Задание: Unit тестване с NUnit 3, Moq и (по избор) CI

Цел

Да изградите пълен работен процес за модулно тестване върху избрана от вас C# библиотека (една от предоставените 6 теми), използвайки **NUnit 3**, параметризирани тестове, жизнен цикъл **SetUp/TearDown**, базови проверки за изключения, **mock**-ове с **Moq** и кратка интеграция към CI (по избор). Заданието стъпва върху помощните материали от приложението.

Част 1: Подготовка на тестов проект с NUnit 3

1. Създайте **нов .NET Class Library** проект за тестове, напр. ChosenTheme.Tests.
2. През **NuGet** инсталирайте:
    - NUnit
    - NUnit3TestAdapter
    - Microsoft.NET.Test.Sdk
3. Добавете **Project Reference** към избраната библиотека (една от 6-те).
4. **Build** на решението и отворете **Test Explorer**; уверете се, че тестовете се откриват по атрибута [Test].

📌 Подсказка: Прегледайте примерните стъпки и пояснения за NUnit, Test Explorer и атрибутите за тестове в приложението.

Част 2: Проучване на модела и базови тестове

1. Разгледайте публичните API на избраната библиотека (класове, интерфейси, свойства, методи).
2. Създайте **минимален набор Smoke тестове** за конструктори и начални стойности (напр. по подобие на примерите за “герой” и инициализационни проверки). Използвайте Assert/Is за твърдения.

Част 3: Жизнен цикъл на тестовете (SetUp/TearDown)

1. Премахнете дублиран код, като въведете:

- [OneTimeSetUp] / [OneTimeTearDown] – логика преди/след всички тестове в класа.
- [SetUp] / [TearDown] – логика преди/след **всеки** тест (напр. реинициализация на обект).

2. Добавете **тест за изключение** с Assert.Throws<…>() за невалидни входове (напр. отрицателни стойности), по аналогия с примерите.

Част 4: Параметризирани тестове

Покрийте поведение с **няколко входни стойности** чрез:

- [TestCase(...)] – различни аргументи към един и същ метод.
- [Values(...)] – деклариране на множество стойности за параметър.
- [Range(start, end, step)] – автоматично генериране на поредица стойности.

Използвайте ги за „нормални“ и гранични случаи. При нужда добавете и проверки с Does/Has от NUnit.

Част 5: Тестове по специфика

Освен базова функционалност, добавете **специфични** проверки за конкретните подтипове във вашата тема (напр. ефект от „повишаване на ниво“, специални полета/ограничения, инварианти). Насоките следват подхода от секцията „Тестване в специфика“.

Част 6: Mocking с Moq

1. Инсталирайте Moq през NuGet.
2. Създайте поне **един mock-тест**, който изолира тествания клас от зависимост (интерфейс/база).
3. Демонстрирайте:
   - new Mock<T>(), използване на mock.Object;
   - Setup на метод/свойство;
   - (по желание) Protected() и CallBase() за частичен mock на защитени методи, по аналогия с примерите.

Част 7 (по избор, бонус): Непрекъсната интеграция (CI)

Добавете файл за CI конфигурация (напр. *Travis CI*), който:

- възстановява пакети, билдва решението и стартира dotnet test за вашия тестов проект.
  Може да използвате примерната структура на .travis.yml от приложението и да я адаптирате за вашата версия на .NET.

Какво да предадете

1. Репозитори/ZIP със:
   - ChosenTheme.Tests (NUnit 3) + референция към избраната библиотека.
   - Тестови класове, организирани по функционални зони (напр. „база“, „подтипове/специфика“).
   - Поне **1 mock тест** с Moq.
   - (Бонус) CI конфигурация.
2. Кратък **README** с:
   - как да се пуснат тестовете (CLI и/или VS Test Explorer);
   - кои сценарии/гранични случаи са покрити;
   - известни ограничения/TODO.

Минимални изисквания (оценяване)

- ✅ Проект с **NUnit 3** и откриваеми тестове в **Test Explorer**.
- ✅ Използвани **[SetUp]/[TearDown]** и поне една двойка **[OneTimeSetUp]/[OneTimeTearDown]**.
- ✅ Най-малко **3 параметризирани теста** (микс от [TestCase], [Values], [Range]).
- ✅ Тестове за **изключения** при невалидни входове.
- ✅ **1+ Moq тест** за изолиране на зависимост.

- ⭐ (бонус) Работещ **CI** pipeline, който стартира dotnet test.

Полезни насоки (от приложението)

- Структурирайте тестовете по класове/зони и следвайте трите фази: **Arrange–Act–Assert**; ползвайте Assert/Is.
- Наблегнете на **граничните случаи** и по-голямо покритие; добрият тест изпълнява съществена част от кода.
- При провал на тест – четете съобщенията от Test Explorer и добавяйте собствени пояснения към Assert.