Snapshot testing

Un approccio innovativo per la verifica dei test

Ivano Matrisciano

corso di Software Project Management and Evolution

2024

Panoramica

1. Introduzione allo **snapshot testing**
2. Esempio: **Verify** per .NET
3. Conclusioni

Panoramica

1. Introduzione allo **snapshot testing**
2. Esempio: **Verify** per .NET
3. Conclusioni

Introduzione allo **snapshot testing**

In questa sezione:

- Cosa è lo snapshot testing
- Scenari d’uso
- Traction
- Che aspetto ha uno snapshot test
- Pipeline dello snapshot testing
- Gestione degli snapshot

1. Introduzione: cosa è lo snapshot testing

- Lo snapshot testing è una **tecnica di verifica** che permette di semplificare la stesura di casi di test
- Prevede di **memorizzare un output** della funzione sotto test come *known good*
- Si controlla che la stessa funzione, con gli stessi input, dia lo stesso risultato
- Così facendo non è necessario indicare manualmente l'expected result e le asserzioni

1. Introduzione: scenari d’uso

Ad oggi lo snapshot testing viene utilizzato principalmente per:

- **Test di interfacce (UI testing)**
  - Si può verificare che una schermata non sia variata rispetto ad uno snapshot buono, senza scrivere asserzioni complicate
- **Test di unità**
  - Si può verificare che il risultato ottenuto non sia variato rispetto a quello *known good*
- **Test di integrazione**
  - Si possono agevolmente verificare non solo i valori delle proprietà, ma anche la struttura di DTO e header

1. Introduzione: chi usa lo snapshot testing

30 000
snapshot tests

(per l'app su iOS)

> 1 000
snapshot tests

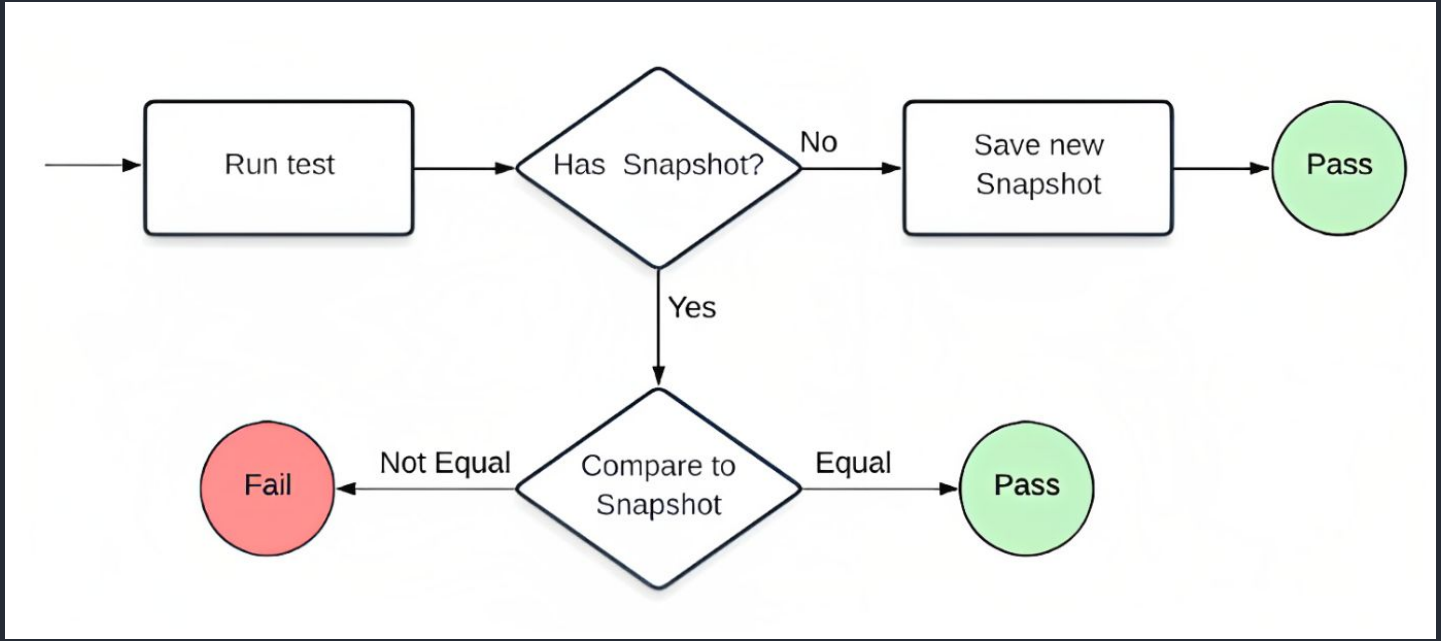
> 1 000
snapshot tests

Altre aziende che usano di snapshot testing includono: Uber, Lyft, Robinhood

1. Introduzione: esempio di snapshot test

```
[Fact]
| 0 references
public async Task CalcolaMetricheStudente_DovrebbeRestituireRisultatiCorretti_SnapshotTesting()
{
    // Arrange
    var studente = new Studente("Mario", "Rossi", "N97000123");

    var corso1 = new Corso("Corso 1", Cfu: 9);
    var corso2 = new Corso("Corso 2", Cfu: 6);

    List<RegistrazioneEsame> esami = [
        new RegistrazioneEsame(studente, corso1, 30, Lode: true),
        new RegistrazioneEsame(studente, corso2, 20, Lode: false),
    ];

    var repositoryEsami = Substitute.For<IRegistrazioneEsameRepository>();
    repositoryEsami.GetEsamiStudenteAsync(studente).Returns(esami);

    var metricheStudenteService = new MetricheStudenteService(repositoryEsami);

    // Act
    var result = await metricheStudenteService.CalcolaMetricheStudenteAsync(studente);

    // Verifica snapshot
    await Verify(result);
}
```

1. Introduzione: pipeline del sistema di verifica



1. Introduzione: gestione degli snapshot

- Per ognuno dei test scritti è necessario **salvare** su disco lo snapshot, cioè un file contenente l'expected result propriamente serializzato
  - è necessario **versionare** gli snapshot

- Ogni volta che uno snapshot differisce da quello salvato, il test fallirà
  - alla **prima esecuzione** non esiste uno snapshot *known-good* e il test fallirà

Panoramica

1. Introduzione allo **snapshot testing**
2. Esempio: **Verify** per .NET
3. Conclusioni

Esempio: **Verify** per .NET

In questa sezione:

- Progetto di esempio
- Setup del progetto di test
- Snapshot per test di unità
- Verifica di eccezioni
- Snapshot per test di integrazione
- Snapshot per test di interfacce

2. Esempio: il progetto

Obiettivo: calcolare le metriche di uno studente, intese come media aritmetica, media ponderate e numero di lodi per gli esami sostenuti

```
4 references
public record Studente(
    string Nome,
    string Cognome,
    string Matricola);
1 reference
public record Corso(
    string Nome,
    int Cfu);
4 references
public record RegistrazioneEsame(
    Studente Studente,
    Corso Corso,
    int Voto,
    bool Lode = false);
2 references
public record MetricheStudente(
    Studente Studente,
    double MediaAritmetica,
    double MediaPonderata,
    int NumeroDiLodi);
```

```
2 references
public class MetricheStudenteService(IRegistrazioneEsameRepository registrazioneEsameRepository)
{
    2 references | 2/2 passing
    public async Task<MetricheStudente> CalcolaMetricheStudenteAsync(Studente studente)
    {
        ArgumentNullException.ThrowIfNull(studente, nameof(studente));

        var esamiSostenuti = await registrazioneEsameRepository.GetEsamiStudenteAsync(studente);

        double mediaAritmetica = esamiSostenuti.Average(esame => esame.Voto);
        double mediaPonderata = CalcolaMediaPonderata(esamiSostenuti);
        int numeroDiLodi = esamiSostenuti.Count(esame => esame.Lode);

        return new MetricheStudente(studente, mediaAritmetica, mediaPonderata, numeroDiLodi);
    }

    1 reference
    private double CalcolaMediaPonderata(IEnumerable<RegistrazioneEsame> esamiSostenuti)
    {
        int totaleCfu = esamiSostenuti.Sum(esame => esame.Corso.Cfu);
        if (totaleCfu == 0)
            return 0;

        double sommaPesata = esamiSostenuti.Sum(esame => (double) esame.Corso.Cfu * esame.Voto);
        return sommaPesata / totaleCfu;
    }
}
```

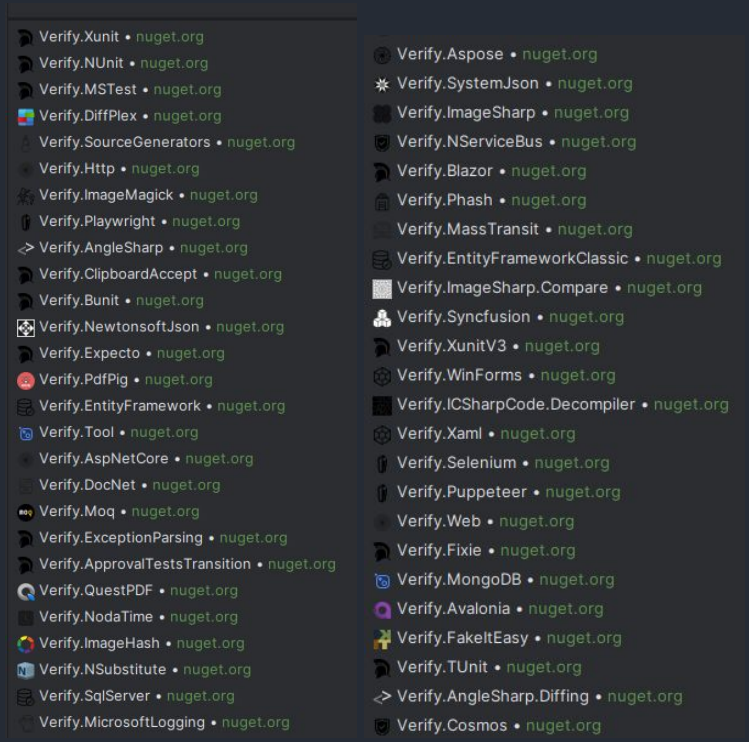
2. Esempio: il progetto di test

Prepariamo dunque un progetto di test per la nostra applicazione

Ingredienti:

- Un framework di test
  - xUnit
- Una libreria per mocking
  - NSubstitute
- Helper per le asserzioni
  - FluentAssertions
- Una libreria di Snapshot Testing
  - Verify (MIT license ╰(*°▽°*)╯)

Alcuni degli infiniti (*) pacchetti Verify →



(*): il numero di pacchetti non è infinito

2. Esempio: un test di unità classico

```
[Fact]
0 references
public async Task CalcolaMetricheStudente_DovrebbeRestituireRisultatiCorretti()
{
    // Arrange
    var studente = new Studente("Mario", "Rossi", "N97000123");

    var corso1 = new Corso("Corso 1", Cfu: 9);
    var corso2 = new Corso("Corso 2", Cfu: 6);

    List<RegistrazioneEsame> esami = [
        new RegistrazioneEsame(studente, corso1, 30, Lode: true),
        new RegistrazioneEsame(studente, corso2, 20, Lode: false),
    ];

    var repositoryEsami = Substitute.For<IRegistrazioneEsameRepository>();
    repositoryEsami.GetEsamiStudenteAsync(studente).Returns(esami);

    var metricheStudenteService = new MetricheStudenteService(repositoryEsami);

    var expected = new MetricheStudente(
        Studente: studente,
        MediaAritmetica: 25,
        MediaPonderata: 26,
        NumeroDiLodi: 1
        );

    // Act
    var actual = await metricheStudenteService.CalcolaMetricheStudenteAsync(studente);

    // Assert
    actual.Should().BeEquivalentTo(expected);
}
```

Preparazione dei dati

Mock della repository dei voti

Definizione dei valori attesi

Azione e asserzione

2. Esempio: un test di unità con snapshot

```
[Fact]
| 0 references
public async Task CalcolaMetricheStudente_DovrebbeRestituireRisultatiCorretti_SnapshotTesting()
{
    // Arrange
    var studente = new Studente("Mario", "Rossi", "N97000123");

    var corso1 = new Corso("Corso 1", Cfu: 9);
    var corso2 = new Corso("Corso 2", Cfu: 6);

    List<RegistrazioneEsame> esami = [
        new RegistrazioneEsame(studente, corso1, 30, Lode: true),
        new RegistrazioneEsame(studente, corso2, 20, Lode: false),
    ];

    var repositoryEsami = Substitute.For<IRegistrazioneEsameRepository>();
    repositoryEsami.GetEsamiStudenteAsync(studente).Returns(esami);

    var metricheStudenteService = new MetricheStudenteService(repositoryEsami);

    // Act
    var result = await metricheStudenteService.CalcolaMetricheStudenteAsync(studente);

    // Verifica snapshot
    await Verify(result);
}
```

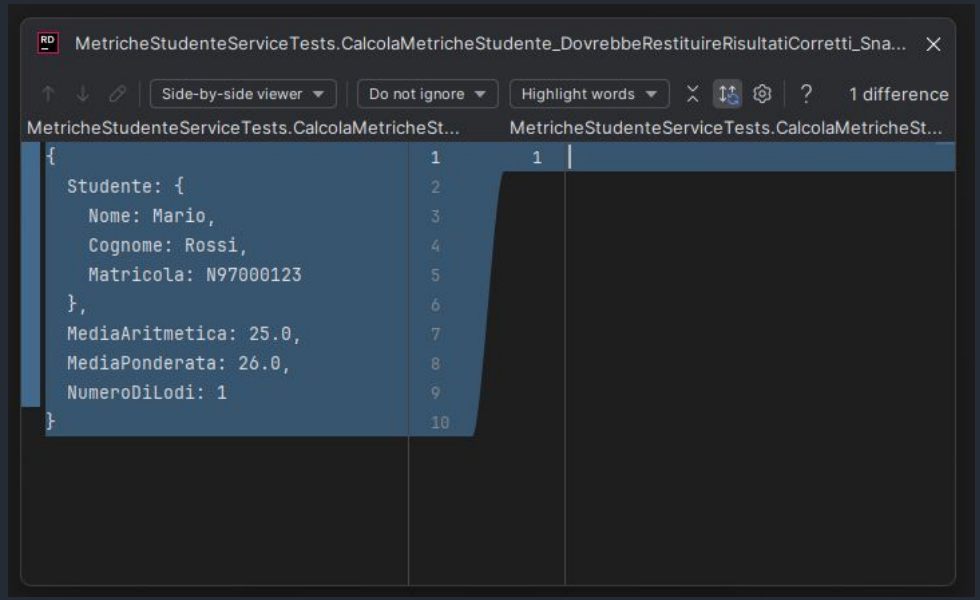
Preparazione dei dati

Mock della repository dei voti

Azione e verifica

2. Esempio: inizializzazione dello snapshot

- Alla prima esecuzione il test fallisce in quanto non esiste un *known-good*
- Si aprirà il nostro merge tool per permetterci di visionare le differenze e modificare il *known-good*
- Fissato lo snapshot, il test passerà finché il metodo produrrà gli stessi risultati



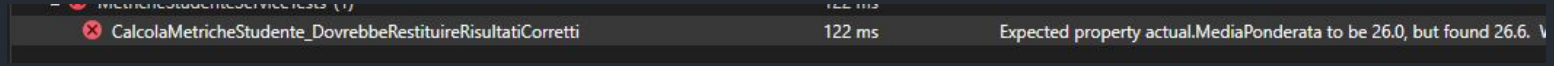
2. Esempio: modifica alla business logic

Ora le lodi valgono un voto in più nel calcolo della media ponderata

```
8 references
public record RegistrazioneEsame(
    Studente Studente,
    Corso Corso,
    int Voto,
    bool Lode = false)
{
    1 reference
    public int VotoConBonusLode
    {
        get => Lode ? Voto + 1 : Voto;
    }
}
```

```
2 references
public class MetricheStudenteService(IRegistrazioneEsameRepository registrazioneEsameRepository)
{
    2 references | 2/2 passing
    public async Task<MetricheStudente> CalcolaMetricheStudenteAsync(Studente studente)
    {
        ArgumentNullException.ThrowIfNull(studente, nameof(studente));

        var esamiSostenuti = await registrazioneEsameRepository.GetEsamiStudenteAsync(studente);

        double mediaAritmetica = esamiSostenuti.Average(esame => esame.Voto);
        double mediaPonderata = CalcolaMediaPonderata(esamiSostenuti);
        int numeroDiLodi = esamiSostenuti.Count(esame => esame.Lode);

        return new MetricheStudente(studente, mediaAritmetica, mediaPonderata, numeroDiLodi);
    }

    1 reference
    private double CalcolaMediaPonderata(IEnumerable<RegistrazioneEsame> esamiSostenuti)
    {
        int totaleCfu = esamiSostenuti.Sum(esame => esame.Corso.Cfu);
        if (totaleCfu == 0)
            return 0;

        double sommaPesata = esamiSostenuti.Sum(esame => (double) esame.Corso.Cfu * esame.VotoConBonusLode);
        return sommaPesata / totaleCfu;
    }
}
```

2. Esempio: modifica alla business logic (1/2)




```
[Fact]
0 references
public async Task CalcolaMetricheStudente_DovrebbeRestituireRisultatiCorretti()
{
    // Arrange
    var studente = new Studente("Mario", "Rossi", "N97000123");

    var corso1 = new Corso("Corso 1", Cfu: 9);
    var corso2 = new Corso("Corso 2", Cfu: 6);

    List<RegistrazioneEsame> esami = [
        new RegistrazioneEsame(studente, corso1, 30, Lode: true),
        new RegistrazioneEsame(studente, corso2, 20, Lode: false),
    ];

    var repositoryEsami = Substitute.For<IRegistrazioneEsameRepository>();
    repositoryEsami.GetEsamiStudenteAsync(studente).Returns(esami);

    var metricheStudenteService = new MetricheStudenteService(repositoryEsami);

    var expected = new MetricheStudente(
        Studente: studente,
        MediaAritmetica: 25,
        MediaPonderata: 26,
        NumeroDiLodi: 1
        );

    // Act
    var actual = await metricheStudenteService.CalcolaMetricheStudenteAsync(studente);

    // Assert
    actual.Should().BeEquivalentTo(expected);
}
```

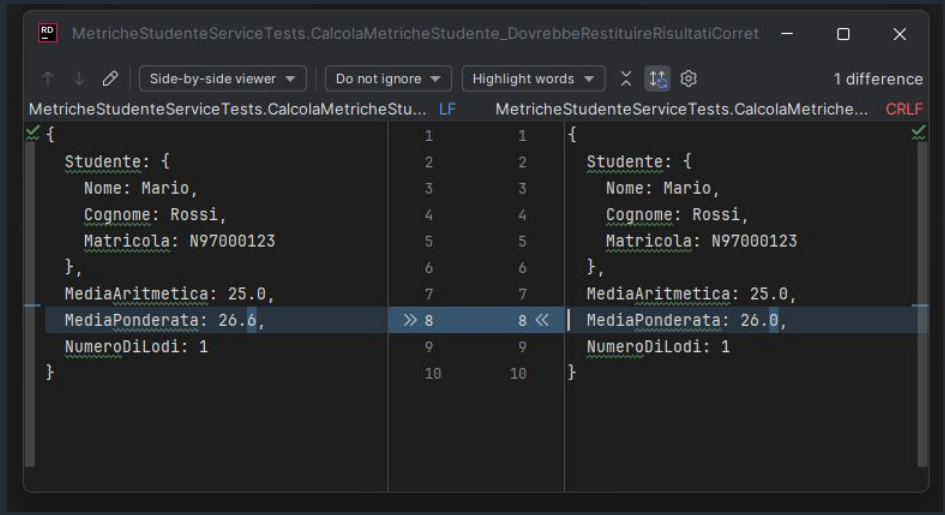
Il test di unità fallisce:

```
  Messaggio:
Expected property actual.MediaPonderata to
be 26.0, but found 26.6.
```

Sta a noi andare a modificare il test affinché corrisponda ai nuovi requisiti

2. Esempio: modifica alla business logic (2/2)

- Lo snapshot test fallisce poiché trova valori diversi rispetto a quelli memorizzati e segnati come buoni
- Possiamo dunque usare il merge tool per accettare o rigettare i nuovi valori
- Accettate le modifiche, la prossima esecuzione del test avrà successo

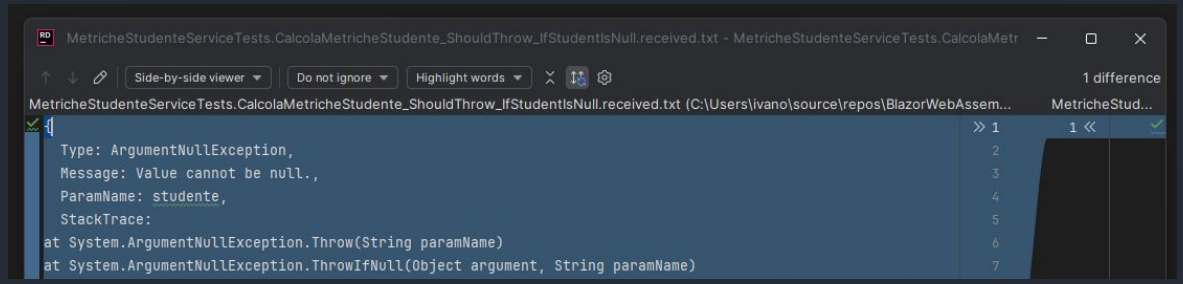


2. Esempio: verifica di eccezioni

```
[Fact]
0 references
public async Task CalcolaMetricheStudente_ShouldThrow_IfStudentIsNull()
{
    // Arrange
    var repositoryEsami = Substitute.For<IRegistrazioneEsameRepository>();

    var metricheStudenteService = new MetricheStudenteService(repositoryEsami);

    // Act
    Task<MetricheStudente> action() =>
        metricheStudenteService.CalcolaMetricheStudenteAsync(null!);

    // Verifica snapshot
    await ThrowsTask(action);
}
```

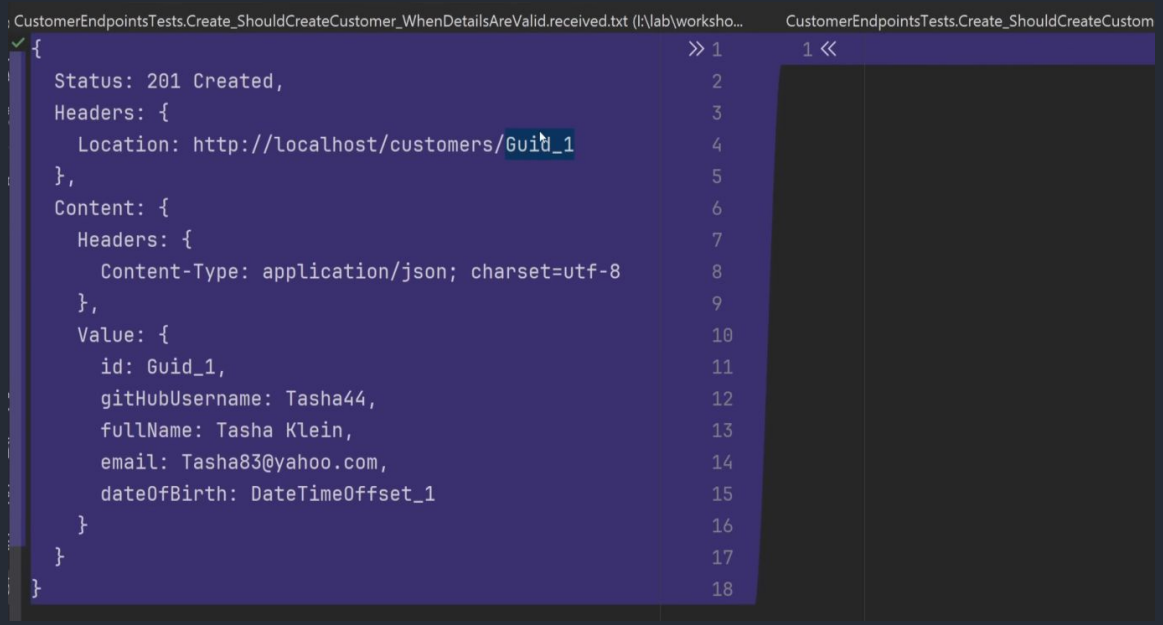
Nella verifica di eccezioni viene controllato non solo il tipo e il messaggio dell'eccezione, ma anche eventuali parametri aggiuntivi e lo stack trace della stessa.

Dallo stack trace vengono eliminati i numeri di linea per rendere il test più resiliente a modifiche del sorgente.



2. Esempio: integration testing

Esempio di verifica di risposta per un endpoint HTTP



Possiamo verificare in un colpo solo:

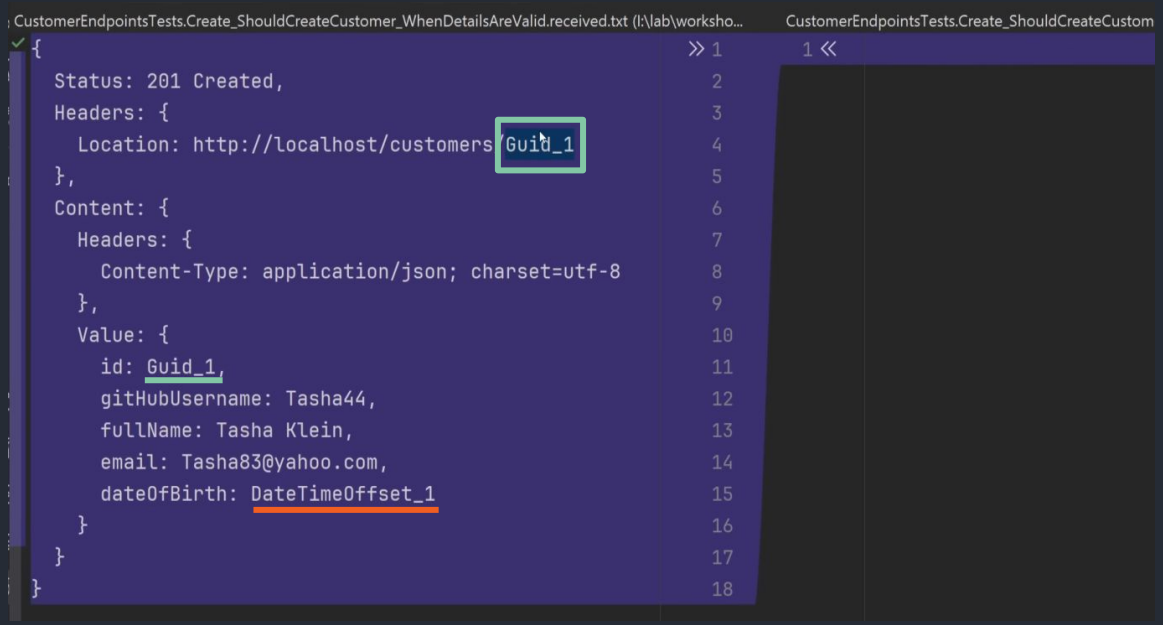
- status code della risposta
- headers di risposta
- valore del contenuto
- headers del contenuto
- struttura del documento

Da: The Only Type of Testing You Need

2. Esempio: su Guid, DateTime e simili

Che succede se dobbiamo testare metodi che producono **risultati non deterministici**?

- Esempio: Guid (generati casualmente), DateTime (cambiano in ogni momento)



La libreria Verify è in grado di individuare il **tipo dei parametri** in test e sostituisce con dei marker i valori di proprietà stocastiche.

È possibile attivare lo stripping inline di Guid

Da: The Only Type of Testing You Need

2. Esempio: UI testing (1/2)

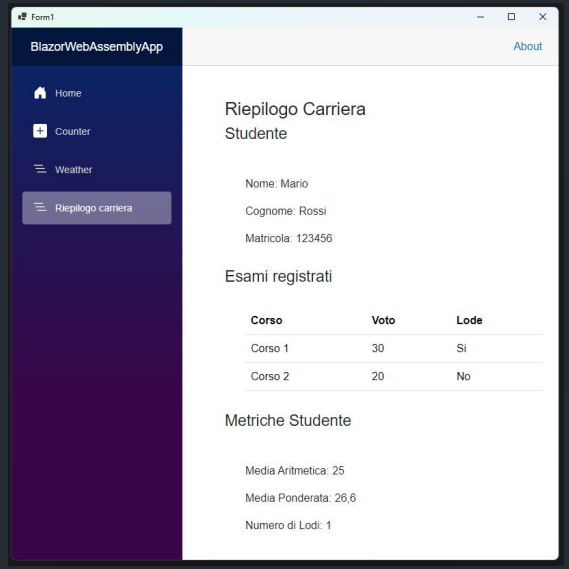
Markup della pagina da testare

```razor
@page "/riepilogo-carriera/{Matricola}"
@using BlazorWebAssemblyApp.Models
@using BlazorWebAssemblyApp.Services

<style>
    .content{
        margin: 30px;
    }
</style>

<h3>Riepilogo Carriera</h3>

@if (Studente is not null)
{
    <h4>Studente</h4>
    <div class="content">
        <p>Nome: @Studente.Nome</p>
        <p>Cognome: @Studente.Cognome</p>
        <p>Matricola: @Studente.Matricola</p>
    </div>

    @if (RegistrazioniEsami is not null)
    {
        <h4>Esami registrati</h4>
        <table class="table content">
            <thead>
                <tr>
                    <th>Corso</th>
                    <th>Voto</th>
                    <th>Lode</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var registrazioneEsame in RegistrazioniEsami)
                {
                    <tr>
                        <td>@registrazioneEsame.Corso.Nome</td>
                        <td>@registrazioneEsame.Voto</td>
                        <td>@(registrazioneEsame.Lode ? "Si" : "No")</td>
                    </tr>
                }
            </tbody>
        </table>
    }

    @if (MetricheStudente is not null)
    {
        <h4>Metriche Studente</h4>
        <div class="content">
            <p>Media Aritmetica: @MetricheStudente.MediaAritmetica</p>
            <p>Media Ponderata: @MetricheStudente.MediaPonderata</p>
            <p>Numero di Lodi: @MetricheStudente.NumeroDiLodi</p>
        </div>
    }
}
```

Codice della pagina da testare

```razor
@code {
    [Inject]
    private IMetricheStudenteService _metricheStudenteService { get; set; } = default!;
    [Inject]
    private IRegistrazioneEsameRepository registrazioneEsameRepository { get; set; } = default!;

    [Parameter]
    public string? Matricola { get; set; }

    Studente? Studente = null;
    MetricheStudente? MetricheStudente = null;
    IEnumerable<RegistrazioneEsame>? RegistrazioniEsami = null;

    protected override async Task OnInitializedAsync()
    {
        await base.OnInitializedAsync();

        //Facciamo finta di aver recuperato l'utente dal database
        Studente = new Studente("Mario", "Rossi", Matricola ?? "000000");

        RegistrazioniEsami = await registrazioneEsameRepository.GetEsamiStudenteAsync(Studente);
        MetricheStudente = await _metricheStudenteService.CalcolaMetricheStudenteAsync(Studente);
    }
}
```

Interfaccia risultante



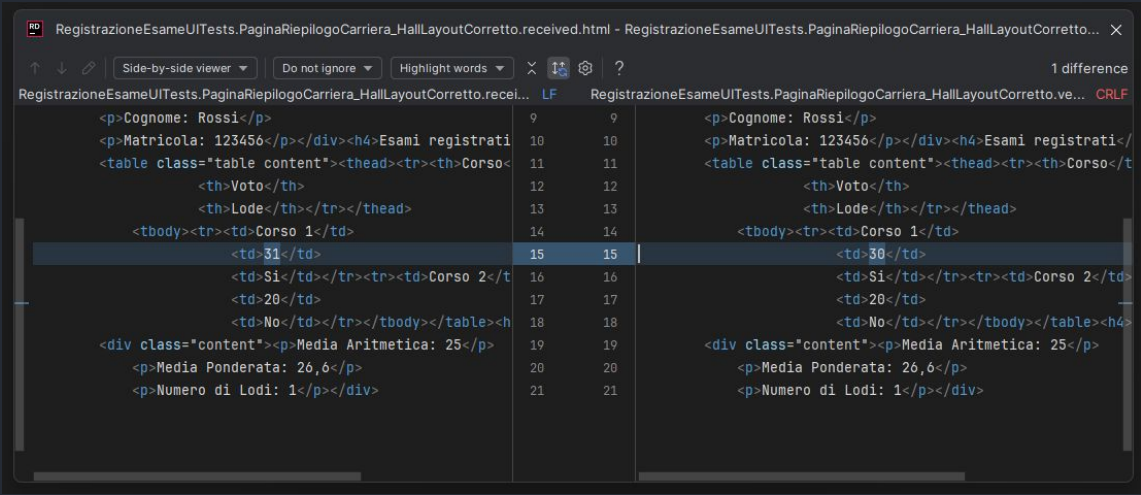
2. Esempio: UI testing (2/2)

TEST DI INTERFACCIA

```
0 references
public class RegistrazioneEsameUITests
{
    [Fact]
    | 0 references
    public async Task PaginaRiepilogoCarriera_HaIlLayoutCorretto()
    {
        //Arrange
        var template = new RiepilogoCarriera() // Il componente sotto test
        {
            Matricola = "123456", //I parametri in ingresso nel componente
        };

        //Setup del mock
        var studente = new Studente("Mario", "Rossi", "123456");
        var corso1 = new Corso("Corso 1", Cfu: 9);
        var corso2 = new Corso("Corso 2", Cfu: 6);

        List<RegistrazioneEsame> esami = [
            new RegistrazioneEsame(studente, corso1, 30, Lode: true),
            new RegistrazioneEsame(studente, corso2, 20, Lode: false),
        ];

        var repositoryEsami = Substitute.For<IRegistrazioneEsameRepository>();
        repositoryEsami.GetEsamiStudenteAsync(studente).Returns(esami);

        var metricheStudenteService = new MetricheStudenteService(repositoryEsami);

        //Setup del DI
        ServiceProvider serviceProvider = new ServiceCollection()
            .AddSingleton<IRegistrazioneEsameRepository>(repositoryEsami)
            .AddSingleton<IMetricheStudenteService>(metricheStudenteService)
            .BuildServiceProvider();

        //Render del componente
        var target = Render.Component(serviceProvider, template: template);

        //Verifica
        await Verify(target);
    }
}
```

BUG: per qualche motivo il servizio che ottiene gli esami sostenuti sta includendo il bonus della lode nel voto

Il test fallisce poiché il rendering del componente differisce dallo snapshot.



2. Esempio: il test dei test

È possibile aggiungere un caso di test che controlli che il progetto sia stato configurato correttamente.

Questo test fallirà, ad esempio, se il sistema di versionamento non è configurato per mantenere gli snapshot *known-good* e ignorare i risultati temporanei

```
0 references
public class VerifyConfigurationTests
{
    [Fact]
    | 0 references
    public async Task VerifyIsConfiguredCorrectly()
    {
        await VerifyChecks.Run();
    }
}
```

Panoramica

1. Introduzione allo **snapshot testing**
2. Esempio: **Verify** per .NET
3. Conclusioni

Conclusioni

In questa sezione:

- Vantaggi dello snapshot testing
- Svantaggi dello snapshot testing
- Best practices per snapshot testing

3. Conclusioni: vantaggi

Principali vantaggi dello snapshot testing:

1. Viene velocizzata la stesura dei test
    a. Si possono scrivere più test a parità di effort!
2. Si possono facilmente verificare oggetti complessi
    a. Come le GUI!
3. Oltre a verificare la correttezza dei dati si può verificare la struttura dell'oggetto prodotto
    a. Utile per test di integrazione!

3. Conclusioni: svantaggi

Principali svantaggi dello snapshot testing:

1. Fragilità: piccoli cambiamenti possono far fallire il test
    a. Problematico soprattutto per GUI
2. Mancanza di contesto: potrebbe essere difficile capire cosa esattamente un dato test sta verificando
3. Tendenza ad accettare il cambiamento invece di investigare accuratamente il motivo del fallimento

3. Conclusioni: best practices

Best practices da adottare se si sceglie di utilizzare snapshot testing:

1. Usare gli snapshot test durante le code review
2. Trattare gli snapshot *known-good* come fossero parte del sorgente
   a. Includere gli snapshot nel sistema di versionamento!
3. Scrivere snapshot con scope ridotto
   a. Riduce la quantità di test che potrebbero fallire a causa di un cambiamento, riducendo la tendenza ad accettare i cambiamenti
4. Scrivere descrizioni dettagliate per i test
   a. Necessarie per fornire il giusto contesto

Riferimenti

- Victor Pezzi Gazzinelli Cruz, Henrique Rocha and Marco Tulio Valente,
  Snapshot Testing in Practice: Benefits and Drawbacks
  Journal of Systems and Software Volume 204, October 2023, 111797
  https://doi.org/10.1016/j.jss.2023.111797
- Nick Chapsas, *The Only Type of Testing You Need*,
  https://www.youtube.com/watch?v=JG4zt9CnIl4
- Verify documentation,
  https://github.com/VerifyTests/Verify

Il codice sorgente utilizzato durante la demo è disponibile qui:
https://github.com/imatrisciano/SnapshotTestingPlayground

Grazie.