



# C# / .NET

(chatbot: разработка слоя доступа к данным,  
модульное тестирование)

Артём Трофимушкин

# Взаимосвязи между компонентами

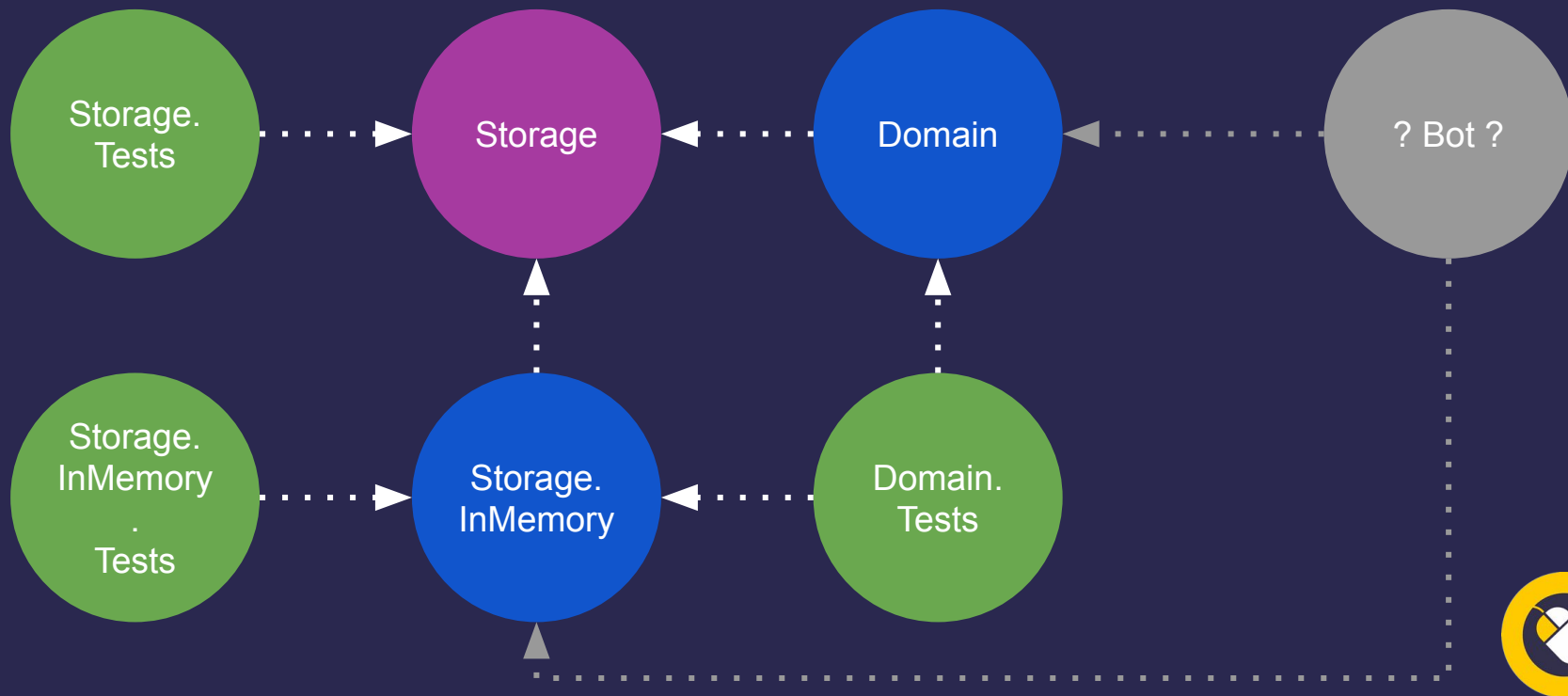
---

Основные сборки, отвечающей за логику работы собственно ремайндера:

- **Reminder.Storage**
  - Библиотека с описанием интерфейсов и классов, которые будут использоваться конкретными реализациями хранилища данных
  - Ей в пару будет создана сборка `Reminder.Storage.Tests` для тестов логики классов, если она будет.
- **Reminder.Storage.InMemory**
  - Библиотека с реализацией хранилища данных в памяти. Реализует все интерфейсы `Reminder.Storage.Core`.
  - Ей в пару будет создана сборка `Reminder.Storage.InMemory.Tests` для тестов логики хранилища.
- **Reminder.Domain**
  - Основная библиотека логики.



# Взаимосвязи между компонентами



# Библиотека `Reminder.Storage`

---

- `Reminder.Storage.ReminderItem` класс
  - Определяет сущность единственной записи будильника которая будет лежать в хранилище данных.
  - Должен содержать, как минимум, необходимые по заданию поля с датой и текстом.
  - Должен также содержать уникальный идентификатор для быстрого поиска внутри хранилища.
- `Reminder.Storage.ReminderItemStatus` перечисление
  - Определяет возможные статусы для единственной записи будильника которая будет лежать в хранилище данных.
  - Как минимум отправлено/не отправлено.
- `Reminder.Storage.IReminderStorage` интерфейс
  - Описывает интерфейс хранилища данных.
  - Должен, как минимум, содержать методы по записи данных в хранилище и получению по данным из хранилища.



# Библиотека `Reminder.Storage.InMemory`

---

- `Storage.InMemory.IReminderStorage` интерфейс
  - Описывает интерфейс хранилища данных.
  - Должен, как минимум, содержать методы по записи данных в хранилище и получению по данным из хранилища.



# Модульное тестирование (Unit tests)

Модульные тесты позволяют разработчикам и тест-инженерам быстро искать логические ошибки в реализации классов.

Для модульных тестов **создаются отдельные проекты модульных тестов**, как правило, по одному проекту на один тестируемый проект в союшене.

Существует несколько форматов написания модульных тестов для платформы .NET Core:

- MSTest Test Project (.NET Core)
- **NUnit Test Project (.NET Core)**
- xUnit Test Project (.NET Core)

Они отличаются синтаксическими элементами оформления тестов и (немного) встроенными возможностями (см. [таблицу сравнения](#)).



# Базовые атрибуты тестовых проектов

Обычно в одном классе теста покрывается одна единица тестирования. Это может быть класс, или даже метод.

Типичный файл проекта модульных тестов - это обычный класс, размеченный специальными атрибутами:

```
public class UnitTest1
{
    [Test]           ← ----- Атрибут тестового метода
    public void TestMethod1()
    {
        ClassToBeTested obj = new ClassToBeTested();
        Assert.IsNull(obj); ← ----- Тестовое утверждение
    }
}
```

Это и есть элементарный тест



# Расширенные атрибуты тестовых проектов

Атрибут `[TestCase]` представляет набор тестов, которые выполняют один и тот же код, но имеют разные входные аргументы. С помощью конструктора атрибута `[TestCase]` можно указать значения для этих входных аргументов.

```
[TestCase(0)]  
[TestCase(1)]  
[TestCase(int.MaxValue)]  
public void ReturnFalseGivenValuesLessThan2(int value)  
{  
    ClassToBeTested cls = new ClassToBeTested();  
    bool result = cls.IsPositive(value);  
  
    Assert.IsTrue(result);  
}
```





# Расширенные атрибуты тестовых проектов

---

Также для сложных сценариев можно использовать методы настройки среды тестирования, которые будут подготавливать окружение для запуска.

- `[SetUp]` – Метод помеченный этим атрибутом будет запускаться перед каждым тестом в классе
- `[TearDown]` – Метод помеченный этим атрибутом будет запускаться после каждого теста в классе
- `[OneTimeSetUp]` – Метод помеченный этим атрибутом будет запускаться перед запуском первого теста в классе (и перед методом помеченным атрибутом `[SetUp]`, если такой есть)
- `[OneTimeTearDown]` – Метод помеченный этим атрибутом будет запускаться после запуска последнего теста в классе (и после метода помеченного атрибутом `[TearDown]`, если такой есть)



# InternalsVisibleTo для доступа к internal-членам

---

Чтобы обеспечить доступ к своим internal-членам (например для тестирования их в проектах unit-тестов) необходимо добавить в начало любого файла \*.cs такие строки:

```
using System.Runtime.CompilerServices;  
  
[assembly: InternalsVisibleTo("Reminder.Storage.InMemory.Tests")]
```

Для этого можно создавать отдельный файл AssemblyInfo.cs в котором и будут находиться только эти строки.



## Полезные ссылки

---

- Документация Microsoft: Модульное тестирование кода C# с использованием MSTest и .NET Core:  
<https://docs.microsoft.com/ru-ru/dotnet/core/testing/unit-testing-with-mstest>
- Документация Microsoft: Модульное тестирование C# в .NET Core с использованием dotnet test и xUnit:  
<https://docs.microsoft.com/ru-ru/dotnet/core/testing/unit-testing-with-dotnet-test>
- Документация Microsoft: Модульное тестирование кода C# с использованием NUnit и .NET Core:  
<https://docs.microsoft.com/ru-ru/dotnet/core/testing/unit-testing-with-nunit>
- Habr: Unit тесты на практике: <https://habr.com/ru/post/191986>



# Самостоятельная работа

---

Добавить к проектам `Reminder.Storage.Core` и `Reminder.Storage.InMemory` по одному проекту `MSTest Test Project (.NET Core)` с названиями

- `Reminder.Storage.Core.Tests`
- `Reminder.Storage.InMemory.Tests`

Покройте тестами

- Свойство `TimeToAlarm` класса `ReminderItem` библиотеки `Reminder.Storage.Core`,
- Метод `Add(ReminderItem item)` класса `ReminderStorage` библиотеки `Reminder.Storage.InMemory`.



# Домашняя работа

---

Завершить реализацию интерфейса `IReminderStorage` в библиотеке `Reminder.Storage.InMemory` и покрыть тестами код реализации в библиотеке `Reminder.Storage.InMemory.Tests`.



# Спасибо за внимание.

