permessi minimi**, necessari per il corretto funzionamento dell'app. Si applica il cosiddetto “principio del minimo privilegio”: un'applicazione deve avere i permessi strettamente necessari per il suo funzionamento, nulla di più.

### 13.2.3.1. Gestione degli account (diversi da root) per MariaDb

Per creare un account in MariaDb si può utilizzare la console a riga di comando (mysql monitor), oppure applicazioni con interfaccia grafica che permettano di gestire gli account (HeidiSQL, MySQL Workbench, etc.). L'interfaccia da riga di comando è disponibile in tutti gli ambienti ed è quella che verrà illustrata di seguito.

Dopo esserci accertati che il database MariaDb sia stato attivato, apriamo la console a riga di comando (terminal):



Se il percorso di mysql.exe è già registrato nelle variabili d'ambiente dell'utente o di sistema basterà digitare il comando:

```
mysql -u root -p
```

invio

la password di root

Se il percorso di mysql.exe non è nelle variabili d'ambiente basta aprire la console nella directory dove si trova il programma mysql.exe. Ad esempio, nel caso di Laragon portable con la versione di MariaDb installata:

```
laragon-portable\bin\mysql\mariadb-10.6.7-winx64\bin
```

A screenshot of a Windows command prompt window titled "C:\Windows\System32\cmd.exe - mysql -u root -p". The window shows the following text:

```
C:\Windows\System32\cmd.exe - mysql -u root -p
Microsoft Windows [Version 10.0.22000.593]
(c) Microsoft Corporation. All rights reserved.

C:\Users\genna\Documents\ScuolaLocal\laragon-portable\bin\mysql\mariadb-10.6.7-winx64\bin>mysql -u root -p
Enter password: *****
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 10.6.7-MariaDB-log mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

### 13.2.3.1.1.Gli account attivi sul DBMS

Per vedere gli account attivi ci sono diverse opzioni:

```
select User, Host, Password from mysql.user;
```

```
MariaDB [(none)]> select User, Host, Password from mysql.user;
+-----+-----+-----+
| User | Host | Password |
+-----+-----+-----+
| mariadb.sys | localhost |          |
| root | localhost | *81F5E21E35407D884A6CD4A731AEBFB6AF209E1B |
| root | malafronter-leg5 |          |
| root | 127.0.0.1 |          |
| root | ::1 |          |
+-----+-----+-----+
5 rows in set (0.001 sec)
```

Oppure:

```
select User, Host, JSON_DETAILED(Priv) from mysql.global_priv \G
```

```
MariaDB [(none)]> select User, Host, JSON_DETAILED(Priv) from mysql.global_priv \G
***** 1. row *****
      User: mariadb.sys
      Host: localhost
JSON_DETAILED(Priv): {
  "access": 0,
  "plugin": "mysql_native_password",
  "authentication_string": "",
  "account_locked": true,
  "password_last_changed": 0
}
***** 2. row *****
      User: root
      Host: localhost
JSON_DETAILED(Priv): {
  "access": 549755813887,
  "version_id": 100607,
  "plugin": "mysql_native_password",
  "authentication_string": "*81F5E21E35407D884A6CD4A731AEBFB6AF209E1B",
  "password_last_changed": 1648286753
}
***** 3. row *****
```

Oppure:

```
select User, Host, json_extract(Priv, '$.authentication_string') as password from mysql.global_priv \G
```

```
MariaDB [(none)]> select User, Host, json_extract(Priv, '$.authentication_string') as password from mysql.global_priv \G
***** 1. row *****
      User: mariadb.sys
      Host: localhost
password: ""
***** 2. row *****
      User: root
      Host: localhost
password: "*81F5E21E35407D884A6CD4A731AEBFB6AF209E1B"
***** 3. row *****
```

[https://mariadb.com/kb/en/json\\_extract/](https://mariadb.com/kb/en/json_extract/)

<https://mariadb.com/kb/en/jsonpath-expressions/>

Oppure da HeidiSQL:

The screenshot shows two windows from the HeidiSQL application. The top window displays the 'global\_priv' table in the 'mysql' database. The bottom window shows the 'User manager - root' interface.

**Top Window (HeidiSQL):**

- Table View:** Shows the 'global\_priv' table with the following data:

Host	User	Priv
localhost	mariadb.sys	{"access":0,"plugin":"mysql_native_password","auth...}
localhost	root	{"access":549755813887,"version_id":100607,"plugin"...}
malafronte-leg5	root	{"access":18446744073709551615}
127.0.0.1	root	{"access":18446744073709551615}
::1	root	{"access":18446744073709551615}

- Database Filter:** Shows the structure of the 'mysql' database, including tables like columns\_priv, column\_stats, db, event, func, general\_log, and global\_priv.

**Bottom Window (User manager - root):**

- User Accounts:** A list of user accounts:
 

Username	Host
mariadb.sys	localhost
root	localhost
root	malafronte-leg5
root	127.0.0.1
root	::1
- Credentials Tab:** Configuration for the selected 'root' account:
  - User name: root
  - From host: localhost
  - Password: \*81F5E21E35407D884A6CD4A731AEBFB6AF209E1B
  - Repeat password: (empty)
- Limitations Tab:** Not visible in the screenshot.
- SSL options Tab:** Not visible in the screenshot.
- Allow access to:** A list of global privileges checked:
  - Global privileges
  - EXECUTE
  - PROCESS
  - SELECT
  - SHOW DATABASES
  - SHOW VIEW
  - ALTER
  - ALTER ROUTINE
  - CREATE
  - CREATE ROUTINE
  - CREATE TABLESPACE
  - CREATE TEMPORARY TABLES
  - CREATE VIEW
  - DELETE
  - DROP
  - EVENT
  - INDEX
  - INSERT
  - REFERENCES
  - TRIGGER
  - UPDATE

Come si vede dagli output dei comandi da riga di comando, oppure da HeidiSQL, ci sono già diversi account creati nel DBMS. L'account root è presente più volte perché l'installazione di default di MariaDb crea più account per l'utente root, differenziandoli per l'host da cui si connette. Questo è un concetto molto importante e riguarda la sicurezza: lo stesso utente potrebbe avere alcuni

permessi/privilegi quando si connette in localhost e altri quando si connette da remoto. Nel caso dell'utente root alcuni account sono senza password, ad esempio quello che utilizza 127.0.0.1 come specifica dell'host, oppure ::1. Ovviamente per questioni di sicurezza andrebbe impostata la password anche per questi account.

L'account mariadb.sys è un account speciale senza password (ma locked, ossia non è possibile collegarsi a MariaDb tramite esso) e serve per alcuni scopi amministrativi del DBMS. Non bisogna impostare la password per questo account.

<https://dba.stackexchange.com/questions/280219/should-i-set-up-a-password-for-mariadb-syslocalhost>

<https://www.thegeekdiary.com/what-is-the-purpose-of-mysql-syslocalhost-user/>

<https://www.thegeekdiary.com/what-are-reserved-user-accounts-in-mysql/>

### 13.2.3.1.2. Creazione di un account

Per creare un account da riga di comando:

<https://mariadb.com/kb/en/create-user/>

```
CREATE USER 'myuser'@'localhost' IDENTIFIED BY 'mypassword';
```

L'account creato non ha nessun permesso: può connettersi al DBMS ma non può fare nessuna azione. Non può vedere nemmeno i database esistenti sul DBMS.

### 13.2.3.1.3. Rimozione di un account

Per rimuovere un account:

<https://mariadb.com/kb/en/drop-user/>

```
DROP USER IF EXISTS 'nome_account'@'localhost';
```

### 13.2.3.2. Gestione dei permessi di un account

<https://mariadb.com/kb/en/grant/>

Per assegnare i permessi ad un account l'istruzione principale è GRANT

Ci sono permessi che consentono di creare database, permessi che consentono di effettuare operazioni sulle tabelle di un database, etc.

Ad esempio, supponendo di voler dare all'account myuser il permesso di lettura, inserimento, modifica e cancellazione di righe, sulla tabella table\_name del database database\_name l'istruzione per la definizione dei permessi è:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON database_name.table_name TO  
'myuser'@'localhost';
```

Nota: la tabella table\_name deve esistere!

Per assegnare una serie di permessi su tutte le tabelle di un database (anche quelle che non esistono ancora) ad un determinato account si può usare la wildcard \*:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON database_name.* TO  
'myuser'@'localhost';
```

Per assegnare una serie di permessi su tutte le tabelle di tutti i database gestiti dal DBMS:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO 'myuser'@'localhost';
```

Per vedere di quali permessi gode un determinato account si può usare il comando SHOW GRANTS

**FOR user;**

Ad esempio, per verificare quali permessi ha l'utente 'myuser'@'localhost':

SHOW GRANTS FOR 'myuser'@'localhost';

```
MariaDB [(none)]> SHOW GRANTS FOR 'myuser'@'localhost';
+-----+
| Grants for myuser@localhost
+-----+
| GRANT USAGE ON `.*` TO `myuser`@`localhost` IDENTIFIED BY PASSWORD '*FABE5482D5AADF36D028AC443D117BE1180B9725'
| GRANT SELECT, INSERT, UPDATE, DELETE ON `database_name`.* TO `myuser`@`localhost`
+-----+
2 rows in set (0.001 sec)
```

Per assegnare tutti i permessi ad un account su un determinato database:

GRANT ALL PRIVILEGES ON database\_name.\* TO 'myuser'@'localhost';

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON database_name.* TO 'myuser'@'localhost';
Query OK, 0 rows affected (0.012 sec)

MariaDB [(none)]> SHOW GRANTS FOR 'myuser'@'localhost';
+-----+
| Grants for myuser@localhost
+-----+
| GRANT USAGE ON `.*` TO `myuser`@`localhost` IDENTIFIED BY PASSWORD '*FABE5482D5AADF36D028AC443D117BE1180B9725'
| GRANT ALL PRIVILEGES ON `database_name`.* TO `myuser`@`localhost`
+-----+
2 rows in set (0.000 sec)
```

Per dare all'account la possibilità di creare un altro account con i suoi stessi privilegi (GRANT OPTION):

GRANT ALL PRIVILEGES ON database\_name.\* TO 'myuser'@'localhost' WITH GRANT OPTION;

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON database_name.* TO 'myuser'@'localhost' WITH GRANT OPTION;
Query OK, 0 rows affected (0.002 sec)

MariaDB [(none)]> SHOW GRANTS FOR 'myuser'@'localhost';
+-----+
| Grants for myuser@localhost
+-----+
| GRANT USAGE ON `.*` TO `myuser`@`localhost` IDENTIFIED BY PASSWORD '*FABE5482D5AADF36D028AC443D117BE1180B9725'
| GRANT ALL PRIVILEGES ON `database_name`.* TO `myuser`@`localhost` WITH GRANT OPTION
+-----+
2 rows in set (0.000 sec)
```

Per rimuovere i privilegi ad un dato account (REVOKE):

REVOKE ALL PRIVILEGES ON database\_name.\* FROM 'myuser'@'localhost';

Per rimuovere la GRANT OPTION:

REVOKE GRANT OPTION ON database\_name.\* FROM 'myuser'@'localhost';

```

MariaDB [(none)]> GRANT ALL PRIVILEGES ON database_name.* TO 'myuser'@'localhost' WITH GRANT OPTION;
Query OK, 0 rows affected (0.011 sec)

MariaDB [(none)]> REVOKE ALL PRIVILEGES ON database_name.* FROM 'myuser'@'localhost';
Query OK, 0 rows affected (0.002 sec)

MariaDB [(none)]> SHOW GRANTS FOR 'myuser'@'localhost';
+-----+
| Grants for myuser@localhost |
+-----+
| GRANT USAGE ON *.* TO `myuser`@`localhost` IDENTIFIED BY PASSWORD '*FABE5482D5AADF36D028AC443D117BE1180B9725' |
| GRANT USAGE ON `database_name`.* TO `myuser`@`localhost` WITH GRANT OPTION |
+-----+
2 rows in set (0.000 sec)

MariaDB [(none)]> REVOKE GRANT OPTION ON database_name.* FROM 'myuser'@'localhost';
Query OK, 0 rows affected (0.010 sec)

MariaDB [(none)]> SHOW GRANTS FOR 'myuser'@'localhost';
+-----+
| Grants for myuser@localhost |
+-----+
| GRANT USAGE ON *.* TO `myuser`@`localhost` IDENTIFIED BY PASSWORD '*FABE5482D5AADF36D028AC443D117BE1180B9725' |
+-----+
1 row in set (0.000 sec)

```

Se si vuole rimuovere solo una determinata lista di permessi basta sostituire a ALL PRIVILEGES la lista dei privilegi. Ad esempio: SELECT, INSERT, UPDATE, DELETE.

**Per modificare la password di un utente:**

```

ALTER USER 'myuser'@'localhost' IDENTIFIED BY 'nuovapassword';

FLUSH PRIVILEGES;

```

### 13.2.3.2.1.Flush

<https://mariadb.com/kb/it/flush/>

Dopo la modifica dei permessi di un account è sempre consigliato eseguire il comando FLUSH PRIVILEGES che pulisce o carica varie cache utilizzate internamente al DBMS MariaDb.

FLUSH PRIVILEGES;

```

MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.001 sec)

```

### 13.2.3.3. Creazione di un account specifico per una Web Application

Supponendo di voler creare un account limitato che permetta ad un'applicazione Web di poter effettuare CRUD operations su un database (Create, Read, Update, Delete), avendo i permessi minimi richiesti (principio del minimo privilegio) si può creare un account come il seguente:

```

CREATE USER 'myapp_user'@'localhost' IDENTIFIED BY 'mypassword';

GRANT SELECT, INSERT, UPDATE, DELETE ON myapp_database.* TO
'myapp_user'@'localhost';

FLUSH PRIVILEGES;

```

Supponendo che il database utilizzato dalla Web Application si chiami myapp\_database.

- La SELECT è richiesta per poter leggere dati
- La INSERT è richiesta per poter inserire nuovi dati

- La UPDATE è richiesta per poter modificare dati già inseriti
- La DELETE è richiesta per poter cancellare dati dal database

Questi indicati sopra sono i permessi minimi necessari per far funzionare la web application, supponendo che il database sia già stato creato. In effetti, dal punto di vista della sicurezza sarebbe meglio affidare la creazione del database (con eventuale migrazione) ad un account di MariaDb con i permessi necessari per la creazione del database e di tutte le strutture necessarie (ad esempio Index, Trigger, Temporary Table, etc.) e poi usare un account limitato con solo i permessi sopra indicati per il servizio ordinario.

### 13.2.3.3.1.Creazione del database mediante uno script SQL (in produzione)

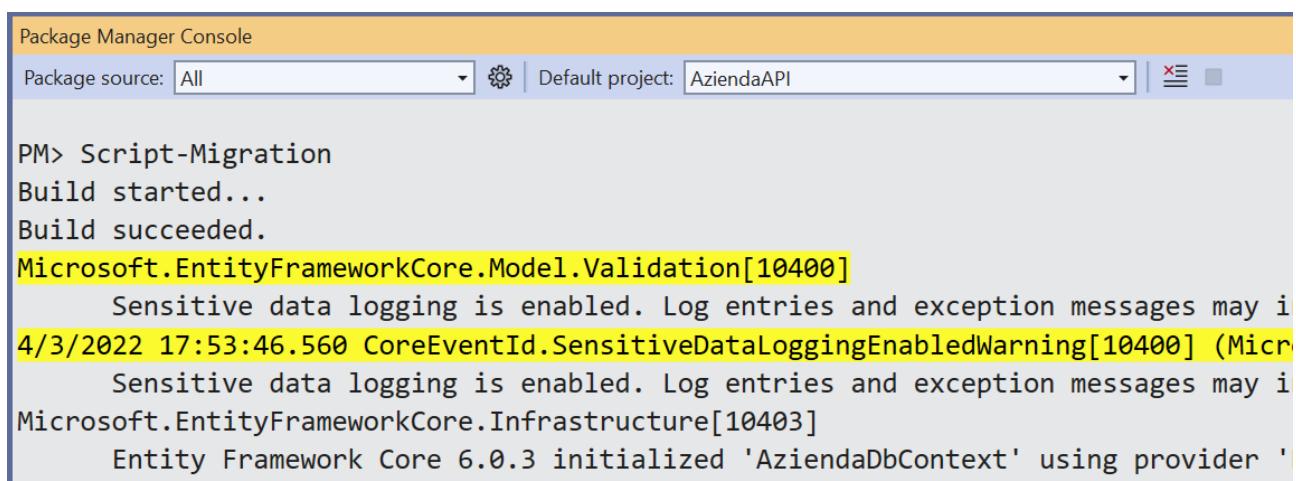
<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/applying?tabs=vs#basic-usage-1>

Prerequisiti: Il server di MariaDb è in esecuzione e raggiungibile con i parametri della connection string.

Dalla Packet Manager Console di Visual Studio digitare il comando:

**Script-Migration**

Visual Studio genera un file SQL con tutte le istruzioni necessarie per creare il database e applicare l'ultima migrazione effettuata.



```
Package Manager Console
Package source: All | Default project: AziendaAPI | x | E

PM> Script-Migration
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Model.Validation[10400]
    Sensitive data logging is enabled. Log entries and exception messages may i
4/3/2022 17:53:46.560 CoreEventId.SensitiveDataLoggingEnabledWarning[10400] (Micr
        Sensitive data logging is enabled. Log entries and exception messages may i
Microsoft.EntityFrameworkCore.Infrastructure[10403]
    Entity Framework Core 6.0.3 initialized 'AziendaDbContext' using provider '
```

Ad esempio, applicando questo comando al progetto Azienda API viene prodotto un file con estensione .sql nella cartella: **AziendaAPI\AziendaAPI\obj\Debug\net6.0** che contiene le istruzioni SQL per la creazione del database. Allo script generato da EF Core aggiungiamo in cima le due righe evidenziate in giallo con le istruzioni per creare il database aziendaapi e salviamo nuovamente il file.

```
CREATE DATABASE IF NOT EXISTS aziendaapi;
use aziendaapi;
CREATE TABLE IF NOT EXISTS `__EFMigrationsHistory` (
    `MigrationId` varchar(150) CHARACTER SET utf8mb4 NOT NULL,
    `ProductVersion` varchar(32) CHARACTER SET utf8mb4 NOT NULL,
    CONSTRAINT `PK__EFMigrationsHistory` PRIMARY KEY (`MigrationId`)
) CHARACTER SET=utf8mb4;

START TRANSACTION;

ALTER DATABASE CHARACTER SET utf8mb4;
```

```

CREATE TABLE `Aziende` (
    `AziendaId` int NOT NULL AUTO_INCREMENT,
    `Nome` nvarchar(100) NOT NULL,
    `Indirizzo` nvarchar(100) NULL,
    CONSTRAINT `PK_Aziende` PRIMARY KEY (`AziendaId`)
) CHARACTER SET=utf8mb4;

CREATE TABLE `Prodotti` (
    `ProdottoId` int NOT NULL AUTO_INCREMENT,
    `AziendaId` int NOT NULL,
    `Nome` nvarchar(100) NOT NULL,
    `Descrizione` nvarchar(200) NULL,
    CONSTRAINT `PK_Prodotti` PRIMARY KEY (`ProdottoId`),
    CONSTRAINT `FK_Prodotti_Aziende_AziendaId` FOREIGN KEY (`AziendaId`) REFERENCES `Aziende`(`AziendaId`) ON DELETE CASCADE
) CHARACTER SET=utf8mb4;

CREATE TABLE `Sviluppatori` (
    `SviluppatoreId` int NOT NULL AUTO_INCREMENT,
    `AziendaId` int NOT NULL,
    `Nome` nvarchar(40) NOT NULL,
    `Cognome` nvarchar(40) NOT NULL,
    CONSTRAINT `PK_Sviluppatori` PRIMARY KEY (`SviluppatoreId`),
    CONSTRAINT `FK_Sviluppatori_Aziende_AziendaId` FOREIGN KEY (`AziendaId`) REFERENCES `Aziende`(`AziendaId`) ON DELETE CASCADE
) CHARACTER SET=utf8mb4;

CREATE TABLE `SviluppaProdotti` (
    `ProdottoId` int NOT NULL,
    `SviluppatoreId` int NOT NULL,
    CONSTRAINT `PK_SviluppaProdotti` PRIMARY KEY (`SviluppatoreId`, `ProdottoId`),
    CONSTRAINT `FK_SviluppaProdotti_Prodotti_ProdottoId` FOREIGN KEY (`ProdottoId`) REFERENCES `Prodotti`(`ProdottoId`) ON DELETE CASCADE,
    CONSTRAINT `FK_SviluppaProdotti_Sviluppatori_SviluppatoreId` FOREIGN KEY (`SviluppatoreId`) REFERENCES `Sviluppatori`(`SviluppatoreId`) ON DELETE CASCADE
) CHARACTER SET=utf8mb4;

INSERT INTO `Aziende` (`AziendaId`, `Indirizzo`, `Nome`)
VALUES (1, 'One Microsoft Way, Redmond, WA 98052, Stati Uniti', 'Microsoft');

INSERT INTO `Aziende` (`AziendaId`, `Indirizzo`, `Nome`)
VALUES (2, '1600 Amphitheatre Pkwy, Mountain View, CA 94043, Stati Uniti', 'Google');

INSERT INTO `Aziende` (`AziendaId`, `Indirizzo`, `Nome`)
VALUES (3, '1 Apple Park Way Cupertino, California, 95014-0642 United States', 'Apple');

```

```

INSERT INTO `Prodotti` (`ProdottoId`, `AziendaId`, `Descrizione`, `Nome`)
VALUES (1, 1, 'Applicazione per la gestione delle Note', 'SuperNote');
INSERT INTO `Prodotti` (`ProdottoId`, `AziendaId`, `Descrizione`, `Nome`)
VALUES (2, 1, 'Applicazione per la visione di film in streaming', 'My Cinema');
INSERT INTO `Prodotti` (`ProdottoId`, `AziendaId`, `Descrizione`, `Nome`)
VALUES (3, 2, 'Applicazione per il cad 3d', 'SuperCad');

INSERT INTO `Sviluppatori` (`SviluppatoreId`, `AziendaId`, `Cognome`, `Nome`)
VALUES (1, 1, 'Rossi', 'Mario');
INSERT INTO `Sviluppatori` (`SviluppatoreId`, `AziendaId`, `Cognome`, `Nome`)
VALUES (2, 1, 'Verdi', 'Giulio');
INSERT INTO `Sviluppatori` (`SviluppatoreId`, `AziendaId`, `Cognome`, `Nome`)
VALUES (3, 2, 'Bianchi', 'Leonardo');

INSERT INTO `SviluppaProdotti` (`ProdottoId`, `SviluppatoreId`)
VALUES (1, 1);

INSERT INTO `SviluppaProdotti` (`ProdottoId`, `SviluppatoreId`)
VALUES (1, 2);

INSERT INTO `SviluppaProdotti` (`ProdottoId`, `SviluppatoreId`)
VALUES (3, 3);

CREATE INDEX `IX_Prodotti_AziendaId` ON `Prodotti` (`AziendaId`);

CREATE INDEX `IX_SviluppaProdotti_ProdottoId` ON `SviluppaProdotti` (`ProdottoId`);

CREATE INDEX `IX_Sviluppatori_AziendaId` ON `Sviluppatori` (`AziendaId`);

INSERT INTO `__EFMigrationsHistory` (`MigrationId`, `ProductVersion`)
VALUES ('20220331183438_InitialMigrate', '6.0.3');

COMMIT;

```

Nel caso del progetto mostrato il file è:

C:\Users\genna\source\repos\4IA\LessonPrepare\MinAPI\MinAPIExplorer\AziendaAPI\AziendaAPI\obj\Debug\net6.0\t5ggyk3t.sql

Per la creazione del database a partire dello script si può usare il monitor di mysql (riga di comando), oppure uno strumento di management del database dotato di interfaccia grafica con un account avente i privilegi necessari. Ad esempio, supponendo di cancellare il database aziendaapi e di volerlo ricreare da capo utilizzando il monitor di mysql, si può procedere come segue:

Collegamento a MariaDb come account amministrativo (ad esempio root)

SHOW DATABASES; -- per vedere l'elenco dei database gestiti

```
C:\Windows\System32\cmd.exe - mysql -u root -p
Microsoft Windows [Version 10.0.22000.593]
(c) Microsoft Corporation. All rights reserved.

C:\Users\genna\Documents\ScuolaLocal\laragon-portable\bin\mysql\mariadb-10.6.7-winx64\bin>mysql -u root -p
Enter password: ****
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 47
Server version: 10.6.7-MariaDB-log mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| utentifilmdb |
+-----+
5 rows in set (0.001 sec)

MariaDB [(none)]>
```

Usciamo dal monitor di MariaDb e dalla riga di comando eseguiamo il seguente comando:

```
mysql -u root -p
<C:\Users\genna\source\repos\4IA\LessonPrepare\MinAPI\MinAPIExplorer\AziendaAPI\AziendaAPI\obj\Debug\net6.0\t5ggyk3t.sql
```

```
C:\Windows\System32\cmd.exe
C:\Users\genna\Documents\ScuolaLocal\laragon-portable\bin\mysql\mariadb-10.6.7-winx64\bin>mysql -u root -p <C:\Users\genna\source\repos\4IA\LessonPrepare\MinAPI\MinAPIExplorer\AziendaAPI\AziendaAPI\obj\Debug\net6.0\t5ggyk3t.sql
Enter password: ****
C:\Users\genna\Documents\ScuolaLocal\laragon-portable\bin\mysql\mariadb-10.6.7-winx64\bin>
```

In alternativa è possibile scrivere `mysql -u root - root_password <script.sql` mettendo la `root_password` direttamente nella riga di comando.

In alternativa è anche possibile non uscire dal mysql monitor usando il comando `source` come mostrato qui:

<https://dev.mysql.com/doc/refman/8.0/en/mysql-batch-commands.html>

In questo caso però, quando si usa il comando `source` su Windows, bisogna scrivere i separatori di directory al contrario (forward slash) e bisogna fare attenzione ai casi in cui ci siano spazi nei nomi delle cartelle (in quel caso bisogna mettere gli apici):

<https://stackoverflow.com/a/8940431>

Connettiamoci nuovamente per verificare che il database è stato creato:

```

C:\Windows\System32\cmd.exe - mysql -u root -p
C:\Users\genna\Documents\ScuolaLocal\laragon-portable\bin\mysql\mariadb-10.6.7-winx64\bin>mysql -u root -p <C:\Users\genna\source\repos\4IA\LessonPrepare\MinAPI\MinAPIExplorer\AziendaAPI\AziendaAPI\obj\Debug\net6.0\t5ggk3t.sql
Enter password: ****

C:\Users\genna\Documents\ScuolaLocal\laragon-portable\bin\mysql\mariadb-10.6.7-winx64\bin>mysql -u root -p
Enter password: ****
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 55
Server version: 10.6.7-MariaDB-log mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| aziendaapi |
| information_schema |
| mysql |
| performance_schema |
| sys |
| utentifilmdb |
+-----+
6 rows in set (0.001 sec)

MariaDB [(none)]> .

```

Con il database creato a partire dallo script è possibile eseguire l'applicazione Web con un account limitato:

```

CREATE USER 'aziendaapi_user'@'localhost' IDENTIFIED BY
'aziendaapi_password';

GRANT SELECT, INSERT, UPDATE, DELETE ON aziendaapi.* TO
'aziendaapi_user'@'localhost';

FLUSH PRIVILEGES;

```

Se facciamo partire l'applicazione web da Visual Studio con la stringa di connessione:

```

"ConnectionStrings": {
    "AziendaAPIConnection": "Server=localhost;port=3306;Database=aziendaapi;User Id=
aziendaapi_user;Password=aziendaapi_password;"}
}

```

Verifichiamo che il programma funziona ed effettua tutte le operazioni richieste.

### 13.2.3.3.2. Creazione del database mediante codice (per sviluppo e testing)

<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/applying?tabs=vs#apply-migrations-at-runtime>

Prerequisiti: Il server di MariaDb è in esecuzione e raggiungibile con i parametri della connection string.

It's possible for the application itself to apply migrations programmatically, typically during startup. While productive for local development and testing of migrations, this approach is inappropriate for managing production databases, for the following reasons:

- If multiple instances of your application are running, both applications could attempt to apply the migration concurrently and fail (or worse, cause data corruption).
- Similarly, if an application is accessing the database while another application migrates it, this

can cause severe issues.

- The application must have elevated access to modify the database schema. It's generally good practice to limit the application's database permissions in production.
- It's important to be able to roll back an applied migration in case of an issue. The other strategies provide this easily and out of the box.
- The SQL commands are applied directly by the program, without giving the developer a chance to inspect or modify them. This can be dangerous in a production environment.

Attenzione: questo approccio è raccomandato solo in caso di sviluppo e testing, ma non in produzione.

La creazione della migrazione direttamente da codice richiede che l'account utilizzato per connettersi al database abbia tutti i permessi necessari non soltanto per effettuare le CRUD operations (SELECT, INSERT, UPDATE, DELETE), ma anche per creare le tabelle, creare indici, modificare database, etc.

La creazione del database automatizzata tramite codice, richiede, come già mostrato nell'esempio 13.2.2, un codice del tipo:

```
//https://stackoverflow.com/questions/42355481/auto-create-database-in-entity-framework-core  
//https://stackoverflow.com/a/55232983  
//crea il database se non esiste a partire dalla migrazione (che deve esistere)  
//se non è stata fatta la migrazione il database non verrà creato  
using (var serviceScope = app.Services.CreateScope())  
{  
    var context = serviceScope.ServiceProvider.GetRequiredService<  
ApplicationDbContext>();  
    context.Database.Migrate();  
    //vedere IN ALTERNATIVA a Migrate anche  
    //context.Database.EnsureCreated();  
}
```

In tal caso però l'account utilizzato per la connessione al database deve avere anche i permessi utilizzati nella migration per la creazione del database e delle relative tabelle, ad esempio:

```
GRANT CREATE, SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, INDEX, SHOW  
VIEW, TRIGGER, LOCK TABLES, CREATE TEMPORARY TABLES ON myapp_database.*  
TO 'myapp_user'@'localhost';  
  
FLUSH PRIVILEGES;
```

In questo modo l'account `myapp_user@localhost` può creare il database `myapp_database` e tutte le relative tabelle.

In alternativa si può anche creare un account con tutti i permessi sul database specifico:

```
GRANT ALL PRIVILEGES ON myapp_database.* TO 'myapp_user'@'localhost';  
  
FLUSH PRIVILEGES;
```

Il dettaglio sulla creazione manuale delle tabelle da riga di comando sarà oggetto di studio del linguaggio SQL nel prossimo anno scolastico.

### **13.2.3.3.2.1. Esempio: Riscrittura del codice dell'esempio Azienda API con account specifico**

Come utente root, da riga di comando di MariaDb scrivere:

```
CREATE USER 'aziendaapi_user'@'localhost' IDENTIFIED BY
```

```
'aziendaapi_password';
GRANT ALL PRIVILEGES ON aziendaapi.* TO 'aziendaapi_user'@'localhost';
FLUSH PRIVILEGES;
```

Nel file di configurazione del progetto (file appsettings.json) cambiare la stringa di connessione come segue:

```
"ConnectionStrings": {
    "AziendaAPIConnection": "Server=localhost;port=3306;Database=aziendaapi;User Id=
aziendaapi_user;Password=aziendaapi_password;"}
```

Se si prova a cancellare il database e a far ripartire l'applicazione, l'istruzione context.Database.Migrate(); si occuperà di ricreare a runtime il database applicando anche l'ultima migrazione.

### 13.3.Binding dei dati con le Minimal API

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis>

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-6.0#parameter-binding>

In questa sezione si analizza in dettaglio un aspetto molto importante delle Minimal API riguardante il Binding dei dati delle richieste, ossia il meccanismo di collegamento i dati presenti nella richiesta con variabili tipizzate gestite dagli handlers delle rotte. Per altri aspetti relativi al funzionamento delle minimal API si rimanda al link in giallo all'inizio di questo paragrafo.

Vediamo prima alcuni aspetti generali relativi al binding.

Supponiamo di creare la classe Person:

```
public class Person
{
    public string FirstName {get; set;}
    public string LastName {get; set;}
}
```

nel nostro file Program.cs possiamo definire un metodo Post come il seguente:

```
app.MapPost("/people", (Person person) =>
{
    return Results.NoContent();
});
```

In questo caso quando facciamo la post viene automaticamente effettuata la deserializzazione dell'oggetto JSON passato dall'utente in un oggetto Person usando il serializzatore predefinito di Microsoft System.Text.Json.Deserialize.

In generale il sistema delle min API cerca di capire da dove prendere certi parametri. Ad esempio, supponiamo di avere il metodo post seguente:

```
app.MapPost("/people/{id:int}", (int id, Person person) =>
{
    return Results.NoContent();
});
```

Nell'esempio precedente l'id viene preso dalla route, mentre l'oggetto Person viene preso dal body. Se dovessi inserire un servizio:

```
public class PeopleService
{
//definizione del servizio
}
```

Andremmo a registrare tale servizio dopo la creazione del builder:

```
builder.Services.AddScoped<PeopleService>();
```

Se volessimo utilizzare questo servizio in una rotta potremmo iniettarlo direttamente nella lambda:

```
app.MapPost("/people/{id:int}", (int id, Person person, PeopleService peopleService) =>
{
    return Results.NoContent();
});
```

Volendo potremmo mettere un attributo [FromService] prima di PeopleService, ma è superfluo, perché viene dedotto in automatico.

Vediamo i binding speciali. Ci sono alcuni binding che ci permettono di accedere ad oggetti che definiscono in maniera completa lo stato della richiesta, della risposta, del context, dell'utente:

```
app.MapGet("/binding", (HttpRequest request, HttpResponse response, HttpContext context,
ClaimsPrincipal user) =>
{
    return Results.NoContent();
});
```

Questi binding speciali sono utili quando vogliamo analizzare i parametri di ingresso e quando il formato dei dati non è quello standard (JSON).

Vediamo alcuni aspetti tratti direttamente dalla documentazione ufficiale.

Parameter binding is the process of converting request data into strongly typed parameters that are expressed by route handlers. A binding source determines where parameters are bound from. Binding sources can be explicit or inferred based on HTTP method and parameter type.

Supported binding sources:

- Route values
- Query string
- Header
- Body (as JSON)
- Services provided by dependency injection
- Custom

Vediamo alcuni esempi:

```
var builder = WebApplication.CreateBuilder(args);

// Added as service
builder.Services.AddSingleton<Service>();

var app = builder.Build();

app.MapGet("/{id}", (int id,
                     int page,
                     [FromHeader(Name = "X-CUSTOM-HEADER")] string customHeader,
                     Service service) => { });
```

```
class Service { }
```

The following table shows the relationship between the parameters used in the preceding example and the associated binding sources.

Parameter	Binding Source
id	route value
page	query string
customHeader	header
service	Provided by dependency injection

The HTTP methods GET, HEAD, OPTIONS, and DELETE don't implicitly bind from body. To bind from body (as JSON) for these HTTP methods, [bind explicitly with \[FromBody\]](#) or read from the [HttpRequest](#).

The following example POST route handler uses a binding source of body (as JSON) for the person parameter:

```
var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.MapPost("/", (Person person) => { });

record Person(string Name, int Age);
```

The parameters in the preceding examples are all bound from request data automatically. To demonstrate the convenience that parameter binding provides, the following example route handlers show how to read request data directly from the request:

```
app.MapGet("/{id}", (HttpRequest request) =>
{
    var id = request.RouteValues["id"];
    var page = request.Query["page"];
    var customHeader = request.Headers["X-CUSTOM-HEADER"];

    // ...
});

app.MapPost("/", async (HttpRequest request) =>
{
    var person = await request.ReadFromJsonAsync<Person>();

    // ...
});
```

Attributes can be used to explicitly declare where parameters are bound from.

```
using Microsoft.AspNetCore.Mvc;

var builder = WebApplication.CreateBuilder(args);

// Added as service
builder.Services.AddSingleton<Service>();

var app = builder.Build();
```

```
app.MapGet("/{id}", ([FromRoute] int id,
    [FromQuery(Name = "p")] int page,
    [FromServices] Service service,
    [FromHeader(Name = "Content-Type")] string contentType)
=> {});
```

```
class Service { }
```

```
record Person(string Name, int Age);
```

#### Parameter Binding Source

id	route value with the name id
page	query string with the name "p"
service	Provided by dependency injection
contentType	header with the name "Content-Type"

Parameters declared in route handlers are treated as required:

- If a request matches the route, the route handler only runs if all required parameters are provided in the request.
- Failure to provide all required parameters results in an error.

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/products", (int pageNumber) => $"Requesting page {pageNumber}");

app.Run();
```

#### URI result

/products?pageNumber=3	3 returned
/products	BadHttpRequestException: Required parameter "int pageNumber" was not provided from query string.
/products/1	HTTP 404 error, no matching route

To make pageNumber optional, define the type as optional or provide a default value:

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/products", (int? pageNumber) => $"Requesting page {pageNumber ?? 1}");

string ListProducts(int pageNumber = 1) => $"Requesting page {pageNumber}";

app.MapGet("/products2", ListProducts);

app.Run();
```

#### URI result

/products?pageNumber=3	3 returned
/products	1 returned
/products2	1 returned

The preceding nullable and default value applies to all sources:

```
var builder = WebApplication.CreateBuilder(args);
```

```

var app = builder.Build();

app.MapPost("/products", (Product? product) => { });

```

app.Run();

The preceding code calls the method with a null product if no request body is sent.

**NOTE:** If invalid data is provided and the parameter is nullable, the route handler is *not* run.

```

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/products", (int? pageNumber) => $"Requesting page {pageNumber ?? 1}");

app.Run();

```

URI	result
/products?pageNumber=3	3 returned
/products	1 returned
/products/two	BadHttpRequestException: Failed to bind parameter "Nullable<int> pageNumber" from "two".

## Special Types

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-6.0#special-types>

The following types are bound without explicit attributes:

- [HttpContext](#): The context which holds all the information about the current HTTP request or response:
 

```
app.MapGet("/", (HttpContext context) =>
    context.Response.WriteAsync("Hello World"));
```
- [HttpRequest](#) and [HttpResponse](#): The HTTP request and HTTP response:
 

```
app.MapGet("/", (HttpRequest request, HttpResponse response) =>
    response.WriteAsync($"Hello World {request.Query["name"]}"));
```
- [CancellationToken](#): The cancellation token associated with the current HTTP request:
 

```
app.MapGet("/", async (CancellationToken cancellationToken) =>
    await MakeLongRunningRequestAsync(cancellationToken));
```
- [ClaimsPrincipal](#): The user associated with the request, bound from [HttpContext.User](#):
 

```
app.MapGet("/", (ClaimsPrincipal user) => user.Identity.Name);
```

### 13.3.1. Custom Binding

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-6.0#custom-binding>

There are two ways to customize parameter binding:

1. For route, query, and header binding sources, bind custom types by adding a static [TryParse](#) method for the type.

## 2. Control the binding process by implementing a **BindAsync** method on a type.

### 13.3.1.1. TryParse

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-6.0#tryparse>

TryParse has two APIs:

```
public static bool TryParse(string value, T out result);  
public static bool TryParse(string value, IFormatProvider provider, T out result);
```

Cosa succederebbe se, ad esempio, volessimo effettuare una conversione automatica di un dato da stringa ad un oggetto custom?

Ad esempio, supponiamo di voler creare una rotta /map che riporta come query string le coordinate gps di un punto nella forma /map?point=43.65,7.42 e supponiamo di voler ottenere queste coordinate deserializzate all'interno di un oggetto Point che ha due properties, una per latitudine (X) e una per longitudine (Y). Creiamo un custom ModelBinder con un metodo statico TryParse

```
var builder = WebApplication.CreateBuilder(args);  
var app = builder.Build();  
  
// GET /map?Point=12.3,10.1  
app.MapGet("/map", (Point point) => $"Point: {point.X}, {point.Y}");  
  
app.Run();  
  
public class Point  
{  
    public double X { get; set; }  
    public double Y { get; set; }  
  
    public static bool TryParse(string? value, IFormatProvider? provider,  
                               out Point? point)  
    {  
        // Format is "(12.3,10.1)"  
        var trimmedValue = value?.TrimStart('(').TrimEnd(')');  
        var segments = trimmedValue?.Split(',',  
                                         StringSplitOptions.RemoveEmptyEntries |  
StringSplitOptions.TrimEntries);  
        if (segments?.Length == 2  
            && double.TryParse(segments[0], out var x)  
            && double.TryParse(segments[1], out var y))  
        {  
            point = new Point { X = x, Y = y };  
            return true;  
        }  
  
        point = null;  
        return false;  
    }  
}
```

In questo caso, siccome l'oggetto Point ha un metodo tryParse, quando riceviamo il parametro dalla query string viene invocato il tryParse per provare a fare la deserializzazione custom.

Come fa il sistema a capire che deve invocare il metodo TryParse? il sistema vede che la lambda accetta in input un parametro di tipo Location che ha il metodo TryParse.

Il TryParse è usato per recuperare dati da query string, dall'header o da una rottura. Nel caso in cui avessimo nel body della richiesta un oggetto complesso che non sia nel formato JSON dovremmo fare un custom Binder con BindAsync.

### 13.3.1.2. BindAsync

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-6.0#bindasync>

BindAsync has the following APIs:

```
public static ValueTask<T> BindAsync(HttpContext context, ParameterInfo parameter);  
public static ValueTask<T> BindAsync(HttpContext context);
```

Esempio di codice che usa il BindAsync:

```
using System.Reflection;  
  
var builder = WebApplication.CreateBuilder(args);  
var app = builder.Build();  
  
// GET /products?SortBy=xyz&SortDir=Desc&Page=99  
app.MapGet("/products", (PagingData pageData) => $"SortBy:{pageData.SortBy}, " +  
    $"SortDirection:{pageData.SortDirection}, CurrentPage:{pageData.CurrentPage}");  
  
app.Run();  
  
public class PagingData  
{  
    public string? SortBy { get; init; }  
    public SortDirection SortDirection { get; init; }  
    public int CurrentPage { get; init; } = 1;  
  
    public static ValueTask<PagingData?> BindAsync(HttpContext context,  
                                                ParameterInfo parameter)  
    {  
        const string sortByKey = "sortBy";  
        const string sortDirectionKey = "sortDir";  
        const string currentPageKey = "page";  
  
        Enum.TryParse<SortDirection>(context.Request.Query[sortDirectionKey],  
                                      ignoreCase: true, out var sortDirection);  
        int.TryParse(context.Request.Query[currentPageKey], out var page);  
        page = page == 0 ? 1 : page;  
  
        var result = new PagingData  
        {  
            SortBy = context.Request.Query[sortByKey],  
            SortDirection = sortDirection,  
            CurrentPage = page  
        };  
    }  
}
```

```

        return ValueTask.FromResult<PagingData?>(result);
    }
}

public enum SortDirection
{
    Default,
    Asc,
    Desc
}

```

Nota importante: se su un oggetto sono definiti sia il TryParse, sia il BindAsync, il metodo TryParse non verrà mai richiamato, perché il BindAsync è più generale del TryParse.

### 13.3.1.3. Binding failures

When binding fails, the framework logs a debug message and returns various status codes to the client depending on the failure mode.

Failure mode	Nullable Parameter Type	Binding Source	Status code
{ParameterType}.TryParse returns false	yes	route/query/header	400
{ParameterType}.BindAsync returns null	yes	custom	400
{ParameterType}.BindAsync throws	does not matter	custom	500
Failure to deserialize JSON body	does not matter	body	400
Wrong content type (not application/json)	does not matter	body	415

### 13.3.1.4. Binding Precedence

The rules for determining a binding source from a parameter:

1. Explicit attribute defined on parameter (From\* attributes) in the following order:
  0. Route values: [\[FromRoute\]](#)
  1. Query string: [\[FromQuery\]](#)
  2. Header: [\[FromHeader\]](#)
  3. Body: [\[FromBody\]](#)
  4. Service: [\[FromServices\]](#)
2. Special types
  0. HttpContext
  1. HttpRequest
  2. HttpResponseMessage
  3. ClaimsPrincipal
  4. CancellationToken
3. Parameter type has a valid BindAsync method.
4. Parameter type is a string or has a valid TryParse method.

0. If the parameter name exists in the route template e.g. `app.Map("/todo/{id}", (int id) => {});`, then it's bound from the route.
1. Bound from the query string.
5. If the parameter type is a service provided by dependency injection, it uses that service as the source.
6. The parameter is from the body.

### 13.3.2. Customize JSON binding

The body binding source uses [System.Text.Json](#) for de-serialization. It is **not** possible to change this default, but the binding can be customized using other techniques described previously. To customize JSON serializer options, use code similar to the following:

```
using Microsoft.AspNetCore.Http.Json;

var builder = WebApplication.CreateBuilder(args);

// Configure JSON options.
builder.Services.Configure<JsonOptions>(options =>
{
    options.SerializerOptions.IncludeFields = true;
});

var app = builder.Build();

app.MapPost("/products", (Product product) => product);

app.Run();

class Product
{
    // These are public fields, not properties. Di norma i campi non sono serializzati.
    // In questo caso i campi sono serializzati in base all'impostazione scelta.
    public int Id;
    public string? Name;
}
```

The preceding code:

- Configures both the input and output default JSON options.
- Returns the following JSON

JSON

```
{
    "id": 1,
    "name": "Joe Smith"
}
```

When posting

JSON

```
{
```

```

    "Id": 1,
    "Name": "Joe Smith"
}

```

Ad esempio, per eliminare i loop dalla serializzazione di oggetti (che si presenta quando ci sono navigation properties incrociate tra oggetti), si può scrivere:

```

// Configure JSON options.
//https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-
//migrate-from-newtonsoft-how-to?pivot=dotnet-6-0#preserve-object-references-and-
//handle-loops
// https://stackoverflow.com/a/66562656
builder.Services.Configure<Microsoft.AspNetCore.Http.Json.JsonOptions>(options =>
{
    options.SerializerOptions.ReferenceHandler =
System.Text.Json.Serialization.ReferenceHandler.IgnoreCycles;
});

```

## 13.4. Configuration in ASP.NET Core

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-6.0#configuration>

The following code reads from the configuration system:

```

var app = WebApplication.Create(args);

var message = app.Configuration["HelloKey"] ?? "Hello";

app.MapGet("/", () => message);

app.Run();

```

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-6.0>

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-6.0#appsettingsjson>

Consider the following appsettings.json file:

```

{
    "Position": {
        "Title": "Editor",
        "Name": "Joe Smith"
    },
    "MyKey": "My appsettings.json Value",
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft": "Warning",
            "Microsoft.Hosting.Lifetime": "Information"
        }
    },
    "AllowedHosts": "*"
}

```

Nelle Minimal API è possibile accedere ai dati nel file di configurazione usando la property Configuration, come negli esempi di seguito riportati:

```
var builder = WebApplication.CreateBuilder(args);
var myKeyValue = builder.Configuration["MyKey"];
var title = builder.Configuration["Position:Title"];
//...oppure
var app = builder.Build();
var name = app.Configuration["Position:Name"];
var defaultLogLevel = app.Configuration["Logging:LogLevel:Default"];
```

## 13.5.Route Handlers

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-6.0#route-handlers>

Route handlers are methods that execute when the route matches. Route handlers can be a function or any shape, including synchronous or asynchronous. Route handlers can be a lambda expression, a local function, an instance method or a static method.

### 13.5.1. Lambda expression

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();
app.MapGet("/inline", () => "This is an inline lambda");
var handler = () => "This is a lambda variable";
app.MapGet("/", handler);
app.Run();
```

### 13.5.2. Local Function

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();
string LocalFunction() => "This is local function";
app.MapGet("/", LocalFunction);
app.Run();
```

### 13.5.3. Instance method

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();
var handler = new HelloHandler();
app.MapGet("/", handler.Hello);
app.Run();
class HelloHandler
{
    public string Hello()
    {
        return "Hello Instance method";
    }
}
```

### 13.5.4. Static method

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();
app.MapGet("/", HelloHandler.Hello);
app.Run();
class HelloHandler
{
    public static string Hello()
    {
        return "Hello static method";
    }
}
```

```
    }  
}
```

### 13.5.5. Name routes and link generation

Routes can be given names in order to generate URLs to the route. Using a named route avoids having to hard code paths in an app:

```
var builder = WebApplication.CreateBuilder(args);  
var app = builder.Build();  
  
app.MapGet("/hello", () => "Hello named route")  
    .WithName("hi");  
  
app.MapGet("/", (LinkGenerator linker) =>  
    $"The link to the hello route is {linker.GetPathByName("hi", values: null)}");  
  
app.Run();
```

The preceding code displays The link to the hello route is /hello from the / endpoint.

Route names are inferred from method names if specified:

```
var builder = WebApplication.CreateBuilder(args);  
var app = builder.Build();  
  
string Hi() => "Hello there";  
app.MapGet("/hello", Hi);  
  
app.MapGet("/", (LinkGenerator linker) =>  
    $"The link to the hello route is {linker.GetPathByName("Hi", values: null)}");  
  
app.Run();
```

**REVIEW:** {linker.GetPathByName("Hi", values: null)} is null in the preceding code.

**NOTE:** Route names are case sensitive.

Route names:

- Must be globally unique.
- Are used as the OpenAPI operation id when OpenAPI support is enabled. See the [OpenAPI section](#) for more details.

## 13.6.Responses

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-6.0#responses>

Route handlers support the following types of return values:

- IResult based - This includes Task<IResult> and ValueTask<IResult>
- string - This includes Task<string> and ValueTask<string>
- T (Any other type) - This includes Task<T> and ValueTask<T>

Return value	Behavior	Content-Type
IResult	The framework calls <a href="#">IResult.ExecuteAsync</a>	Decided by the IResult implementation
string	The framework writes the string directly to the response	text/plain
T (Any other type)	The framework will JSON serialize the response	application/json

Esempi:

### 13.6.1. string return values

```
app.MapGet("/hello", () => "Hello World");
```

### 13.6.2. JSON return values

```
app.MapGet("/hello", () => new { Message = "Hello World" });
```

### 13.6.3. IResult return values

```
app.MapGet("/hello", () => Results.Ok(new { Message = "Hello World" }));

app.MapGet("/api/todoitems/{id}", async (int id, TodoDb db) =>
    await db.Todos.FindAsync(id)
    is Todo todo
    ? Results.Ok(todo)
    : Results.NotFound()
    .Produces<Todo>(StatusCodes.Status200OK)
    .Produces(StatusCodes.Status404NotFound);
```

Altri esempi di IResult:

```
app.MapGet("/hello", HelloHandler);
app.MapPost("/people/{id:int}", Save).Produces(StatusCodes.Status204NoContent);

private IResult HelloHandler(string name)
{
    var result = new {Message = $"Hello, {name}!"}; //Anonymous type object
    return Results.Ok(result); //restituisce un application/json; charset=utf-8 e
status code 200
}

private IResult Save(int id, Person person, PeopleService peopleService)
{
    return Results.NoContent();
```

}

## 13.7. ASP.NET Core Service Lifetimes

<https://stackoverflow.com/questions/38138100/addtransient-addscoped-and-addsingleton-services-differences>

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection>

<https://www.ezzylearning.net/tutorial/asp-net-core-service-lifetimes-infographic>

ASP.NET Core supports the dependency injection (DI) software design pattern that allows us to register services and control how these services will be instantiated and injected in different components. Some services will be instantiated for a short time and will be available only in a particular component and request. Some will be instantiated just once and will be available throughout the application. Here are the service lifetimes available in ASP.NET Core.

### 13.7.1. Singleton

A single instance of the service class is created, stored in memory and reused throughout the application. We can use Singleton for services that are expensive to instantiate. We can register Singleton service using the AddSingleton method as follows:

```
services.AddSingleton<IPersonService, PersonService>();
```

### 13.7.2. Scoped

The service instance will be created once per request. All middlewares, MVC controllers, etc. that participate in handling of a single request will get the same instance. A good candidate for a scoped service is an Entity Framework context. We can register Scoped service using the AddScoped method as follows:

```
services.AddScoped<IPersonService, PersonService>();
```

### 13.7.3. Transient

Transient lifetime services are created each time they're requested. This lifetime works best for lightweight, stateless services. We can register Transient service using the AddTransient method as follows:

```
services.AddTransient<IPersonService, PersonService>();
```

If you want to visualize the above concepts then here is an infographic for your quick reference.

### Singleton

- Only one service instance is created and shared across all requests.
- We need to be aware of concurrency and threading issues

Request 1

Request 2

Instance

Instance

Instance

### Scoped

- One service instance is created for each request and reused throughout the request.
- Request is considered as scope.

Request 1

Request 2

Instance

Instance

Instance

Instance

Instance

Instance

### Transient

- A new service instance is created every time even if it is the same request.
- It is most common and always the safest option if you are worried about multithreading.

Request 1

Request 2

Instance

Instance

Instance

Instance

Instance

Instance

## 13.8.Data Validation

Supponiamo di voler fare una richiesta di tipo post di un oggetto Person e di voler effettuare la validazione dell'input, ossia verificare che il dato di input soddisfi certe condizioni. Supponiamo di scrivere la classe Person come segue:

```
public class Person
{
    [Required]
    public string FirstName {get; set;}
    [Required]
    public string LastName {get; set;}
}
```

In questo caso se facessimo una post e non mettessimo, ad esempio, il LastName non verrebbe generato nessun messaggio d'errore automatico, come invece accade nel caso di ASP.NET API con controllers (non minimal). Dunque l'annotazione sui campi delle classi non funziona con le Min API. Non solo, ma se volessimo, ad esempio, imporre una condizione sui valori inseriti dovremmo scrivere manualmente del codice per gestire la situazione.

Per ottenere una validazione dell'input con le Min API si può ricorrere a librerie di terze parti come MiniValidate (<https://github.com/DamianEdwards/MiniValidation>), oppure FluentValidator (<https://fluentvalidation.net/>). In questi appunti vedremo come usare FluentValidator che è un progetto più maturo di MiniValidate.

### 13.8.1. FluentValidator

<https://fluentvalidation.net/>

FluentValidator è una libreria molto usata per la validazione dello stato degli oggetti C#. Può essere usata in qualsiasi progetto C#.

Permette di specificare i vincoli su un oggetto utilizzando una notazione Fluent. Ad esempio, supponendo di avere una classe Customer definita come segue:

```
public class Customer
{
    public int Id { get; set; }
    public string Surname { get; set; }
    public string Forename { get; set; }
    public decimal Discount { get; set; }
    public string Address { get; set; }
}
```

FluentValidator permette di specificare i vincoli sullo stato dell'oggetto Customer attraverso la definizione di un CustomerValidator definito come segue:

```
public class CustomerValidator : AbstractValidator<Customer>
{
    public CustomerValidator()
    {
        RuleFor(x => x.Surname).NotEmpty();
        RuleFor(x => x.Forename).NotEmpty().WithMessage("Please specify a first name");
        RuleFor(x => x.Discount).NotEqual(0).When(x => x.HasDiscount);
        RuleFor(x => x.Address).Length(20, 250);
        RuleFor(x => x.Postcode).Must(BeAValidPostcode).WithMessage("Please specify a valid
postcode");
    }

    private bool BeAValidPostcode(string postcode)
    {
        // custom postcode validating logic goes here
    }
}
```

Andiamo con ordine e vediamo come installare e utilizzare questa libreria.

### 13.8.1.1. Installazione

Per progetto generici, come ad esempio i progetti console, basta installare da NuGet il pacchetto:

`FluentValidation`

Per avere l'integrazione di questa libreria in un progetto ASP.NET Core occorre invece installare da NuGet il pacchetto:

`FluentValidation.AspNetCore`

### 13.8.1.2. Getting started with validation

<https://docs.fluentvalidation.net/en/latest/start.html>

To define a set of validation rules for a particular object, you will need to create a class that inherits from `AbstractValidator<T>`, where `T` is the type of class that you wish to validate.

For example, imagine that you have a Customer class:

```
public class Customer
{
    public int Id { get; set; }
    public string Surname { get; set; }
    public string Forename { get; set; }
    public decimal Discount { get; set; }
    public string Address { get; set; }
}
```

You would define a set of validation rules for this class by inheriting from `AbstractValidator<Customer>`:

```
using FluentValidation;

public class CustomerValidator : AbstractValidator<Customer>
{}
```

The validation rules themselves should be defined in the validator class's constructor.

To specify a validation rule for a particular property, call the `RuleFor` method, passing a lambda expression that indicates the property that you wish to validate. For example, to ensure that the `Surname` property is not null, the validator class would look like this:

```
using FluentValidation;

public class CustomerValidator : AbstractValidator<Customer>
{
    public CustomerValidator()
    {
        RuleFor(customer => customer.Surname).NotNull();
    }
}
```

To run the validator, instantiate the validator object and call the `Validate` method, passing in the object to validate.

```
Customer customer = new Customer();
CustomerValidator validator = new CustomerValidator();

ValidationResult result = validator.Validate(customer);
```

The `Validate` method returns a `ValidationResult` object. This contains two properties:

- `IsValid` - a boolean that says whether the validation succeeded.
- `Errors` - a collection of `ValidationFailure` objects containing details about any validation failures.

The following code would write any validation failures to the console:

```
using FluentValidation.Results;

Customer customer = new Customer();
CustomerValidator validator = new CustomerValidator();

ValidationResult results = validator.Validate(customer);

if(! results.IsValid)
{
    foreach(var failure in results.Errors)
    {
        Console.WriteLine("Property " + failure.PropertyName + " failed validation. Error was: " +
failure.ErrorMessage);
    }
}
```

You can chain multiple validators together for the same property:

```
using FluentValidation;

public class CustomerValidator : AbstractValidator<Customer>
{
    public CustomerValidator()
    {
        RuleFor(customer => customer.Surname).NotNull().NotEqual("foo");
    }
}
```

This would ensure that the surname is not null and is not equal to the string ‘foo’.

### 13.8.1.3. Throwing Exceptions

Instead of returning a `ValidationResult`, you can alternatively tell FluentValidation to throw an exception if validation fails by using the `ValidateAndThrow` method:

```
Customer customer = new Customer();
CustomerValidator validator = new CustomerValidator();

validator.ValidateAndThrow(customer);
```

This throws a `ValidationException` which contains the error messages in the Errors property.

Note `ValidateAndThrow` is an extension method, so you must have the `FluentValidation` namespace imported with a `using` statement at the top of your file in order for this method to be available.

```
using FluentValidation;
```

The `ValidateAndThrow` method is helpful wrapper around FluentValidation's options API, and is the equivalent of doing the following:

```
validator.Validate(customer, options => options.ThrowOnFailures());
```

If you need to combine throwing an exception with [Rule Sets](#), or validating individual properties, you can combine both options using this syntax:

```
validator.Validate(customer, options =>
{
    options.ThrowOnFailures();
    options.IncludeRuleSets("MyRuleSets");
    options.IncludeProperties(x => x.Name);
});
```

#### 13.8.1.4. Complex Properties

Validators can be re-used for complex properties. For example, imagine you have two classes, Customer and Address:

```
public class Customer
{
    public string Name { get; set; }
    public Address Address { get; set; }
}

public class Address
{
    public string Line1 { get; set; }
    public string Line2 { get; set; }
    public string Town { get; set; }
    public string County { get; set; }
    public string Postcode { get; set; }
}
```

... and you define an AddressValidator:

```
public class AddressValidator : AbstractValidator<Address>
{
    public AddressValidator()
    {
        RuleFor(address => address.Postcode).NotNull();
        //etc
    }
}
```

... you can then re-use the AddressValidator in the CustomerValidator definition:

```
public class CustomerValidator : AbstractValidator<Customer>
```

```

{
    public CustomerValidator()
    {
        RuleFor(customer => customer.Name).NotNull();
        RuleFor(customer => customer.Address).SetValidator(new AddressValidator());
    }
}

```

... so when you call `Validate` on the CustomerValidator it will run through the validators defined in both the CustomerValidator and the AddressValidator and combine the results into a single ValidationResult.

If the child property is null, then the child validator will not be executed.

Instead of using a child validator, you can define child rules inline, eg:

```
RuleFor(customer => customer.Address.Postcode).NotNull()
```

In this case, a null check will not be performed automatically on Address, so you should explicitly add a condition

```
RuleFor(customer => customer.Address.Postcode).NotNull().When(customer => customer.Address != null)
```

### 13.8.1.5. FluentValidator with ASP.NET Core

<https://docs.fluentvalidation.net/en/latest/aspnet.html>

#### 13.8.1.5.1. Un esempio di applicazione di FluentValidator

Come esempio di applicazione di FluentValidator modifichiamo il progetto PizzaStoreSQLite aggiungendo un controllo sulla lunghezza del nome e della descrizione delle pizze. Supponiamo che la lunghezza minima del nome di una pizza debba essere di due caratteri, mentre la descrizione debba contenere almeno 7 caratteri.

Apriamo il progetto PizzaStoreSQLite, già sviluppato in precedenza. Aggiungiamo da NuGet il pacchetto:

`FluentValidation.AspNetCore`

Creiamo la **cartella Validators** con al suo interno la classe PizzaValidator:

```
//file PizzaValidator.cs
```

```
using FluentValidation;
using PizzaStoreSQLite.Models;
```

```
namespace PizzaStoreSQLite.Validators
{
    public class PizzaValidator : AbstractValidator<Pizza>
    {
        public PizzaValidator()
        {
            RuleFor(p => p.Name).NotEmpty().MinimumLength(2); //lunghezza minima del nome
            RuleFor(p => p.Description).NotEmpty().MinimumLength(7); //lunghezza minima della descrizione
        }
    }
}
```

```
    }  
}
```

Iniettiamo il servizio di validazione nella pipeline dell'App. Nel file Program.cs, dopo aver creato l'oggetto builder scriviamo:

```
builder.Services.AddFluentValidation(fv =>  
fv.RegisterValidatorsFromAssemblyContaining<Pizza>());
```

Modifichiamo il metodo che consente di fare la post e la put di una pizza:

```
app.MapPost("/pizza", async (PizzaDb db, IValidator<Pizza> validator, Pizza pizza) =>  
{  
    var pizzaValidationResult = validator.Validate(pizza);  
    if (!pizzaValidationResult.IsValid)  
    {  
        var errors = new { errors = pizzaValidationResult.Errors.Select(x =>  
x.ErrorMessage) };  
        return Results.BadRequest(errors);  
    }  
    await db.Pizzas.AddAsync(pizza);  
    await db.SaveChangesAsync();  
    return Results.Created($"/pizza/{pizza.Id}", pizza);  
});  
  
app.MapPut("/pizza/{id}", async (PizzaDb db, IValidator < Pizza > validator, Pizza  
updatePizza, int id) =>  
{  
    //trovo la pizza con l'id specificato nel database  
    var pizza = await db.Pizzas.FindAsync(id);  
    if (pizza is null) return Results.NotFound();  
    //verifico che la pizza caricata dall'utente rispetti i requisiti sulla pizza  
    var pizzaValidationResult = validator.Validate(updatePizza);  
    if (!pizzaValidationResult.IsValid)  
    {  
        var errors = new { errors = pizzaValidationResult.Errors.Select(x =>  
x.ErrorMessage) };  
        return Results.BadRequest(errors);  
    }  
    pizza.Name = updatePizza.Name;  
    pizza.Description = updatePizza.Description;  
    await db.SaveChangesAsync();  
    return Results.NoContent();  
});
```

Verifichiamo che il sistema si comporti come ci si aspetta. Proviamo a inserire una pizza con un nome oppure con una descrizione che non rispetta i requisiti di validazione e vediamo che effettivamente il sistema risponde con un codice 400 Bad Request con anche la descrizione degli errori.

**POST** /pizza

Parameters

No parameters

Request body required

application/json

```
{
  "id": 0,
  "name": "A",
  "description": "Una pi"
}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'https://localhost:7203/pizza' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d {
    "id": 0,
    "name": "A",
    "description": "Una pi"
  }'
```

Request URL

<https://localhost:7203/pizza>

Server response

Code	Details	Links
400 <small>Undocumented</small>	Error: response status is 400 Response body <pre>{   "errors": [     "The length of 'Name' must be at least 2 characters. You entered 1 characters.",     "The length of 'Description' must be at least 7 characters. You entered 6 characters."   ] }</pre> Response headers <pre>content-type: application/json; charset=utf-8 date: Tue, 05 Apr 2022 17:22:44 GMT server: Kestrel</pre>	No links
200	Success	

Ne caso in cui il dato inserito rispetti i requisiti tutto procede normalmente come già visto in precedenza.

The screenshot shows a Swagger UI interface for a POST request to the '/pizza' endpoint. The request body is defined as follows:

```
{
  "id": 0,
  "name": "Capricciosa",
  "description": "Pizza con mozzarella, prosciutto cotto, carciofi, olive nere"
}
```

The response section shows a successful 201 status with the following JSON response body:

```
{
  "id": 5,
  "name": "Capricciosa",
  "description": "Pizza con mozzarella, prosciutto cotto, carciofi, olive nere"
}
```

Below the response, there is a 'Responses' section with a 'Code' table:

Code	Description	Links
200	Success	No links

Per comodità si riporta il codice completo del file Program.cs e la struttura del progetto:

```
//file Program.cs

using Microsoft.OpenApi.Models;
using Microsoft.EntityFrameworkCore;
using PizzaStoreSQLite.Data;
using PizzaStoreSQLite.Models;
using FluentValidation.AspNetCore;
using FluentValidation;

var builder = WebApplication.CreateBuilder(args);
var connectionString = builder.Configuration.GetConnectionString("Pizzas") ?? "Data
Source=DefaultPizzas.db";
// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
//builder.Services.AddDbContext<PizzaDb>(options =>
options.UseInMemoryDatabase("items"));
//builder.Services.AddSqlite<PizzaDb>(connectionString);
builder.Services.AddDbContext<PizzaDb>(options =>
options.UseSqlite(connectionString));
```

```

builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "PizzaStore API",
        Description = "Making the Pizzas you love",
        Version = "v1"
    });
});
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
builder.Services.AddFluentValidation(fv =>
fv.RegisterValidatorsFromAssemblyContaining<Pizza>());
var app = builder.Build();
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "PizzaStore API V1");
    });
}
app.UseHttpsRedirection();
//la rotta / è indicata a solo scopo illustrativo
//ExcludeFromDescription() serve a non documentare questa rotta
app.MapGet("/", () => "Hello World!).ExcludeFromDescription();
app.MapGet("/pizzas", async (PizzaDb db) => await db.Pizzas.ToListAsync());
app.MapPost("/pizza", async (PizzaDb db, IValidator<Pizza> validator, Pizza pizza) =>
{
    var pizzaValidationResult = validator.Validate(pizza);
    if (!pizzaValidationResult.IsValid)
    {
        var errors = new { errors = pizzaValidationResult.Errors.Select(x =>
x.ErrorMessage) };
        return Results.BadRequest(errors);
    }
    await db.Pizzas.AddAsync(pizza);
    await db.SaveChangesAsync();
    return Results.Created($"/pizza/{pizza.Id}", pizza);
});
app.MapGet("/pizza/{id}", async (PizzaDb db, int id) => await
db.Pizzas.FindAsync(id));
app.MapPut("/pizza/{id}", async (PizzaDb db, IValidator < Pizza > validator, Pizza
updatePizza, int id) =>
{
    //trovo la pizza con l'id specificato nel database
    var pizza = await db.Pizzas.FindAsync(id);
    if (pizza is null) return Results.NotFound();
    //verifico che la pizza caricata dall'utente rispetti i requisiti sulla pizza
    var pizzaValidationResult = validator.Validate(updatePizza);
    if (!pizzaValidationResult.IsValid)
    {
        var errors = new { errors = pizzaValidationResult.Errors.Select(x =>
x.ErrorMessage) };
        return Results.BadRequest(errors);
    }
    pizza.Name = updatePizza.Name;
    pizza.Description = updatePizza.Description;
    await db.SaveChangesAsync();
    return Results.NoContent();
});
app.MapDelete("/pizza/{id}", async (PizzaDb db, int id) =>
{
    var pizza = await db.Pizzas.FindAsync(id);

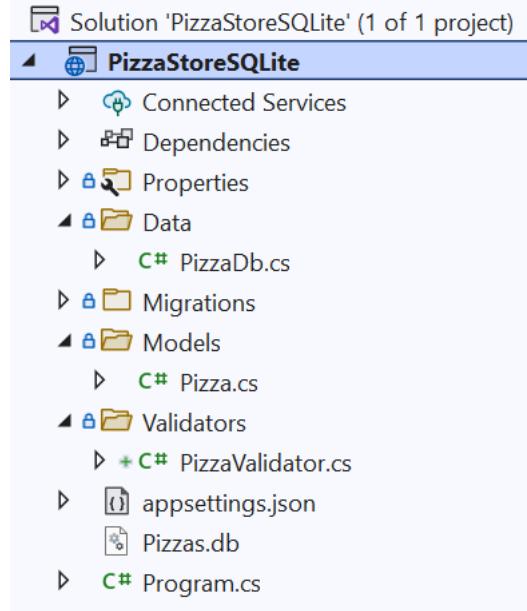
```

```

    if (pizza == null)
    {
        return Results.NotFound();
    }
    db.Pizzas.Remove(pizza);
    await db.SaveChangesAsync();
    return Results.Ok();
});
app.Run();

```

Struttura del progetto:



## 13.9. Messa in sicurezza delle REST API

Un aspetto essenziale delle REST API è dato dalla protezione dei dati che esse gestiscono. Eccetto alcuni casi particolari come, ad esempio, le API pubbliche di servizi come Wikipedia (che pure sono limitate nel numero di API Call per ora), la stragrande maggioranza delle API prevede un meccanismo di sicurezza per permetterne l'utilizzo solo ad applicazioni client autorizzate. La necessità di restringere l'uso delle API solo a soggetti specifici è essenziale quando i dati gestiti dalle API sono riservati (si pensi a dati personali, oppure a dati di transazioni bancarie, etc.), oppure quando l'utilizzo stesso delle API è subordinato al possesso di determinati requisiti: ad esempio, quando si introducono ruoli specifici (come Administrator, oppure Standard User) nell'accesso alle API.

La messa in sicurezza delle API passa attraverso alcuni elementi essenziali, quali:

- **Authentication:**

<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-6.0>

l'applicazione client deve autenticarsi verso il server API per dimostrare di essere un soggetto noto al server. L'autenticazione è il processo di determinazione dell'identità di un utente.

- **Authorization:**

<https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction?view=aspnetcore-6.0>

l'applicazione client deve avere il permesso di accedere al servizio che richiede. L'autorizzazione si riferisce al processo che determina ciò che un utente è in grado di eseguire. Ad esempio, un utente amministratore può creare una raccolta documenti, aggiungere documenti, modificare documenti ed eliminarli. Un utente non amministratore che lavora con la libreria è autorizzato solo a leggere i documenti.

L'autorizzazione è ortogonale e indipendente dall'autenticazione. Tuttavia, l'autorizzazione richiede un meccanismo di autenticazione. L'autenticazione è il processo per verificare chi è un utente.

- **Utilizzo di un protocollo sicuro per la trasmissione dei dati tra client e server (tipicamente https):**

L'utilizzo del protocollo https (<https://www.cloudflare.com/learning/ssl/what-is-https/>) (ovvero http over TLS/SSL) è fondamentale per impedire che un soggetto terzo rispetto al client autenticato e al server Web API possa intercettare i dati che vengono trasmessi tra client e server, oppure possa addirittura impersonificare un'altra identità (facendo richieste al posto del legittimo client). Il protocollo https è, di fatto, il protocollo http, sopra un layer TLS/SSL che permette una connessione criptata e l'autenticazione del server attraverso un certificato SSL valido. Gli aspetti specifici della sicurezza del protocollo https verranno approfonditi nel corso di informatica e di sistemi e reti del quinto anno. Per il momento, dal punto di vista delle applicazioni server web, si può osservare per supportare il protocollo https un server web deve essere provvisto di un certificato digitale valido. Il certificato digitale deve essere a sua volta firmato da una certification Authority (CA) valida. Un certificato digitale non può essere autoprodotto (i certificati autoprodotti possono essere usati solo in fase di testing, ma mai in produzione) e deve essere richiesto ad una CA. Molte CA rilasciano certificati digitali a pagamento. C'è anche la possibilità di utilizzare certificati digitali validi mediante un servizio gratuito come Let's Encrypt (<https://letsencrypt.org/>).

### 13.9.1. Meccanismi di autenticazione e autorizzazione

Di seguito verranno riportati i meccanismi di autenticazione definiti nello standard OpenAPI <https://swagger.io/specification/>:

The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service

without access to source code, documentation, or through network traffic inspection.

<https://swagger.io/docs/specification/authentication/>

OpenAPI uses the term **security scheme** for authentication and authorization schemes. OpenAPI 3.0 lets you describe APIs protected using the following security schemes:

- HTTP authentication schemes (they use the Authorization header):
  - [Basic](#)
  - [Bearer](#)
  - other HTTP schemes as defined by [RFC 7235](#) and [HTTP Authentication Scheme Registry](#)
- [API keys](#) in headers, query string or cookies
  - [Cookie authentication](#)
- [OAuth 2](#)
- [OpenID Connect Discovery](#)

In questa guida verranno trattati soltanto gli schemi di sicurezza **Basic** e quelli basati sul concetto di API key, distinguendo tra i casi di **API key nell'header o nella query string** dal caso di **cookie authentication**. In quest'ultimo caso si parlerà più propriamente di cookie based authentication/authorization.

Gli altri meccanismi di autenticazione verranno trattati l'anno prossimo con riferimento alle architetture basate su microservizi.

### 13.9.1.1. Basic Authentication

[https://en.wikipedia.org/wiki/Basic\\_access\\_authentication](https://en.wikipedia.org/wiki/Basic_access_authentication)

<https://swagger.io/docs/specification/authentication/basic-authentication/>

[Basic authentication](#) is a simple authentication scheme built into the HTTP protocol. The client sends HTTP requests with the `Authorization` header that contains the word `Basic` word followed by a space and a base64-encoded string `username:password`. A request using basic authentication for the user `daniel` with the password `password` looks like this:

```
GET / HTTP/1.1
Host: example.com
Authorization: Basic ZGFuaWVsOnBhc3N3b3Jk
```

**Note:** Because base64 is easily decoded, Basic authentication should only be used together with other security mechanisms such as HTTPS/SSL.

Si provi ad esempio con: <https://www.base64encode.org/>

When using basic authentication for an API, this header is usually sent in every request. The credentials become more or less an API key when used as authentication for the application. Even if it represents a username and password, it's still just a static string.

In theory, the password could be changed once in a while, but that's usually not the case. As with the

API keys, these credentials could leak to third parties. Granted, since credentials are sent in a header, they are less likely to end up in a log somewhere than using a query or path parameter, as the API key might do.

Using basic authentication for authenticating users is usually not recommended since sending the user credentials for every request would be considered bad practice. If HTTP Basic Auth is only used for a single request, it still requires the application to collect user credentials. The user has no means of knowing what the app will use them for, and the only way to revoke the access is to change the password.

### 13.9.1.1.1.Benefits

HTTP Basic Auth is a standardized way to send credentials. The header always looks the same, and the components are easy to implement. It's easy to use and might be a decent authentication for applications in server-to-server environments.

### 13.9.1.1.2.Drawbacks

When a user is authenticated, the application is required to collect the password. From the user perspective, it's not possible to know what the app does with the password. The application will gain full access to the account, and there's no other way for the user to revoke the access than to change the password. Passwords are long-lived tokens, and if an attacker would get a hold of a password, it will likely go unnoticed. When used to authenticate the user, multi-factor authentication is not possible.

## 13.9.1.2. API Key authentication/authorization

<https://swagger.io/docs/specification/authentication/api-keys/>

Some APIs use API keys for authorization. An API key is a token that a client provides when making API calls. The key can be sent in the query string:

```
1. GET /something?api_key=abcdef12345
```

or as a request header:

```
1. GET /something HTTP/1.1  
2. X-API-Key: abcdef12345
```

or as a cookie:

```
1. GET /something HTTP/1.1  
2. Cookie: X-API-KEY=abcdef12345
```

API keys are supposed to be a secret that only the client and server know. Like Basic authentication, API key-based authentication is only considered secure if used together with other security mechanisms such as HTTPS/SSL.

<https://nordicapis.com/the-difference-between-http-auth-api-keys-and-oauth/>

Using API keys is a way to authenticate an application accessing the API, without referencing an actual user. The app adds the key to each API request, and the API can use

the key to identify the application and authorize the request. The key can then be used to perform things like rate limiting, statistics, and similar actions. For instance, Google Cloud accepts the API key with a query parameter like this:

```
curl -X POST https://language.googleapis.com/v1/documents:analyzeEntities?key=API_KEY
```

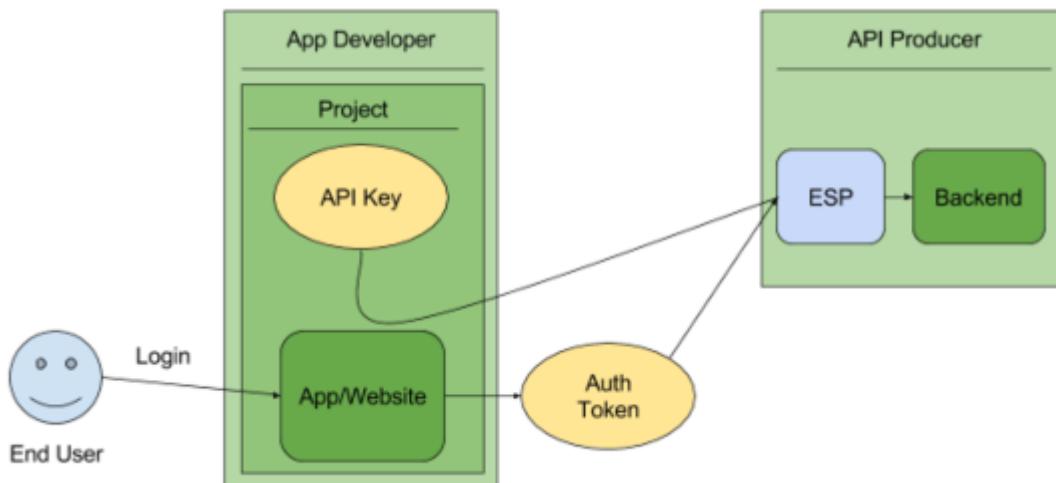
Cloudflare requires the API key to be sent in a custom header:

```
curl https://api.cloudflare.com/client/v4/zones/cd7d0123e301230df9514d \
-H "Content-Type:application/json" \
-H "X-Auth-Key:1234567893feefc5f0q5000bfo0c38d90bbeb" \
-H "X-Auth-Email:example@example.com"
```

Le API Keys sono usate per identificare i progetti (le applicazioni oppure i siti web), non gli utenti:

[https://cloud.google.com/endpoints/docs/openapi/when-why-api-key#api\\_keys\\_are\\_for\\_projects\\_authentication\\_is\\_for\\_users](https://cloud.google.com/endpoints/docs/openapi/when-why-api-key#api_keys_are_for_projects_authentication_is_for_users)

- API keys identify the calling project — the application or site — making the call to an API.
- Authentication tokens identify a user — the person — that is using the app or site.



By identifying the calling project, you can use API keys to associate usage information with that project. API keys allow the [Extensible Service Proxy \(ESP\)](#) to reject calls from projects that haven't been granted access or enabled in the API.

### 13.9.1.2.1. Benefits

<https://nordicapis.com/the-difference-between-http-auth-api-keys-and-oauth/>

It's relatively easy for clients to use API keys. Even though most providers use different methods, adding a key to the API request is quite simple.

### 13.9.1.2.2. Drawbacks

<https://nordicapis.com/the-difference-between-http-auth-api-keys-and-oauth/>

The API key only identifies the application, not the user of the application. It's often difficult to keep the key a secret. For server-to-server communication, it's possible to hide the key using TLS and restrict the access to only be used in backend scenarios. However, since many other types of clients

will consume the APIs, the keys are likely to leak.

Request URLs can end up in logs. JavaScript applications have more or less everything out in the open. Mobile apps are easy to decompile, and so on. Thus, developers shouldn't rely on API keys for more than identifying the client for statistical purposes. Furthermore, API keys are also not standardized, meaning every API has a unique implementation.

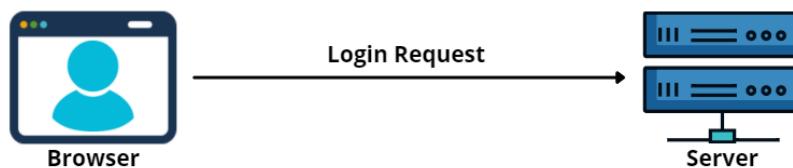
### 13.9.1.3. *Cookie based authentication/authorization*

<https://swagger.io/docs/specification/authentication/cookie-authentication/>

<https://blog.bitsrc.io/web-authentication-cookies-vs-tokens-8e47d5a96d34>

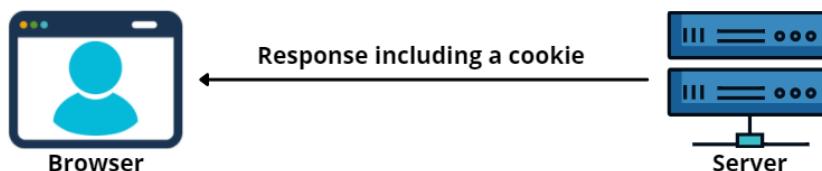
Cookie authentication uses [HTTP cookies](#) to authenticate client requests and maintain session information. It works as follows:

1. The client sends a login request to the server.



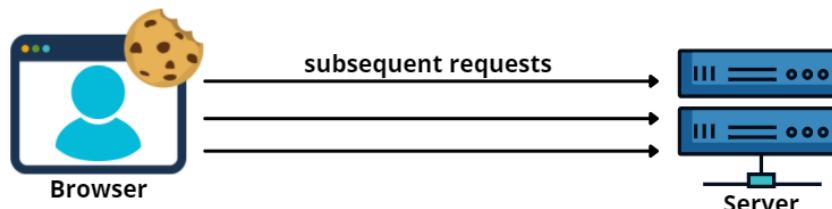
2. On the successful login, the server response includes the [Set-Cookie](#) header that contains the cookie name, value, expiry time and some other info. Here is an example that sets the cookie named `JSESSIONID`:

```
0. Set-Cookie: JSESSIONID=abcde12345; Path=/; HttpOnly
```



3. The client needs to send this cookie in the [Cookie](#) header in all subsequent requests to the server.

```
0. Cookie: JSESSIONID=abcde12345
```



4. On the logout operation, the server sends back the `Set-Cookie` header that causes the cookie to expire.

Il server può creare una sessione (nella memoria RAM del server oppure in un database), ossia una collection di dati associati all'utente, che utilizza il cookie di sessione come chiave di accesso: dato il cookie di sessione (SessionID) è univocamente determinato lo spazio di memoria all'interno del server riservato all'utente. In questo modo il server potrà riconoscere l'utente e associare ad esso informazioni che lo riguardano. In questo modo, ad esempio, è possibile fare in modo che, dopo aver fatto il login, l'utente venga riconosciuto in tutte le pagine successive che visita sul sito: ogni volta che l'utente accede ad una pagina del sito, il browser manda automaticamente il cookie di sessione in tutte le richieste che fa verso il server.

Note: Cookie authentication is vulnerable to Cross-Site Request Forgeries (CSRF) attacks, so it should be used together with other security measures, such as [CSRF tokens](#).

[https://en.wikipedia.org/wiki/Cross-site\\_request\\_forgery](https://en.wikipedia.org/wiki/Cross-site_request_forgery)

[https://it.wikipedia.org/wiki/Cross-site\\_request\\_forgery](https://it.wikipedia.org/wiki/Cross-site_request_forgery)

Il **Cross-site request forgery**, abbreviato **CSRF** o anche **XSRF**, è una vulnerabilità a cui sono esposti i siti web dinamici quando sono progettati per ricevere richieste da un [client](#) senza meccanismi per controllare se la richiesta sia stata inviata intenzionalmente oppure no. Diversamente dal [cross-site scripting](#) (XSS), che sfrutta la fiducia di un utente in un particolare sito, il CSRF sfrutta la fiducia di un sito nel browser di un utente.

In un attacco CSRF un hacker attacca l'utente inviandogli un link malevolo (ad esempio mediante e-mail) che scatena un'azione su un sito web a cui la vittima aveva precedentemente avuto accesso e di cui aveva memorizzato il cookie di autorizzazione/sessione.

In un attacco di tipo XSS un hacker attacca un sito web, sfruttandone le sue vulnerabilità e iniettandovi codice javascript malevolo che verrà successivamente eseguito nel browser della vittima.

[https://en.wikipedia.org/wiki/Cross-site\\_scripting](https://en.wikipedia.org/wiki/Cross-site_scripting)

[https://it.wikipedia.org/wiki/Cross-site\\_scripting](https://it.wikipedia.org/wiki/Cross-site_scripting)

Un esempio di attacco XSS non persistente:

[https://it.wikipedia.org/wiki/Cross-site\\_scripting#Non-persistent](https://it.wikipedia.org/wiki/Cross-site_scripting#Non-persistent)

Un esempio di attacco XSS persistente:

[https://it.wikipedia.org/wiki/Cross-site\\_scripting#Attacco\\_persistente](https://it.wikipedia.org/wiki/Cross-site_scripting#Attacco_persistente)

Prevenzione di attacchi di tipo XSS:

[https://it.wikipedia.org/wiki/Cross-site\\_scripting#Come\\_difendersi](https://it.wikipedia.org/wiki/Cross-site_scripting#Come_difendersi)

Un esempio di attacco CSRF:

[https://it.wikipedia.org/wiki/Cross-site\\_request\\_forgery#Esempio\\_di\\_attacco](https://it.wikipedia.org/wiki/Cross-site_request_forgery#Esempio_di_attacco)

Prevenzione di attacchi di tipo CSRF:

[https://en.wikipedia.org/wiki/Cross-site\\_request\\_forgery#Prevention](https://en.wikipedia.org/wiki/Cross-site_request_forgery#Prevention)

A differenza dell'API Key authentication/authorization, l'autenticazione basata su cookie (con o

senza sessioni) può essere usata per identificare l'utente e non solo il progetto (l'app).

### 13.9.1.4. Implementazione dei meccanismi di sicurezza in .NET Core

In questa guida verranno implemetati diversi meccanismi di autenticazione, ed in particolare:

Basic authentication/authorization

ApiKey authentication/authorization

Session based authentication/authorization

Cookie based authentication/authorization (senza sessioni)

Per ognuno di questi meccanismi verrà mostrato un esempio base e alla fine verrà mostrato un esempio completo nel quale verranno applicati diversi meccanismi di autenticazione ed autorizzazione. La parte relativa alla bearer authentication e ai meccanismi come OAuth2 e Open ID connect verranno analizzati l'anno prossimo.

#### 13.9.1.4.1. Esempio di Basic authentication/authorization

L'implementazione della Basic Authentication in .NET Core è resa molto semplice grazie all'utilizzo di una libreria che può essere scaricata dal repository NuGet:

NuGet: Install-Package AspNetCore.Authentication.Basic

Codice sorgente: <https://github.com/mihirdilip/aspnetcore-authentication-basic>

Creiamo un progetto del tipo ASP.NET Core Web API (minimal, vale a dire senza controllers), chiamato BasicAuthenticationSample.

Installiamo il pacchetto AspNetCore.Authentication.Basic

In questo progetto, per mantenere l'esempio semplice e focalizzato sull'argomento della Basic Authentication assumeremo che le credenziali di accesso siano presenti in un Repository in memoria. In uno scenario reale, queste credenziali saranno recuperate da un database.

L'esempio mostrato di seguito è presente nel repository di Github:

<https://github.com/GreppiDev/Info4IA2122NetworkProgramming>

Creiamo una cartella **Models** con al suo interno il file User.cs

```
namespace BasicAuthenticationSample.Models
{
    public class User
    {
        public string Username { get; set; } = null!;
        public string Password { get; set; } = null!;
        public string? Role { get; set; }
    }
}
```

Creiamo la cartella **Repositories** con al suo interno i file:

```
//file IUserRepository.cs
using BasicAuthenticationSample.Models;

namespace BasicAuthenticationSample.Repositories
{
    public interface IUserRepository
    {
        Task<User?> GetUserByUsername(string username);
        Task<IEnumerable<User>> GetUsers();
    }
}
```

```

//file InMemoryUserRepository.cs

using BasicAuthenticationSample.Models;

namespace BasicAuthenticationSample.Repositories
{
    public class InMemoryUserRepository : IUserRepository
    {
        private List<User> _users = new List<User>
        {
            new User { Username = "TestUser1", Password = "1234" , Role="Administrator"},  

            new User { Username = "TestUser2", Password = "1234" , Role="Standard"},  

            new User { Username = "TestUser3", Password = "1234" },  

            new User { Username = "TestUser4", Password = "1234" }
        };
        public Task<User?> GetUserByUsername(string username)
        {
            return Task.FromResult(_users.FirstOrDefault(u => u.Username == username));
        }

        public Task<IEnumerable<User>> GetUsers()
        {
            return Task.FromResult<IEnumerable<User>>(_users);
        }
    }
}

```

Creiamo la cartella **Services** con dentro il servizio **BasicUserValidationService**:

```

//file BasicUserValidationService.cs

using AspNetCore.Authentication.Basic;
using BasicAuthenticationSample.Repositories;

namespace BasicAuthenticationSample.Services
{
    internal class BasicUserValidationService : IBasicUserValidationService
    {
        private readonly ILogger<BasicUserValidationService> _logger;
        private readonly IUserRepository _userRepository;

        public BasicUserValidationService(ILogger<BasicUserValidationService> logger,
IUserRepository userRepository)
        {
            _logger = logger;
            _userRepository = userRepository;
        }

        public async Task<bool> IsValidAsync(string username, string password)
        {
            try
            {
                var user = await _userRepository.GetUserByUsername(username);
                var isValid = user != null && user.Password == password;
                return isValid;
            }
            catch (Exception e)
            {
                _logger.LogError(e, e.Message);
                throw;
            }
        }
    }
}

```

Scriviamo il codice del file **Program.cs** come segue:

```

using AspNetCore.Authentication.Basic;
using Microsoft.AspNetCore.Authorization;
using BasicAuthenticationSample.Repositories;
using BasicAuthenticationSample.Services;
using System.Security.Claims;
using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

```

```

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
//builder.Services.AddSwaggerGen();
//https://www.c-sharpcorner.com/article/basic-authentication-in-swagger-open-api-net-5/
//le due istruzioni seguenti impostano Swagger per utilizzare utilizzare la basic authentication per
//tutti gli endpoints
//E' possibile fare in modo che che Swagger mostri l'opzione per la security solo sugli endpoint che
//lo richiedono effettivamente, ma
//la configurazione è più complessa è verrà mostrata in un prossimo esempio
#region Configure Swagger
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo { Title = "BasicAuth", Version = "v1" });
    c.AddSecurityDefinition("basic", new OpenApiSecurityScheme
    {
        Name = "Authorization",
        Type = SecuritySchemeType.Http,
        Scheme = "basic",
        In = ParameterLocation.Header,
        Description = "Basic Authorization header using the Bearer scheme."
    });
    c.AddSecurityRequirement(new OpenApiSecurityRequirement
    {
        {
            new OpenApiSecurityScheme
            {
                Reference = new OpenApiReference
                {
                    Type = ReferenceType.SecurityScheme,
                    Id = "basic"
                }
            },
            Array.Empty<string>()
        }
    });
});
#endregion
// Add User repository to the dependency container.
builder.Services.AddScoped<IUserRepository, InMemoryUserRepository>();

// Add the Basic scheme authentication here..
// It requires Realm to be set in the options if SuppressWWWAuthenticateHeader is not set.
// If an implementation of IBasicUserValidationService interface is registered in the dependency
// register as well as OnValidateCredentials delegate on options.Events is also set then this delegate
// will be used instead of an implementation of IBasicUserValidationService.
builder.Services.AddAuthentication(BasicDefaults.AuthenticationScheme)

    // The below AddBasic without type parameter will require OnValidateCredentials delegate on
    // options.Events to be set unless an implementation of IBasicUserValidationService interface is
    // registered in the dependency register.
    // Please note if both the delegate and validation server are set then the delegate will be used
    // instead of BasicUserValidationService.
    //.AddBasic(options =>

        // The below AddBasic with type parameter will add the BasicUserValidationService to the
        // dependency register.
        // Please note if OnValidateCredentials delegate on options.Events is also set then this delegate
        // will be used instead of BasicUserValidationService.
        .AddBasic<IBasicUserValidationService>(options =>
    {
        options.Realm = "Sample Web API";

        //// Optional option to suppress the browser login dialog for ajax calls.
        //options.SuppressWWWAuthenticateHeader = true;

        //// Optional option to ignore authentication if AllowAnonymous metadata/filter attribute is
        //added to an endpoint.
        options.IgnoreAuthenticationIfAllowAnonymous = true;

        //// Optional events to override the basic original logic with custom logic.
        //// Only use this if you know what you are doing at your own risk. Any of the events can be
        //assigned.
        options.Events = new BasicEvents
    {

```

```

    //// A delegate assigned to this property will be invoked just before validating
credentials.
    OnValidateCredentials = async (context) =>
    {
        // custom code to handle credentials, create principal and call Success method on
context.
        var userRepository =
context.HttpContext.RequestServices.GetRequiredService<IUserRepository>();
        var user = await userRepository.GetUserByUsername(context.Username);
        var isValid = user != null && user.Password == context.Password;
        if (isValid)
        {
            context.Response.Headers.Add("ValidationCustomHeader", "From
OnValidateCredentials");
            var claims = new List<Claim>()
            {
                //i claims sono delle attestazioni che vengono fatte sul soggetto che
effettua le richieste
                new Claim(ClaimTypes.NameIdentifier, context.Username, ClaimValueTypes.String,
context.Options.ClaimsIssuer),
                new Claim(ClaimTypes.Name, context.Username, ClaimValueTypes.String,
context.Options.ClaimsIssuer),
                new Claim("CustomClaimType", "Custom Claim Value - from
OnValidateCredentials"),
            };
            if (user?.Role != null)
            {
                claims.Add(new Claim(ClaimTypes.Role, user.Role, ClaimValueTypes.String,
context.Options.ClaimsIssuer));
            }
            context.Principal = new ClaimsPrincipal(new ClaimsIdentity(claims,
context.Scheme.Name));
            context.Success();
        }
        else
        {
            context.NoResult();
        }
    },
    //// A delegate assigned to this property will be invoked before a challenge is sent back
to the caller when handling unauthorized response.
//OnHandleChallenge = async (context) =>
//{
//    // custom code to handle authentication challenge unauthorized response.
//    context.Response.StatusCode = StatusCodes.Status401Unauthorized;
//    context.Response.Headers.Add("ChallengeCustomHeader", "From OnHandleChallenge");
//    await context.Response.WriteAsync("{\"CustomBody\":\"From OnHandleChallenge\"}");
//    context.Handled(); // important! do not forget to call this method at the end.
//},
    //// A delegate assigned to this property will be invoked if Authorization fails and
results in a Forbidden response.
//OnHandleForbidden = async (context) =>
//{
//    // custom code to handle forbidden response.
//    context.Response.StatusCode = StatusCodes.Status403Forbidden;
//    context.Response.Headers.Add("ForbidCustomHeader", "From OnHandleForbidden");
//    await context.Response.WriteAsync("{\"CustomBody\":\"From OnHandleForbidden\"}");
//    context.Handled(); // important! do not forget to call this method at the end.
//},
    //// A delegate assigned to this property will be invoked when the authentication
succeeds. It will not be called if OnValidateCredentials delegate is assigned.
    //// It can be used for adding claims, headers, etc to the response.
    //OnAuthenticationSucceeded = (context) =>
    //{
//        //custom code to add extra bits to the success response.
//        context.Response.Headers.Add("SuccessCustomHeader", "From
OnAuthenticationSucceeded");
//        var customClaims = new List<Claim>
//        {

```

```

        //      new Claim("CustomClaimType", "Custom Claim Value - from
OnAuthenticationSucceeded")
        //    };
        //    context.AddClaims(customClaims);
        //    //or can add like this - context.Principal.AddIdentity(new
ClaimsIdentity(customClaims));
        //    return Task.CompletedTask;
        //},
        // A delegate assigned to this property will be invoked when the authentication fails.
//OnAuthenticationFailed = (context) =>
//{
//    // custom code to handle failed authentication.
//    context.Fail("Failed to authenticate");
//    return Task.CompletedTask;
//}

};

// All the requests will need to be authorized.
// Alternatively, add [Authorize] attribute to Controller or Action Method where necessary.
//builder.Services.AddAuthorization(options =>
//{
//    options.FallbackPolicy = new AuthorizationPolicyBuilder()
//        .RequireAuthenticatedUser()
//        .Build();
//});
builder.Services.AddAuthorization();

// Configure JSON options. Questo serve solo nel caso di loop nella serializzazione di oggetti.
// Nel caso di oggetti DTO senza navigation property il problema non si pone e non c'è bisogno di
impostare questa opzione
//https://stackoverflow.com/a/66562656
//https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-migrate-from-newtonsoft-how-to?pivots=dotnet-6-0#preserve-object-references-and-handle-loops
builder.Services.Configure<Microsoft.AspNetCore.Http.Json.JsonOptions>(options =>
{
    options.SerializerOptions.ReferenceHandler =
System.Text.Json.Serialization.ReferenceHandler.IgnoreCycles;
});

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthentication(); // NOTE: DEFAULT TEMPLATE DOES NOT HAVE THIS, THIS LINE IS REQUIRED AND
HAS TO BE ADDED!!!

app.UseAuthorization();

var summaries = new[]
{
    "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
};

app.MapGet("/weatherforecast", [Authorize(AuthenticationSchemes = BasicDefaults.AuthenticationScheme,
Roles = "Administrator, Standard")] () =>
{
    var forecast = Enumerable.Range(1, 5).Select(index =>
        new WeatherForecast
        (
            DateTime.Now.AddDays(index),
            Random.Shared.Next(-20, 55),
            summaries[Random.Shared.Next(summaries.Length)])
    )
        .ToArray();
    return forecast;
}
);

```

```

        })
        .WithName("GetWeatherForecast").Produces(StatusCodes.Status200OK).Produces(StatusCodes.Status403Forbidden).Produces(StatusCodes.Status401Unauthorized);

    app.MapGet("/show-claims", [Authorize(AuthenticationSchemes = BasicDefaults.AuthenticationScheme, Roles = "Administrator")]) (ClaimsPrincipal user) =>
    {
        return Results.Ok(user.Claims.ToList());
    })
    .WithName("showClaims").Produces(StatusCodes.Status200OK).Produces(StatusCodes.Status403Forbidden).Produces(StatusCodes.Status401Unauthorized);

    app.MapGet("/ciao-mondo", () =>
    {
        return Results.Ok(new { message = "Ciao Mondo. Questa rotta non richiede autorizzazione" });
    })
    .WithName("ciaoMondo").Produces(StatusCodes.Status200OK);

app.Run();

internal record WeatherForecast(DateTime Date, int TemperatureC, string? Summary)
{
    public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
}

```

Si noti che la classe BasicUserValidationService poteva anche non essere scritta, dal momento che nell'esempio è stato fatto uso dell'evento OnValidateCredential per avere la possibilità di mostrare anche l'utilizzo dei Claims per la definizione dei ruoli.

Si osservi cosa succede quando si effettua una richiesta, guardando gli headers della richiesta e della risposta con i developer tools del browser (F12 e poi selezionare il tab Network).

Si noti anche che i browser memorizzano le credenziali per la basic authentication fino a che non vengono chiusi del tutto, oppure fino a che non si effettua un restart del browser. Alcuni browser, come Chrome, rimangono attivi in background anche quando si chiudono tutte le schede. In tal caso, per testare la basic authentication, conviene fare un restart del browser, aprendo una finestra (tab) e scrivendo **chrome://restart** e poi premendo invio (nel caso di Chrome, Edge, Opera). Nel caso di Edge funziona anche **edge://restart**, mentre nel caso di Opera si può usare **opera://restart**. Nel caso di Firefox bisogna aprire un tab e scrivere **about:profiles** e premere invio; a questo punto si apre una pagina dove c'è un pulsante per riavviare il browser.

In alternativa al restart del browser c'è la possibilità di aprire una pagina di navigazione in incognito.

### 13.9.1.4.2. Esempio di ApiKey authentication/authorization

L'implementazione della ApiKey Authentication in .NET Core è resa molto semplice grazie all'utilizzo di una libreria che può essere scaricata dal repository NuGet:

NuGet: Install-Package AspNetCore.Authentication.ApiKey

Codice sorgente: <https://github.com/mihirdilip/aspnetcore-authentication-apikey>

Creiamo un progetto del tipo ASP.NET Core Web API (minimal, vale a dire senza controllers), chiamato ApiKeyAuthenticationSample.

Installiamo il pacchetto AspNetCore.Authentication.ApiKey

In questo progetto, per mantenere l'esempio semplice e focalizzato sull'argomento della ApiKey Authentication assumeremo che le credenziali di accesso siano presenti in un Repository in memoria. In uno scenario reale, queste credenziali saranno recuperate da un database.

L'esempio mostrato di seguito è presente nel repository di Github:

<https://github.com/GreppiDev/Info4IA2122NetworkProgramming>

Creiamo una cartella **Models** con al suo interno il file ApiKey.cs

```
//file ApiKey.cs

using AspNetCore.Authentication.ApiKey;
using System.Security.Claims;

namespace ApiKeyAuthenticationSample.Models
{
    class ApiKey : IApiKey
    {
        public ApiKey(string key, string owner, List<Claim>? claims = null)
        {
            Key = key;
            OwnerName = owner;
            Claims = claims ?? new List<Claim>();
        }

        public string Key { get; }
        public string OwnerName { get; }
        public IReadOnlyCollection<Claim> Claims { get; }
    }
}
```

Creiamo la cartella **Repositories** con dentro i file:

IApiKeyRepository.cs e InMemoryApiKeyRepository.cs.

//file: IApiKeyRepository.cs

```
using AspNetCore.Authentication.ApiKey;

namespace ApiKeyAuthenticationSample.Repositories
{
    public interface IApiKeyRepository
    {
        Task<IApiKey?> GetApiKeyAsync(string key);
    }
}
```

//file: InMemoryApiKeyRepository.cs

```
using AspNetCore.Authentication.ApiKey;
using ApiKeyAuthenticationSample.Models;
using System.Security.Claims;

namespace ApiKeyAuthenticationSample.Repositories
{
    public class InMemoryApiKeyRepository : IApiKeyRepository
    {

        private List<IApiKey> _cache = new List<IApiKey>
        {
            new ApiKey("Key1", "Admin", new List<Claim>()
            {
                new Claim(ClaimTypes.NameIdentifier, "Admin"),
                new Claim(ClaimTypes.GivenName, "Admin"),
                new Claim(ClaimTypes.Role, "Administrator")
            }),
            new ApiKey("Key2", "User", new List<Claim>()
            {
                new Claim(ClaimTypes.NameIdentifier, "User"),
                new Claim(ClaimTypes.GivenName, "User"),
                new Claim(ClaimTypes.Role, "Standard")
            }),
        };

        public Task<IApiKey?> GetApiKeyAsync(string key)
        {
            var apiKey = _cache.FirstOrDefault(k => k.Key.Equals(key,
StringComparison.OrdinalIgnoreCase));
            return Task.FromResult(apiKey);
        }
    }
}
```

Creiamo la cartella **Services** con dentro il file ApiKeyProvider.cs

```
//file: ApiKeyProvider.cs

using AspNetCore.Authentication.ApiKey;
using ApiKeyAuthenticationSample.Repositories;

namespace ApiKeyAuthenticationSample.Services
{
    class ApiKeyProvider : IApiKeyProvider
    {
        private readonly ILogger<IApiKeyProvider> _logger;
        private readonly IApiKeyRepository _apiKeyRepository;

        public ApiKeyProvider(ILogger<IApiKeyProvider> logger, IApiKeyRepository apiKeyRepository)
        {
            _logger = logger;
            _apiKeyRepository = apiKeyRepository;
        }

        public async Task<IApiKey?> ProvideAsync(string key)
        {
            try
            {
                return await _apiKeyRepository.GetApiKeyAsync(key);
            }
            catch (System.Exception exception)
            {
                _logger.LogError(exception, exception.Message);
                throw;
            }
        }
    }
}
```

Scriviamo il codice del file **Program.cs** come segue:

```
//file Program.cs

using AspNetCore.Authentication.ApiKey;
using ApiKeyAuthenticationSample.Repositories;
using ApiKeyAuthenticationSample.Services;
using Microsoft.AspNetCore.Authorization;
using System.Security.Claims;
using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
//builder.Services.AddSwaggerGen();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "ApiKey Authentication",
        Description = "A simple project describing ApiKey Authentication",
        Version = "v1"
    });
    //*** le due istruzioni seguenti (in commento) impostano Swagger per utilizzare ApiKey per tutti gli endpoints
});
```

```

//https://stackoverflow.com/questions/57943550/how-to-document-api-key-authentication-using-
swashbuckle-aspnetcore-v5-0-0-rc2
//https://stackoverflow.com/a/57944310
c.AddSecurityDefinition("ApiKey", new OpenApiSecurityScheme()
{
    Type = SecuritySchemeType.ApiKey,
    In = ParameterLocation.Header,
    Name = "X-API-KEY",
    Description = "Some routes require api key. Rules are covered",
});
c.AddSecurityRequirement(new OpenApiSecurityRequirement()
{
    {
        new OpenApiSecurityScheme
        {
            Reference = new OpenApiReference { Type = ReferenceType.SecurityScheme, Id = "ApiKey" }
        },
        new string[] { }
    }
});
}

// Add User repository to the dependency container.
builder.Services.AddScoped<IApiKeyRepository, InMemoryApiKeyRepository>();

// Add the ApiKey scheme authentication here..
// It requires Realm to be set in the options if SuppressWWWAuthenticateHeader is not set.
// If an implementation of IApiKeyProvider interface is registered in the dependency register as well
// as OnValidateKey delegate on options.Events is also set then this delegate will be used instead of an
// implementation of IApiKeyProvider.
builder.Services.AddAuthentication(ApiKeyDefaults.AuthenticationScheme)

    // The below AddApiKeyInHeaderOrqueryParams without type parameter will require OnValidateKey
    // delegate on options.Events to be set unless an implementation of IApiKeyProvider interface is
    // registered in the dependency register.

    // Please note if OnValidateKey delegate on options.Events is also set then this delegate will be
    // used instead of ApiKeyProvider.*
    //.AddApiKeyInHeaderOrqueryParams(options =>

        // The below AddApiKeyInHeaderOrqueryParams with type parameter will add the ApiKeyProvider to the
        // dependency register.

        // Please note if OnValidateKey delegate on options.Events is also set then this delegate will be
        // used instead of ApiKeyProvider.
        .AddApiKeyInHeaderOrqueryParams<ApiKeyProvider>(options =>
    {
        options.Realm = "Sample Web API";
        options.KeyName = "X-API-KEY";
    }
}

```

```

    //// Optional option to suppress the browser login dialog for ajax calls.
    //options.SuppressWWWAuthenticateHeader = true;

    //// Optional option to ignore extra check of ApiKey string after it is validated.
    //options.ForLegacyIgnoreExtraValidatedApiKeyCheck = true;

    //// Optional option to ignore authentication if AllowAnonymous metadata/filter attribute is
    //added to an endpoint.
    options.IgnoreAuthenticationIfAllowAnonymous = true;

    //// Optional events to override the ApiKey original logic with custom logic.
    //// Only use this if you know what you are doing at your own risk. Any of the events can be
    //assigned.
    options.Events = new ApiKeyEvents
    {

        //// A delegate assigned to this property will be invoked just before validating the api
        //key.
        //OnValidateKey = async (context) =>
        //{
        //    // custom code to handle the api key, create principal and call Success method on
        //context.
        //    var apiKeyRepository =
        context.HttpContext.RequestServices.GetRequiredService<IApiKeyRepository>();
        //    var apiKey = await apiKeyRepository.GetApiKeyAsync(context.ApiKey);
        //    var isValid = apiKey != null && apiKey.Key.Equals(context.ApiKey,
        StringComparison.OrdinalIgnoreCase);
        //    if (isValid)
        //    {
        //        context.Response.Headers.Add("ValidationCustomHeader", "From OnValidateKey");
        //        var claims = new[]
        //        {
        //            new Claim(ClaimTypes.NameIdentifier, apiKey.OwnerName,
        ClaimValueTypes.String, context.Options.ClaimsIssuer),
        //            new Claim(ClaimTypes.Name, apiKey.OwnerName, ClaimValueTypes.String,
        context.Options.ClaimsIssuer),
        //            new Claim("CustomClaimType", "Custom Claim Value - from OnValidateKey")
        //        };
        //        context.Principal = new ClaimsPrincipal(new ClaimsIdentity(claims,
        context.Scheme.Name));
        //        context.Success();
        //    }
        //    else
        //    {
        //        context.NoResult();
        //    }
        //}

        //// A delegate assigned to this property will be invoked just before validating the api
        //key.
    }

```

```

    //// NOTE: Same as above delegate but slightly different implementation which will give
    same result.

    //OnValidateKey = async (context) =>
    //{
    //    // custom code to handle the api key, create principal and call Success method on
    context.
    //
    //    var apiKeyRepository =
    context.HttpContext.RequestServices.GetRequiredService<IApiKeyRepository>();
    //
    //    var apiKey = await apiKeyRepository.GetApiKeyAsync(context.ApiKey);
    //
    //    var isValid = apiKey != null && apiKey.Key.Equals(context.ApiKey,
    StringComparison.OrdinalIgnoreCase);
    //
    //    if (isValid)
    //
    //    {
    //        context.Response.Headers.Add("ValidationCustomHeader", "From OnValidateKey");
    //
    //        var claims = new[]
    //
    //        {
    //            new Claim("CustomClaimType", "Custom Claim Value - from OnValidateKey")
    //
    //        };
    //
    //        context.ValidationSucceeded(apiKey.OwnerName, claims); // claims are optional
    //
    //    }
    //
    //    else
    //
    //    {
    //        context.ValidationFailed();
    //
    //    }
    //
    //},
    //

    //// A delegate assigned to this property will be invoked before a challenge is sent back
    to the caller when handling unauthorized response.

    //OnHandleChallenge = async (context) =>
    //{
    //
    //    // custom code to handle authentication challenge unauthorized response.
    //
    //    context.Response.StatusCode = StatusCodes.Status401Unauthorized;
    //
    //    context.Response.Headers.Add("ChallengeCustomHeader", "From OnHandleChallenge");
    //
    //    await context.Response.WriteAsync("{\"CustomBody\":\"From OnHandleChallenge\"}");
    //
    //    context.Handled(); // important! do not forget to call this method at the end.
    //
    //},
    //

    //// A delegate assigned to this property will be invoked if Authorization fails and
    results in a Forbidden response.

    //OnHandleForbidden = async (context) =>
    //{
    //
    //    // custom code to handle forbidden response.
    //
    //    context.Response.StatusCode = StatusCodes.Status401Unauthorized;
    //
    //    context.Response.Headers.Add("ForbidCustomHeader", "From OnHandleForbidden");
    //
    //    await context.Response.WriteAsync("{\"CustomBody\":\"From OnHandleForbidden\"}");
    //
    //    context.Handled(); // important! do not forget to call this method at the end.
    //
    //},
    //

    //// A delegate assigned to this property will be invoked when the authentication
    succeeds. It will not be called if OnValidateKey delegate is assigned.

```

```

    //// It can be used for adding claims, headers, etc to the response.
    //OnAuthenticationSucceeded = (context) =>
    //{
    //    //custom code to add extra bits to the success response.
    //    context.Response.Headers.Add("SuccessCustomHeader", "From
OnAuthenticationSucceeded");
    //    var customClaims = new List<Claim>
    //{
    //        new Claim("CustomClaimType", "Custom Claim Value - from
OnAuthenticationSucceeded")
    //    };
    //    context.AddClaims(customClaims);
    //    //or can add like this - context.Principal.AddIdentity(new
ClaimsIdentity(customClaims));
    //    return Task.CompletedTask;
    //},
    //// A delegate assigned to this property will be invoked when the authentication fails.
    //OnAuthenticationFailed = (context) =>
    //{
    //    // custom code to handle failed authentication.
    //    context.Fail("Failed to authenticate");
    //    return Task.CompletedTask;
    //}
};

};

// All the requests will need to be authorized.
// Alternatively, add [Authorize] attribute to Controller or Action Method where necessary.
//builder.Services.AddAuthorization(options =>
//{
//    options.FallbackPolicy = new AuthorizationPolicyBuilder()
//        .RequireAuthenticatedUser()
//        .Build();
//});
builder.Services.AddAuthorization();

// Configure JSON options. Questo serve solo nel caso di loop nella serializzazione di oggetti.
// Nel caso di oggetti DTO senza navigation property il problema non si pone e non c'è bisogno di
impostare questa opzione
//https://stackoverflow.com/a/66562656
//https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-migrate-from-newtonsoft-how-to?pivots=dotnet-6-0#preserve-object-references-and-handle-loops
builder.Services.Configure<Microsoft.AspNetCore.Http.Json.JsonOptions>(options =>
{
    options.SerializerOptions.ReferenceHandler =
System.Text.Json.Serialization.ReferenceHandler.IgnoreCycles;
});

```

```

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthentication();      // NOTE: DEFAULT TEMPLATE DOES NOT HAVE THIS, THIS LINE IS REQUIRED AND
HAS TO BE ADDED!!!

app.UseAuthorization();

var summaries = new[]
{
    "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
};

app.MapGet("/weatherforecast", [Authorize(AuthenticationSchemes = ApiKeyDefaults.AuthenticationScheme,
Roles = "Administrator, Standard")] () =>
{
    var forecast = Enumerable.Range(1, 5).Select(index =>
        new WeatherForecast
        (
            DateTime.Now.AddDays(index),
            Random.Shared.Next(-20, 55),
            summaries[Random.Shared.Next(summaries.Length)]
        )
        .ToArray();
    return forecast;
})
.WithName("GetWeatherForecast").Produces(StatusCodes.Status200OK).Produces(StatusCodes.Status403Forbidden).Produces(StatusCodes.Status401Unauthorized);

app.MapGet("/show-claims", [Authorize(AuthenticationSchemes = ApiKeyDefaults.AuthenticationScheme,
Roles = "Administrator")] (ClaimsPrincipal user) =>
{
    return Results.Ok(user.Claims.ToList());
})
.WithName("showClaims").Produces(StatusCodes.Status200OK).Produces(StatusCodes.Status403Forbidden).Produces(StatusCodes.Status401Unauthorized);

app.MapGet("/ciao-mondo", () =>
{
    return Results.Ok(new { message = "Ciao Mondo. Questa rotta non richiede autorizzazione" });
}

```

```

    }

    .WithName("ciaoMondo").Produces(StatusCodes.Status200OK);

app.Run();

internal record WeatherForecast(DateTime Date, int TemperatureC, string? Summary)
{
    public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
}

```

Per testare il progetto ApiKeyAuthenticationSample utilizziamo Postman con il file:

ApiKeyAuthenticationSampleClient.postman\_collection.json

Il cui contenuto è:

```
{
  "info": {
    "_postman_id": "131adecd-f94d-4057-8c61-dffae040094e",
    "name": "ApiKeyAuthenticationSampleClient",
    "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
  },
  "item": [
    {
      "name": "Get Weather Forecasts no key",
      "request": {
        "auth": {
          "type": "noauth"
        },
        "method": "GET",
        "header": [],
        "url": {
          "raw": "{{base_url}}/weatherforecast",
          "host": [
            "{{base_url}}"
          ],
          "path": [
            "weatherforecast"
          ]
        }
      },
      "response": []
    },
    {
      "name": "Get Weather Forecasts with key",
      "request": {
        "method": "GET",
        "header": [],
        "url": {
          "raw": "{{base_url}}/weatherforecast",
          "host": [

```

```

        "{{base_url}}"
    ],
    "path": [
        "weatherforecast"
    ]
}
},
{
    "name": "Show Claims",
    "request": {
        "method": "GET",
        "header": [],
        "url": {
            "raw": "{{base_url}}/show-claims",
            "host": [
                "{{base_url}}"
            ],
            "path": [
                "show-claims"
            ]
        }
    },
    "response": []
}
],
"auth": {
    "type": "apikey",
    "apikey": [
        {
            "key": "value",
            "value": "Key2",
            "type": "string"
        },
        {
            "key": "key",
            "value": "X-API-Key",
            "type": "string"
        }
    ]
},
"event": [
    {
        "listen": "prerequest",
        "script": {
            "type": "text/javascript",
            "exec": [
                ""
            ]
        }
    }
]

```

```

        }
    },
    {
        "listen": "test",
        "script": {
            "type": "text/javascript",
            "exec": [
                ""
            ]
        }
    }
],
"variable": [
    {
        "key": "base_url",
        "value": "https://localhost:44304/"
    }
]
}

```

### 13.9.1.4.3. Esempio di Session based authentication/authorization

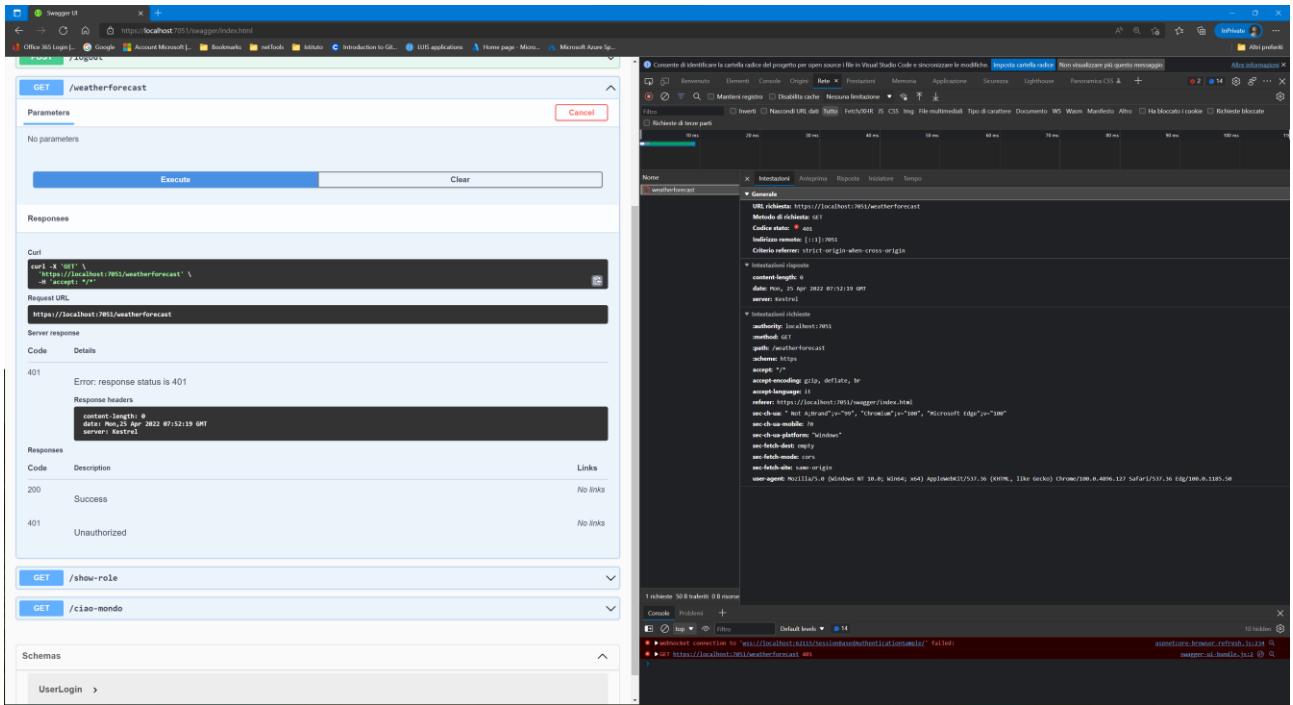
L'implementazione della authentication/authorization basato sull'uso delle sessioni può essere fatto molto semplicemente in maniera non strutturata, senza fare ricorso a pattern specifici dell'architettura .NET Core. Basta creare una sessione al login, nella quale viene inserito un campo che permette di individuare univocamente l'utente e poi verificare in tutte le richieste successive, se la richiesta proviene da un utente che ha effettuato il login e se quell'utente ha i permessi necessari per accedere alla risorsa. Questo approccio è molto semplice, ma non permette di usare i meccanismi automatici del framework .NET Core. Questo metodo viene mostrato per illustrare il concetto di sessione che sarà molto utilizzato nello sviluppo di siti web l'anno prossimo; tuttavia, il metodo preferito per implementare la cookie based authentication in .NET core è quello che verrà illustrato nel paragrafo seguente.

Questo meccanismo di autenticazione/autorizzazione rientra anch'esso nella categoria cookie based perché il server per gestire le sessioni invia un cookie di sessione al client. Questo cookie di sessione deve essere inviato dal client in tutte le richieste successive effettuate verso il server. Il cookie di sessione è di fatto una stringa utilizzata come chiave per recuperare la sessione dell'utente che è nella memoria del server. Nel caso dei browser l'invio dei cookie al server è automatico: un browser invia automaticamente i cookie al server che gli ha impostato tali cookie. Nel caso di un'applicazione client, ad esempio in C#, l'invio del cookie di sessione deve essere gestito dall'applicazione (deve essere inserito nell'header delle richieste verso il server).

Gli svantaggi di questo meccanismo di autenticazione sono quelli già visti in generale per la cookie based authentication in generale.

The screenshot displays two side-by-side browser windows. The left window shows the Swagger UI for the 'SessionBasedAuthenticationSample' API, specifically the '/login' endpoint. It includes a 'POST' request form with a JSON body containing 'username' and 'password' fields, both set to 'TestUser1'. Below the form are sections for 'Responses' (Code 200, Headers, Body) and 'Request URL' (curl command). The right window shows the Microsoft Edge developer tools Network tab. A red box highlights the 'Cookies' section for a request to 'localhost:7051/login'. It shows a cookie named 'asp.NET\_SessionId' with the value 't6mzv1jw1qk2r0xgk2zmxqfzgtr1wz'. The bottom status bar of the browser indicates 'azmettura.browser.refresh\_incid'.

Se non si effettua il login il cookie di sessione non verrà inviato e l'accesso alle rotte che richiedono login non verrà concesso:



Creiamo un progetto del tipo ASP.NET Core Web API (minimal, vale a dire senza controllers), chiamato SessionBasedAuthenticationSample.

In questo progetto, per mantenere l'esempio semplice e focalizzato sull'argomento della Session Based Authentication, assumeremo che le credenziali di accesso siano presenti in un Repository in memoria. In uno scenario reale, queste credenziali saranno recuperate da un database.

L'esempio mostrato di seguito è presente nel repository di Github:

<https://github.com/GreppiDev/Info4IA2122NetworkProgramming>

Creiamo la cartella **Models** con all'interno i seguenti file:

//file User.cs

```
namespace SessionBasedAuthenticationSample.Models
{
    public class User
    {
        public string Username { get; set; } = null!;
        public string Password { get; set; } = null!;
        public string? Role { get; set; }
    }
}
```

//file UserLogin.cs

```
namespace PizzaStoreApiKeyMariaDb.Models
{
    public class UserLogin
    {
        public string Username { get; set; } = null!;
        public string Password { get; set; } = null!;
    }
}
```

Creiamo la cartella **Repositories** con al suo interno i file:

//file IUserRepository.cs

```
using SessionBasedAuthenticationSample.Models;

namespace SessionBasedAuthenticationSample.Repositories
{
    public interface IUserRepository
    {
        Task<User?> GetUserByUsername(string username);
    }
}
```

```

        Task<IEnumerable<User>> GetUsers();
    }
}

//file InMemoryUserRepository.cs

using SessionBasedAuthenticationSample.Models;

namespace SessionBasedAuthenticationSample.Repositories
{
    public class InMemoryUserRepository : IUserRepository
    {
        private List<User> _users = new List<User>
        {
            new User { Username = "TestUser1", Password = "1234" , Role="Administrator" },
            new User { Username = "TestUser2" , Password = "1234" , Role="Standard" },
            new User { Username = "TestUser3" , Password = "1234" },
            new User { Username = "TestUser4" , Password = "1234" }
        };
        public Task<User?> GetUserByUsername(string username)
        {
            return Task.FromResult(_users.FirstOrDefault(u => u.Username == username));
        }

        public Task<IEnumerable<User>> GetUsers()
        {
            return Task.FromResult<IEnumerable<User>>(_users);
        }
    }
}

```

Scriviamo il file Program.cs con il seguente codice:

```

using Microsoft.AspNetCore.Session;
using PizzaStoreApiKeyMariaDb.Models;
using SessionBasedAuthenticationSample.Models;
using SessionBasedAuthenticationSample.Repositories;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Add User repository to the dependency container.
builder.Services.AddScoped<IUserRepository, InMemoryUserRepository>();

//Configuro le sessioni
//https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-6.0#configure-session-state
builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromSeconds(1000); //trascorso questo tempo la sessione viene
    resettata. Non il cookie di sessione
    //impostazioni per i cookie
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});

//in questo esempio non saranno usati meccanismi di autenticazione e autorizzazione forniti dal
framework
//verrà utilizzato solamente il concetto di sessione e l'applicazione applicherà i criteri di
autenticazione e autorizzazione
//definiti nella business logic dell'applicazione:
//per accedere alle rotte protette bisogna essere logged in
//per alcune rotte oltre che essere logged in bisogna anche avere un ruolo specifico nelle
impostazioni dell'account

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

```

```

}

app.UseHttpsRedirection();
//uso delle sessioni
app.UseSession();

var summaries = new[]
{
    "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
};

//Public route that let a client be identified via session cookie.
//On success the user is authenticated in subsequent protected routes
app.MapPost("/login", async (IUserRepository userRepository, HttpContext context, UserLogin user) =>
{
    if (!string.IsNullOrEmpty(user.Username) &&
        !string.IsNullOrEmpty(user.Password))
    {
        //tolgo eventuali spazi
        user.Username = user.Username.Trim();
        //verifico ci sia un utente con i campi forniti
        User? userInRepo = await userRepository.GetUserByUsername(user.Username);
        if (userInRepo != null)
        {
            //verifico la corrispondenza tra email e password
            bool validCredentials =
                user.Username == userInRepo.Username &&
                user.Password == userInRepo.Password;
            if (validCredentials)
            {
                //impostiamo lo username dell'utente dentro la sessione
                context.Session.SetString("userName", user.Username);
                return Results.Ok();
            }
        }
    }
    return Results.Unauthorized();
}).WithName("Login").Produces(StatusCodes.Status200OK).Produces(StatusCodes.Status401Unauthorized); ;

//Public route that let the client to log out if it was logged in.
//On success the user is logged out, if he was logged in. The route returns 200 Ok in any case.
app.MapPost("/Logout", (HttpContext context) =>
{
    //verifico che l'utente sia logged in
    bool loggedIn = !string.IsNullOrEmpty(context.Session.GetString("userName"));
    //se l'utente è logged in effettuo la cancellazione della sessione
    if (loggedIn)
    {
        //rimuovo tutti i dati della sessione
        context.Session.Clear();
        //invalido il cookie di sessione
        var sessionCookie = context.Request.Cookies[SessionDefaults.CookieName];
        if (sessionCookie != null)
        {
            //creo un cookie con le stesse caratteristiche del cookie di sessione,
            //ma con un expiration date nel passato
            CookieOptions option = new();
            //le impostazioni sono quelle impostate per il cookie di sessione, eccetto che per
            l'Expires
            //https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-
            6.0#session-options
            //https://stackoverflow.com/a/71373231
            option.Expires = DateTime.UtcNow.AddDays(-10);
            option.Path = SessionDefaults.CookiePath;
            option.HttpOnly = true;
            option.IsEssential = true;
            option.SameSite = SameSiteMode.Lax;
            //appendo il cookie alla risposta
            context.Response.Cookies.Append(SessionDefaults.CookieName, string.Empty, option);
        }
    }
    return Results.Ok();
}).WithName("Logout").Produces(StatusCodes.Status200OK);

app.MapGet("/weatherforecast", async (IUserRepository userRepository, HttpContext context) =>

```

```

{
    //verifico che l'utente sia logged in
    var userNameInSession = context.Session.GetString("userName");
    bool loggedIn = !string.IsNullOrEmpty(userNameInSession);
    //se l'utente è logged in vado avanti
    if (loggedIn)
    {
        //verifico i ruoli
        //recupero l'utente dal repository
        User? userInRepo = await userRepository.GetUserByUsername(userNameInSession!);
        if (userInRepo != null && (userInRepo.Role == "Administrator" || userInRepo.Role == "Standard"))
        {
            var forecast = Enumerable.Range(1, 5).Select(index =>
                new WeatherForecast
                {
                    DateTime.Now.AddDays(index),
                    Random.Shared.Next(-20, 55),
                    summaries[Random.Shared.Next(summaries.Length)]
                })
                .ToArray();
            return Results.Ok(forecast);
        }
    }
    return Results.Unauthorized();
}
.WithName("GetWeatherForecast").Produces(StatusCodes.Status200OK).Produces(StatusCodes.Status401Unauthorized);

app.MapGet("/show-role", async (IUserRepository userRepository, HttpContext context) =>
{
    //verifico che l'utente sia logged in
    var userNameInSession = context.Session.GetString("userName");
    bool loggedIn = !string.IsNullOrEmpty(userNameInSession);
    //se l'utente è logged in vado avanti
    if (loggedIn)
    {
        //verifico i ruoli
        //recupero l'utente dal repository
        User? userInRepo = await userRepository.GetUserByUsername(userNameInSession!);
        if (userInRepo != null && userInRepo.Role == "Administrator")
        {
            return Results.Ok(new { UserRole = userInRepo.Role });
        }
    }
    return Results.Unauthorized();
})
.WithName("showRole").Produces(StatusCodes.Status200OK).Produces(StatusCodes.Status401Unauthorized);

app.MapGet("/ciao-mondo", () =>
{
    return Results.Ok(new { message = "Ciao Mondo. Questa rotta non richiede autorizzazione" });
})
.WithName("ciaoMondo").Produces(StatusCodes.Status200OK);

app.Run();

internal record WeatherForecast(DateTime Date, int TemperatureC, string? Summary)
{
    public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
}

```

Come si può vedere dal metodo che effettua il logout, per eliminare una sessione si procede in due step:

- 1) si eliminano i dati dalla sessione - `context.Session.Clear();`
- 2) si invalida il cookie di sessione – si crea un cookie con le stesse opzioni del cookie di sessione, ad eccezione del parametro `Expires`, che viene impostato ad un valore già passato nel tempo. Il valore del cookie di sessione viene posto a `String.Empty`. In

questo modo il browser non manderà più in automatico il cookie di sessione (perché scaduto). Se l'applicazione client non fosse un browser, e provasse comunque a inviare il cookie, l'accesso non sarebbe concesso perché il cookie sarebbe comunque scaduto.

#### 13.9.1.4.4. Esempio di cookie based authentication/authorization (senza sessioni)

<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/cookie?view=aspnetcore-6.0>

<https://docs.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-6.0>

Questo tipo di autenticazione utilizza i cookie per individuare univocamente l'utente che effettua la richiesta e per autorizzarne l'accesso alle rotte dell'applicazione, ma, a differenza del meccanismo session based, in questo caso non sono utilizzate le sessioni per riconoscere l'utente. L'identificativo dell'utente, assieme ad altre attestazioni sull'utente, come ad esempio, il nome, la e-mail, il ruolo, etc. (i cosiddetti claims), sono memorizzate all'interno di un cookie di autorizzazione "auth\_cookie". Le informazioni inserite nel cookie di autorizzazione sono crittografate e firmate digitalmente dal server per evitare che possano essere accedute, oppure contraffatte da utenti malevoli. Il cookie di autorizzazione viene inviato dall'applicazione client per tutte le richieste fatte dopo il login, in maniera analoga a quanto fatto nel caso delle sessioni. In questo caso, però, non c'è bisogno di accedere al database per riconoscere l'utente: l'identità dell'utente e i suoi claims sono già nel cookie di autorizzazione. Questo aspetto rende interessante questo meccanismo di autenticazione e autorizzazione perché non necessita delle sessioni in memoria nel server (e questo aspetto è importante quando si realizzano applicazioni multi-server con reverse-proxy come bilanciatori di carico) e non necessita nemmeno dell'accesso al database per verificare i permessi/ruoli dell'utente, poiché questi sono crittografati nel cookie di autorizzazione. In questo modo si migliorano anche le performance dell'applicazione web che non deve necessariamente accedere al database per andare a recuperare le informazioni di base dell'utente.

Gli svantaggi di questo meccanismo di autenticazione sono i soliti associati all'utilizzo dei cookie: possibilità di attacchi di tipo Cross-Site Request Forgeries (CSRF) e XSS. Tuttavia questi attacchi possono essere mitigati con l'utilizzo di opportune tecniche di programmazione. Per approfondimenti, si veda, ad esempio, la guida ufficiale Microsoft:

<https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery?view=aspnetcore-6.0>

Con il meccanismo di autenticazione basato sui cookie, il cookie inviato dal client è l'elemento distintivo dell'identità dell'utente. Se ci dovessero essere delle modifiche fatte dal backend dell'applicazione sui permessi dell'utente, (ad esempio, venisse cambiato il ruolo dell'utente), queste verrebbero riflesse in cambiamenti effettivi sulle autorizzazioni del cookie di autorizzazione solo dopo che l'utente ha effettuato un logout, oppure è stato generato un nuovo cookie di autorizzazione in seguito ad una nuova richiesta da parte del client. Nel caso si voglia avere un controllo puntuale sul cookie di autorizzazione, per fare in modo che questo rifletta le modifiche fatte dal backend ci sono due strade perseguitibili:

- rinnovare periodicamente il cookie di autorizzazione con intervalli periodici di pochi minuti (ad esempio 5, 10 minuti) quando ci sono nuove richieste da parte del client
- effettuare la validazione del cookie ad ogni richiesta per verificare che le informazioni presenti nel cookie di autorizzazione siano compatibili con le informazioni presenti nel database.

Nel primo caso si privilegiano le performance a discapito del controllo assoluto sui permessi dell'utente che potrebbero essere cambiati in un lasso temporale ristretto.

Nel secondo caso si privilegia la sicurezza in senso assoluto a discapito delle performance dell'applicazione server che deve effettuare una validazione del cookie di autorizzazione e un accesso al database ad ogni nuova richiesta effettuata con il cookie di autorizzazione.

**La scelta se adottare una strategia, piuttosto che un'altra dipende dal contesto applicativo.**

Creiamo un progetto del tipo ASP.NET Core Web API (minimal, vale a dire senza controllers), chiamato CookieBasedAuthenticationSample.

In questo progetto, per mantenere l'esempio semplice e focalizzato sull'argomento della Cookie Based Authentication, assumeremo che le credenziali di accesso siano presenti in un Repository in memoria. In uno scenario reale, queste credenziali saranno recuperate da un database.

L'esempio mostrato di seguito è presente nel repository di Github:

<https://github.com/GreppiDev/Info4IA2122NetworkProgramming>

Creiamo la cartella **Models** con all'interno i seguenti file:

//file Users.cs

```
namespace CookieBasedAuthenticationSample.Models
{
    public class User
    {
        public string Username { get; set; } = null!;
        public string Password { get; set; } = null!;
        public string? Role { get; set; }
        public DateTime LastChanged { get; set; }
    }
}
```

//file UserLogin.cs

```
namespace CookieBasedAuthenticationSample.Models
{
    public class UserLogin
    {
        public string Username { get; set; } = null!;
        public string Password { get; set; } = null!;
    }
}
```

Creiamo la cartella **Repositories** con dentro i file:

//file IUserRepository.cs

```
using CookieBasedAuthenticationSample.Models;

namespace CookieBasedAuthenticationSample.Repositories
{
    public interface IUserRepository
    {
        Task<User?> GetUserByUsername(string username);
        Task<IEnumerable<User>> GetUsers();
        public Task<bool> ValidateLastChangedAsync(string? userIdentifer, string?
lastChanged);
    }
}
```

//file InMemoryUserRepository.cs

```
using CookieBasedAuthenticationSample.Models;
using System.Globalization;

namespace CookieBasedAuthenticationSample.Repositories
{
    public class InMemoryUserRepository : IUserRepository
    {
        private List<User> _users = new List<User>
        {
            new User { Username = "TestUser1", Password = "1234" , Role="Administrator",
LastChanged=DateTime.UtcNow },
            new User { Username = "TestUser2", Password = "1234" , Role="Standard",
LastChanged=DateTime.UtcNow },
        };
    }
}
```

```

        new User { Username = "TestUser3", Password = "1234" ,
LastChanged=DateTime.UtcNow},
        new User { Username = "TestUser4", Password = "1234",
LastChanged=DateTime.UtcNow }
    };
    public Task<User?> GetUserByUsername(string username)
{
    return Task.FromResult(_users.FirstOrDefault(u => u.Username == username));
}

public Task<IEnumerable<User>> GetUsers()
{
    return Task.FromResult<IEnumerable<User>>(_users);
}

public async Task<bool> ValidateLastChangedAsync(string? userNameIdentifier, string?
lastChanged)
{
    //recupero l'utente dal repository. In una situazione reale l'utente verrà
    //prelevato dal database
    if (userNameIdentifier != null && lastChanged != null &&
DateTime.TryParse(lastChanged, CultureInfo.InvariantCulture, DateTimeStyles.RoundtripKind, out
DateTime lChanged))
    {

        var userInRepo = _users.Where(u=>
u.Username==userNameIdentifier).FirstOrDefault();
        if (userInRepo != null)
        {
            //se un utente è stato modificato dopo il lastChanged recuperato
            //dal claim del cookie bisogna invalidare il cookie
            return await Task.FromResult(userInRepo.LastChanged <= lChanged);
        }
    }
    return false;
}
}
}

```

Creiamo la cartella **Events** con dentro il file CustomCookieAuthenticationEvents.cs

```

//file CustomCookieAuthenticationEvents.cs

using CookieBasedAuthenticationSample.Repositories;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using System.Security.Claims;

namespace CookieBasedAuthenticationSample.Events
{
    public class CustomCookieAuthenticationEvents : CookieAuthenticationEvents
    {
        private readonly IUserRepository _userRepository;

        public CustomCookieAuthenticationEvents(IUserRepository userRepository)
        {
            _userRepository = userRepository;
            //https://www.blinkingcaret.com/2018/07/18/secure-an-asp-net-core-web-api-using-cookies/
            OnRedirectToLogin = redirectContext =>
            {
                redirectContext.HttpContext.Response.StatusCode = 401;
                return Task.CompletedTask;
            };
            //questo serve per fare in modo che quando un utente è logged in, ma non ha i permessi
            //richiesti
            //venga restituito il codice 403 (forbidden) al posto del valore di default 404 (che è not
            //found)
            OnRedirectToAccessDenied = redirectContext =>
            {
                redirectContext.HttpContext.Response.StatusCode = 403;
                return Task.CompletedTask;
            };
        }
    }
}

```

```

    //https://docs.microsoft.com/en-us/aspnet/core/security/authentication/cookie?view=aspnetcore-6.0#react-to-back-end-changes
    //la validazione del cookie di autorizzazione non è obbligatoria e dipende dal tipo di applicazione
    //fare la validazione del cookie di autorizzazione ad ogni richiesta ha un impatto sulle prestazioni
    public override async Task ValidatePrincipal(CookieValidatePrincipalContext context)
    {
        var userPrincipal = context.Principal;

        var lastChanged = userPrincipal?.Claims.Where(c => c.Type == "LastChanged").Select(c => c.Value).FirstOrDefault();
        var userNameIdentifier = userPrincipal?.Claims.Where(c => c.Type == ClaimTypes.NameIdentifier).Select(c => c.Value).FirstOrDefault();

        if (!string.IsNullOrEmpty(lastChanged) && !string.IsNullOrEmpty(userNameIdentifier) &&
            await _userRepository.ValidateLastChangedAsync(userNameIdentifier, lastChanged))
        {
            return;
        }
        context.RejectPrincipal();

        await context.HttpContext.SignOutAsync(
            CookieAuthenticationDefaults.AuthenticationScheme);
    }
}

```

Scriviamo il codice del file **Program.cs** come segue:

```

//file Program.cs

using CookieBasedAuthenticationSample.Events;
using CookieBasedAuthenticationSample.Models;
using CookieBasedAuthenticationSample.Repositories;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Authorization;
using System.Security.Claims;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Configure JSON options. Questo serve solo nel caso di loop nella serializzazione di oggetti. Ad esempio quando si serializza un oggetto Key che ha un riferimento
//di tipo User, il serializzatore scopre che User ha una collection di Key e per ognuno di questi effettua ancora una serializzazione, andando in loop. Per evitare questo
//si può istruire il serializzatore di ignorare i loop. Nel caso di oggetti DTO senza navigation property il problema non si pone e non c'è bisogno di impostare questa opzione
//https://stackoverflow.com/a/66562656
//https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-migrate-from-newtonsoft-how-to?pivots=dotnet-6-0#preserve-object-references-and-handle-loops
builder.Services.Configure<Microsoft.AspNetCore.Http.Json.JsonOptions>(options =>
{
    options.SerializerOptions.ReferenceHandler =
System.Text.Json.Serialization.ReferenceHandler.IgnoreCycles;
});

//Configuro Cookie based authentication
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(options =>
{
    options.ExpireTimeSpan = TimeSpan.FromMinutes(20);
    options.SlidingExpiration = true;
    //options.AccessDeniedPath = "/Forbidden/";
    //https://www.blinkingcaret.com/2018/07/18/secure-an-asp-net-core-web-api-using-cookies/
    options.Cookie.Name = "auth_cookie";
})

```

```

        options.Cookie.SameSite = SameSiteMode.None;
        //https://www.blinkingcaret.com/2018/07/18/secure-an-asp-net-core-web-api-using-cookies/
        options.EventsType = typeof(CustomCookieAuthenticationEvents);
    });
builder.Services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();
//in questo caso si è scelto di utilizzare un AdSingleton per InMemoryRepository per avere un'unica
istanza in memoria della lista
//degli utenti e fare in modo che i tempi del LastChanged fossero univoci e corrispondenti a quelli in
cui è stato fatto partire il servizio
//se avessimo utilizzato un AddScoped per InMemoryUserRepository l'esempio non avrebbe funzionato
perché con AdScoped ad ogni richiesta
//viene ricreata un'istanza di InMemoryUserRepository e quindi anche i tempi relativi a LastChanged
sarebbero stati calcolati in base al nuovo
//valore di DateTime.UtcNow invalidando tutto il meccanismo.
//In un esempio reale con database si può tranquillamente utilizzare AdScoped perché i tempi del
LastChanged non sono simulati, ma corrispondono
//ai tempi nei quali il database ha modificato i dati dell'utente
builder.Services.AddSingleton<IUserRepository, InMemoryUserRepository>();
builder.Services.AddScoped<CustomCookieAuthenticationEvents>();

builder.Services.AddAuthorization();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseAuthentication();
app.UseAuthorization();

var summaries = new[]
{
    "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
};

//Public route that let a client be identified via authentication cookie.
//On success the user is authenticated in subsequent protected routes
app.MapPost("accounts/login", async (IUserRepository userRepository, IHttpContextAccessor ctxAccessor,
UserLogin user) =>
{
    if (!string.IsNullOrEmpty(user.Username) &&
        !string.IsNullOrEmpty(user.Password))
    {
        //tolgo eventuali spazi
        user.Username = user.Username.Trim();
        //verifico ci sia un utente con i campi forniti
        User? userInRepo = await userRepository.GetUserByUsername(user.Username);
        if (userInRepo != null)
        {
            //verifico la corrispondenza tra email e password
            bool validCredentials =
                user.Username == userInRepo.Username &&
                user.Password == userInRepo.Password;
            if (validCredentials)
            {
                //utente logged in
                //impostiamo il cookie di autorizzazione
                await SetAuthorizationCookie(ctxAccessor, userInRepo);
                return Results.NoContent();
            }
        }
    }
    return Results.Unauthorized();
}).WithName("Login");
Produces(StatusCodes.Status204NoContent);
Produces(StatusCodes.Status401Unauthorized);

```

```

//Public route that let the client to log out if it was logged in.
//On success the user is logged out, if he was logged in.
app.MapPost("accounts/logout", (IHttpContextAccessor ctxAccessor) =>
{
    ctxAccessor.HttpContext?.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
    return Results.NoContent();
}).WithName("logout").Produces(StatusCodes.Status204NoContent);

app.MapGet("/weatherforecast", [Authorize(AuthenticationSchemes =
CookieAuthenticationDefaults.AuthenticationScheme, Roles = "Standard,Administrator")] () =>
{
    var forecast = Enumerable.Range(1, 5).Select(index =>
        new WeatherForecast
        (
            DateTime.Now.AddDays(index),
            Random.Shared.Next(-20, 55),
            summaries[Random.Shared.Next(summaries.Length)])
    )
    .ToArray();
    return forecast;
})
.WithName("GetWeatherForecast").Produces(StatusCodes.Status200OK).Produces(StatusCodes.Status403Forbidden).Produces(StatusCodes.Status401Unauthorized);

app.MapGet("/show-claims", [Authorize(AuthenticationSchemes =
CookieAuthenticationDefaults.AuthenticationScheme, Roles = "Administrator")] (ClaimsPrincipal user) =>
{
    return Results.Ok(user.Claims.ToList());
})
.WithName("showClaims").Produces(StatusCodes.Status200OK).Produces(StatusCodes.Status403Forbidden).Produces(StatusCodes.Status401Unauthorized);

app.MapGet("/ciao-mondo", () =>
{
    return Results.Ok(new { message = "Ciao Mondo. Questa rotta non richiede autorizzazione" });
})
.WithName("ciaoMondo").Produces(StatusCodes.Status200OK);

app.Run();

```

```

static async Task SetAuthorizationCookie(IHttpContextAccessor ctxAccessor, User user)
{
    //tutte le volte che cambiamo i dati dell'utente Role, EmailAddress, IsEmailConfirmed dobbiamo
    reinviare il cookie al client con i nuovi dati
    var claims =new List<Claim>
    {
        new Claim(ClaimTypes.NameIdentifier, user.Username),
        new Claim("LastChanged", user.LastChanged.ToString("O"))//l'opzione O garantisce il round
    trip della conversione da DateTime a string e viceversa
    };
    if (user.Role != null)
    {
        claims.Add(new Claim(ClaimTypes.Role, user.Role));
    }
    var claimsIdentity = new ClaimsIdentity(
        claims, CookieAuthenticationDefaults.AuthenticationScheme);

    var authProperties = new AuthenticationProperties
    {
        //AllowRefresh =<bool>
        // Refreshing the authentication session should be allowed.

        //ExpiresUtc = DateTimeOffset.UtcNow.AddMinutes(10),
        // The time at which the authentication ticket expires. A
        // value set here overrides the ExpireTimeSpan option of
        // CookieAuthenticationOptions set with AddCookie.

        //IsPersistent = true,
        // Whether the authentication session is persisted across
        // multiple requests. When used with cookies, controls
    };
}

```

```

// whether the cookie's lifetime is absolute (matching the
// lifetime of the authentication ticket) or session-based.

//IssuedUtc = <DateTimeOffset>,
// The time at which the authentication ticket was issued.

//RedirectUri = <string>
// The full path or absolute URI to be used as an http
// redirect response value.

};

if (ctxAccessor.HttpContext != null)
{
    await ctxAccessor.HttpContext.SignInAsync(
        CookieAuthenticationDefaults.AuthenticationScheme,
        new ClaimsPrincipal(claimsIdentity),
        authProperties);
}

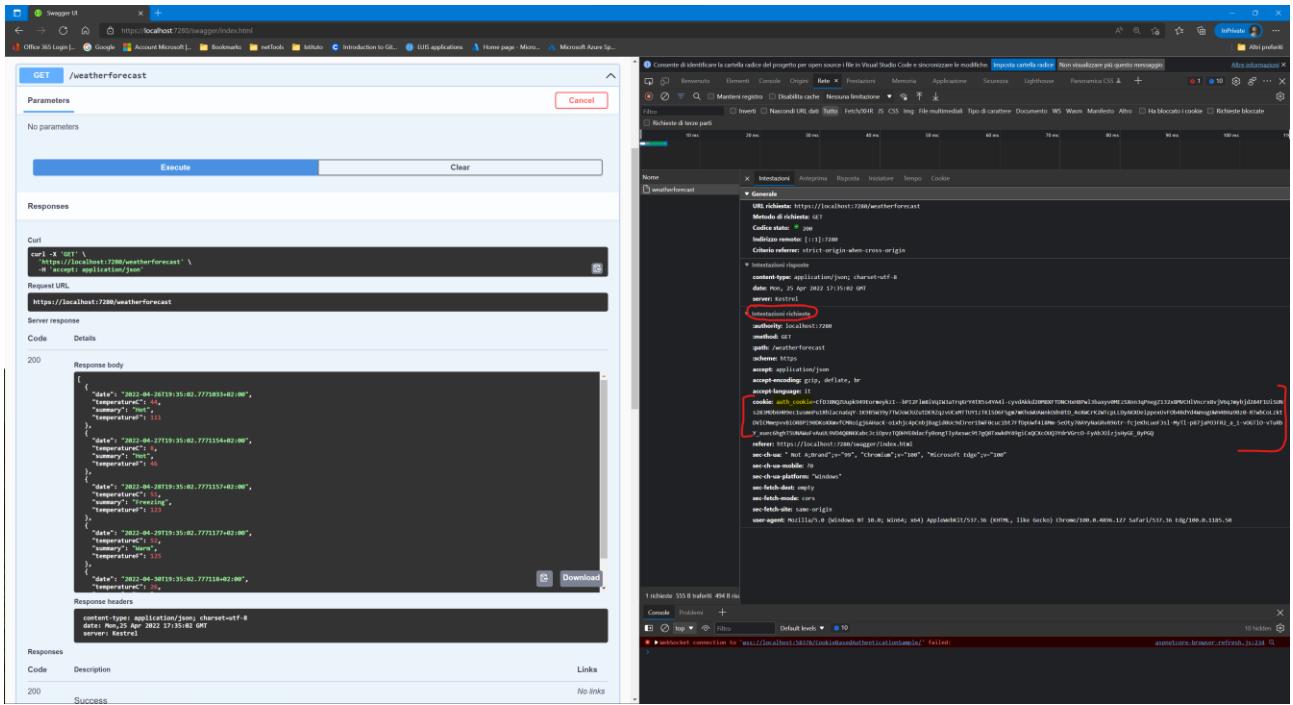
internal record WeatherForecast(DateTime Date, int TemperatureC, string? Summary)
{
    public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
}

```

Quando si effettua il login il server risponde con il cookie di autorizzazione “auth\_cookie”. Questo cookie contiene un security ticket con al suo interno le informazioni sull’identità dell’utente e sui suoi claims. Le informazioni sono criptate e firmate digitalmente dal server.

The screenshot shows a browser developer tools Network tab. A POST request to '/accounts/login' is selected. In the 'Cookies' section, a cookie named 'auth\_cookie' is listed. The cookie value is obscured by a large red box. The cookie is marked as 'Secure' and 'HttpOnly'. The URL of the request is https://localhost:7288/accounts/login.

Quando si effettua una qualunque altra richiesta al server, il browser invia il cookie di autorizzazione:



Il server legge il cookie di autorizzazione (lo decripta e controlla la firma digitale) ed estrae da esso le informazioni sull'identità e sui claims.

### 13.9.1.4.5.PizzaStoreAPI - Un esempio completo di Web API con user management e key management

Si vuole realizzare un servizio di Web API che consenta di accedere ad un servizio per la creazione di oggetti di tipo Pizza (con nome e descrizione). La particolarità di questo servizio è che oltre alle rotte per l'accesso alle API delle Pizze **/api/pizzas** (con le solite CRUD operations) ci sarà anche la gestione degli utenti **/accounts** (sviluppatori di software) che si iscrivono al servizio per poter creare delle chiavi da usare per le loro applicazioni, ed, ovviamente, la gestione delle chiavi **/keys**.

Ogni utente che si iscrive alla piattaforma crea un proprio account con una serie di dati personali, tra cui anche la e-mail, che dovrà essere verificata tramite un servizio di invio e-mail.

Gli utenti, iscritti alla piattaforma e che hanno verificato la propria e-mail, possono creare delle chiavi per poter accedere al servizio **/api/pizzas**. Una volta creata una chiave sarà possibile effettuare CRUD operations sulle api delle pizze senza doversi più autenticare, ma semplicemente inserendo la chiave nell'header o nella query string della richiesta.

Questo servizio denominato PizzaStore API è simile, come tipologia, a quello offerto da OpenWeatherMap (<https://openweathermap.org/>): anche in quel caso ci si registra al servizio e poi si possono creare delle chiavi da associare alle applicazioni che si svilupperanno. Ciascuna di queste chiavi permette di identificare l'applicazione client (il progetto), ma non il singolo utente che utilizza l'applicazione client. Ad esempio, se facessimo l'applicazione per Android MyCrazyPizza che utilizza i servizi di <https://pizzastoreapi.azurewebsites.net/api/pizzas>, questa applicazione sarebbe autenticata/autorizzata nei confronti della Web API attraverso la chiave che il suo sviluppatore ha utilizzato per l'applicazione. In nessun modo la Web API app potrebbe distinguere le richieste che arrivano da un particolare utente dell'applicazione MyCrazyPizza da quelle che provengono da un altro utilizzatore della stessa app. L'unica cosa che la Web API app potrebbe sapere è qual è il progetto per cui è stata concessa la chiave di accesso al servizio **/api/pizzas** ed eventualmente applicare un billing (tariffazione) sulla base del numero di richieste che arrivano per chiave di progetto, oppure delle restrizioni sulla base del tipo di chiave utilizzata.

Le funzionalità da implemtare sono:

#### User management:

- 1) Ci si registra con una e-mail valida (arriva l'e-mail per confermare l'account – eventualmente bisogna controllare nella casella dello spam). La password deve essere almeno di 5 caratteri, con almeno un carattere maiuscolo, uno minuscolo e un carattere speciale.
- 2) Quando arriva la e-mail si clicca sul link per validare la e-mail, oppure si fa la post su /accounts/verify-e-mail con il token usa getta che è stato inviato. Fino a che non si fa la validazione della e-mail alcune operazioni non sono abilitate (deve essere implementato un middleware per una policy che verifica se l'utente ha l'e-mail verificata).
- 3) Si fa il login con e-mail e password scelte in fase di registrazione
- 4) Si può controllare il proprio profilo, eventualmente si può anche modificarlo (anche solo parzialmente, ad esempio mettendo solo le properties del nome oppure la e-mail...)
- 5) Se si dimentica la password c'è la possibilità di recuperarla con una post su /accounts/forgot-password fornendo la propria e-mail. Se l'indirizzo e-mail è registrato nel sistema viene inviato un messaggio di posta a quell'indirizzo con un token per resettare la password tramite una POST sulla rotta /accounts/reset-password.
- 6) Gli utenti che si registrano autonomamente sono Standard users. Solo un Administrator può cambiare il ruolo di un utente mediante una POST su /accounts/change-role
- 7) Solo gli Administrator possono vedere ed eventualmente cancellare gli account che si sono registrati sulla piattaforma.

#### Key Management:

- 8) Un utente registrato e con e-mail verificata può creare fino a 10 chiavi. Ogni chiave ha un nome, un valore e una expiration date. Il valore delle chiavi è una stringa generata con un algoritmo crittografico ed è unica nel database.
- 9) Una volta che un utente ha una chiave può usarla per utilizzare il servizio /api/pizzas
- 10) devono essere implementati due meccanismi di sicurezza:
  - a. cookie based authentication/authorization per lo user management e il key management
  - b. api key authentication/authorization per il servizio /api/pizzas
- 11) Le password e i token di sicurezza devono essere generati con algoritmi crittografici. Un token ha una validità temporale di 24 ore e una volta utilizzato non può più essere riutilizzato.

#### Pizza API Management:

- 12) Visualizzazione e gestione delle pizze per tutti gli utenti in possesso di una chiave valida:

Creazione e modifica di pizze solo per utenti che hanno il ruolo di amministratori (una stessa chiave può abilitare uno o più servizi in base al ruolo dell'utente che possiede quella chiave). Ad esempio, se una chiave è di proprietà di un utente standard potrà solo far vedere le pizze. Se una chiave è di proprietà di un utente amministratore, quella chiave potrà fare anche creare, modificare o cancellare delle pizze.

In questo progetto verrà utilizzato in localhost il DBMS MariaDb, mentre per il deployment su Azure verrà utilizzato Azure SQL Server. Questa scelta è dettata dal fatto che la licenza studenti di Azure permette un uso limitato di MariaDb, mentre SQL Server è gratuito senza particolari limiti (12 mesi). L'utilizzo dell'ORM Entity Framework Core (EF Core) renderà il passaggio da un DBMS ad un altro praticamente impercettibile a meno del cambiamento di pochissime righe di codice.

In questo esempio verranno utilizzati i seguenti componenti:

- la libreria MailKit per l'invio delle email: <https://github.com/jstedfast/MailKit> - NuGet: MailKit
- una classe per la gestione dei servizi crittografici necessari per la generazione delle chiavi, delle password e dei token per la validazione delle email o per il recupero della password.

La libreria MailKit rende piuttosto semplice l'invio di una email (permette anche la ricezione con IMAP o POP3). Ad esempio, per inviare una email basta scrivere un codice come il seguente:

<https://github.com/jstedfast/MailKit#sending-messages>

```
using System;

using MailKit.Net.Smtp;
using MailKit;
using MimeKit;

namespace TestClient {
    class Program
    {
        public static void Main (string[] args)
        {
            var message = new MimeMessage ();
            message.From.Add (new MailboxAddress ("Joey Tribbiani", "joey@friends.com"));
            message.To.Add (new MailboxAddress ("Mrs. Chanandler Bong", "chandler@friends.com"));
            message.Subject = "How you doin'?;

            message.Body = new TextPart ("plain") {
                Text = @"Hey Chandler,
I just wanted to let you know that Monica and I were going to go play some paintball, you in?
-- Joey"
            };

            using (var client = new SmtpClient ()) {
                client.Connect ("smtp.friends.com", 587, false);

                // Note: only needed if the SMTP server requires authentication
                client.Authenticate ("joey", "password");

                client.Send (message);
                client.Disconnect (true);
            }
        }
    }
}
```

Per inviare una email è possibile utilizzare un proprio account di posta, ma in tal caso bisogna verificare quali sono le impostazioni del proprio provider in termini di protocolli di sicurezza e di porte utilizzate. A questo link: <https://jasonwatmore.com/post/2022/03/11/net-6-send-an-email-via-mailkit>

[smtp-with-mailkit](#) è possibile trovare un tutorial che spiega come configurare la connessione con diversi provider di posta. In questo esempio useremo un account di posta fake, utile per fare prove, chiamato ethereal.email <https://ethereal.email/>. Un altro servizio simile è <https://mailtrap.io/>.

Con ethereal.email basta un click per creare un account di posta fake:

<https://ethereal.email/create>

Per la gestione degli algoritmi crittografici verrà utilizzata la classe CryptoService:

```
using Microsoft.AspNetCore.Cryptography.KeyDerivation;
using System.Security.Cryptography;
using System.Text;

namespace PizzaStoreApiKey.Services
{
    public class CryptoService : ICryptoService
    {
        internal static readonly char[] chars =
            "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890".ToCharArray();

        public string GenerateKeyValue(int keySize=30)
        {
            //https://stackoverflow.com/a/1344255
            byte[] data = new byte[4 * keySize];
            using (var crypto = RandomNumberGenerator.Create())
            {
                crypto.GetBytes(data);
            }
            StringBuilder result = new StringBuilder(keySize);
            for (int i = 0; i < keySize; i++)
            {
                var rnd = BitConverter.ToInt32(data, i * 4);
                var idx = rnd % chars.Length;

                result.Append(chars[idx]);
            }
            return result.ToString();
        }

        /// <summary>
        /// Esegue l'hash della password
        /// </summary>
        /// <param name="password">password di cui si vuole calcolare l'hash</param>
        /// <param name="salt">salt utilizzato per effettuare l'hash</param>
        /// <param name="keySize">La dimensione in bit dello hash generato (multiplo di 8)</param>
        /// <returns>L'hash della password come array di byte</returns>
        public byte[] GeneratePasswordHash(string password, byte[] salt, int keySize=256)
        {
            return HashingAlgorithm(password, salt, keySize);
        }

        /// <summary>
        /// Esegue l'hash della password
        /// </summary>
        /// <param name="password">password di cui si vuole calcolare l'hash</param>
        /// <param name="salt">salt utilizzato per effettuare l'hash</param>
        /// <param name="keySize">La dimensione in bit dello hash generato (multiplo di 8)</param>
        /// <returns>L'hash della password come array di byte</returns>
        private static byte[] HashingAlgorithm(string password, byte[] salt, int keySize = 256)
        {
            //https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/consumer-apis/password-hashing?view=aspnetcore-6.0
            // derive a 256-bit subkey (use HMACSHA256 with 100,000 iterations)
            return KeyDerivation.Pbkdf2(
                password: password,
                salt: salt,
                prf: KeyDerivationPrf.HMACSHA256,
                iterationCount: 100000,
                numBytesRequested: keySize / 8);
        }

        /// <summary>
        /// Genera un salt per effettuare un algoritmo di hashing su una password
        /// </summary>
        /// <param name="saltSize">La dimensione in bit del salt generato (multiplo di 8)</param>
        /// <returns>Il salt come array di byte</returns>
        public byte[] GenerateSalt(int saltSize=128)
```

```
        //https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/consumer-apis/password-hashing?view=aspnetcore-6.0
        byte[] salt = new byte[saltSize / 8];
        using (var randomGenerator = RandomNumberGenerator.Create())
        {
            randomGenerator.GetNonZeroBytes(salt);
        }
        return salt;
    }
    /// <summary>
    /// Verifica se passwordToValidate è valida. Calcola l'hashing di passwordToValidate con lo stesso algoritmo e lo stesso salt usato per creare
    /// l'hash dell password salvato nel database. Se i due hash coincidono la password inserita è valida
    /// </summary>
    /// <param name="passwordToValidate"></param>
    /// <param name="salt"></param>
    /// <param name="passwordHash"></param>

    /// <returns>true se passwordToValidate produce lo stesso hashing di passwordHash con il medesimo salt e algoritmo di hashing</returns>
    public bool IsPasswordValid(string passwordToValidate, byte[] salt, byte[] passwordHash)
    {
        //https://stackoverflow.com/a/4331313
        byte[] calculatedHash = HashingAlgorithm(passwordToValidate, salt, passwordHash.Length*8);
        return calculatedHash.SequenceEqual(passwordHash);
    }

    /// <summary>
    /// Genera un token per la validazione della mail, oppure per il recupero della password.
    /// Genera un hashing con algoritmo SHA256 a partire da dati pseudo-casuali. Il risultato è codificato con codifica Base64. La lunghezza della stringa risultato è data da:
    /// 4*(256/8)/3 --> 42.66 (deve essere un multiplo di 4)--> 44 caratteri
    /// </summary>
    /// <returns>il token </returns>
    public string GenerateToken()
    {
        //https://stackoverflow.com/a/14644367
        byte[] time = BitConverter.GetBytes(DateTime.UtcNow.ToBinary());
        byte[] key = Guid.NewGuid().ToByteArray();

        var token = time.Concat(key).ToArray();
        string hex;
        //https://stackoverflow.com/a/70351657
        using (var alg = SHA256.Create())
        {
            var hashValue = alg.ComputeHash(token);
            //https://stackoverflow.com/questions/13378815/base64-length-calculation
            //https://stackoverflow.com/a/311179
            //https://en.wikipedia.org/wiki/Base64
            hex = Convert.ToString(hashValue);
        }
        return hex;
    }
}
```

La classe CryptoService implementa l'interfaccia ICryptoService:

```
namespace PizzaStoreApiKey.Services
{
    public interface ICryptoService
    {
        /// <summary>
        /// Genera un token di sicurezza
        /// </summary>
        /// <returns>il token di sicurezza</returns>
        public string GenerateToken();
        /// <summary>
        /// Genera una chiave di lunghezza data
        /// </summary>
        /// <param name="keySize">La lunghezza della chiave in numero di caratteri</param>
        /// <returns>La chiave generata</returns>
        public string GenerateKeyValue(int keySize = 30);
        /// <summary>
```

```

    /// Restituisce un salt per la generazione dell'hash di una chiave, usando un algoritmo
    crittografico - System.Security.Cryptography.RandomNumberGenerator.
    /**
     * </summary>
     * <param name="saltSize">Numero di byte del salt</param>
     * <returns>Il salt come array di byte</returns>
    public byte[] GenerateSalt(int saltSize = 128);
    /**
     * Genera l'hash della password con il salt fornito
     * </summary>
     * <param name="password">password di cui si vuole creare l'hash</param>
     * <param name="salt">salt per la generazione dell'hash</param>
     * <param name="keySize">numero di byte dell'hash</param>
     * <returns></returns>
    public byte[] GeneratePasswordHash(string password, byte[] salt, int keySize = 256);
    /**
     * Verifica che passwordToValidate sia valida, calcolando l'hash con il salt e confrontando
     il risultato
     * con passwordHash memorizzato.
     * </summary>
     * <param name="passwordToValidate">password da validare</param>
     * <param name="salt">salt per generare l'hash della password da validare</param>
     * <param name="passwordHash">l'hash con cui confrontare il risultato dell'hash
     calcolato</param>
     * <param name="hashingAlgorithm">algoritmo di hashing da utilizzare</param>
     * <returns></returns>
    public bool IsPasswordValid(string passwordToValidate, byte[] salt, byte[] passwordHash);

}
}

```

Questi algoritmi non verranno discussi in dettaglio, poiché molti concetti verranno ripresi in sistemi e reti l'anno prossimo, ma ci si limiterà a capirne l'utilità e il funzionamento nell'ambito del progetto corrente.

Questo è un esempio abbastanza completo e con molte classi, pertanto non si partirà con la scrittura del progetto da zero, ma si discuterà direttamente il progetto che può essere scaricato direttamente da Github: <https://github.com/GreppiDev/Info4IA2122NetworkProgramming>

### 13.9.1.4.5.1. Setup del progetto a partire dall'esempio su GitHub

Scaricare il progetto di esempio StudentVersion/PizzaStoreApiKey da GitHub:

<https://github.com/GreppiDev/Info4IA2122NetworkProgramming>.

Aprire il file appsettings.json per configurare la connessione al database e i parametri necessari per far funzionare l'invio delle email.

Inizialmente il file di appsettings.json si presenta come riportato di seguito:

```
{
  "ConnectionStrings": {
    "PizzaStoreConnection": 
      "Server=localhost;port=3306;Database=pizzastore_apikey_cookie;UserId=root;Password=root;" 
  },
  "AppSettings": {
    "AdminInitialPassword": "password primo account admin",
    "AdminInitialEmail": "email primo account admin",
    "AdminSurname": "Admin surname",
    "AdminGivenName": "Admin name",
    "MaxNumberOfKeysPerUser": 10,
    "EmailFrom": "ethereal email address",
    "EmailSenderName": "Pizza Store API",
    "SmtpHost": "ENTER YOUR OWN SMTP OPTIONS OR CREATE FREE TEST ACCOUNT IN ONE CLICK
AT https://ethereal.email/",
    "SmtpPort": 587,
    "SmtpUser": "ethereal email address",
    "SmtpPass": "ethereal password",
  }
}
```

```

    "GenerateMessageIdFrom": "ethereal.email"
},
"Logging": {
    "LogLevel": {
        "Default": "Information",
        "Microsoft.AspNetCore": "Warning"
    }
},
"AllowedHosts": "*"
}

```

Il progetto utilizza una stringa di connessione verso un database di MariaDb in localhost chiamato pizzastore\_apikey\_cookie con le credenziali di accesso di root. Modificare eventualmente username e password in modo che la connessione funzioni sul proprio DBMS in locale.

Il progetto prevede di inserire un account con il ruolo di amministratore nel database già dalla prima migrazione del database. In questo modo quando l'app partirà per il primo deployment sarà già configurato un account amministratore. Questa scelta è dovuta al fatto che per impostazione predefinita, quando ci si registra nella web app si è automaticamente utenti standard e solo un account amministratore può cambiare il ruolo di un account della piattaforma. Se non fosse stato inserito un account amministratore nei dati della migrazione, sarebbe stato necessario inserirlo manualmente direttamente nel database, utilizzando l'interfaccia a riga di comando (mysql monitor), oppure l'interfaccia grafica (HeidiSQL). L'account amministratore avrà come username la e-mail istituzionale della scuola e come password una password di propria scelta, che deve rispettare i seguenti criteri: almeno cinque caratteri, almeno un carattere maiuscolo, almeno uno minuscolo ed almeno un carattere speciale.

Il progetto prevede un account di posta elettronica, utilizzato per inviare i messaggi di posta agli utenti che si registrano alla piattaforma web. Questo account di posta è scorrelato dall'account del primo admin. In questo caso per rendere il setup semplice e uniforme per tutti gli studenti si utilizzerà un account e-mail fake fornito da <https://ethereal.email>. Chi volesse utilizzare un account di posta vero e inviare realmente le email potrà farlo utilizzando le indicazioni riportate nelle sezioni precedenti di questa guida.

Creiamo un account di e-mail fake con un solo click all'indirizzo: <https://ethereal.email/create>

**Account credentials**

NB! these credentials are shown only once. If you do not write these down then you have to create a new account.

Name	leland.schinner
Username	leland.schinner79@ethereal.email (also works as a real inbound email address)
Password	bNBsyBPwupQyapbkeA

[Download as CSV](#) [Open Mailbox](#)

**Nodemailer configuration**

```
const transporter = nodemailer.createTransport({
  host: 'smtp.ethereal.email',
  port: 587,
  auth: {
    user: 'leland.schinner79@ethereal.email',
    pass: 'bNBsyBPwupQyapbkeA'
  }
});
```

**PHPMailer configuration**

```
$mail = new PHPMailer(true);
$mail->isSMTP();
$mail->Host = 'smtp.ethereal.email';
$mail->SMTPOAuth = true;
$mail->Username = 'leland.schinner79@ethereal.email';
$mail->Password = 'bNBsyBPwupQyapbkeA';
$mail->SMTPSecure = 'tls';
$mail->Port = 587;
```

**SwiftMailer configuration**

```
$transport = (new Swift_SmtpTransport('smtp.ethereal.email', 587, 'tls'))
->setUsername('leland.schinner79@ethereal.email')
->setPassword('bNBsyBPwupQyapbkeA');
```

**SMTP configuration**

Host	smtp.ethereal.email
Port	587
Security	STARTTLS
Username	leland.schinner79@ethereal.email
Password	bNBsyBPwupQyapbkeA

[Feedback](#)

Modifichiamo i dati presenti nel file appsettings.json come segue:

```
{
  "ConnectionStrings": {
    "PizzaStoreConnection": {
      "Server": "localhost",
      "port": 3306,
      "Database": "pizzastore_apikey_cookie",
      "UserId": "root",
      "Password": "root"
    }
  },
  "AppSettings": {
    "AdminInitialPassword": "SuperSegreta#1",
    "AdminInitialEmail": "gennaro.malafronte@issgreppi.it",
    "AdminSurname": "Malafronte",
    "AdminGivenName": "Gennaro",
    "MaxNumberOfKeysPerUser": 10,
    "EmailFrom": "leland.schinner79@ethereal.email",
    "EmailSenderName": "Pizza Store API",
    "SmtpHost": "smtp.ethereal.email",
    "SmtpPort": 587,
    "SmtpUser": "leland.schinner79@ethereal.email",
    "SmtpPass": "bNBsyBPwupQyapbkeA",
    "GenerateMessageIdFrom": "ethereal.email"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  }
}
```

```
        "AllowedHosts": "*"
    }
```

Ovviamente bisogna inserire come dati del primo admin quelli della propria casella email istituzionale del Greppi.

Il progetto dell'app prevede che i dati inseriti in appsettings.json siano utilizzati nella migrazione del database. Infatti, guardando il codice della classe **AppDbContext** si vede che c'è una sezione del metodo **OnModelCreating** che esegue proprio la creazione del primo utente di tipo Administrator a partire dai dati di appsettings.json:

```
//***sezione per generare il primo admin nel database
    CryptoService cs = new();
    var adminSalt = cs.GenerateSalt();
    var adminPasswordHash =
    cs.GeneratePasswordHash(_appSettings.AdminInitialPassword, adminSalt);
    var timeCreationFirstAdmin = DateTime.UtcNow;
    //https://stackoverflow.com/questions/40744245/use-appsettings-json-to-
    configure-dbcontext-mapping
    modelBuilder.Entity<User>().HasData(
        new User
        {
            UserId = 1,
            GivenName = _appSettings.AdminGivenName,
            Surname = _appSettings.AdminSurname,
            EmailAddress = _appSettings.AdminInitialEmail,
            IsEmailConfirmed = true,
            Role = UserRole.Administrator,
            ValidationToken = cs.GenerateToken(),
            ValidationTokenExpireDate = DateTime.UtcNow,
            PasswordHash = adminPasswordHash,
            Salt = adminSalt,
            CreatedOn = timeCreationFirstAdmin,
            UpdatedOn = timeCreationFirstAdmin,
        });
    //***
```

Dopo aver controllato che MariaDb sia in esecuzione in localhost, effettuiamo la prima migrazione in modo che i dati inseriti in appsettings.json siano effettivamente inseriti nel database.

Add-Migration InitialMigrate

Controlliamo il file di migrazione e verifichiamo che sia stato effettivamente creato il primo admin della piattaforma.

Facciamo partire l'applicazione da Visual Studio:

The screenshot shows the Swagger UI interface for the PizzaStore API V1. At the top, there's a navigation bar with tabs for 'Swagger UI' and 'localhost:7155/swagger/index.html'. The main header includes the 'Swagger' logo, the text 'Supported by SMARTBEAR', and a dropdown menu 'Select a definition' set to 'PizzaStore API V1'. Below the header, the title 'PizzaStore API' is displayed with a version 'v1' and an 'OAS3' badge. A link to '/swagger/v1/swagger.json' is shown. The main content area is titled 'PizzaStoreApiKey' and contains a list of API endpoints under the 'POST /accounts/register' section. Each endpoint is represented by a button with its method (e.g., POST, GET) and path. Some endpoints have a lock icon indicating they require authentication. The interface uses a light blue and white color scheme with green highlights for successful operations.

## PizzaStore API v1 OAS3

/swagger/v1/swagger.json

Make the pizzas you love, but with a little of security in mind

Authorize

### PizzaStoreApiKey

POST /accounts/register

GET /accounts/verify-email

POST /accounts/verify-email

POST /accounts/login

POST /accounts/logout

POST /accounts/forgot-password

POST /accounts/reset-password

GET /accounts/reset-password

POST /accounts/change-password

GET /accounts/profile

POST /accounts/change-profile

Da HeidiSQL verifichiamo che sia stato creato il database associato alla web application:

Proviamo a fare il login come amministratori mediante la richiesta: **POST /accounts/login**

The screenshot shows the Swagger UI interface for a REST API. In the 'Curl' section, a POST command is shown:

```
curl -X 'POST' \
  'https://localhost:7155/accounts/login' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{' \
    "emailAddress": "gennaro.malafronte@issgreppi.it", \
    "password": "SuperSegreta#1" \
}'
```

A red circle highlights the password value in the JSON payload. Below the curl command, the 'Request URL' is listed as `https://localhost:7155/accounts/login`. Under 'Server response', the status code **204** is circled in red, indicating a successful login. The 'Response headers' section shows:

```
cache-control: no-cache,no-store
date: Tue,26 Apr 2022 17:57:26 GMT
expires: Thu,01 Jan 1970 00:00:00 GMT
pragma: no-cache
server: Kestrel
```

The 'Responses' section lists two entries:

Code	Description	Links
204	No Content	No links
401	Unauthorized	No links

Possiamo verificare il nostro profilo facendo la richiesta: **GET /accounts/profile**

The screenshot shows the Swagger UI interface for a REST API. A blue button labeled 'GET /accounts/profile' is highlighted, with the note 'Require authentication with role Standard,Administrator'. The 'Parameters' section indicates 'No parameters'. Below the button are 'Execute' and 'Clear' buttons.

In the 'Responses' section, the status code **200** is circled in red. The 'Response body' shows a JSON object:

```
{ "userId": 1, "givenName": "Gennaro", "surname": "Malafronte", "emailAddress": "gennaro.malafronte@issgreppi.it", "isEmailConfirmed": true, "role": 1 }
```

The 'Response headers' section shows:

```
content-type: application/json; charset=utf-8
date: Tue,26 Apr 2022 17:58:55 GMT
server: Kestrel
```

Proviamo a fare il logout con la richiesta **POST /accounts/logout** e successivamente a registrarcì

con un altro account (dobbiamo fornire un indirizzo email valido e diverso da quelli già presenti nella piattaforma), facendo una nuova richiesta: **POST /accounts/register**

## PizzaStoreApiKey

POST /accounts/register

Parameters

No parameters

Request body required

application/json

```
{  
  "givenName": "Gennareo",  
  "surname": "Malafronte",  
  "password": "SuperSegreta#1",  
  "confirmPassword": "superSegreta#1",  
  "emailAddress": "██████████",  
  "confirmEmailAddress": "██████████"  
}
```

La risposta della web application è la seguente:

Responses

Curl

```
curl -X 'POST' \
  'https://localhost:7155/accounts/register' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "givenName": "Gennaro",
    "surname": "Malafrente",
    "password": "SuperSegreta#1",
    "confirmPassword": "SuperSegreta#1",
    "emailAddress": "gennaro.malafrente@outlook.com",
    "confirmEmailAddress": "gennaro.malafrente@outlook.com"
}'
```

Request URL

```
https://localhost:7155/accounts/register
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "code": 200,   "message": [     "Account created. An email with a validation token has been sent"   ] }</pre> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Tue, 26 Apr 2022 18:04:50 GMT server: Kestrel</pre>

Copy Download

Controlliamo l'account di posta per trovare il token di validazione della e-mail. Se avessimo usato un account di posta vero per inviare le email troveremmo un vero messaggio di posta elettronica nella nostra casella di posta (si può controllare questo provando ad iscriversi al servizio <https://pizzastoreapi.azurewebsites.net/>, in questo caso bisogna controllare che la mail non sia finita nello spam). In questo esempio, siccome ethereal.email è una fake email vedremo il messaggio di posta nell'account di ethereal.mail:

The screenshot shows the Ethereal Email inbox. A red box highlights the subject line "PizzaStore - Account creato: conferma indirizzo email". Another red box highlights the recipient "To: [REDACTED]". A red arrow points from the "From" field "Ethereal Email <info@ethereal.email>" to the "This is your ethereal mailbox" message below it. The timestamp "Today at 20:04" is also circled.

The screenshot shows the message details page for the confirmation email. A red box highlights the subject "Subject: PizzaStore - Account creato: conferma indirizzo email". Another red box highlights the recipient "To: [REDACTED]". A red arrow points from the "Time: Today at 20:04" timestamp to the "Message-ID: <FDXQDANLOGU4.322YS1VBOK2C2@ethereal.email>" field. Below the message content, a red box highlights the "HTML" radio button and the validation token "K6RAz2tumgH3lfUo9Zahc29wwwAqql2YruBBq/dxIg=".

Controlliamo il database da HeidiSQL prima di cliccare sul link:

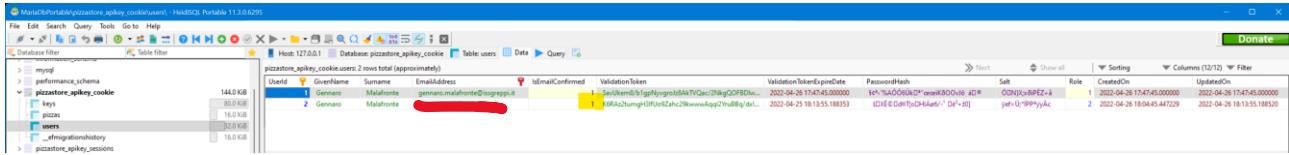
The screenshot shows the HeidiSQL database interface connected to the "pizzaone\_apkey\_cookie" database. The "users" table is selected, displaying two rows of data. The "IsEmailConfirmed" column for both rows is set to "0".

User ID	GivenName	Surname	EmailAddress	IsEmailConfirmed	ValidationToken	ValidationTokenExpireDate	PasswordHash	Salt	Role	CreatedOn	UpdatedOn
1	Gennaro	Malafronte	gennaro.malafronte@isoppi.it	0	Seckdem3tgMyng0dAA7VQsc2hkgQOFew...	2022-04-26 17:40:45,000000	1k+IA0063C*9anK80Dv8 AD#	CNNUjz8PEZ+4	1	2022-04-26 17:41:45,000000	2022-04-26 17:41:45,000000
2	Gennaro	Malafronte	gennaro.malafronte@isoppi.it	0	KIRAz2tumgH3lfUo9Zahc29wwwAqql2YruBBq/dxIg=	2022-04-27 18:04:45,40749	IDIEFDmH7yOHauf* Df*851	yel+U-PPNyAc	2	2022-04-26 18:04:45,447229	2022-04-26 18:04:45,447229

Clicchiamo sul link di conferma e verifichiamo che il campo IsEmailConfirmed sia passato a true:

The screenshot shows a browser window with the URL "https://localhost:7155/accounts/". The response body is a JSON object with a "code" field of 200 and a "message" field containing the string "E-mail validated".

```
{
  "code": 200,
  "message": [
    "E-mail validated"
  ]
}
```



Verificare che sia possibile effettuare le azioni specificate di seguito.

**Come utenti standard, oppure amministratori**, dopo aver effettuato il login è possibile:

- effettuare il logout tramite la richiesta: **POST /accounts/logout**
  - vedere il proprio profilo tramite la richiesta: **GET /accounts/profile**
  - cambiare il proprio profilo tramite la richiesta: **GET /accounts/change-profile** (qui si può cambiare nome, cognome ed indirizzo email)
  - cambiare la propria password con la richiesta: **POST /accounts/change-password**
  - creare una propria chiave tramite la richiesta: **POST /keys**
  - vedere le proprie chiavi mediante la richiesta: **GET /keys**
  - ricercare le proprie chiavi mediante la richiesta: **GET /keys/{keyName}**
  - eliminare una propria chiave mediante la richiesta: **DELETE /keys/{keyName}**

**Come utenti anonimi** (che non hanno effettuato il login) è possibile:

- registrarsi al servizio mediante la richiesta: **POST /accounts/register**
  - verificare il proprio indirizzo e-mail mediante le richieste:  
**GET /accounts/verify-email**  
**POST /accounts/verify-email**
  - effettuare il login tramite la richiesta: **POST /accounts/login**
  - recuperare la password mediante la richiesta: **POST /accounts/forgot-password** (in questo caso bisogna specificare un indirizzo e-mail valido; se questo indirizzo è presente sulla piattaforma, verrà inviata una e-mail a quell'indirizzo con un token valido 24 ore)
  - impostare la nuova password a seguito di una richiesta: **POST /accounts/reset-password** (in questo caso bisogna specificare la e-mail dell'account per cui si sta cambiando la password, il token precedentemente inviato tramite e-mail, la nuova password, scritta due volte per la conferma della nuova password)

**Come utenti amministratori**, dopo aver effettuato il login è possibile:

- ottenere un elenco di tutti gli account presenti sulla piattaforma, mediante una richiesta:  
**GET /accounts**
  - ottenere uno specifico account registrato sulla piattaforma, mediante la richiesta:  
**GET /accounts/{email}**
  - ottenere una lista di tutte le chiavi di uno specifico utente di una piattaforma, mediante la richiesta: **GET /accounts/{userId}/keys**
  - eliminare una chiave di uno specifico utente, mediante la richiesta:  
**DELETE /accounts/{userId}/keys/{keyName}**
  - eliminare un account mediante la richiesta: **DELETE /accounts/{userId}**. In questo caso

verranno eliminate anche tutte le chiavi associate a quell'account.

### 13.9.1.4.5.2. Struttura del progetto

#### 13.9.1.4.5.2.1. Il data model

Tra i vari file del model si riportano i seguenti, che rappresentano le entity che sono convertite in tabelle del database. In particolare l'entity BaseUserEntity serve per poter implementare un meccanismo automatico di tracciamento delle date di creazione e di ultima modifica degli utenti. Questo aspetto riveste una certa importanza, poiché il meccanismo di validazione del cookie di autorizzazione valida il cookie ad ogni richiesta fatta dal client e controlla se la data di ultima modifica dell'utente nel database è successiva alla data "LastChanged" inserita nel claim del cookie di autorizzazione. Nel caso in cui fosse stata fatta una modifica dal backend dell'app sull'utente (ad esempio un cambio di ruolo, di e-mail, etc.) la data UpdatedOn sarebbe successiva alla data recuperata dal cookie di autorizzazione (auth\_cookie) e l'applicazione rigetterebbe il cookie costringendo il client ad effettuare un nuovo login.

//file: BaseUserEntity.cs

```
namespace PizzaStoreApiKey.Models
{
    public class BaseUserEntity
    {
        public DateTime CreatedOn { get; set; }
        public DateTime UpdatedOn { get; set; }
    }
}
```

//file: User.cs

```
using System.ComponentModel.DataAnnotations.Schema;

namespace PizzaStoreApiKey.Models
{
    public enum UserRole:byte
    {
        Administrator=1, //con 0 la migration non inserisce correttamente i valori convertiti se si usa
        una conversione in stringa nel database
        Standard
    }
    public class User:BaseUserEntity
    {

        public int UserId { get; set; }

        [Column(TypeName = "nvarchar(60)")]
        public string GivenName { get; set; } = null!;

        [Column(TypeName = "nvarchar(60)")]
        public string Surname { get; set; } = null!;

        [Column(TypeName = "nvarchar(80)")]
        public string EmailAddress { get; set; } = null!;

        //valore booleano che indica se la mail dell'account è stata verificata oppure no
        //il valore di default è false
        public bool IsEmailConfirmed { get; set; }

        //utilizza un token a lunghezza fissa di 44 caratteri (vedere il metodo che genera il token
        per il calcolo della lunghezza)
        [Column(TypeName = "nchar(44)")]
        public string ValidationToken { get; set; } = null!;

        public DateTime ValidationTokenExpireDate { get; set; }

        //chiave salvata come HMACSHA256
        [Column(TypeName = "binary(32)")]
        public byte[] PasswordHash { get; set; } = null!;

        //salt salvato come SHA2-256
    }
}
```

```

    [Column(TypeName = "binary(16)")]
    public byte[] Salt { get; set; } = null!;

    public UserRole Role { get; set; }
    public ICollection<Key> Keys { get; set; } = new HashSet<Key>();

}

```

//file Key.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace PizzaStoreApiKey.Models
{
    public class Key
    {
        public int Id { get; set; }

        [Column(TypeName = "nvarchar(80)")]
        public string Name { get; set; } = null!;

        [Column(TypeName = "nchar(30)")]
        public string Value { get; set; } = null!;
        public bool IsValid { get; set; }
        public DateTime? ExpireTime { get; set; }
        public int UserId { get; set; }
        public User User { get; set; } = null!;

    }
}

```

//file Pizza.cs

```

namespace PizzaStoreApiKey.Models
{
    public class Pizza
    {
        public int Id { get; set; }
        public string? Name { get; set; }
        public string? Description { get; set; }
    }
}

```

La cartella Data del progetto contiene il file AppDbContext.cs che recupera i dati per la migrazione direttamente dal file appsettings.cs, utilizzando una classe helper, chiamata AppSettings.

//file: AppDbContext.cs

```

using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Options;
using PizzaStoreApiKey.Helpers;
using PizzaStoreApiKey.Models;
using PizzaStoreApiKey.Services;

namespace PizzaStoreApiKey.Data
{
    public class AppDbContext : DbContext
    {
        //https://stackoverflow.com/questions/40744245/use-appsettings-json-to-configure-dbcontext-
mapping
        private readonly AppSettings _appSettings;
        public AppDbContext(DbContextOptions<AppDbContext> options, IOptions<AppSettings> appSettings)
: base(options)
        {
            _appSettings = appSettings.Value;
        }
        public DbSet<Pizza> Pizzas => Set<Pizza>();
        public DbSet<User> Users => Set<User>();
        public DbSet<Key> Keys => Set<Key>();
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {

```

```

//https://stackoverflow.com/questions/41246614/entity-framework-core-add-unique-
constraint-code-first
modelBuilder.Entity<User>().HasIndex(u => u.EmailAddress).IsUnique();
modelBuilder.Entity<User>().Property(u => u.IsEmailConfirmed).HasDefaultValue(false);
modelBuilder.Entity<User>().Property(u => u.Role).HasDefaultValue(UserRole.Standard);
//https://docs.microsoft.com/en-us/ef/core/modeling/value-conversions?tabs=data-
annotations
modelBuilder.Entity<User>().HasIndex(k => new {k.Name, k.UserId }).IsUnique();
modelBuilder.Entity<User>().HasIndex(k => k.Value).IsUnique();

modelBuilder.Entity<Pizza>().HasData(
    new Pizza { Id = 1, Name = "Montemagno", Description = "Pizza shaped like a great
mountain" },
    new Pizza { Id = 2, Name = "The Galloway", Description = "Pizza shaped like a
submarine, silent but deadly" },
    new Pizza { Id = 3, Name = "The Noring", Description = "Pizza shaped like a Viking
helmet, where's the mead" }
);

/***sezione per generare il primo admin nel database
CryptoService cs = new();
var adminSalt = cs.GenerateSalt();
var adminPasswordHash = cs.GeneratePasswordHash(_appSettings.AdminInitialPassword,
adminSalt);
var timeCreationFirstAdmin = DateTime.UtcNow;
//https://stackoverflow.com/questions/40744245/use-appsettings-json-to-configure-
dbcontext-mapping
modelBuilder.Entity<User>().HasData(
    new User
    {
        UserId = 1,
        GivenName = _appSettings.AdminGivenName,
        Surname = _appSettings.AdminSurname,
        EmailAddress = _appSettings.AdminInitialEmail,
        IsEmailConfirmed = true,
        Role = UserRole.Administrator,
        ValidationToken = cs.GenerateToken(),
        ValidationTokenExpireDate = DateTime.UtcNow,
        PasswordHash = adminPasswordHash,
        Salt = adminSalt,
        CreatedOn = timeCreationFirstAdmin,
        UpdatedOn = timeCreationFirstAdmin,
    });
/***
}

#region snippet_auto_track_user
//https://threewill.com/how-to-auto-generate-created-updated-field-in-ef-core/
//la parte di codice in questo snippet serve a tracciare in automatico i tempi di creazione e
di ultima modifica delle entity di tipo User
public override int SaveChanges(bool acceptAllChangesOnSuccess)
{
    OnBeforeSaving();
    return base.SaveChanges(acceptAllChangesOnSuccess);
}

public override async Task<int> SaveChangesAsync(
    bool acceptAllChangesOnSuccess,
    CancellationToken cancellationToken = default(CancellationToken))
{
    OnBeforeSaving();
    return (await base.SaveChangesAsync(acceptAllChangesOnSuccess,
        cancellationToken));
}

private void OnBeforeSaving()
{
    var entries = ChangeTracker.Entries();
    var utcNow = DateTime.UtcNow;

    foreach (var entry in entries)
    {
        // for entities that inherit from BaseEntity,

```

```

        // set UpdatedOn / CreatedOn appropriately
        if (entry.Entity is BaseEntity trackable)
        {
            switch (entry.State)
            {
                case EntityState.Modified:
                    // set the updated date to "now"
                    trackable.UpdatedOn = utcNow;

                    // mark property as "don't touch"
                    // we don't want to update on a Modify operation
                    entry.Property(nameof(trackable.CreatedOn)).IsModified = false;
                    break;

                case EntityState.Added:
                    // set both updated and created date to "now"
                    trackable.CreatedOn = utcNow;
                    trackable.UpdatedOn = utcNow;
                    break;
            }
        }
    }
}
#endifregion
}
}

```

Nella cartella Helpers ci sono i file:

```

//file AppSettings.cs

namespace PizzaStoreApiKey.Helpers
{
    public class AppSettings
    {
        public string AdminInitialPassword { get; set; } = string.Empty;
        public string AdminInitialEmail { get; set; } = string.Empty;
        public string AdminSurname { get; set; } = string.Empty;
        public string AdminGivenName { get; set; } = string.Empty;
        public int MaxNumberOfKeysPerUser { get; set; }
        public string EmailFrom { get; set; } = string.Empty;
        public string EmailSenderName { get; set; } = string.Empty;
        public string SmtpHost { get; set; } = string.Empty;
        public int SmtpPort { get; set; }
        public string SmtpUser { get; set; } = string.Empty;
        public string SmtpPass { get; set; } = string.Empty;
        public string GenerateMessageIdFrom { get; set; } = string.Empty;
    }
}

```

```

//file AppParams.cs

namespace PizzaStoreApiKey.Helpers
{
    public static class AppParams
    {
        public const int UserGivenNameMinLen = 7;
        public const int UserGivenMaxLen = 30;
        public const int UserSurnameMinLen = 7;
        public const int UserSurnameMaxLen = 30;
        public const int PasswordMinLen = 5;
        public const int PizzaNameMinLen = 5;
        public const int PizzaDescriptionMinLen = 7;
        public const int KeyNameMinLen = 5;
        public const int KeyNameMaxLen = 30;
        public const string LastChangedClaimType = "LastChanged";
        public const string VerifiedEmailClaimType = "VerifiedEmail";
        public const string VerifiedEmailClaimValue = "Verified";
    }
}

```

Il meccanismo di validazione dei cookie implementa una verifica ad ogni nuova richiesta. Per fare ciò è stata creata la classe CustomCookieAuthenticationEvents nella quale si è iniettata la dipendenza

da IUserRepository:

```
//file CustomCookieAuthenticationEvents.cs

using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using PizzaStoreApiKey.Helpers;
using PizzaStoreApiKey.Services;
using System.Security.Claims;

namespace PizzaStoreApiKey.Events
{
    public class CustomCookieAuthenticationEvents : CookieAuthenticationEvents
    {
        private readonly IUserRepository _userRepository;

        public CustomCookieAuthenticationEvents(IUserRepository userRepository)
        {
            _userRepository = userRepository;
            //https://www.blinkingcaret.com/2018/07/18/secure-an-asp-net-core-web-api-using-cookies/
            OnRedirectToLogin = redirectContext =>
            {
                redirectContext.HttpContext.Response.StatusCode = 401;
                return Task.CompletedTask;
            };
            //questo serve per fare in modo che quando un utente è logged in, ma non ha i permessi
            richiesti
            //venga restituito il codice 401 (forbidden) al posto del valore di default 404 (che è not
            found)
            OnRedirectToAccessDenied = redirectContext =>
            {
                redirectContext.HttpContext.Response.StatusCode = 403;
                return Task.CompletedTask;
            };

            //https://docs.microsoft.com/en-us/aspnet/core/security/authentication/cookie?view=aspnetcore-
            6.0#react-to-back-end-changes
            public override async Task ValidatePrincipal(CookieValidatePrincipalContext context)
            {
                var userPrincipal = context.Principal;

                var lastChanged = userPrincipal?.Claims.Where(c => c.Type ==
AppParams.LastChangedClaimType).Select(c => c.Value).FirstOrDefault();
                var userId = userPrincipal?.Claims.Where(c => c.Type ==
ClaimTypes.NameIdentifier).Select(c => c.Value).FirstOrDefault();

                if (!string.IsNullOrEmpty(lastChanged) && !string.IsNullOrEmpty(userId) &&
                    await _userRepository.ValidateLastChangedAsync(userId, lastChanged))
                {
                    return;
                }
                context.RejectPrincipal();

                await context.HttpContext.SignOutAsync(
                    CookieAuthenticationDefaults.AuthenticationScheme);
            }
        }
    }
}

//file IUserRepository.cs

namespace PizzaStoreApiKey.Services
{
    public interface IUserRepository
    {
        public Task<bool> ValidateLastChangedAsync(string? userId, string? lastChanged);
    }
}
```

```

//file UserRepository.cs

using PizzaStoreApiKey.Data;
using System.Globalization;

namespace PizzaStoreApiKey.Services
{
    public class UserRepository : IUserRepository
    {
        //inietto la dipendenza dal database
        //https://docs.microsoft.com/en-us/ef/core/dbcontext-configuration/#dbcontext-in-dependency-
        injection-for-aspnet-core
        private readonly AppDbContext _DbContext;

        public UserRepository(AppDbContext DbContext)
        {
            _DbContext = DbContext;
        }

        public async Task<bool> ValidateLastChangedAsync(string? userId, string? lastChanged)
        {
            //recupero l'utente dal database

            if (userId != null && lastChanged!=null && int.TryParse(userId, out int id) &&
                DateTime.TryParse(lastChanged, CultureInfo.InvariantCulture, DateTimeStyles.RoundtripKind, out
                DateTime lChanged))
            {
                var userInDb = await _DbContext.Users.FindAsync(id);
                if(userInDb != null)
                {
                    //se un utente è stato modificato dopo il lastChanged recuperato dal claim del
                    cookie bisogna invalidare il cookie
                    return await Task.FromResult(userInDb.UpdatedOn <= lChanged);
                }
            }
            return false;
        }
    }
}

```

L'invio delle e-mail è fatto attraverso il servizio EmailService:

```

using Microsoft.Extensions.Options;
using PizzaStoreApiKey.Helpers;
using MailKit.Net.Smtp;
using MailKit.Security;
using MimeKit;
using MimeKit.Text;
using MimeKit.Utils;

namespace PizzaStoreApiKey.Services
{
    public class EmailService : IEmailService
    {
        private readonly AppSettings _appSettings;

        public EmailService(IOptions<AppSettings> appSettings)
        {
            _appSettings = appSettings.Value;
        }

        public void Send(string to, string subject, string bodyHtml, string? from = null)
        {
            // create message
            var email = new MimeMessage();
            email.From.Add(new MailboxAddress(_appSettings.EmailSenderName, from ??
            _appSettings.EmailFrom));
            email.To.Add(MailboxAddress.Parse(to+" <" + to + ">"));
            email.Subject = subject;
            email.Body = new TextPart(TextFormat.Html) { Text = bodyHtml };
            if (!string.IsNullOrEmpty(_appSettings.GenerateMessageIdFrom))
            {
                email.MessageId = MimeUtils.GenerateMessageId(_appSettings.GenerateMessageIdFrom);
            }
            // send email
            using var smtp = new SmtpClient();

```

```

        smtp.Connect(_appSettings.SmtpHost, _appSettings.SmtpPort, SecureSocketOptions.StartTls);
        smtp.Authenticate(_appSettings.SmtpUser, _appSettings.SmtpPass);
        smtp.Send(email);
        smtp.Disconnect(true);
    }
}

```

Per gli endpoint che prevedono un input da parte dell'utente sono stati creati una serie di modelli che semplificano la gestione dell'input:

- UserSignUp → per la registrazione di un nuovo utente
- UserVerifyEmailInput → per la verifica dell'email
- UserChangePasswordInput → per il cambio della password
- UserResetPasswordInput → per il reset della password
- UserForgotPasswordInput → per chiedere un token per il reset della password
- UserChangeroleInput → per il cambio di ruolo di un utente
- UserChangeProfileInput → per il cambio del profilo dell'utente
- KeyInput → per la creazione di una nuova chiave
- Pizza → per la creazione di una nuova pizza

Per ognuna di queste classi di dati di input è stata creata una classe di validazione tramite FluentValidator (per i dettagli si veda il codice del progetto su GitHub):

- UserSignUpValidator
- UserVerifyEmailInputValidator
- UserChangePasswordInputValidator
- UserResetPasswordInputValidator
- UserForgotPasswordInputValidator
- UserChangeroleInputValidator
- UserChangeProfileInputValidator
- KeyInputValidator
- PizzaValidator

Per la classe UserLogin non è stato previsto un validatore poiché l'unica assunzione fatta sul login è che i campi email e password siano non nulli e non vuoti.

Per la restituzione di oggetti di tipo Key e di tipo User sono stati creati dei tipi DTO equivalenti:

- KeyDTO
- UserDTO

Per gli altri dettagli si rimanda al codice del progetto, disponibile su GitHub. Di seguito si riporta il codice del file Program.cs:

//file Program.cs

```

using FluentValidation.AspNetCore;
using Microsoft.EntityFrameworkCore;

```

```

using Microsoft.OpenApi.Models;
using AspNetCore.Authentication.ApiKey;
using PizzaStoreApiKey.Data;
using PizzaStoreApiKey.EndPoints;
using PizzaStoreApiKey.Models;
using PizzaStoreApiKey.Services;
using PizzaStoreApiKey.Helpers;
using PizzaStoreApiKey.Filters.Swagger;
using Microsoft.AspNetCore.Authentication.Cookies;
using PizzaStoreApiKey.Events;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "PizzaStore API",
        Description = "Make the pizzas you love, but with a little of security in mind",
        Version = "v1"
    });
    //*** le due istruzioni seguenti (in commento) impostano Swagger per utilizzare ApiKey per tutti gli endpoints
    //https://stackoverflow.com/questions/57943550/how-to-document-api-key-authentication-using-swashbuckle-aspnetcore-v5-0-0-rc2
    //https://stackoverflow.com/a/57944310
    //c.AddSecurityDefinition("ApiKey", new OpenApiSecurityScheme()
    //{
    //    Type = SecuritySchemeType.ApiKey,
    //    In = ParameterLocation.Header,
    //    Name = "X-API-KEY",
    //    Description = "Pizza Store API. Permette di effettuare modifiche al database delle pizze solo se si è in possesso di una ApiKey. Sono gestiti anche i ruoli",
    //});
    //c.AddSecurityRequirement(new OpenApiSecurityRequirement()
    //{
    //    {
    //        new OpenApiSecurityScheme
    //        {
    //            Reference = new OpenApiReference { Type = ReferenceType.SecurityScheme, Id =
    "ApiKey" }
    //        },
    //        new string[] { }
    //    }
    //});
});

```

```

//*** il filtro e l'addSecurityDefinition permettono di definire (PER SWAGGER) meccanismi di sicurezza differenti per le diverse rotte
//queste impostazioni non hanno nulla a che fare con il comportamento dell'App: servono solo ad istruire Swagger per usare uno specifico meccanismo di sicurezza
//https://stackoverflow.com/questions/58694220/asp-net-core-2-2-swagger-endpoint-specific-security-definition

// Set the custom operation filter
c.OperationFilter<AuthorizationOperationFilter>();

// Add ApiKey Authentication Scheme
var apiKeySecurityScheme = new OpenApiSecurityScheme
{
    Type = SecuritySchemeType.ApiKey,
    In = ParameterLocation.Header,
    Name = "X-API-KEY",
    Description = "Pizza Store API. You can perform CRUD operations on Pizza store APIs only if a valid ApiKey is provided in the header or query string." +
        " Administrator and Standard roles are covered.",
    Reference = new OpenApiReference
    {
        Id = "ApiKey",
        Type = ReferenceType.SecurityScheme
    }
};

c.AddSecurityDefinition(apiKeySecurityScheme.Reference.Id, apiKeySecurityScheme);

// Add cookie based Authentication Scheme
var cookieBasedSecurityScheme = new OpenApiSecurityScheme
{
    Type = SecuritySchemeType.ApiKey,
    In = ParameterLocation.Cookie,
    Name = "auth_cookie",
    Description = "Pizza Store API. After login you can perform accounts and keys related requests. Administrator and Standard roles are covered." +
        " In the browser (via Swagger UI) cookie management is done automatically:" +
        " just login and then perform other account and/or key related API calls. You just don't need to explicitly set the cookie in the Swagger UI." +
        " However if you get the \"auth_cookie\" after login (e. g. inspecting browser request via dev tools) and you put it in the Swagger UI authorization textbox, you get" +
        " curl equivalent instruction to test your application with \"auth_cookie\"." +
        "For Pizza related authentication you need X-API-KEY authorization, which is independent from cookie based authorization.",
    Reference = new OpenApiReference
    {
        Id = "Cookies",
        Type = ReferenceType.SecurityScheme
    }
};

c.AddSecurityDefinition(cookieBasedSecurityScheme.Reference.Id, cookieBasedSecurityScheme);

```

```

//Add note on roles for methods requiring authentication
c.OperationFilter<SwaggerAppendAuthorizeToSummaryOperationFilter>();

//Ignore properties with [SwaggerIgnore] attribute
c.SchemaFilter<SwaggerIgnoreFilter>();

//gestione dello schema delle classi DTO in Swagger
//https://stackoverflow.com/questions/40644052/rename-model-in-swashbuckle-6-swagger-with-asp-net-
core-web-api
//https://stackoverflow.com/a/40684060
c.CustomSchemaIds(schemaIdStrategy);
static string schemaIdStrategy(Type currentClass)
{
    string returnedValue = currentClass.Name;
    if (returnedValue.EndsWith("DTO"))
        returnedValue = returnedValue.Replace("DTO", string.Empty);
    return returnedValue;
}

});

//connessione al database MariaDb
var connectionString = builder.Configuration.GetConnectionString("PizzaStoreConnection");
var serverVersion = ServerVersion.AutoDetect(connectionString);
builder.Services.AddDbContext<AppDbContext>(
    dbContextOptions => dbContextOptions
        .UseMySql(connectionString, serverVersion)
        // The following three options help with debugging, but should
        // be changed or removed for production.
        .LogTo(Console.WriteLine, LogLevel.Information)
        .EnableSensitiveDataLogging()
        .EnableDetailedErrors()
);

//connessione al database SQL Server (Express)
//var connectionString = builder.Configuration.GetConnectionString("PizzaStoreConnection");
//builder.Services.AddDbContext<AppDbContext>(
//    dbContextOptions => dbContextOptions
//        .UseSqlServer(connectionString)
//);

// Configure JSON options. Questo serve solo nel caso di loop nella serializzazione di oggetti. Ad
// esempio quando si serializza un oggetto Key che ha un riferimento
//di tipo User, il serializzatore scopre che User ha una collection di Key e per ognuno di questi
//effettua ancora una serializzazione, andando in loop. Per evitare questo
//si può istruire il serializzatore di ignorare i loop. Nel caso di oggetti DTO senza navigation
//property il problema non si pone e non c'è bisogno di impostare questa opzione
//https://stackoverflow.com/a/66562656

```

```

//https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-migrate-from-newtonsoft-how-to?pivots=dotnet-6-0#preserve-object-references-and-handle-loops
builder.Services.Configure<Microsoft.AspNetCore.Http.Json.JsonOptions>(options =>
{
    options.SerializerOptions.ReferenceHandler =
    System.Text.Json.Serialization.ReferenceHandler.IgnoreCycles;
});

//Inietto il servizio CryptoService
builder.Services.AddScoped<ICryptoService, CryptoService>();

//Inietto il servizio per accedere alle AppSettings
// configure strongly typed settings object
//https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/options?view=aspnetcore-6.0#bind-hierarchical-configuration
builder.Services.Configure<AppSettings>(builder.Configuration.GetSection("AppSettings"));

//Inietto il servizio e-mail:
builder.Services.AddScoped<IEmailService, EmailService>();

//configurazione dei validatori dell'input
builder.Services.AddFluentValidation(fv => fv.RegisterValidatorsFromAssemblyContaining<UserSignUp>());
builder.Services.AddFluentValidation(fv => fv.RegisterValidatorsFromAssemblyContaining<KeyInput>());
builder.Services.AddFluentValidation(fv => fv.RegisterValidatorsFromAssemblyContaining<Pizza>());
builder.Services.AddFluentValidation(fv =>
fv.RegisterValidatorsFromAssemblyContaining<UserForgotPasswordInput>());
builder.Services.AddFluentValidation(fv =>
fv.RegisterValidatorsFromAssemblyContaining<UserResetPasswordInput>());
builder.Services.AddFluentValidation(fv =>
fv.RegisterValidatorsFromAssemblyContaining<UserVerifyEmailInput>());
builder.Services.AddFluentValidation(fv =>
fv.RegisterValidatorsFromAssemblyContaining<UserChangePasswordInput>());
builder.Services.AddFluentValidation(fv =>
fv.RegisterValidatorsFromAssemblyContaining<UserChangeRoleInput>());
builder.Services.AddFluentValidation(fv =>
fv.RegisterValidatorsFromAssemblyContaining<UserChangeProfileInput>());

//Configuro Cookie based authentication
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(options =>
{
    options.ExpireTimeSpan = TimeSpan.FromMinutes(20);
    options SlidingExpiration = true;
    //options.AccessDeniedPath = "/Forbidden/";
    //https://www.blinkingcaret.com/2018/07/18/secure-an-asp-net-core-web-api-using-cookies/
    options.Cookie.Name = "auth_cookie";
    options.Cookie.SameSite = SameSiteMode.None;
    //https://www.blinkingcaret.com/2018/07/18/secure-an-asp-net-core-web-api-using-cookies/
}

```

```

        options.EventsType = typeof(CustomCookieAuthenticationEvents);
    });
builder.Services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();
builder.Services.AddScoped<IUserRepository, UserRepository>();
builder.Services.AddScoped<CustomCookieAuthenticationEvents>();

//configurazione per APIKey Authentication
// Add ApiKeyRepository to the dependency container.
builder.Services.AddScoped<IApiKeyRepository, ApiKeyRepository>();

// Add the ApiKey scheme authentication here..
// It requires Realm to be set in the options if SuppressWWWAuthenticateHeader is not set.
// If an implementation of IApiKeyProvider interface is registered in the dependency register as well
// as OnValidateKey delegate on options.Events
// is also set then this delegate will be used instead of an implementation of IApiKeyProvider.
builder.Services.AddAuthentication(ApiKeyDefaults.AuthenticationScheme)
    .AddApiKeyInHeaderOrqueryParams<IApiKeyProvider>(options =>
{
    options.Realm = "Pizza Store API";
    options.KeyName = "X-API-KEY";

    //// Optional option to suppress the browser login dialog for ajax calls.
    //options.SuppressWWWAuthenticateHeader = true;

    //// Optional option to ignore extra check of ApiKey string after it is validated.
    //options.ForLegacyIgnoreExtraValidatedApiKeyCheck = true;

    //// Optional option to ignore authentication if AllowAnonymous metadata/filter attribute
    //is added to an endpoint.
    options.IgnoreAuthenticationIfAllowAnonymous = true;

    //// Optional events to override the ApiKey original logic with custom logic.
    //// Only use this if you know what you are doing at your own risk. Any of the events can
    //be assigned.
    options.Events = new ApiKeyEvents { };

});

// All the requests will need to be authorized.
// Alternatively, add [Authorize] attribute to Controller or Action Method where necessary.
//builder.Services.AddAuthorization(options =>
//{
//    options.FallbackPolicy = new AuthorizationPolicyBuilder()
//        .RequireAuthenticatedUser()
//        .Build();
//});

builder.Services.AddAuthorization(options =>
{

```

```

    options.AddPolicy("OnlyVerifiedEmailAccount", policy =>
policy.RequireClaim(AppParams.VerifiedEmailClaimType));
});

var app = builder.Build();
app.UseSwagger();

app.UseHttpsRedirection();

//app.UseCors(policy =>
//{
//    policy.AllowAnyHeader();
//    policy.AllowAnyMethod();
//    policy.AllowAnyOrigin();
//    policy.AllowCredentials();
//});

app.UseAuthentication();

app.UseAuthorization();

//configurazione delle rotte
app.MapUserEndpoints();
app.MapKeyEndpoints();
app.MapPizzaEndpoints();
app.UseSwaggerUI(c =>
{
    c.SwaggerEndpoint("/swagger/v1/swagger.json", "PizzaStore API V1");
    //c.RoutePrefix =string.Empty;
});

//migrazione con code first approach - solo development e testing
using (var serviceScope = app.Services.CreateScope())
{
    var context = serviceScope.ServiceProvider.GetRequiredService<AppDbContext>();
    context.Database.Migrate();
    //vedere IN ALTERNATIVA a Migrate anche
    //context.Database.EnsureCreated();
}

//avvio dell'applicazione
app.Run();

```

### **13.9.1.4.5.3. Deployment del progetto su Azure**

Vogliamo effettuare il deployment del progetto su Azure usando le funzionalità di hosting gratuite di Azure e ottenere qualcosa di simile a questo: <https://pizzastoreapi.azurewebsites.net/>

Dopo aver effettuato il testing dell'applicazione in localhost è giunto il momento di effettuare il deployment dell'applicazione su Azure, utilizzando il servizio gratuito di hosting incluso nella

sottoscrizione Azure per studenti <https://azure.microsoft.com/it-it/free/students/> .

Prima di procedere al deployment vero e proprio, procederemo ad una modifica dell'applicazione riguardante il database utilizzato. Infatti, nell'offerta Azure per studenti, è disponibile il database MariaDb, ma solo in un numero limitato di ore (750 ore in un anno). In alternativa è possibile utilizzare un database **Azure SQL Server** che non ha limiti temporali, nell'offerta Azure per Studenti, e si comporta in modo del tutto analogo a MariaDb per le funzionalità richieste dall'applicazione web sviluppata. Inoltre, avendo utilizzato un ORM come EF Core, abbiamo anche il vantaggio che l'applicazione è quasi del tutto indifferente al DBMS utilizzato.

La parte di codice da modificare per utilizzare Azure SQL Server (chè è di fatto Microsoft SQL Server) è la seguente:

1) eliminiamo la parte:

```
////connessione al database MariaDb
//var connectionString =
builder.Configuration.GetConnectionString("PizzaStoreConnection");
//var serverVersion = ServerVersion.AutoDetect(connectionString);
//builder.Services.AddDbContext<AppDbContext>(
//    dbContextOptions => dbContextOptions
//        .UseMySql(connectionString, serverVersion)
//        // The following three options help with debugging, but should
//        // be changed or removed for production.
//        .LogTo(Console.WriteLine, LogLevel.Information)
//        .EnableSensitiveDataLogging()
//        .EnableDetailedErrors()
//);
```

2) Aggiungiamo:

```
//connessione al database SQL Server (Express)
var connectionString =
builder.Configuration.GetConnectionString("PizzaStoreConnection");
builder.Services.AddDbContext<AppDbContext>(
    dbContextOptions => dbContextOptions
        .UseSqlServer(connectionString)
);
```

3) Modifichiamo la stringa di connessione per usare SQL Server LocalDb che è incluso nell'installazione di Visual Studio. Questo è, di fatto, un SQL Server locale che parte quando c'è bisogno, ossia quando Visual Studio determina che c'è una richiesta di connessione e non necessita di username e password per l'accesso:

```
"PizzaStoreConnection":  
"server=(localdb)\mssqllocaldb;database=pizza_store_cookie;trusted_connection=true"
```

4) Cancelliamo la migrazione precedente

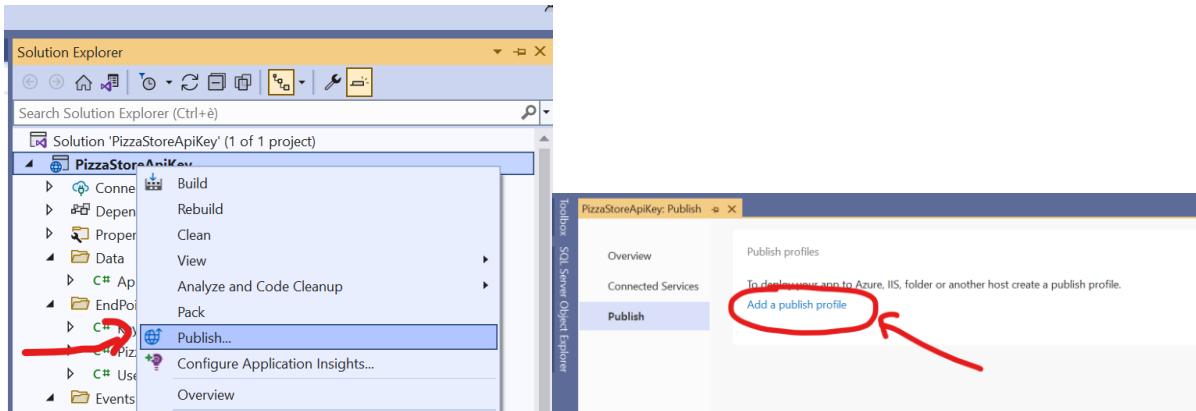
5) Creiamo una nuova migrazione per SQL Server:

```
Add-Migration InitialMigrate
```

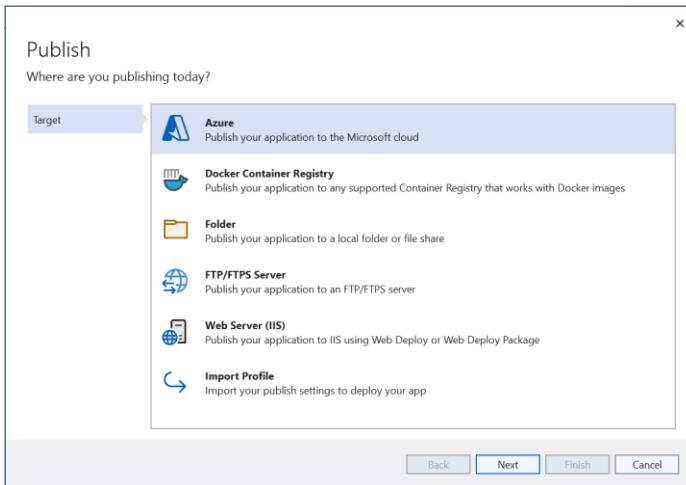
### 13.9.1.4.5.3.1. Deployment su Azure mediante il wizard integrato di Visual Studio.

Prerequisito: è stata effettuata la migrazione del database verso SQL Server.

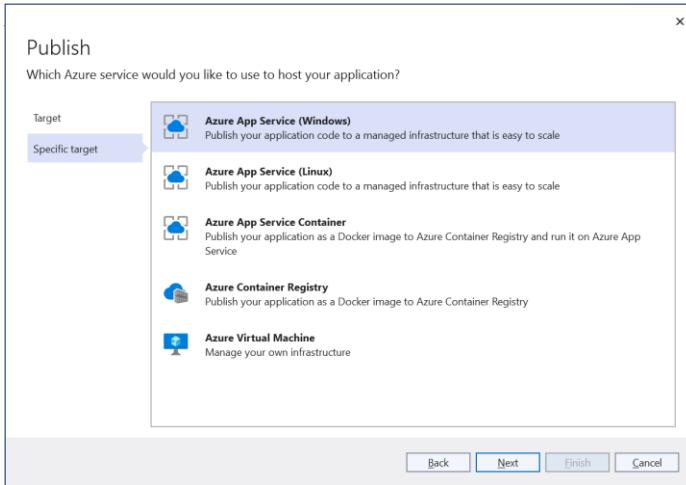
Dopo aver fermato l'eventuale esecuzione in corso della web app, procediamo al deployment utilizzando la funzione Publish: da Solution Explorer si effettua click destro sul progetto e poi dal menu contestuale si seleziona la voce Publish.



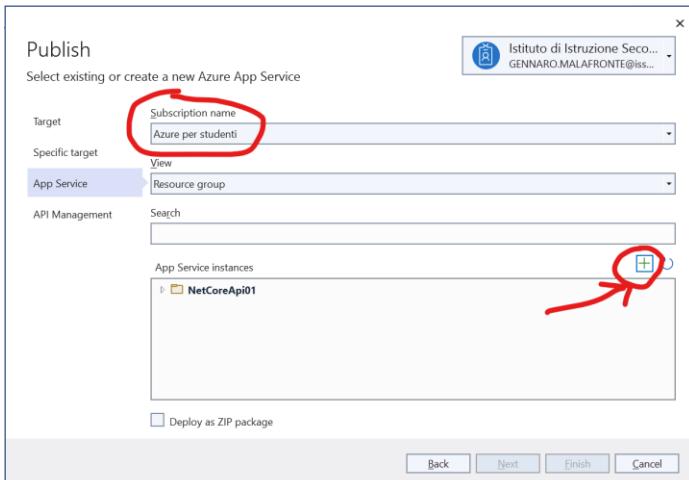
Selezioniamo Azure e poi next:



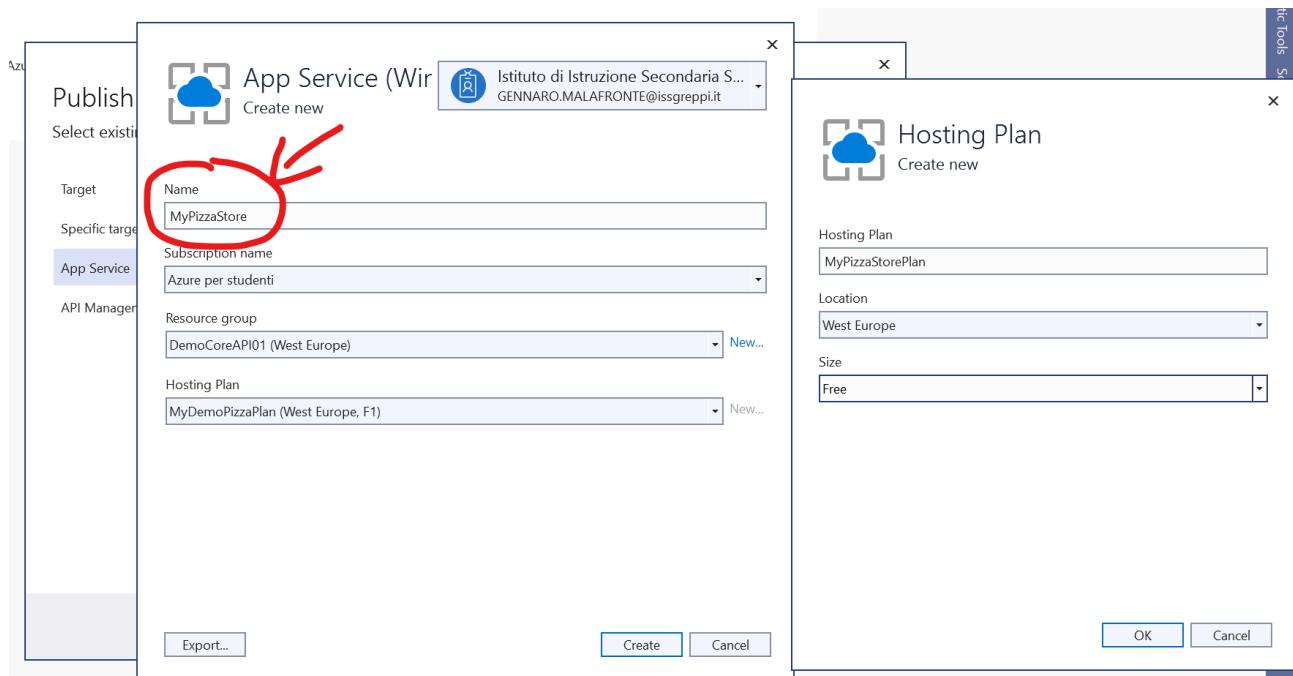
Selzioniamo Azure App Service (Windows) e quindi next:



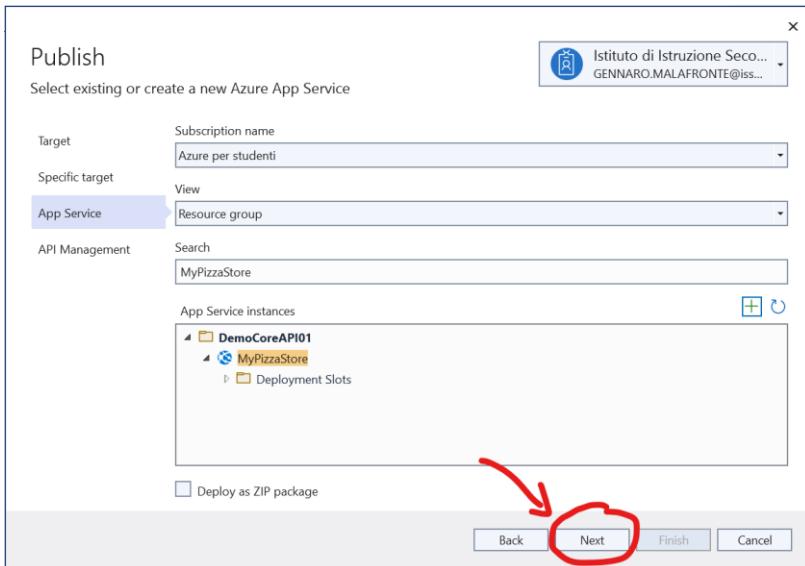
Creiamo un nuovo App Service, oppure selezioniamo un App Service già esistente. In questo caso vediamo come creare un nuovo App Service nel quale effettueremo il deployment dell'app corrente.



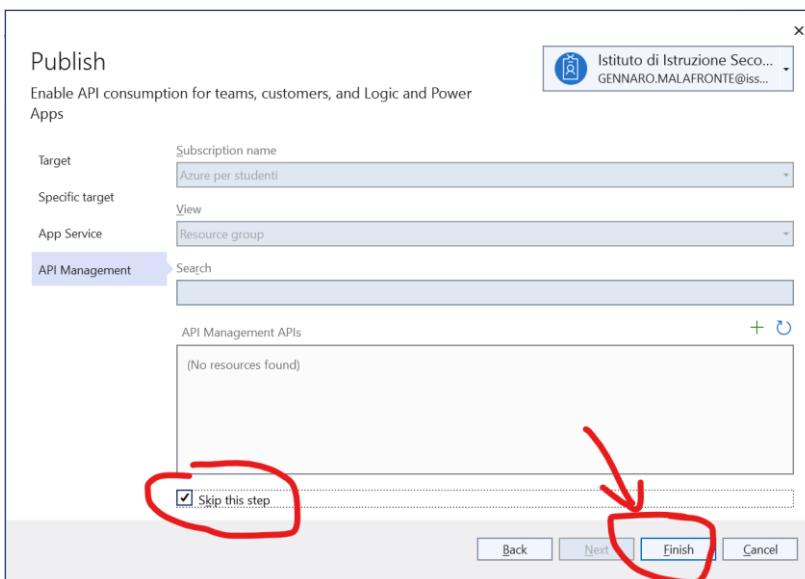
Dopo aver cliccato sul pulsante di aggiunta di un nuovo App Service, diamo un nome univoco alla nostra App: questo nome verrà utilizzato per il nome del dominio dell'app. Con il nome scelto nell'esempio l'applicazione sarà pubblicata su `mypizzastore.azurewebsites.net`. Selezioniamo un resource group nuovo, oppure utilizziamo il resource group già utilizzato in precedenza per i progetti già sviluppati su Azure. Il gruppo di risorse permette di gestire in modo unificato le risorse per uno specifico progetto, quindi, in questo caso, è preferibile creare un nuovo gruppo di risorse. Nell'esempio il gruppo di risorse è stato chiamato DemoCoreAPI01. Selezioniamo un Hosting Plan, scegliendo come Location Western Europe e come Size il piano gratuito (Free). Confermiamo, cliccando su Ok, e poi su Create. Quindi attendiamo alcuni secondi affinché il servizio sia configurato.



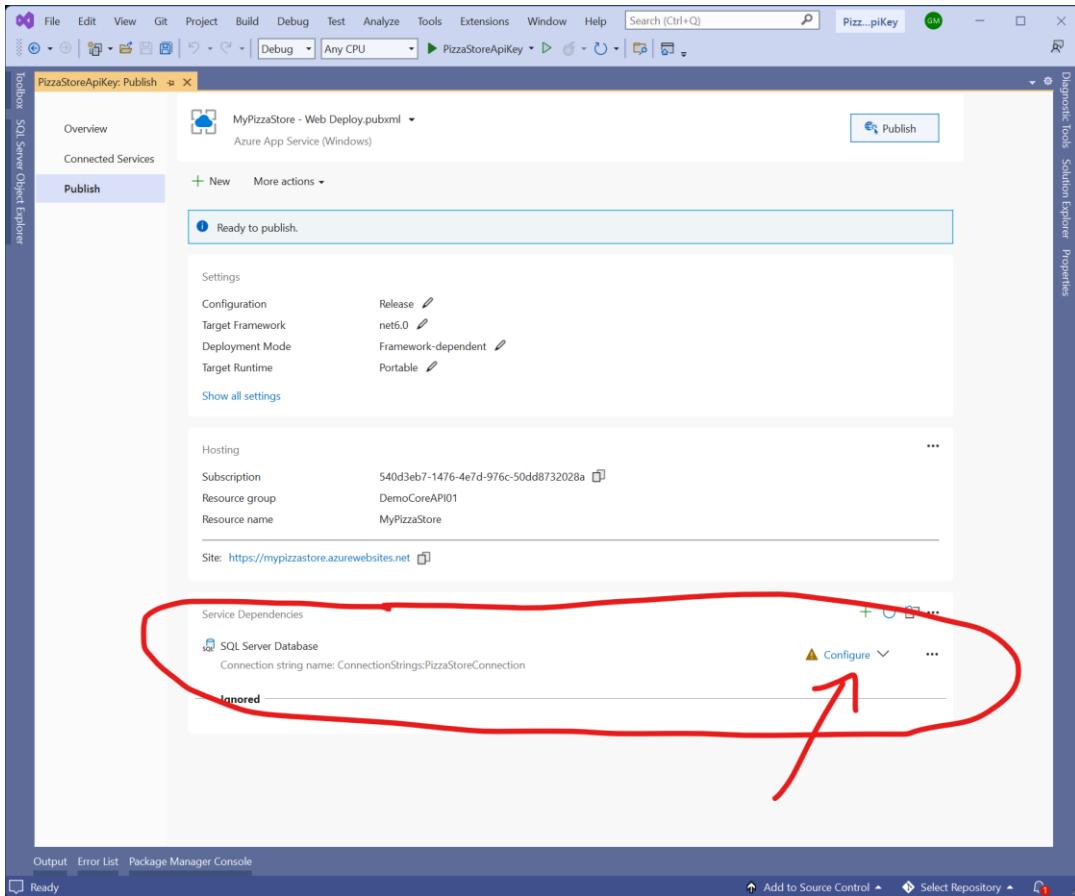
Clicchiamo su Next:



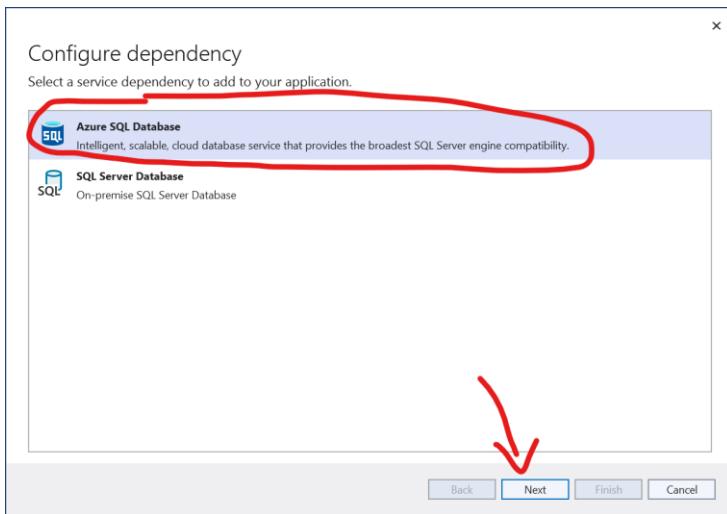
A questo punto il wizard propone di creare un modulo di API Management. Possiamo saltare questo punto selezionando la voce “skip this step” e poi il bottone Finish.



A questo punto la schermata di deployment dovrebbe essere come quella riportata di seguito:



Come si vede il wizard ha riconosciuto che c'è una stringa di connessione al database che va configurata opportunamente. Infatti, attualmente il progetto è configurato per funzionare in locale con MariaDb, oppure con Microsoft SQL Server Express, ma per far funzionare la nostra web app su Azure dobbiamo configurare un servizio database che sia utilizzabile dall'applicazione quando sarà installata su Azure. In questo caso optiamo per la creazione di un nuovo database direttamente su Azure. Clicchiamo su Configure, selezioniamo Azure SQL Database e quindi su Next:



Possiamo selezionare un database già esistente su Azure (se abbiamo già effettuato questa operazione in precedenza), oppure creare un nuovo database, cliccando sul pulsante con il segno “+”.

**Configure Azure SQL Database**

Select a service dependency to add to your application.

Subscription name: Azure per studenti

SQL databases:

Name	Resource group	Location	Server name
PizzaStoreApiKeyDb	NetCoreAPI01	Germany West Central	pizzastoreapikeydbserver

Database name: PizzaStoreApiKey\_db

Subscription name: Azure per studenti

Resource group: DemoCoreAPI01 (West Europe)

Database server: <Required>

Database administrator username (must have permissions to create): <Required>

Database administrator password: <Required>

Buttons: Back, Next, Finish, Cancel, Export..., Create, Cancel

**SQL Server**

Create new

Database server name: pizzastoreapikeydbserver

Location: Germany West Central

Administrator username: ProfAdmin

Administrator password: \*\*\*\*\*

Administrator password (confirm): \*\*\*\*\*

Buttons: OK, Cancel

**Azure SQL Data**

Istituto di Istruzione Secondaria S...  
GENNARO.MALAFRONTI@issgreppi.it

Create new

Database name: PizzaStoreApiKey\_db

Subscription name: Azure per studenti

Resource group: DemoCoreAPI01 (West Europe)

Database server: pizzastoreapikeydbserver\* (Germany West Central)

Database administrator username (must have permissions to create): ProfAdmin

Database administrator password: \*\*\*\*\*

Buttons: Export..., Create, Cancel

Attendiamo la creazione del database. Potrebbe volerci qualche minuto. Selezioniamo quindi il database appena creato dal wizard e quindi clicchiamo sul pulsante Next:

Configure Azure SQL Database

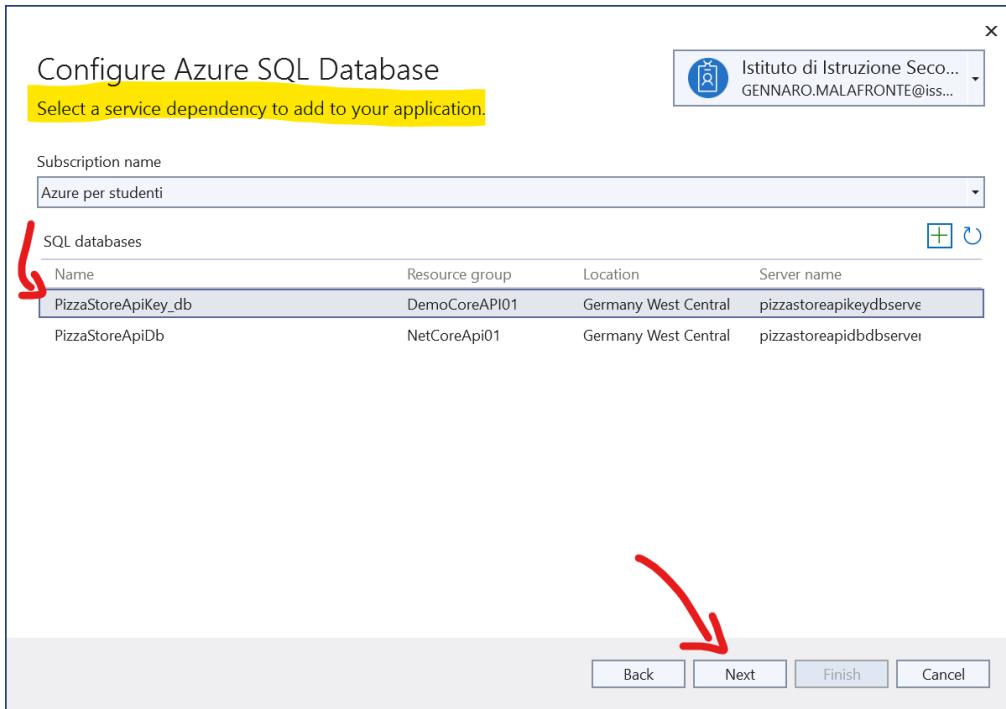
Select a service dependency to add to your application.

Subscription name: Azure per studenti

SQL databases:

Name	Resource group	Location	Server name
PizzaStoreApiKey_db	DemoCoreAPI01	Germany West Central	pizzastoreapikeydbserve
PizzaStoreApiDb	NetCoreApi01	Germany West Central	pizzastoreapidbserve

Back Next Finish Cancel



Recuperiamo i parametri di connessione dalla stringa di connessione:

Configure Azure SQL Database

Provide connection string name and specify how to save it

Database connection string name: ConnectionStrings:PizzaStoreConnection

Database connection user name: <Required>

Database connection password: <Required>

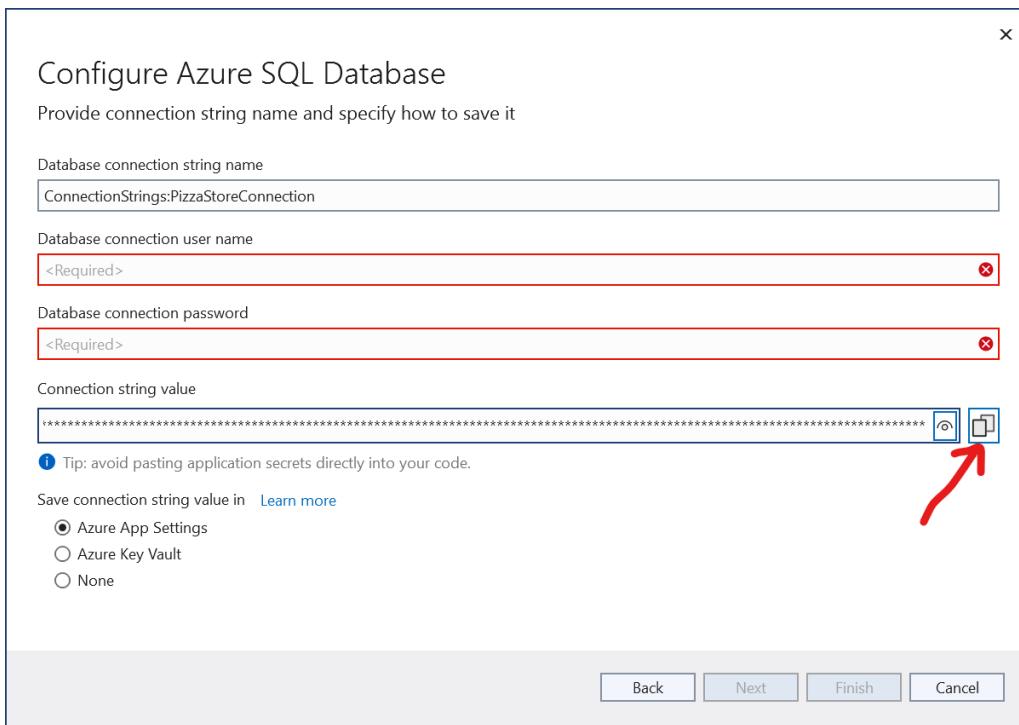
Connection string value: \*\*\*\*\*

Tip: avoid pasting application secrets directly into your code.

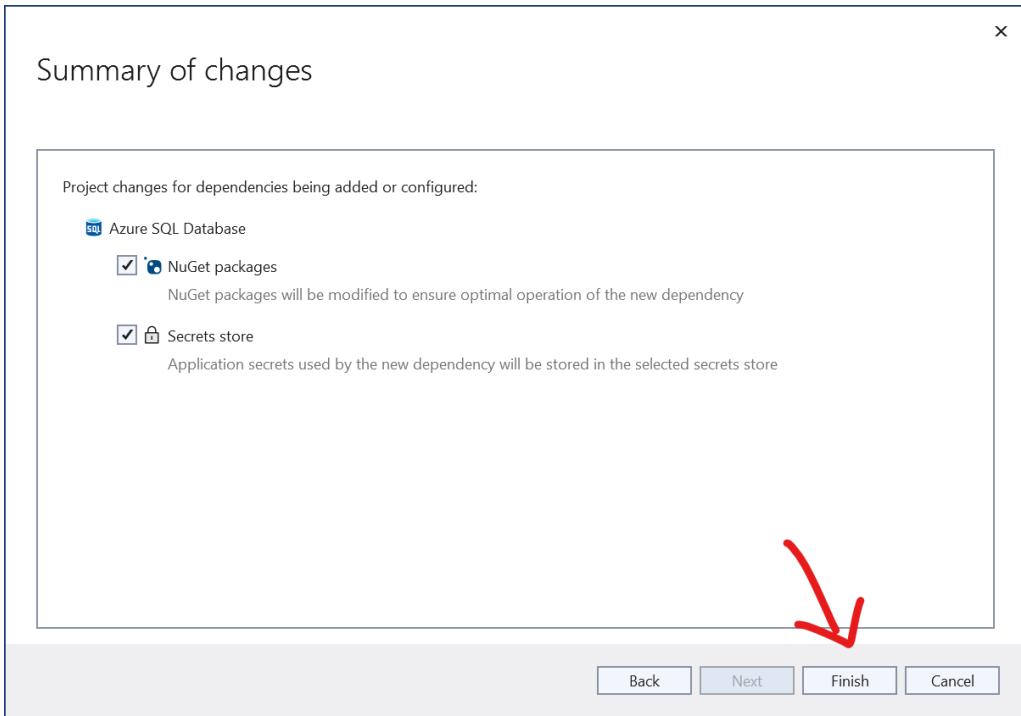
Save connection string value in: [Learn more](#)

Azure App Settings  
 Azure Key Vault  
 None

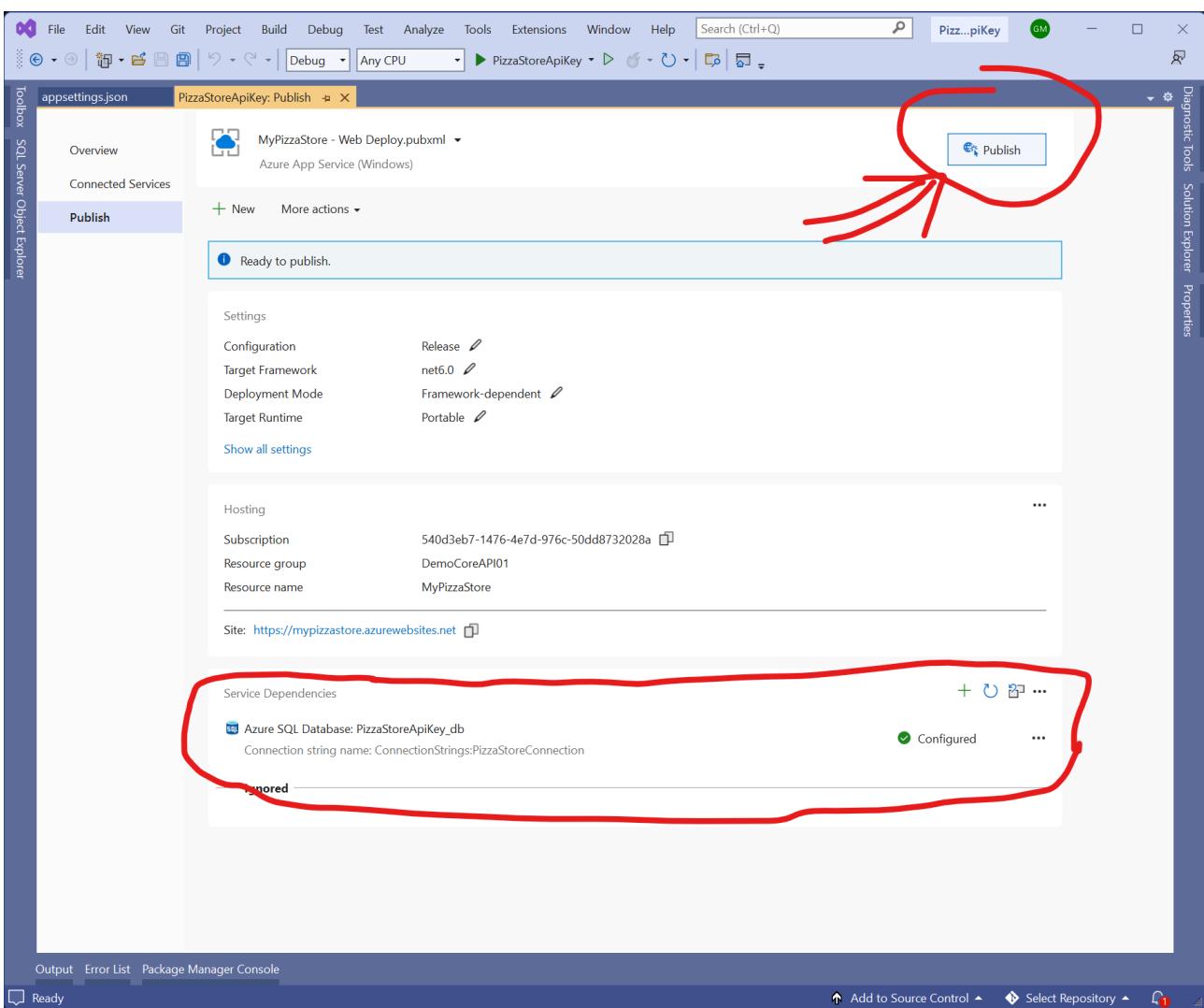
Back Next Finish Cancel



Inseriamo i dati richiesti e clicchiamo su Next e poi su Finish:



A questo punto la connessione al database dovrebbe comparire come configurata correttamente e tutto è pronto per la pubblicazione. Clicchiamo sul pulsante publish:



Dopo alcuni secondi si dovrebbe aprire la pagina del browser al link:  
<https://mypizzastore.azurewebsites.net/>

Tuttavia per vedere la pagina di Swagger bisogna inserire la rottta:

<https://mypizzastore.azurewebsites.net/swagger/index.html>

L'applicazione è online!

Per fare in modo che quando si va sulla rottta principale dell'applicazione si apra direttamente la pagina di Swagger, occorre fare una piccola modifica al codice del file Program.cs:

Decommentare la riga di codice evidenziata sotto:

```
app.UseSwaggerUI(c =>
{
    c.SwaggerEndpoint("/swagger/v1/swagger.json", "PizzaStore API V1");
    c.RoutePrefix = string.Empty;
});
```

Ricompilare l'applicazione e rifare la pubblicazione cliccando su publish:

Se si prova a registrarsi con un nuovo account, bisogna ovviamente ricordare che, avendo usato una mail fake i messaggi di posta inviati dovranno essere recuperati dall'account di posta utilizzato di ethereal.email. L'utilizzo di un account reale di posta elettronica reale (non fake) non è affatto difficile e rende l'applicazione utilizzabile realmente come nel caso di <https://pizzastoreapi.azurewebsites.net/>

### 13.9.1.4.5.3.2. Parte Opzionale – uso di SQL Server Express

Questa parte serve solo nel caso in cui non si sia fatto ricorso all'utilizzo di SQL Server Local Db.

Prima di procedere al deployment su Azure è anche possibile provare ad installare in locale la versione gratuita di Microsoft SQL Server (**Express edition**) dal link: <https://www.microsoft.com/it-it/sql-server/sql-server-downloads> e utilizzare il wizard di installazione con l'opzione “typical install”.

Il DBMS Microsoft SQL Server si installa e, nella configurazione di default, parte all'avvio di Windows, inoltre imposta l'accesso (nelle impostazioni predefinte) con la Windows authentication, per cui non c'è bisogno di specificare (in locale) le credenziali di accesso.

Per testare il progetto con Microsoft SQL Server (express) in locale occorre cambiare la stringa di connessione come segue:

```
"PizzaStoreConnection":  
"server=localhost\\sqlexpress;database=pizza_store_cookie;trusted_connection=true"
```

Prima di testare la connessione al database, occorre fare nuovamente la migrazione, poiché l'operazione di migrazione scrive le istruzioni per la creazione delle tabelle del database in base allo specifico DBMS che dovrà essere utilizzato.

Cancelliamo la precedente migrazione che era stata fatta per MariaDb e creiamo una nuova migrazione, questa volta per SQL Server Express.

Dalla Packet Manager Console di Visual Studio:

```
PM> Add-Migration InitialMigrate  
Build started...  
Build succeeded.  
Microsoft.EntityFrameworkCore.Infrastructure[10403]  
  Entity Framework Core 6.0.4 initialized 'AppDbContext' using provider  
'Microsoft.EntityFrameworkCore.SqlServer:6.0.4' with options: None  
To undo this action, use Remove-Migration.
```

La creazione del database verrà fatta automaticamente dal codice dell'applicazione, dal momento che sono state inserite le istruzioni per la migrazione da codice:

```
//migrazione con code first approach - solo development e testing  
using (var serviceScope = app.Services.CreateScope())  
{  
    var context = serviceScope.ServiceProvider.GetRequiredService<AppDbContext>();  
    context.Database.Migrate();  
    //vedere IN ALTERNATIVA a Migrate anche  
    //context.Database.EnsureCreated();  
}
```

Eseguiamo l'applicazione in locale:

PizzaStore API v1 OAS3

/swagger/v1/swagger.json

Make the pizzas you love, but with a little of security in mind

Authorize

### PizzaStoreApiKey

- POST** /accounts/register
- GET** /accounts/verify-email
- POST** /accounts/verify-email
- POST** /accounts/login
- POST** /accounts/logout
- POST** /accounts/forgot-password
- POST** /accounts/reset-password
- GET** /accounts/reset-password
- POST** /accounts/change-password Require authentication with policy OnlyVerifiedEmailAccount; role Standard,Administrator
- GET** /accounts/profile Require authentication with role Standard,Administrator
- POST** /accounts/change-profile Require authentication with role Standard,Administrator
- POST** /accounts/change-role Require authentication with policy OnlyVerifiedEmailAccount; role Administrator

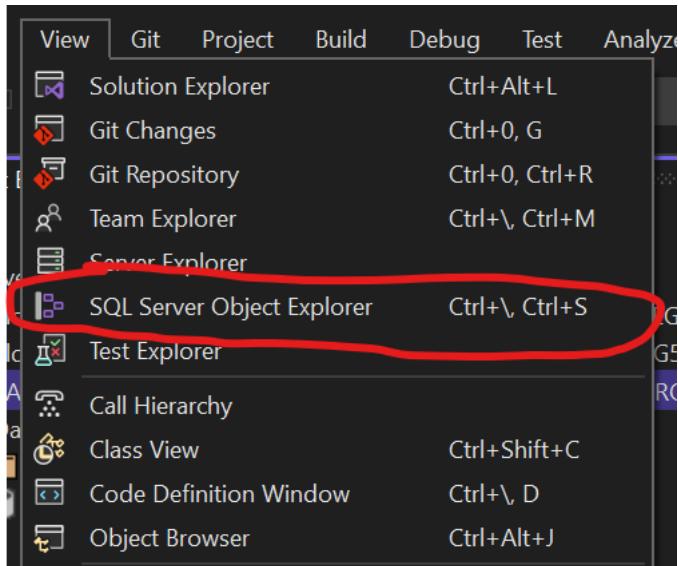
Per gestire, tramite interfaccia grafica, il DBMS SQL Server, si può utilizzare l'applicazione SQL Server Management Studio, che si può scaricare gratuitamente da link:

<https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15#download-ssms>

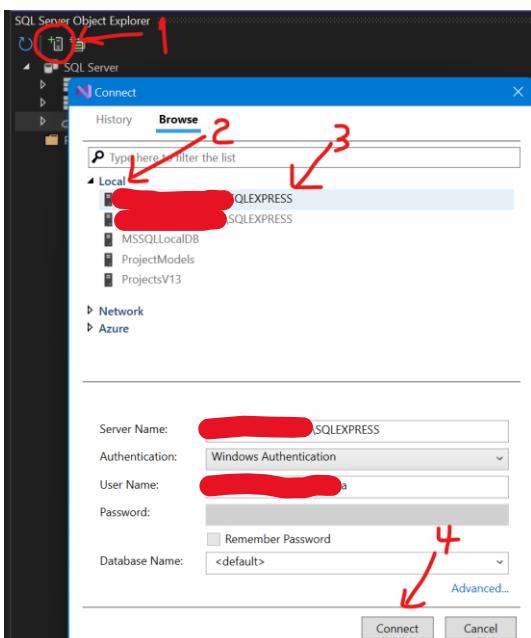
Attenzione! Il programma ha una dimensione di circa 650 MB, per cui non va installato a scuola

User	Username	Surname	EmailAddress	IsEmailConfirmed	ValidationToken	ValidationTokenExpireDate	PasswordHash	Salt	Role	CreatedOn	UpdatedOn
1	Gennaro	Malatona	gennaro.malatona@isegppit.it	1	f7f191e6bf1f3a5f9e032a48a37fb81fc9b199c	2022-04-28 16:15:29.089008	0x1E7D7B7257028724D8A2C8274FF0F1AE03770870F0...	040CE16DC088F924367B108ED49FD0...	1	2022-04-28 16:15:29.0880757	2022-04-28 16:15:29.0880757

Ottimamente, più semplicemente, e senza dover scaricare nessun programma aggiuntivo, utilizzando il connettore integrato in Visual Studio: View → SQL Server Object Explorer



Nel pannello di SQL Server Object Explorer, si crea una nuova connessione, premite il pulsante “Add SQL Server”, seguendo nell’ordine le opzioni da selezionare, mostrate nella figura seguente:



Per visualizzare o manipolare i dati del database, basta selezionare una tabella e poi con click destro del mouse aprire il menù contestuale e selezionare l’opzione View Data.

SQL Server Object Explorer

The screenshot shows the SQL Server Object Explorer interface. A red arrow points from the top right towards the 'dbo.Users' table node. A context menu is open over the 'dbo.Users' node, listing options: Data Comparison..., Script As, View Code, View Designer, View Permissions, View Data (which is highlighted with a blue border), Delete, Rename, Refresh, and Properties. Below the menu, a table is displayed with two rows of data.

	UserId	GivenName	Surname	EmailAddress	IsEmailConfirmed	ValidationToken	ValidationTokenExpiry	PasswordHash	Salt	Role	CreatedOn	UpdatedOn
▶	1	Gennaro	Malafronte	gennaro.malafr...	True	fYLNmw2xFRld...	4/28/2022 4:15...	0x1EF7D7B725...	0x6DCE16DCE0...	1	4/28/2022 4:15...	4/28/2022 4:15...
●	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

## 14. Xamarin

<https://docs.microsoft.com/en-us/xamarin/>

### 14.1. Xamarin Forms

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/>

### 14.2. Xamarin Getting Started

<https://docs.microsoft.com/en-us/xamarin/get-started/>

#### 14.2.1. Che cosa è Xamarin?

<https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin>

Xamarin è una piattaforma open source per la compilazione di applicazioni moderne e a prestazioni elevate per iOS, Android e Windows con .NET. Xamarin è un livello di astrazione che gestisce la comunicazione del codice condiviso con il codice della piattaforma sottostante. Xamarin viene eseguito in un ambiente gestito che offre diversi vantaggi, ad esempio l'allocazione della memoria e la Garbage Collection.

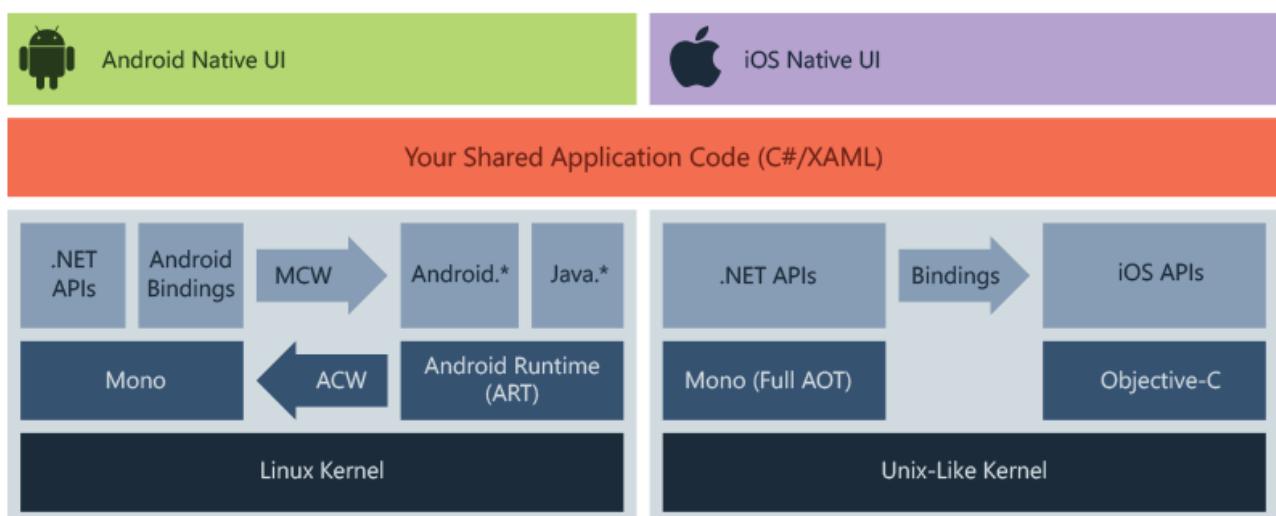
Xamarin consente agli sviluppatori di condividere una media del 90% delle applicazioni su più piattaforme. Questo modello consente agli sviluppatori di scrivere tutta la logica di business in un unico linguaggio (o di riutilizzare il codice dell'applicazione esistente), replicando in ogni piattaforma le prestazioni e l'aspetto originari.

Le applicazioni Xamarin possono essere scritte su PC o Mac e vengono compilate in pacchetti dell'applicazione nativa, ad esempio un file **APK** in Android o un file **IPA** in iOS.

#### Nota

Per la compilazione e la distribuzione di applicazioni per iOS è attualmente necessario un computer MacOS. Per altre informazioni sui requisiti di sviluppo, vedere [Requisiti di sistema](#).

#### 14.2.1.1. Funzionamento di Xamarin



Il diagramma illustra l'architettura generale di un'applicazione Xamarin multiplataforma. Xamarin consente di creare un'interfaccia utente nativa in ogni piattaforma e di scrivere in C# la logica di business da condividere tra le piattaforme. Nella maggior parte dei casi, l'80% del codice dell'applicazione è condivisibile tramite Xamarin.

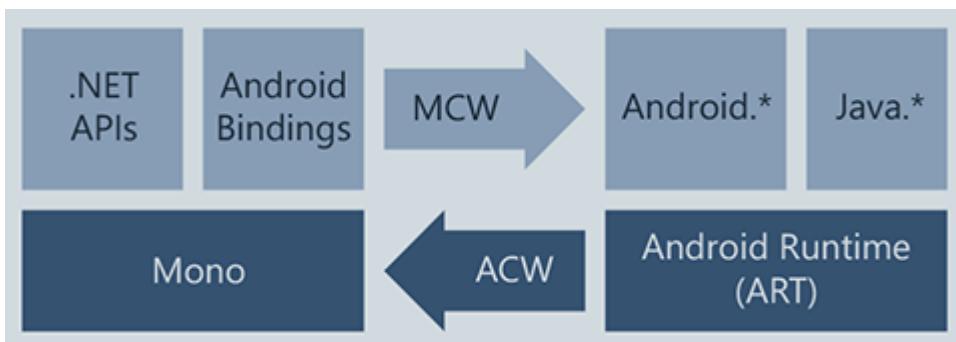
Xamarin è stato progettato basandosi su **Mono**, una versione open source di .NET Framework basata sugli standard ECMA .NET. Mono esiste quasi da quando esiste .NET Framework e viene eseguito nella maggior parte delle piattaforme, tra cui Linux, Unix, FreeBSD e macOS. L'ambiente di esecuzione di Mono gestisce automaticamente attività come l'allocazione della memoria, la Garbage Collection e l'interoperabilità con le piattaforme sottostanti.

Per altre informazioni sull'architettura specifica di una piattaforma, vedere [Xamarin.Android](#) e [Xamarin.iOS](#).

#### 14.2.1.2. Funzionalità aggiuntive

<https://docs.microsoft.com/it-it/xamarin/get-started/what-is-xamarin#added-features>

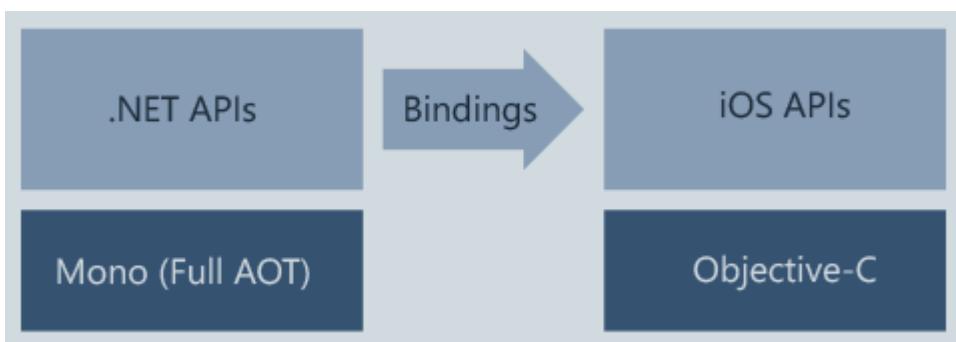
#### 14.2.1.3. Xamarin.Android



Le applicazioni Xamarin.Android vengono compilate da C# nel **linguaggio intermedio (IL)**, che viene quindi compilato in modalità **JIT (Just-In-Time)** in un assembly nativo all'avvio dell'applicazione. Le applicazioni Xamarin.Android vengono eseguite all'interno dell'ambiente di esecuzione di Mono, side-by-side con la macchina virtuale Android Runtime (ART). Xamarin fornisce i binding .NET agli spazi dei nomi Android.\* e Java.\*. L'ambiente di esecuzione di Mono chiama questi spazi dei nomi tramite **MCW (Managed Callable Wrapper)** e fornisce **ACW (Android Callable Wrapper)** ad ART, consentendo a entrambi gli ambienti di richiamare il codice reciprocamente.

Per altre informazioni, vedere [Architettura di Xamarin.Android](#).

#### 14.2.1.4. Xamarin.iOS



Le applicazioni Xamarin.iOS sono compilate completamente in modalità **AOT (Ahead Of Time)** da C# nel codice dell'assembly ARM nativo. Xamarin usa i **selettori** per esporre Objective-C ai **registratori** C# gestiti e poter esporre il codice C# gestito a Objective-C. I selettori e i registratori sono denominati collettivamente "binding" e consentono la comunicazione tra Objective-C e C#.

Per altre informazioni, vedere [Architettura di Xamarin.iOS](#).

#### 14.2.1.5. Xamarin.Essentials

Xamarin.Essentials è una libreria che fornisce API multipiattaforma per le funzionalità native dei

dispositivi. Come Xamarin, Xamarin.Essentials è un'astrazione che semplifica il processo di accesso alle funzionalità native. Di seguito sono riportati alcuni esempi di funzionalità disponibili in Xamarin.Essentials:

- Informazioni sul dispositivo
- File system
- Accelerometro
- Dialer telefono
- Sintesi vocale
- Blocco dello schermo

Per altre informazioni, vedere [Xamarin.Essentials](#).

#### 14.2.1.6. *Xamarin.Forms*

Xamarin.Forms è un framework di interfaccia utente open source. Xamarin.Forms consente agli sviluppatori di compilare applicazioni iOS, Android e Windows da una singola codebase condivisa. Xamarin.Forms consente agli sviluppatori di creare interfacce utente in XAML con code-behind in C#. Queste interfacce utente vengono visualizzate come controlli nativi a prestazioni elevate in ogni piattaforma. Di seguito sono riportati alcuni esempi di funzionalità disponibili in Xamarin.Forms:

- Lingua dell'interfaccia utente XAML
- Data binding
- Movimenti
- Effetti
- Stile

Per altre informazioni, vedere [Xamarin.Forms](#).

#### 14.2.2. **Installazione di Xamarin**

<https://docs.microsoft.com/en-us/xamarin/get-started/installation/>

#### 14.2.3. **Piattaforme supportate da Xamarin.Forms**

È possibile scrivere applicazioni Xamarin.Forms per i sistemi operativi seguenti:

- iOS 9 o versione successiva.
- Android 4.4 (API 19) o versione successiva ([altri dettagli](#)). È tuttavia consigliabile usare Android 5.0 (API 21) come API minima, perché garantisce la compatibilità completa con tutte le librerie di supporto Android, pur continuando ad avere come destinazione la maggior parte dei dispositivi Android.
- Piattaforma UWP (Universal Windows Platform) per Windows 10.

Le app Xamarin.Forms per iOS, Android e la piattaforma UWP (Universal Windows Platform) possono essere compilate in Visual Studio. Per lo sviluppo iOS è tuttavia necessario un Mac connesso alla rete con la versione più recente di Xcode e la versione minima di macOS specificata da Apple. Per altre informazioni, vedere [Requisiti Windows](#).

Le app Xamarin.Forms per iOS e Android possono essere compilate in Visual Studio per Mac. Per altre informazioni, vedere [Requisiti macOS](#).

#### Nota

Lo sviluppo di app con Xamarin.Forms richiede familiarità con [.NET Standard](#).

- **Supporto di piattaforme aggiuntive**

Xamarin.Forms supporta altre piattaforme oltre a iOS, Android e Windows:

- Samsung Tizen
- macOS
- GTK#
- WPF

Lo stato di queste piattaforme è disponibile nella [wiki di GitHub sul supporto delle piattaforme di Xamarin.Forms](#).

- **Supporto della piattaforma Android**

È consigliabile installare la versione più recente di Android SDK Tools e della piattaforma API Android. È possibile eseguire l'aggiornamento alle versioni più recenti usando [Android SDK Manager](#).

La versione di destinazione/compilazione per i progetti Android **deve** inoltre essere impostata su *Usa la piattaforma installata più recente*. La versione minima può tuttavia essere impostata su API 19 in modo da poter continuare a supportare i dispositivi che usano Android 4.4 e versioni successive. Questi valori vengono impostati in **Opzioni progetto**:

- [Visual Studio](#)
- [Visual Studio per Mac](#)

#### 14.2.4. Installazione dell'emulatore Android

<https://docs.microsoft.com/en-us/xamarin/android/get-started/installation/android-emulator/>

##### 14.2.4.1. Accelerazione hardware per le prestazioni dell'emulatore (Hyper-V e HAXM)

<https://docs.microsoft.com/en-us/xamarin/android/get-started/installation/android-emulator/hardware-acceleration?pivot=windows>

#### 14.2.5. Set Up Device for Development (Configurare il dispositivo per lo sviluppo)

<https://docs.microsoft.com/en-us/xamarin/android/get-started/installation/set-up-device-for-development>

#### 14.2.6. Prima App Xamarin.Forms

<https://docs.microsoft.com/en-us/xamarin/get-started/first-app/>

#### 14.2.7. Understanding Android API Levels (Informazioni sui livelli API Android)

<https://docs.microsoft.com/en-us/xamarin/android/app-fundamentals/android-api-levels>

Xamarin.Android espone tre impostazioni di progetto a livello di API Android:

- [Framework di destinazione](#) (Target Framework) – specifica il Framework da usare per la compilazione dell'applicazione. Questo livello API viene usato in fase di *compilazione* da Xamarin.Android.

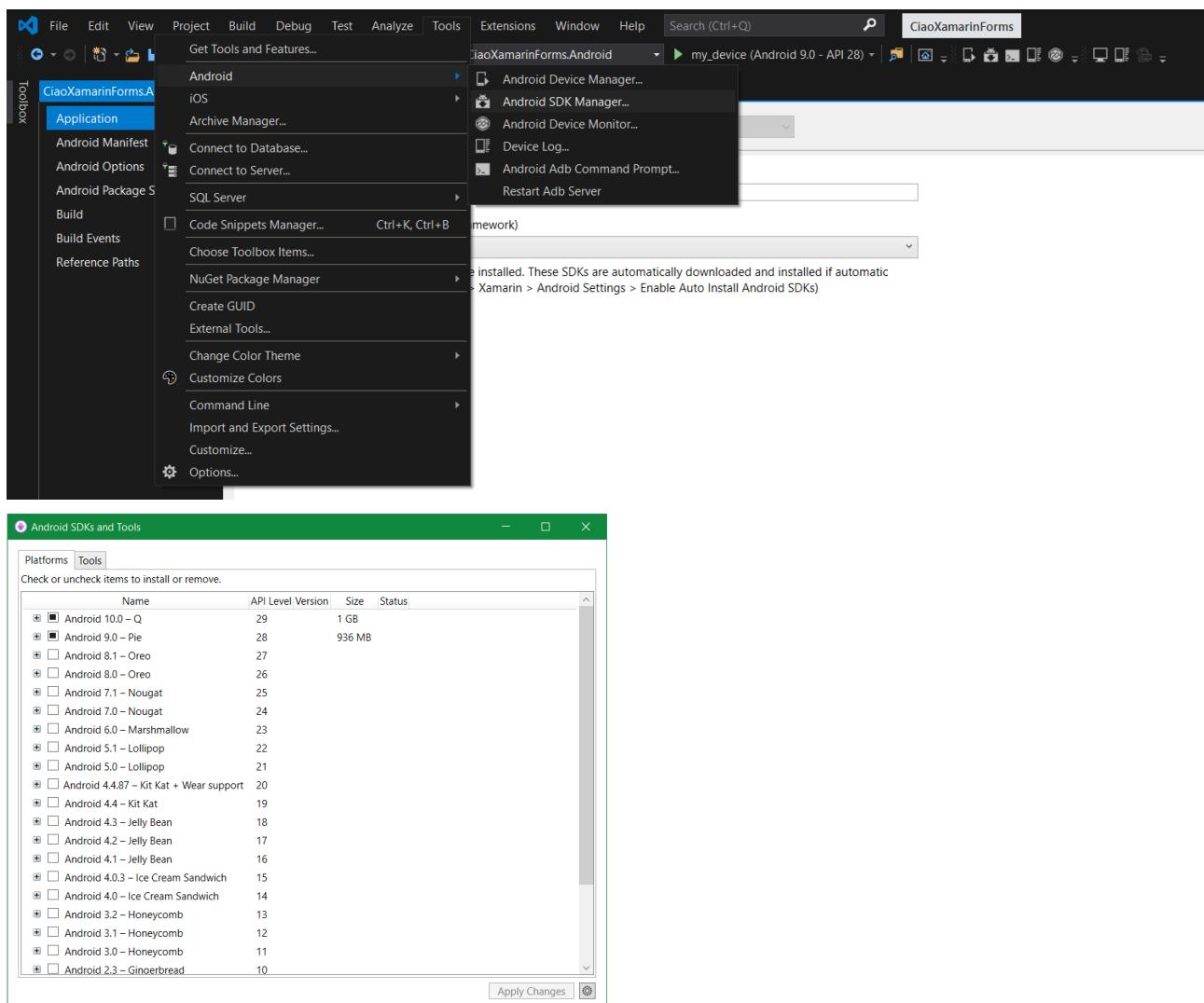
- Versione minima di android (Minimum SDK Version) – specifica la versione di Android meno recente che si vuole che l'app supporti. Questo livello API viene usato in fase di *esecuzione* da Android.
- La versione di Android di destinazione (Target Android Version) – specifica la versione di Android in cui deve essere eseguita l'app. Questo livello API viene usato in fase di *esecuzione* da Android.

Prima di poter configurare un livello API per il progetto, è necessario installare i componenti della piattaforma SDK per tale livello API. Per ulteriori informazioni sul download e sull'installazione dei componenti di Android SDK, vedere [Android SDK installazione](#).

## Nota

A partire dall'agosto 2018, la console di Google Play richiederà che le nuove app siano destinate a livello API 26 (Android 8,0) o versione successiva. Le app esistenti saranno necessarie per il livello API di destinazione 26 o superiore a partire dal 2018 novembre. Per ulteriori informazioni, vedere [migliorare la sicurezza delle app e le prestazioni in Google Play per anni](#).

Per aprire Android SDK Manager: Tools>Android>Android SDK Manager



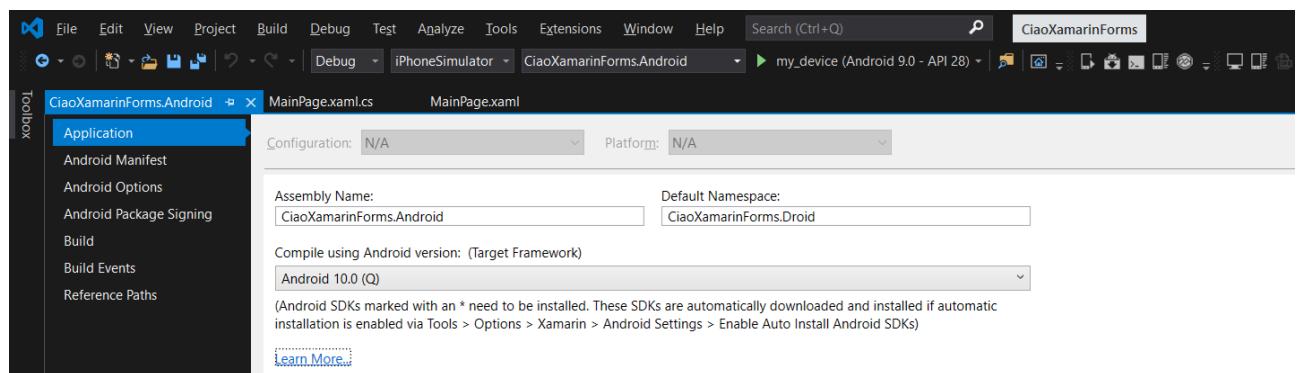
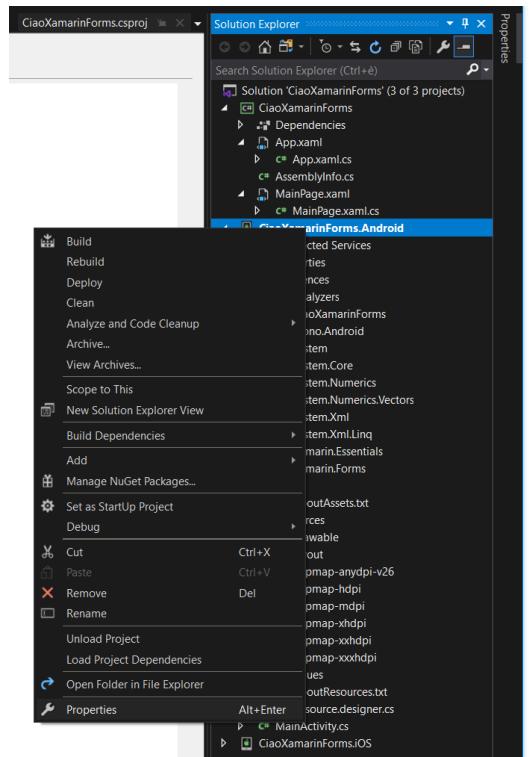
Per compilare un'applicazione Android da Xamarin vengono utilizzate due parametri di compilazione:

- la Target SDK version (corrisponde alla versione di Android rispetto alla quale l'app viene compilata e testata)
- la Minimum SDK version (corrisponde alla versione minima di Android rispetto alla quale l'applicazione è compatibile)

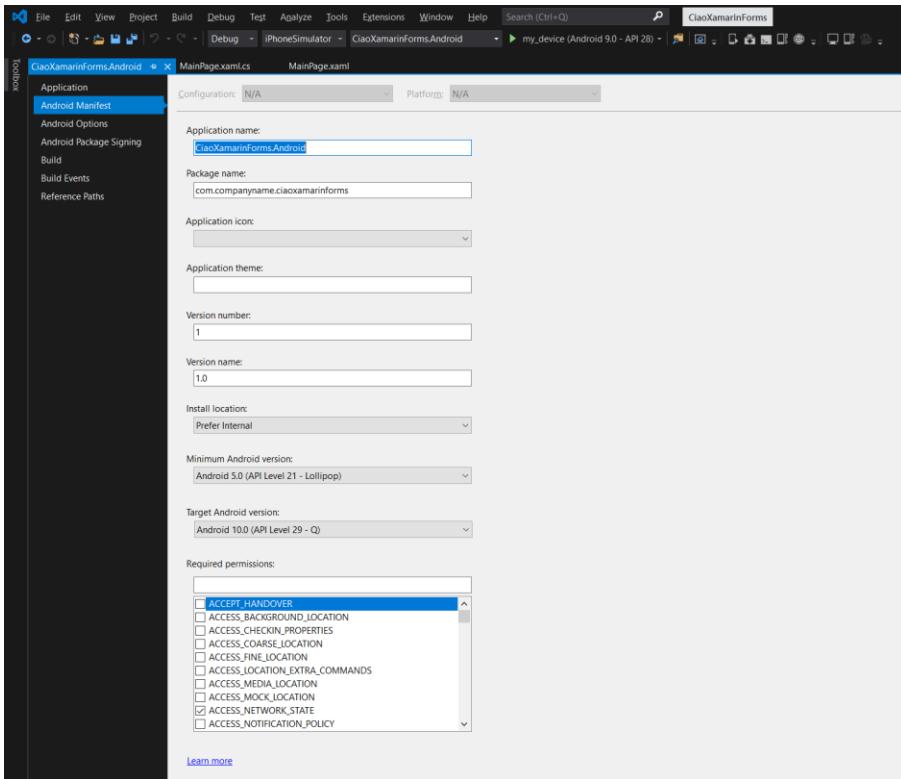
Si raccomanda di usare come Target SDK la versione più recente di Android SDK disponibile e di non andare al di sotto della versione 5.0 di Android (API level 21)

Per la compilazione dell'app Android la versione di SDK utilizzata è definita in:

NomeProgetto.Android → tasto destro → Proprietà → Application



Per vedere la configurazione di Build dell'app Android generata da Xamarin bisogna andare su NomeProgetto.Android → tasto destro → Proprietà → Android Manifest



Per comprendere l'impatto della scelta della minimum SDK version sul market share della propria App considerare anche il grafico riportato qui:

<https://developer.android.com/about/dashboards>

## 14.2.8. Xamarin QuickStart

In questa sezione verrà presentato un primo assaggio di quello che è possibile fare con Xamarin. Ognuno dei concetti qui esposti verrà successivamente ripreso e trattato in dettaglio.

Quick Start: <https://docs.microsoft.com/en-us/xamarin/get-started/quickstarts/>

Esempio App Notes parte 1 (salvataggio di una nota con file): <https://docs.microsoft.com/en-us/xamarin/get-started/quickstarts/app>

Esempio App Notes parte 2 (salvataggio di un elenco di note con file, introduzione al concetto di navigazione mediante Shell): <https://docs.microsoft.com/en-us/xamarin/get-started/quickstarts/navigation>

Esempio App Notes parte 3 (salvataggio di un elenco di note con un database, uso di SQLite per .Net, libreria sqlite-net-pcl): <https://docs.microsoft.com/en-us/xamarin/get-started/quickstarts/database>

Esempio App Notes parte 4 (utilizzo di stili e dizionari di risorse per modificare l'aspetto grafico di un'App): <https://docs.microsoft.com/en-us/xamarin/get-started/quickstarts/styling>

Esempio App Notes Deep Dives (spiegazione dei concetti fondamentali introdotti): <https://docs.microsoft.com/en-us/xamarin/get-started/quickstarts/deepdive>

## 14.2.9. Xamarin Tutorials

Xamarin Forms Tutorials: <https://docs.microsoft.com/en-us/xamarin/get-started/tutorials/>

StackLayout: <https://docs.microsoft.com/en-us/xamarin/get-started/tutorials/stacklayout/>

Grid: <https://docs.microsoft.com/en-us/xamarin/get-started/tutorials/grid/>  
Label: <https://docs.microsoft.com/en-us/xamarin/get-started/tutorials/label/>  
Button: <https://docs.microsoft.com/en-us/xamarin/get-started/tutorials/button/>  
Entry (single line text): <https://docs.microsoft.com/en-us/xamarin/get-started/tutorials/entry/>  
Editor (multi line text): <https://docs.microsoft.com/en-us/xamarin/get-started/tutorials/editor/>  
Image: <https://docs.microsoft.com/en-us/xamarin/get-started/tutorials/image/>  
CollectionView: <https://docs.microsoft.com/en-us/xamarin/get-started/tutorials/collectionview/>  
Pops-up: <https://docs.microsoft.com/en-us/xamarin/get-started/tutorials/pop-ups/>  
Toast:  
<https://stackoverflow.com/a/54744638>  
<https://stackoverflow.com/a/66242105>

You can use Snackbar from [Xamarin Community toolkit](#) package, which uses native implementation in platforms where natively supported, because Toast is deprecated in API level 30, a Snackbar without an Action is equivalent to a Toast.

<https://github.com/xamarin/XamarinCommunityToolkit/blob/main/samples/XCT.Sample/Pages/Views/SnackBarPage.xaml.cs>  
<https://github.com/xamarin/XamarinCommunityToolkit>  
Esercizi su Xamarin.Forms: <https://docs.microsoft.com/it-it/samples/browse/?products=xamarin&term=Xamarin.Forms>  
Percorsi di Microsoft Learn su Xamarin: <https://docs.microsoft.com/en-us/learn/browse/?term=xamarin>

#### 14.2.10. Xamarin Forms XAML

Xaml: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/>  
Xaml Basics: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/xaml-basics/>  
Part 1. Getting Started with XAML: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/xaml-basics/get-started-with-xaml>  
Part 2. Essential XAML Syntax: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/xaml-basics/essential-xaml-syntax>  
Part 3. XAML Markup Extensions: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/xaml-basics/xaml-markup-extensions>  
Part 4. Data Binding Basics: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/xaml-basics/data-binding-basics>  
Part 5. From Data Bindings to MVVM: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/xaml-basics/data-bindings-to-mvvm>  
MVVM in generale: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm>

Command with multiple component: <https://docs.microsoft.com/en-us/answers/questions/619166/how-to-use-command-parameter-with-multiple-component.html>

Controls References: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/controls/>

XAML Controls: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/xaml/xaml-controls>

## 14.2.11.

### 14.3.App fundamentals

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/>

App Class: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/application-class>

App Lifecycle: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/app-lifecycle>

Data Binding (again): <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/data-binding/>

Comportamenti: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/behaviors/>

Elementi dell'interfaccia utente: <https://docs.microsoft.com/en/xamarin/xamarin-forms/user-interface/>

Notifiche locali: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/local-notifications>

#### 14.3.1.1. Navigazione

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/navigation/>

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/controls/pages>

Navigazione Gerarchica (NavigationPage):

<https://docs.microsoft.com/en-us/samples/xamarin/xamarin-forms-samples/navigation-hierarchical/>

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/navigation/hierarchical>

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/controls/pages> (sezione NavigationPage)

Esempi tratti da [xamarin-forms-examples](#)/navigation: Hierarchical, LoginFlow, PassingData

Pagine modali:

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/navigation/modal>

Esempi tratti da [xamarin-forms-examples](#)/navigation: Modal

Tabbed Page:

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/navigation/tabbed-page>

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/controls/pages> (sezione Tabbed Page)

Esempi tratti da [xamarin-forms-examples](#)/navigation: TabbedPage, TabbedPageSVGIcons, TabbedPageWithNavigationPage, Pop-ups

FlyoutPage:

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/navigation/flyoutpage>

Esempi tratti da [xamarin-forms-examples](#)/navigation: FlyoutPage

Shell:

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/shell/>

<https://docs.microsoft.com/en-us/samples/xamarin/xamarin-forms-samples/userinterface-xaminals/>

Esempi tratti da [xamarin-forms-examples](#)/userinterface: Xaminals

CarouselView:

il CarouselPage è stato sostituito dal CarouselView:

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/carouselview/>

<https://docs.microsoft.com/en-us/samples/xamarin/xamarin-forms-samples/userinterface-carouselviewdemos/>

Esempi tratti da [xamarin-forms-examples](#)/userinterface: CarouselViewDemos

### 14.3.2. Xamarin.Essentials

Accesso alle API native con Xamarin.Essentials: <https://docs.microsoft.com/en-us/xamarin/essentials/?context=xamarin/xamarin-forms>

### 14.3.3. Dati e servizi cloud

File: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/data-cloud/data/files?tabs=windows>

Database locale: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/data-cloud/data/databases>

#### 14.3.3.1. Servizi Azure

Inviare e ricevere notifiche push: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/data-cloud/azure-services/azure-notification-hub>

Esempio di notifiche push con Xamarin.Forms: <https://docs.microsoft.com/en-us/samples/xamarin/xamarin-forms-samples/webservices-azurenotificationhub/>

Links sulle notifiche:

<https://docs.microsoft.com/it-it/xamarin/xamarin-forms/data-cloud/azure-services/azure-notification-hub>

<https://docs.microsoft.com/it-it/azure/notification-hubs/xamarin-notification-hubs-push-notifications-android-gcm>

Sorgenti

<https://github.com/Azure/azure-notificationhubs-dotnet/tree/master/Samples/Xamarin/GetStartedXamarinAndroid>

<https://docs.microsoft.com/it-it/azure/app-service-mobile/app-service-mobile-xamarin-forms-get-started-push>

<https://docs.microsoft.com/it-it/azure/notification-hubs/notification-hubs-android-push-notification-google-fcm-get-started> (PHP)

Creazione Push per progetto Android:

<https://docs.microsoft.com/it-it/azure/notification-hubs/notification-hubs-android-push-notification-google-fcm-get-started#prerequisites>

#### 14.3.4. Esempi di applicazioni

Esempi di applicazioni Xamarin:

<https://github.com/xamarin/xamarin-forms-samples>

<https://github.com/xamarin/xamarin-forms-samples/tree/main/FormsGallery>

Tipi di Layout:

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/layouts/layout-options>

FlexLayout:

Visto dall'esempio di FormsGallery nel caso della RefreshView

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/layouts/flex-layout>

RefreshView:

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/refreshview>

<https://docs.microsoft.com/en-us/samples/xamarin/xamarin-forms-samples/userinterface-refreshviewdemo/>

SearchBar:

<https://docs.microsoft.com/it-it/xamarin/xamarin-forms/user-interface/searchbar>

Presente nell'esempio di FormsGallery nel caso della SearchBar e anche con maggiore dettaglio nell'esempio reperibile al link:

<https://docs.microsoft.com/en-us/samples/xamarin/xamarin-forms-samples/userinterface-searchbardemos/>

ListView:

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/listview/>

<https://docs.microsoft.com/en-us/xamarin/get-started/tutorials/listview/?tabs=vswin>

<https://docs.microsoft.com/en-us/samples/xamarin/xamarin-forms-samples/getstarted-tutorials-listviewtutorial/>

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/listview/interactivity>

<https://docs.microsoft.com/en-us/samples/xamarin/xamarin-forms-samples/workingwithlistview/>

<https://docs.microsoft.com/en-us/samples/xamarin/xamarin-forms-samples/userinterface-listview-builtincells/>

CarouselView:

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/carouselview/>

<https://docs.microsoft.com/it-it/samples/xamarin/xamarin-forms-samples/formsgallery/>

(CarouselView demo; nota: CarouselPage è stato sostituito da CarouselView)

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/layouts/frame> (cornice --> Frame)

<https://docs.microsoft.com/it-it/samples/xamarin/xamarin-forms-samples/userinterface-carouselviewdemos/> (l'esempio si può scaricare direttamente e funziona)

esempi interessanti da discutere, tratti dall'esempio carouselviewdemos:

HorizontalTemplateLayoutPage → provare ad aggiungere PeekAreaInsets

PullToRefresh

TableView:

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/tableview>

<https://docs.microsoft.com/en-us/samples/xamarin/xamarin-forms-samples/userinterface-tableview/>

CollectionView:

<https://docs.microsoft.com/en-us/samples/xamarin/xamarin-forms-samples/userinterface-collectionviewdemos/>

SwipeView:

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/swipeview>

Maps:

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/map/>

## **15. .NET ecosystem**

<https://dotnet.microsoft.com>

<https://dotnet.microsoft.com/download>

<https://dotnet.microsoft.com/download/visual-studio-sdks>

### **15.1.Learn .NET**

<https://dotnet.microsoft.com/learn> (il portale di riferimento per imparare le tecnologie .NET)

#### **15.1.1. .NET videos**

<https://dotnet.microsoft.com/learn/videos>

### **15.2..NET Standard**

<https://docs.microsoft.com/en-us/dotnet/standard/net-standard> (definizione)

<https://docs.microsoft.com/en-us/dotnet/standard/frameworks> (framework di destinazione)

### **15.3..NET Core**

<https://docs.microsoft.com/en-us/dotnet/core/>

#### **15.3.1. .NET Core command-line interface (CLI) tools**

<https://docs.microsoft.com/en-us/dotnet/fundamentals/tools-and-productivity>