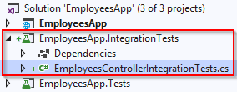
**Лабораторная работа №3**

**Интеграционное тестирование**

**Подготовка нового проекта для интеграционного тестирования**

Сначала необходимо новый проект xUnitс именем EmployeesApp. IntegrationTestsдля целей интеграционного тестирования. После создания проекта необходимо переименовать класс UnitTest1.csв EmployeesControllerIntegrationTests:



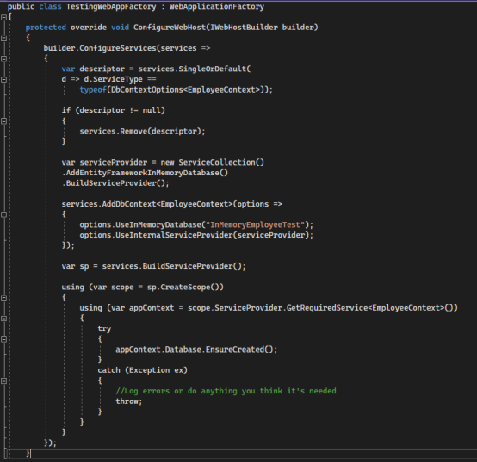
Кроме того, необходимо добавить ссылку на основной проект и установить один пакет NuGet, необходимый для целей тестирования:

AspNetCore.Mvc.Testing - этот пакет предоставляет TestServer и важный класс WebApplicationFactory, чтобы помочь нам загрузить наше приложение в памяти.

Microsoft.EntityFrameworkCore.InMemory - поставщик базы данных в памяти.

Теперь мы можем продолжить.

**Создание конфигурации фабрики In-Memory**

****

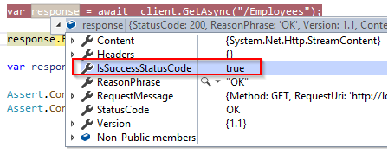
Здесь стоит упомянуть пару вещей. Наш класс реализует класс WebApplicationFactory<Startup> и переопределяет метод ConfigureWebHost. В этом методе мы удаляем регистрацию EmployeeContext из класса Startup.cs. Затем мы добавляем поддержку базы данных в памяти Entity Framework в контейнер DI через класс ServiceCollection.После этого мы добавляем контекст базы данных в контейнер службы и настраиваем его на использование базы данных в памяти вместо реальной базы данных.Далее проверяем, что мы заполняем данные из класса EmployeeContext(те же данные, которые вы вставили в реальную базу данных SQL Server в начале этой серии). Сделав все необходимые приготовления, мы можем вернуться к классу тестирования и приступить к написанию тестов.

**Интеграционное тестирование действия Index**

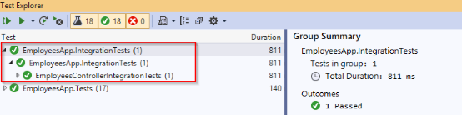
В нашем тестовом классе мы можем найти единственный тестовый метод с именем по умолчанию. Но давайте удалим его и начнем с нуля.

Итак, мы реализуем класс TestingWebAppFactory с интерфейсом IClassFixture и внедряем его в конструктор, где мы создаем экземпляр HttpClient. Интерфейс IClassFixture - это декоратор, который указывает, что тесты в этом классе полагаются на выполнение фикстуры.

Мы используем метод GetAsync для вызова действия на маршруте /Employees, которое является действием Index, и возвращаем результат в виде переменной response. С помощью метода EnsureSuccessStatusCode мы проверяем, что свойство IsSuccessStatusCode имеет значение true:

****

Если значение равно false, это будет означать, что запрос не был успешным, поэтому тест не пройден. Наконец, мы сериализуем наш HTTP-контент в строку с помощью метода ReadAsStringAsync и проверяем, что он содержит двух наших сотрудников:

****

**Заключение**

Мы видим, что тест прошел, и мы успешно вернули наших сотрудников из базы данных в памяти. Если вы хотите убедиться, что мы действительно используем базу данных в памяти, а не настоящую, вы всегда можете остановить службу SQLServer в окне «Службы» и снова запустить тест.