Задание 1.

В данной лабораторной работе мы рассматриваем инструменты для разработки нашего проекта. Так как мы выбрали для реализации проекта C# +ASP Net.Core + MSSQL, то мы составим два анализа по имеющимся инструментам

Анализ 1. Инструменты для взаимодействия кода с базой данных (ORM – object relation mapping)

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | **Entity Framework Core (EF Core)** | **Dapper** | **SQLKata** |
| **Функциональность** | **Полноценный ORM:** отслеживание изменений, миграции, "ленивая" загрузка, LINQ. | **Микро-ORM:** только маппинг результатов запроса на объекты. Ничего лишнего. | **Построитель запросов:** Генерирует SQL из кода. Не выполняет маппинг. |
| **Производительность** | Выше накладные расходы. Требует оптимизации (AsNoTracking, избегание N+1). | **Максимально близко к сырому SQL.** Самый быстрый из-за простоты. | Выше, чем у EF Core, но ниже, чем у Dapper, так как парсит выражения. |
| **Удобство использования** | Очень высокое для CRUD. LINQ позволяет писать запросы на C#. | Прост в изучении. Требует отличного знания SQL. Много рутинного кода. | Удобен для построения сложных динамических запросов. Синтаксис похож на LINQ. |
| **Безопасность (SQL-инъекции)** | Запросы через LINQ безопасны по умолчанию. | Риск при неправильном использовании строк. Обязательно использовать параметры. | Безопасен, так как использует параметризованные запросы. |
| **Совместимость** | Идеально интегрирован в ADO.NET Core. | Работает с любым ADO.NET провайдером (включая MSSQL). | Не зависит от ORM, может работать с Dapper или стандартным ADO.NET |
| **Документация и сообщество** | Официальная документация от Microsoft, огромное сообщество. | Отличная документация, очень популярен в высоконагруженных проектах. | Документация хорошая, но сообщество меньше. |
| **Кроссплатформенность** | Да (как и весь .NET Core). | Да. | Да. |
| **Стоимость** | Бесплатно (Open Source). | Бесплатно (Open Source). | Бесплатно (Open Source). |

Задачей данного анализа является выбор технологии для взаимодействия с БД MSSQL.

Вывод по Анализу 1:

EF Core – лучший выбор для большинства бизнес-приложений, где важна скорость разработки, а сложные запросы встречаются нечасто. Его интеграция с ASP.NET Core и механизм миграций невероятно удобны.

Dapper – следует выбрать, если в проекте есть критические по производительности участки, сложные отчетные запросы или если команда сильно ориентирована на SQL.

SQLKata – отличный компромисс, когда нужна производительность близкая к Dapper, но при этом требуется безопасно строить сложные динамические запросы в коде.

Рекомендация для реализуемого проекта: Использовать EF Core, так как он является стандартом и идеально подходит для MVC, в виду простоты реализации и количества нужных библиотек.

Анализ 2. Инструменты для модульного и интеграционного тестирования.

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | **xUnit** | **NUnit** | **MSTest** |
| **Философия и подход** | Чистота и изоляция. Каждый тест выполняется в собственном экземпляре класса. Использует Fact и Theory. | Гибкость и богатый набор функций. Поддерживает параметризованные тесты через [TestCase]. Классический подход. | Тесная интеграция с Visual Studio. Прост для начала. Использует атрибуты [TestMethod]. |
| **Производительность** | Высокая. По умолчанию создает новый экземпляр тестового класса для каждого теста, что обеспечивает лучшую изоляцию. | Высокая. Может быть настроен для создания экземпляра на тест или на класс. | Ниже, чем у конкурентов. Исторически был медленнее, но в последних версиях разрыв сократился. |
| **Удобство использования** | Чистый и лаконичный синтаксис. Assert.Equal(expected, actual) – порядок аргументов интуитивно понятен. | Очень удобен, особенно для разработчиков, знакомых с JUnit. Множество атрибутов для тонкой настройки. | Простейший порог входа, особенно для новичков в .NET и Visual Studio. |
| **Параметризация тестов** | [Theory] + [InlineData], [MemberData], [ClassData]. Гибко, но данные задаются отдельно. | [TestCase] — позволяет задать параметры и ожидаемый результат прямо в атрибуте. Очень наглядно. | [DataTestMethod] + [DataRow]. Аналогично xUnit, функциональность похожа. |
| **Интеграция с инструментами** | Отлично интегрируется с Visual Studio, Azure DevOps, CLI dotnet test. Фактический стандарт для open-source проектов .NET. | Также отлично интегрируется со всеми основными инструментами. Очень зрелая экосистема. | Идеальная интеграция с Visual Studio (Test Explorer). |
| **Документация и сообщество** | Очень активное и растущее сообщество. Документация хорошая. Используется самой командой .NET. | Огромное, зрелое сообщество. Множество примеров и ресурсов. | Официальная документация от Microsoft. Сообщество большое, но больше сосредоточено на enterprise-среде. |
| **Гибкость и расширяемость** | Очень гибкий. Легко создавать кастомные атрибуты и системы вывода. | Также очень гибок и расширяем. Множество плагинов и расширений. | Менее гибкий по сравнению с xUnit и NUnit. |
| **Стоимость** | Бесплатно (Open Source). | Бесплатно (Open Source). | Бесплатно (поставляется с Visual Studio). |

Вывод по Анализу 2:

xUnit – это рекомендуемый выбор для новых проектов на .NET Core/.NET 5+. Его философия чистоты тестов, высокая производительность и тот факт, что его использует сама команда .NET для тестирования платформы, делают его современным лидером. Синтаксис чист и понятен.

NUnit – это мощный, зрелый и очень гибкий фреймворк. Он является отличным выбором, особенно если команда уже имеет опыт работы с ним или с JUnit. [TestCase] часто считается более удобным для параметризации, чем подход xUnit.

MSTest – самый простой вариант для начала, особенно для разработчиков, которые только осваивают автоматическое тестирование и используют Visual Studio. Однако его меньшая гибкость и (исторически) производительность делают его менее популярным для сложных и больших проектов по сравнению с xUnit и NUnit.

Задание 2.

Исходя из итогов первого задания, у нас получается следующий технологический стек разработки веб-приложения:

C# + MSSQL

xUnit

Entity Framework

ASP NET.Core MVC

C# + ASP NET.Core MVC + Entity Framework Core (EF Core) + xUnit

1. ASP NET.Core – так как мы выбрали язык программирования C# для разработки сайта, то ASP NET.Core – это единственный вариант для реализации сайта используя язык программирования C#, ну и раз он единственный, то и обязательный пункт нашего стека, так как вокруг него начинают работать остальные пункты.
2. Entity Framework Core – данный пункт стека отвечает за связь создаваемого проекта с базой данной, которая реализована с использованием инструментов MSSQL. Как мы уже сказали, данный фреймворк в нашем стеке незаменим для связи кода сайта с базой данных, так как база данных и данный фреймворк являются продуктом компании Microsoft (даже язык программирования C#) они взаимосвязаны, и их будет проще связать, чем остальные рассматриваемые в задании 1 инструменты ORM.
3. xUnit – данный пункт стека отвечает за тестирование самого проекта, проверка его на ошибки и корректную работу его функционала. Мы выбрали xUnit за такие его преимущества как современность стандартов данного тестирования, высокую производительность тестов, а также за то, что он используется самой командой .NET. Поэтому он лидер среди других тестировочных сервисом для нашего сайта.

У всего этого стека есть список преимуществ, из которого стоит выделить следующие пункты:

1. Единая экосистема – все технологии данного стека отлично интегрируются
2. Поддержка Microsoft – гарантия долгосрочной поддержки
3. Высокая производительность - комбинация EF Core.
4. Качество кода - мощная система тестирования