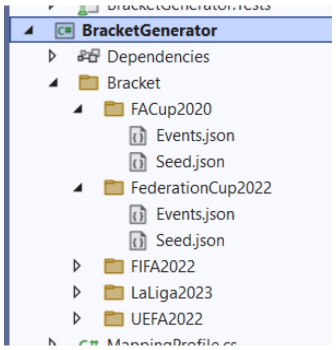
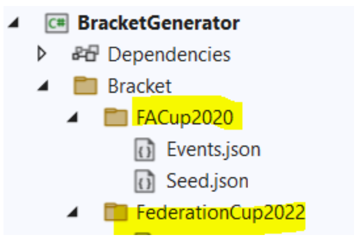
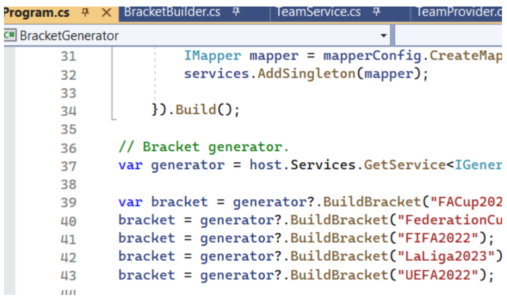
Implementation details.

- This implementation supports, for both group and group less tournaments.
- Also, for now it is only possible to work with the numbers of power of 2.
- Implementation does not support for biases.
- As an example, 64 teams support groups of 32, 16, 8, 4, 2.
- Implementation support any number of teams and groups under above mentioned conditions.
- I have implemented my own formatted seed and JSON files.



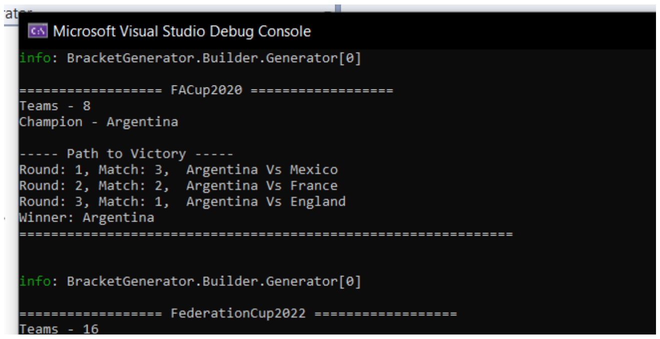
- When you add seed and event file properly, it is possible to generate everything by tournament name that means the folder name. Refer the "Program.cs" file to build the bracket.





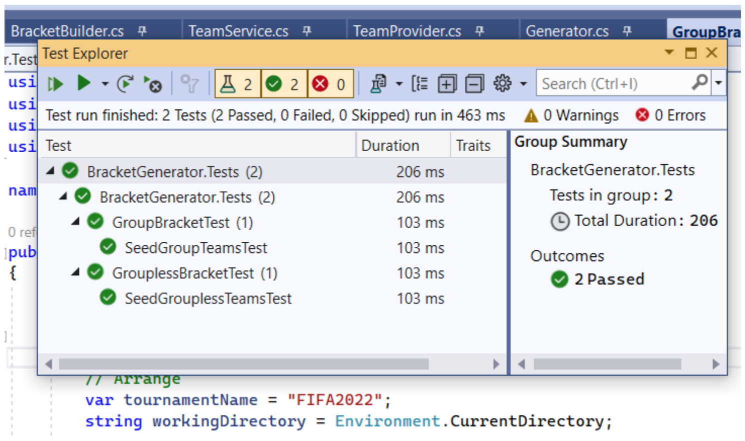
```
            IMapper mapper = mapperConfig.CreateMap
            services.AddSingleton(mapper);

        }).Build();

    // Bracket generator.
    var generator = host.Services.GetService<IGener

    var bracket = generator?.BuildBracket("FACup202
    bracket = generator?.BuildBracket("FederationCu
    bracket = generator?.BuildBracket("FIFA2022");
    bracket = generator?.BuildBracket("LaLiga2023")
    bracket = generator?.BuildBracket("UEFA2022");
```

- Implementation is fully executable, and output can be visible in the console.



```
info: BracketGenerator.Builder.Generator[0]

================= FACup2020 =================
Teams - 8
Champion - Argentina

----- Path to Victory -----
Round: 1, Match: 3,  Argentina Vs Mexico
Round: 2, Match: 2,  Argentina Vs France
Round: 3, Match: 1,  Argentina Vs England
Winner: Argentina
==============================================================


info: BracketGenerator.Builder.Generator[0]

================= FederationCup2022 =================
Teams - 16
```

- From the code it is possible to fetch all the data related to teams, how many rounds and each team have played, round of elimination, opponent details…etc.…
- Considering unit testing I could not be able to cover all the scenarios due to time limitation as I have dropped the weight on implantation. But of course, the written tests are successfully passed.



Test Explorer

Test run finished: 2 Tests (2 Passed, 0 Failed, 0 Skipped) run in 463 ms   0 Warnings   0 Errors

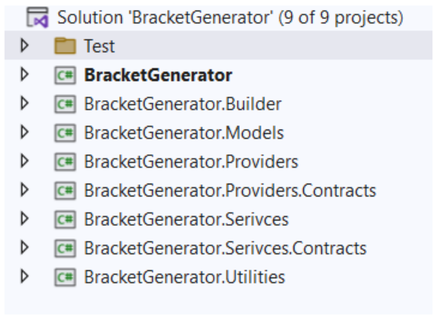
Test	Duration	Traits
BracketGenerator.Tests (2)	206 ms	
BracketGenerator.Tests (2)	206 ms	
GroupBracketTest (1)	103 ms	
SeedGroupTeamsTest	103 ms	
GrouplessBracketTest (1)	103 ms	
SeedGrouplessTeamsTest	103 ms	

Group Summary
BracketGenerator.Tests
Tests in group: 2
Total Duration: 206
Outcomes
2 Passed

```
// Arrange
var tournamentName = "FIFA2022";
string workingDirectory = Environment.CurrentDirectory;
```

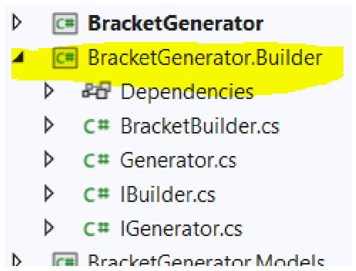
Technical Details

- Project was implemented with a layered architecture breaking down the subsystems which is compatible with “Façade” design pattern.



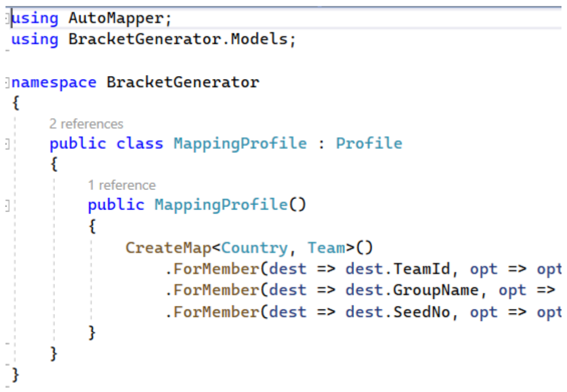
```
Solution 'BracketGenerator' (9 of 9 projects)
  Test
  BracketGenerator
  BracketGenerator.Builder
  BracketGenerator.Models
  BracketGenerator.Providers
  BracketGenerator.Providers.Contracts
  BracketGenerator.Serivces
  BracketGenerator.Serivces.Contracts
  BracketGenerator.Utilities
```

- Dependency injection was used all over the project.
- Building a bracket was implemented according to “Builder” design pattern.



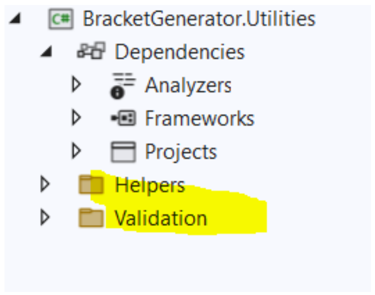
```
BracketGenerator
BracketGenerator.Builder
  Dependencies
  BracketBuilder.cs
  Generator.cs
  IBuilder.cs
  IGenerator.cs
BracketGenerator.Models
```

- SOLID principles were applied with in the implementation. Interfaces and abstract classes were used. It is possible to use virtual methods, partial classes if we need further enhancement which depends.
- “Auto mapper profile was added to the project easily map the entities.



```
using AutoMapper;
using BracketGenerator.Models;

namespace BracketGenerator
{
    2 references
    public class MappingProfile : Profile
    {
        1 reference
        public MappingProfile()
        {
            CreateMap<Country, Team>()
                .ForMember(dest => dest.TeamId, opt => opt
                .ForMember(dest => dest.GroupName, opt =>
                .ForMember(dest => dest.SeedNo, opt => opt
        }
    }
}
```

- Couple of helper methods were implemented to support the project that can be easily used when needed.
- Validations were added using a separate section that can be injected through DI and use anywhere.



```
◢ BracketGenerator.Utilities
    ◢ Dependencies
        ▷ Analyzers
        ▷ Frameworks
        ▷ Projects
    ▷ Helpers
    ▷ Validation
```

- It is essential to add the code comments and breaking regions, but due to time limitation I could not add them.
- Newton JSON library was used to deserialize the data from files.
- Considering unit testing “Fake it easy” library was used to mock the objects.
- Please rebuild and execute.