Testy

Celem lekcji jest omówienie w jaki sposób pisze się testy w programowaniu na przykładach testów jednostkowych i integracyjnych.

Przypomnienie

1. Co to jest LINQ?

- **LINQ** (Language Integrated Query) to narzędzie w języku C# umożliwiające wykonywanie zapytań na różnych źródłach danych (takich jak kolekcje, bazy danych) w sposób zintegrowany z językiem.

2. Do czego służy polecenie Distinct?

- **Distinct** pozwala nam uzyskać unikatowe wyniki np. z bazy danych. Wyświetla wyniki bez duplikatów.

3. Do czego służy polecenie FirstOrDefault?

- **FirstOrDefault** bierze pierwsze wystąpienie danego elementu bądź używa domyślnej wartości, jeżeli nie może znaleźć żadnego elementu. Dla typów referencyjnych, domyślną wartością jest **null**. Oznacza to, że jeśli kolekcja jest pusta lub żaden element nie spełnia warunku, FirstOrDefault zwróci null.

Co to są testy?

Testy w programowaniu to procesy i techniki służące do weryfikacji, czy oprogramowanie działa zgodnie z oczekiwaniami. Testy mają na celu wykrycie błędów, upewnienie się, że funkcje działają poprawnie, oraz że wprowadzone zmiany nie powodują nowych problemów.

Po się pisze testy ?

Pisanie testów w programowaniu jest kluczowym elementem zapewnienia jakości kodu i ma wiele istotnych korzyści:

- **Wykrywanie błędów:** Testy pomagają identyfikować błędy i problemy w kodzie na wczesnym etapie, co ułatwia ich naprawę. Im wcześniej błąd zostanie wykryty, tym mniej kosztowne jest jego naprawienie.
- **Zapewnienie jakości**: Dzięki testom programiści mogą upewnić się, że kod działa zgodnie z oczekiwaniami i spełnia wymagania. Testy sprawdzają, czy wszystkie funkcje działają poprawnie w różnych scenariuszach.
- **Bezpieczeństwo przy zmianach**: Kiedy kod jest modyfikowany lub rozbudowywany, testy automatyczne pomagają upewnić się, że wprowadzone zmiany nie spowodują nowych błędów ani nie zepsują istniejącej funkcjonalności.
- **Dokumentacja**: Testy mogą pełnić funkcję dokumentacji kodu, pokazując, jak powinny działać poszczególne funkcje i jak się ich używa. Mogą one również wyjaśniać zamierzone zachowanie w specyficznych przypadkach.

Rodzaje testów

Dwoma rodzajami testów, na których się skupiamy są:

- **Testy jednostkowe** (unit tests):
  - Testują najmniejsze części aplikacji, takie jak pojedyncze funkcje czy metody.
  - Izolują jednostki kodu, aby upewnić się, że działają one poprawnie niezależnie od innych części systemu.
- **Testy integracyjne** (integration tests):
  - Sprawdzają, czy różne moduły lub usługi współpracują ze sobą poprawnie.
  - Testują interakcje między komponentami aplikacji.

Warto wiedzieć, że w żargonie programistycznym na testy jednostkowe mówi się po prostu **unit testy**.

Przygotowanie testów

Przygotowanie testów składa się głównie z 4 etapów:

1. Zestawienie:

- Przygotowanie środowiska testowego, w tym inicjalizacja obiektów i ustawienie wstępnych warunków potrzebnych do przeprowadzenia testu.

2. Wykonanie:

- Wywołanie testowanej funkcji lub metody z określonymi danymi wejściowymi.

3. Weryfikacja:

- Sprawdzenie, czy wynik działania funkcji jest zgodny z oczekiwaniami.
  Używa się do tego asercji, które porównują rzeczywisty wynik z oczekiwanym.

4. Zakończenie:

- Czyszczenie po teście, usuwanie wszelkich zmian w środowisku, które mogłyby wpłynąć na inne testy.

xUnit

xUnit to framework do testowania jednostkowego (unit testing) dla języka C#.

Główne cechy i zalety xUnit:

- **Lekkość i prostota**: xUnit został zaprojektowany z myślą o prostocie i minimalizmie, co ułatwia jego użycie oraz zrozumienie.
- **Atrybuty**: Używa atrybutów (annotations), aby oznaczać metody testowe i inne elementy, takie jak [Fact], które są bardziej intuicyjne i elastyczne.

Atrybut **[Fact]** w xUnit jest używany do oznaczania metody testowej, która reprezentuje pojedynczy przypadek testowy. Jest to najprostszy sposób na zdefiniowanie testu w xUnit.

```
[Fact]
0 references
public void Dodaj_DwieLiczby_ZwracaPoprawnyWynik()
{
    // Przygotowanie
    var matematyka = new Matematyka();
    int liczba1 = 3;
    int liczba2 = 5;

    // Wykonanie
    int wynik = matematyka.Dodaj(liczba1, liczba2);

    // Sprawdzenie
    Assert.Equal(8, wynik);
}
```