Randomuser

🧭 Guida Didattica – Esercizio “RandomUser”

Obiettivo generale

Creare un’app console in **C#** che interroga l’API pubblica randomuser.me per ottenere una lista di utenti fittizi e stamparli in tabella.

L’obiettivo non è solo “farla funzionare”, ma **imparare a strutturare bene un progetto**, separando le responsabilità e scrivendo codice chiaro, leggibile e testabile.

Struttura del progetto consigliata

Crea la seguente struttura di cartelle:

```
randomuser/
├── RandomUser.sln
└── src/
    └── RandomUser.App/
        ├── Program.cs
        ├── Models/
        ├── Services/
        └── Utilities/
```

Puoi creare la solution e il progetto con i comandi:

```
dotnet new sln -n RandomUser
dotnet new console -n RandomUser.App -o src/RandomUser.App
dotnet sln add src/RandomUser.App
```

Milestone 1 – Impostazione del progetto e obiettivi

Obiettivo: preparare l’ambiente e impostare le prime cartelle.

1. Apri il progetto in Visual Studio Code o Rider.
2. Crea le cartelle:
   - `Models/` → per i record e le classi che rappresentano i dati.
   - `Services/` → per le chiamate HTTP e la logica di business.
   - `Utilities/` → per funzioni di supporto come parser e formatter.
3. In `Program.cs`, lascia solo un messaggio di test:

   ```
   Console.WriteLine("RandomUser App pronta!");
   ```

4. Esegui:

   ```
   dotnet run --project src/RandomUser.App
   ```

   Verifica che tutto funzioni.

Milestone 2 – Definizione dei modelli

Obiettivo: creare le classi che rappresentano i dati dell'applicazione.

Crea il file `Models/RandomUser.cs`:

```
namespace RandomUser.App.Models;

public record RandomUser(
    string FirstName,
    string LastName,
    string Email,
    string Gender,
    string Nat,
    string Phone)
{
    public string FullName => $"{FirstName} {LastName}";
}
```

Crea anche `Models/AppOptions.cs` per rappresentare i parametri passati da riga di comando:

```
namespace RandomUser.App.Models;

public record AppOptions(int Count, string? Gender, string? Nat, string? Se
ed);
```

E infine un modello per il payload JSON: `Models/RandomUserApiResponse.cs`

```
namespace RandomUser.App.Models;

public class RandomUserApiResponse
{
    public List<ResultItem> Results { get; set; } = new();
    public class ResultItem
    {
        public Name Name { get; set; } = new();
        public string Email { get; set; } = "";
        public string Gender { get; set; } = "";
        public string Nat { get; set; } = "";
        public string Phone { get; set; } = "";
    }
    public class Name
    {
        public string First { get; set; } = "";
        public string Last { get; set; } = "";
    }
}
```

il modello rappresenta *solo* la struttura del JSON ricevuto.

Il record `RandomUser` è invece il modello *di dominio*, quello che useremo nel nostro programma.

Milestone 3 – Parsing degli argomenti

Obiettivo: gestire i parametri passati da CLI (`--count`, `--gender`, `--nat`, `--seed`) in modo pulito e validato.

Crea `Utilities/ArgumentParser.cs`:

```
using RandomUser.App.Models;

namespace RandomUser.App.Utilities;

public static class ArgumentParser
{
    public static AppOptions Parse(string[] args)
    {
        int count = 5;
        string? gender = null, nat = null, seed = null;

        for (int i = 0; i < args.Length; i++)
        {
            switch (args[i])
            {
                case "--count":
                    count = int.Parse(args[++i]);
                    break;
                case "--gender":
                    gender = args[++i];
                    break;
                case "--nat":
                    nat = args[++i];
                    break;
                case "--seed":
                    seed = args[++i];
                    break;
            }
        }

        if (count < 1 || count > 500)
            throw new ArgumentException("Il parametro --count deve essere tr
a 1 e 500.");

        return new AppOptions(count, gender, nat, seed);
    }
}
```

TODO studente:

- Aggiungi controlli di validità (solo "male"/"female" per `gender`, nazionalità valide per `nat`).
- Mostra un messaggio d'errore chiaro se mancano parametri.

Milestone 4 – Chiamata HTTP e deserializzazione

Obiettivo: creare un servizio che interroga l'API `https://randomuser.me/api` e restituisce i dati già mappati nei nostri modelli.

Crea `Services/RandomUserService.cs`:

```
using System.Net.Http.Json;
using RandomUser.App.Models;

namespace RandomUser.App.Services;

public class RandomUserService
{
    private readonly HttpClient _http;

    public RandomUserService(HttpClient http) => _http = http;

    public async Task<IReadOnlyList<RandomUser>> FetchAsync(AppOptio
ns opts)
    {
        var url = $"https://randomuser.me/api/?results={opts.Count}";

        if (!string.IsNullOrEmpty(opts.Gender)) url += $"&gender={opts.Gende
r}";
        if (!string.IsNullOrEmpty(opts.Nat)) url += $"&nat={opts.Nat}";
        if (!string.IsNullOrEmpty(opts.Seed)) url += $"&seed={opts.Seed}";

        var response = await _http.GetFromJsonAsync<RandomUserApiRespo
nse>(url)
                       ?? new RandomUserApiResponse();

        return response.Results.Select(r =>
```

```
            new RandomUser(r.Name.First, r.Name.Last, r.Email, r.Gender, r.Nat,
r.Phone)
        ).ToList();
    }
}
```

TODO studente:

- Aggiungi `try/catch` per gestire errori di rete o JSON non valido.
- Imposta un timeout di 10 secondi.
- Stampa un messaggio d'errore se la chiamata fallisce.

Milestone 5 – Formattazione dell'output

Obiettivo: stampare in console una tabella leggibile con i risultati.

Crea `Utilities/ConsoleTableFormatter.cs`:

```
using RandomUser.App.Models;

namespace RandomUser.App.Utilities;

public static class ConsoleTableFormatter
{
    public static void Print(IReadOnlyList<RandomUser> users)
    {
        Console.WriteLine("----------------------------------------------------
----------");
        Console.WriteLine($"{"Full Name",-25} {"Email",-30} {"Nat",-5} {"Gen
der",-8}");
        Console.WriteLine("----------------------------------------------------
----------");

        foreach (var u in users)
            Console.WriteLine($"{u.FullName,-25} {u.Email,-30} {u.Nat,-5} {u.G
ender,-8}");

        Console.WriteLine("----------------------------------------------------
----------");
```

```
    }
}
```

TODO studente:

- Aggiungi la colonna `Phone`.
- Migliora l'allineamento dinamico calcolando la larghezza massima di ogni colonna.

Milestone 6 – Orchestrazione finale

Obiettivo: collegare tutti i pezzi in `Program.cs`.

```
using RandomUser.App.Models;
using RandomUser.App.Services;
using RandomUser.App.Utilities;

var options = ArgumentParser.Parse(args);
var service = new RandomUserService(new HttpClient());

try
{
    var users = await service.FetchAsync(options);
    ConsoleTableFormatter.Print(users);
}
catch (Exception ex)
{
    Console.Error.WriteLine($"Errore: {ex.Message}");
}
```

TODO studente:

- Gestisci separatamente `ArgumentException` (input errati) e `HttpRequestException` (errori di rete).
- Aggiungi un messaggio di benvenuto o help se non vengono passati parametri.

Esecuzione finale

Prova questi comandi:

```
dotnet run --project src/RandomUser.App -- --count 5
dotnet run --project src/RandomUser.App -- --count 10 --gender female --nat IT
dotnet run --project src/RandomUser.App -- --count 5 --seed demo
```

Output atteso:

```
Full Name            Email                    Nat   Gender
-------------------------------------------------------------
Maria Rossi          maria.rossi@example.com    IT    female
Lucia Gallo          lucia.gallo@example.com    IT    female
...
```

Estensioni facoltative

- [ ] Aggiungi **cache locale**: salva i risultati in JSON su file.
- [ ] Implementa **retry** se la rete fallisce temporaneamente.
- [ ] Aggiungi `-out users.csv` per esportare i dati.
- [ ] Scrivi **test unitari** per `ArgumentParser` e `RandomUserService` usando `xUnit`.

Conclusione

A questo punto l'app:

- Accetta input validati.
- Interroga un'API pubblica.
- Gestisce errori e timeout.
- Stampa un output leggibile e coerente.

Il tuo codice è **modulare, estendibile e pulito** — proprio come nei progetti veri.