***Esame C# e RestApi***

**Cos'è un'API?**

**Un'API (Application Programming Interface), o in italiano Interfaccia di Programmazione delle Applicazioni, è un meccanismo che permette a un programma di comunicare e interagire con un altro.**

**L'API definisce le regole e i protocolli per questa comunicazione, stabilendo sia come deve essere fatta la richiesta che il formato della risposta.**

**Cos'è un'API REST?**

**Un'API REST è un tipo di API che segue un insieme di principi di design specifici, chiamati REST (Representational State Transfer). Queste API sono diventate lo standard di fatto per le comunicazioni tra servizi web.**

**Le API REST sono particolarmente utili perché:**

* **Usano richieste HTTP standard: Per comunicare, usano gli stessi metodi del browser web (GET per ottenere dati, POST per creare nuovi dati, PUT per aggiornare dati, DELETE per eliminare dati).**
* **Sono "senza stato": Ogni richiesta è indipendente dalle altre. Il server non ha bisogno di ricordare le richieste precedenti. Questo rende i sistemi più scalabili e affidabili.**

**Le API sono uno strumento fondamentale per:**

* **Flessibilità e Semplificazione: Rendono molto più semplice l'integrazione di nuove funzionalità o servizi. Non devi ricostruire un'intera applicazione da zero; puoi semplicemente connetterla tramite un'API a un altro servizio che ha già le risorse che ti servono.**
* **Condivisione dei Dati: Le API permettono di esporre dati e funzionalità in modo controllato, consentendo alle aziende di condividere informazioni con clienti e partner esterni, il che apre la strada all'innovazione.**
* **Sviluppo Modulare: Supportano lo sviluppo di applicazioni moderne basate su cloud, dove diverse parti dell'applicazione (microservizi) possono essere gestite separatamente e comunicare tra loro tramite API.**

**Confronto tra REST e altre architetture (SOAP, GraphQL)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Caratteristica** | **Rest** | **Soap** | **GraphQL** |
| **Tipo** | **Stile architetturale** | **Protocollo** | **Linguaggio di query API** |
| **Formato Dati** | **JSON (più comune), XML** | **Solo XML** | **JSON** |
| **EndPoint** | **Multipli (una risorsa = un URL)** | **Multipli con WSDL** | **Singolo endpoint** |
| **Semplicità** | **Alta** | **Bassa** | **Media** |
| **Performance** | **Buona, ma rischio over/under-fetching** | **Più lenta (XML)** | **Alta (query mirate)** |
| **Caching** | **Semplice (HTTP cache)** | **Complesso** | **Difficile** |
| **Sicurezza** | |  | | --- | |  |  |  | | --- | | **OAuth, JWT, HTTPS** | | **WS-Security, SSL, HTTPS** | **Manuale, da implementare** |
| **Scenario Ideale** | **Applicazioni web/mobile** | **Enterprise, transazioni complesse** | **Frontend dinamici, real-time** |

**REST = semplice e universale, adatto alla maggior parte delle app.**

**SOAP = pesante ma sicuro e affidabile, perfetto per enterprise e transazioni.**

**GraphQL = flessibile e preciso, ideale quando il client vuole massimo controllo sui dati.**

**Metodi HTTP principali: GET, POST, PUT, DELETE**

**I quattro metodi principali corrispondono alle operazioni fondamentali del database (Create, Read, Update, Delete), spesso riassunte con l'acronimo CRUD.**

**1. GET (Leggi/Read)**

**Il metodo GET serve per richiedere e leggere una risorsa. Non modifica i dati sul server ed è considerato "sicuro" e "idempotente".**

* **A cosa serve: Ottenere la lista di tutti gli elementi, o i dettagli di un singolo elemento specifico.**
* **Esempi:**
  + **GET /libri: Chiede al server la lista di tutti i libri.**
  + **GET /libri/123: Chiede i dettagli del libro con l'ID 123.**

**2. POST (Crea/Create)**

**Il metodo POST serve per creare una nuova risorsa sul server. Si usa quando si inviano dati al server per la prima volta.**

* **A cosa serve: Inviare dati di un modulo, caricare un file, o creare un nuovo record.**
* **Esempi:**
  + **POST /utenti: Invia i dati di un nuovo utente per crearlo.**
  + **POST /ordini: Invia i dati di un nuovo ordine per registrarlo nel sistema.**

**3. PUT (Aggiorna/Update)**

**Il metodo PUT serve per aggiornare o sostituire una risorsa esistente. Come GET, è idempotente.**

* **A cosa serve: Sostituire l'intero contenuto di una risorsa con un nuovo contenuto.**
* **Esempi:**
  + **PUT /utenti/456: Aggiorna tutti i dati dell'utente con l'ID 456 con le nuove informazioni fornite nel corpo della richiesta.**
  + **PUT /prodotti/789: Sostituisce i dettagli del prodotto 789 con i nuovi dati.**

**4. DELETE (Elimina/Delete)**

**Il metodo DELETE serve per eliminare una risorsa specifica dal server.**

* **A cosa serve: Cancellare un record, un file o un utente.**
* **Esempi:**
  + **DELETE /prodotti/789: Elimina il prodotto con l'ID 789.**
  + **DELETE /commenti/101: Cancella il commento con l'ID 101.**

**Codici di stato HTTP più comuni.**

**I codici di stato HTTP sono la risposta del server a una richiesta del client. Sono essenziali per capire se la richiesta ha avuto successo o se c'è stato un errore, permettendo al client di gestire la situazione in modo appropriato.**

**I codici di stato sono organizzati in cinque classi, indicate dalla prima cifra:**

* **1xx (Informativa): La richiesta è stata ricevuta e il processo continua.**
* **2xx (Successo): L'azione richiesta è stata completata con successo.**
* **3xx (Reindirizzamento): Sono necessarie ulteriori azioni per completare la richiesta.**
* **4xx (Errore del client): La richiesta contiene un errore.**
* **5xx (Errore del server): Il server ha fallito nel completare una richiesta apparentemente valida.**

**Codici di Stato Più Comuni**

**2xx - Successo**

* **200 OK: La richiesta ha avuto successo. Questo è il codice più comune per le richieste GET.**
* **201 Created: La risorsa è stata creata con successo. Viene usato in genere dopo una richiesta POST.**
* **204 No Content: La richiesta ha avuto successo, ma non c'è contenuto da restituire nel corpo della risposta. È tipico per le richieste DELETE.**

**4xx - Errore del Client**

* **400 Bad Request: Il server non riesce a comprendere la richiesta a causa di una sintassi non corretta.**
* **401 Unauthorized: La richiesta non può essere completata perché richiede l'autenticazione dell'utente.**
* **403 Forbidden: L'utente è autenticato, ma non ha i permessi per accedere alla risorsa.**
* **404 Not Found: La risorsa che hai cercato non esiste sul server.**

**5xx - Errore del Server**

* **500 Internal Server Error: Un errore generico del server che non è legato a un problema nella tua richiesta. Il server non sa come gestirlo.**
* **503 Service Unavailable: Il server non è in grado di gestire la richiesta, in genere a causa di un sovraccarico o di un'attività di manutenzione.**

**Autenticazione e sicurezza.**

**La sicurezza è un aspetto cruciale in qualsiasi applicazione web, e le API non fanno eccezione. L'autenticazione è il processo per verificare l'identità di chi sta facendo una richiesta.**

**API Key**

**Pensa a una API Key come a una password lunga e segreta che usi per accedere a un servizio. È una stringa di caratteri alfanumerici univoca che il server fornisce a un utente o a un'applicazione.**

* **Come funziona: Ogni volta che il client vuole fare una richiesta a un'API protetta, deve includere la sua API Key, di solito nell'intestazione della richiesta HTTP.**
* **A cosa serve: Il server riceve la richiesta e controlla se la chiave fornita è valida e se ha i permessi per accedere alla risorsa richiesta.**
* **Vantaggi:**
  + **Facile da implementare: È un sistema semplice e veloce da configurare.**
  + **Controllo sull'accesso: Se una chiave viene compromessa, puoi semplicemente disattivarla senza dover reimpostare le password di tutti gli utenti.**
* **Svantaggi:**
  + **Non identificativa: La chiave identifica l'applicazione, non l'utente finale.**
  + **Gestione: Se una chiave viene esposta chiunque può usarla.**

**JWT (JSON Web Token)**

**Un JWT, o JSON Web Token, è un modo più moderno e flessibile per gestire l'autenticazione. Immagina un JWT come un biglietto d'ingresso digitale che contiene informazioni sull'identità dell'utente e sui suoi permessi.**

* **Come funziona:**
  + **Login: Quando un utente fa il login con successo, il server genera un JWT.**
  + **Passaggio: Questo JWT viene inviato al client , che lo memorizza.**
  + **Richieste: Per ogni richiesta successiva a un'API protetta, il client invia questo JWT nell'intestazione della richiesta.**
  + **Verifica: Il server riceve la richiesta, "legge" le informazioni nel JWT e, usando una chiave segreta, verifica che il token sia autentico e non sia stato manomesso.**
* **Vantaggi:**
  1. **Sicurezza: Il token è firmato digitalmente, quindi il server può verificare la sua integrità.**
  2. **Scalabilità: Essendo "senza stato", riduce il carico sul server e lo rende ideale per sistemi distribuiti.**
  3. **Informazioni nel token: Un JWT può contenere informazioni aggiuntive sull'utente, che possono essere utilizzate per i controlli di autorizzazione.**
* **Svantaggi:**
  1. **Invalida: Una volta che un token è stato emesso, è valido fino alla sua scadenza. Non puoi "annullarlo" a meno che non implementi un meccanismo aggiuntivo.**
  2. **Esponibile: Se il token viene intercettato da un malintenzionato, può essere utilizzato fino alla sua scadenza.**

**Descrizione del progetto:**

**Il progetto consiste nella creazione di un’API RESTful che espone servizi per la gestione di una libreria virtuale.  
Le principali caratteristiche sono:**

**Funzionalità previste:**

* + **CRUD sugli Autori: possibilità di creare, leggere, aggiornare ed eliminare gli autori registrati.**
  + **CRUD sui Libri: gestione completa dei libri connessi agli autori.**
  + **Relazioni Autore–Libro: un autore può avere più libri; i libri sono collegati a un autore specifico.**
  + **Routing RESTful: gli endpoint seguono convenzioni REST (es. /api/autori, /api/libri).**
  + **Gestione JSON: input e output delle richieste vengono gestiti in formato JSON.**

**Architettura del codice:**

* **Controllers: espongono gli endpoint HTTP.**
* **Models: definiscono le entità principali (Autore, Libro).**
* **Services: gestiscono la logica applicativa e l’accesso ai dati.**
* **Configurazione: predisposizione per ambienti Debug e Release.**

**Codice sorgente del progetto:**

**https://localhost:7006**

**Documentazione endpoint:**

[**https://localhost:7006/swagger/index.html**](https://localhost:7006/swagger/index.html)

**Repository GitHub:**

[**https://github.com/cdilallo-beep/Esame-C--e-RestApi.git**](https://github.com/cdilallo-beep/Esame-C--e-RestApi.git)

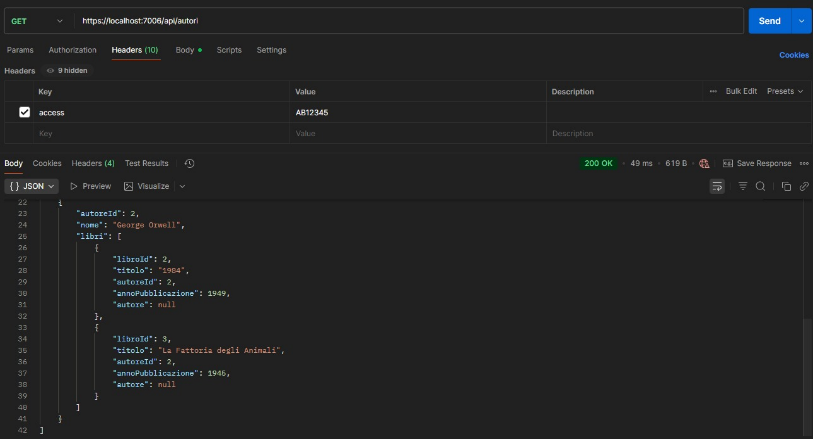
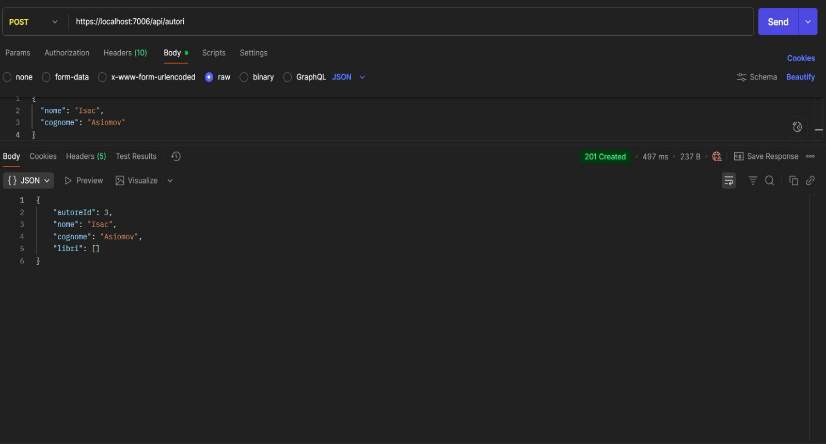
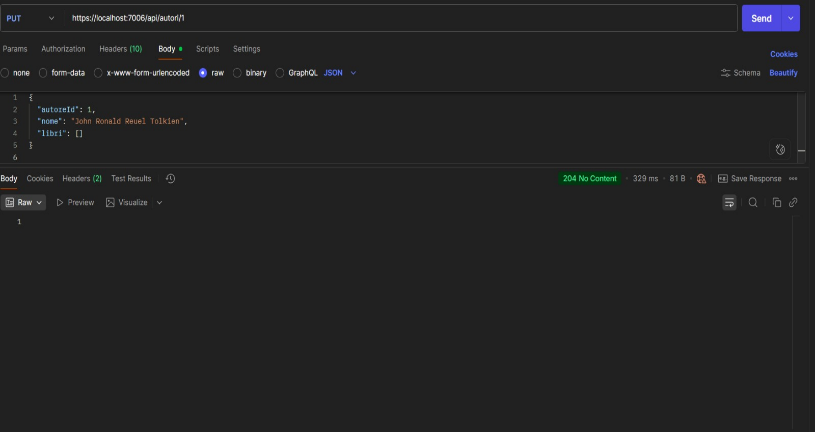
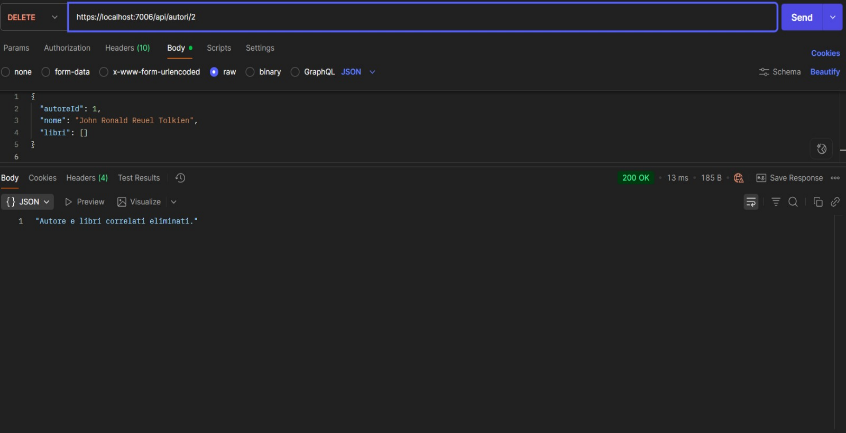
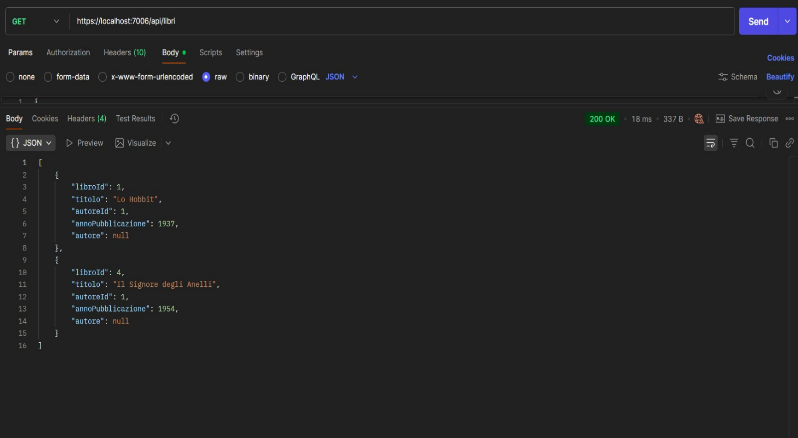
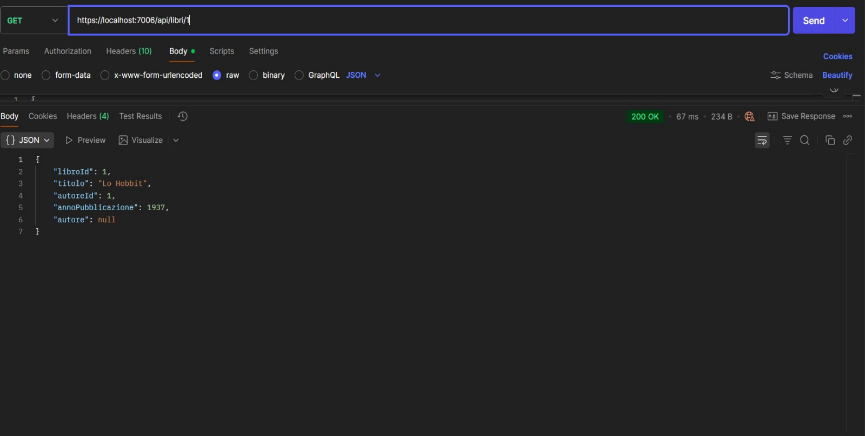
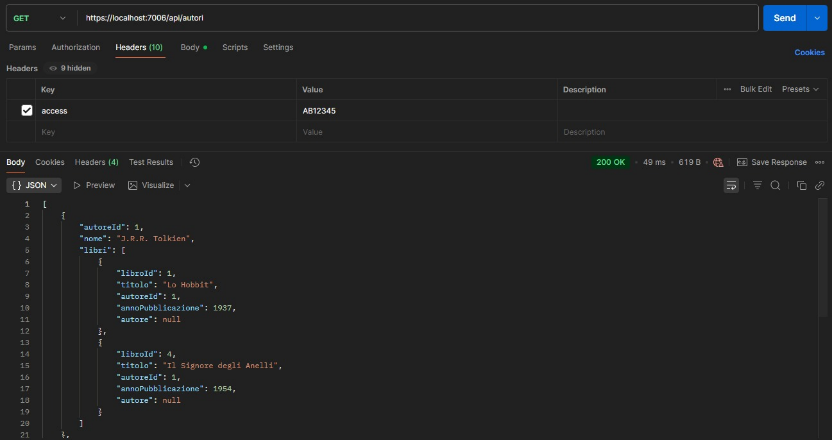
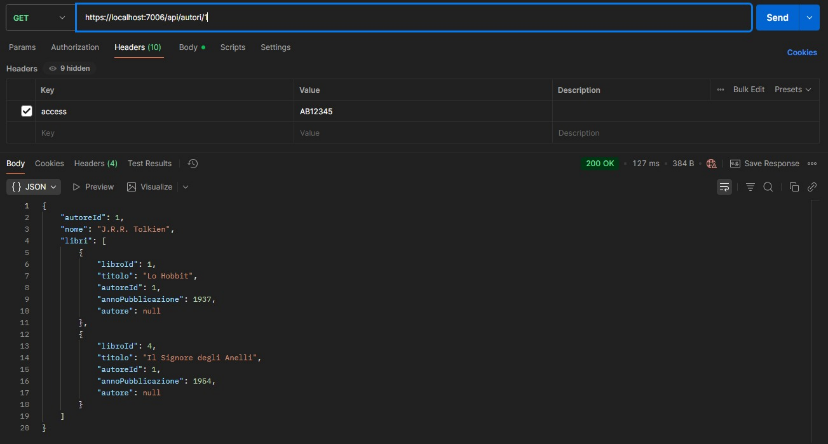
**Diagramma UML delle entità.**

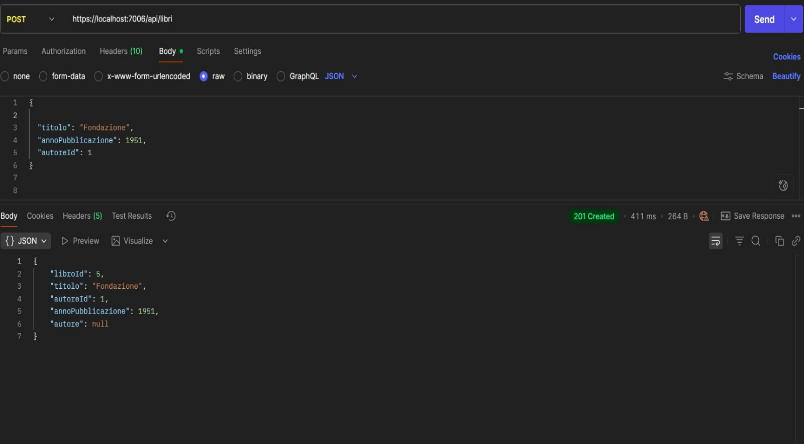
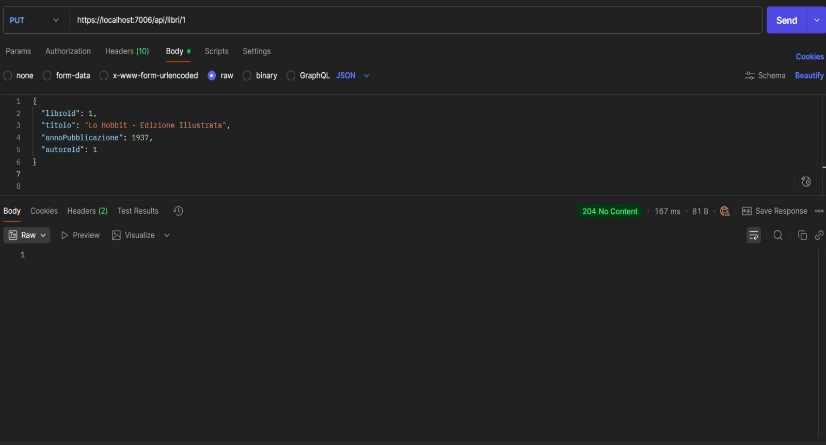
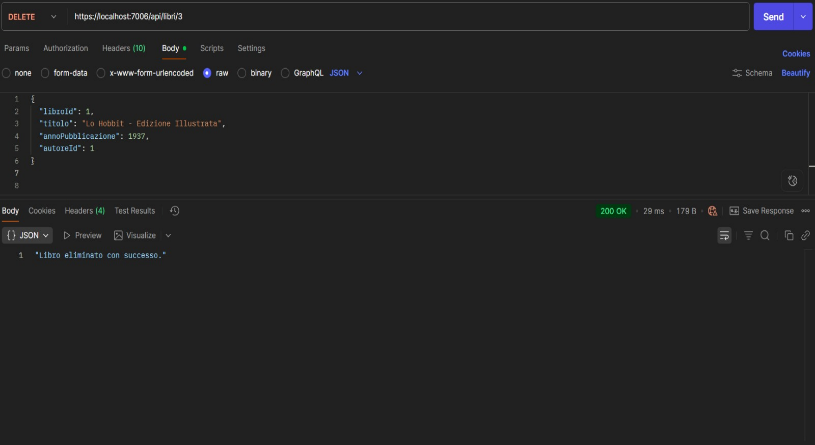
**(1:N), (N:M)**

**Immagine che contiene testo, schermata, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.**

**Screenshot di Postman:**

****



***Cristian Di Lallo***