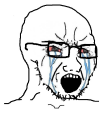


Реконструирование в Entity Framework Core. Формирование архитектуры серверного приложения. (Часть 1)

Введение

В рамках данной работы вы познакомитесь с принципами чистой архитектуры и попробуете применить некоторые принципы в своём проекте, а также при помощи реконструирования Entity Framework Core вы мигрируете модели базы данных в ваш проект используя только одну команду, чтобы в случае необходимости моментально разворачивать базу данных на любом другом сервере СУБД.

Что такое Чистая Архитектура? Это подход, который позволяет разработчикам создавать приложения, которые легко сопровождать (писать новый код или изменять текущий) и масштабировать (увеличить количество хостов). Она основана на принципах SOLID, DRY и KISS, а также на разделении приложения на слои.

Зачем нужна чистая архитектура?

1. Улучшение сопровождаемости

> ℹ️ Чистая архитектура помогает создавать приложения, которые легко сопровождать и изменять. Каждый слой приложения имеет свою ответственность, что позволяет избежать спагетти-кода. Кроме того, каждый слой может быть легко заменен или изменен без влияния на другие слои.

2. Улучшение тестирования

> ℹ️ Чистая архитектура позволяет легко тестировать каждый слой приложения независимо от других слоев. Это позволяет создавать автоматические тесты (Unit Tests) для каждого слоя и обеспечивать высокое качество кода.

3. Улучшение масштабируемости

> ℹ️ Чистая архитектура позволяет создавать приложения, которые легко масштабировать. Каждый слой может быть масштабирован независимо от других слоев, что позволяет обеспечивать высокую производительность и отзывчивость приложения.

4. Уменьшение зависимостей

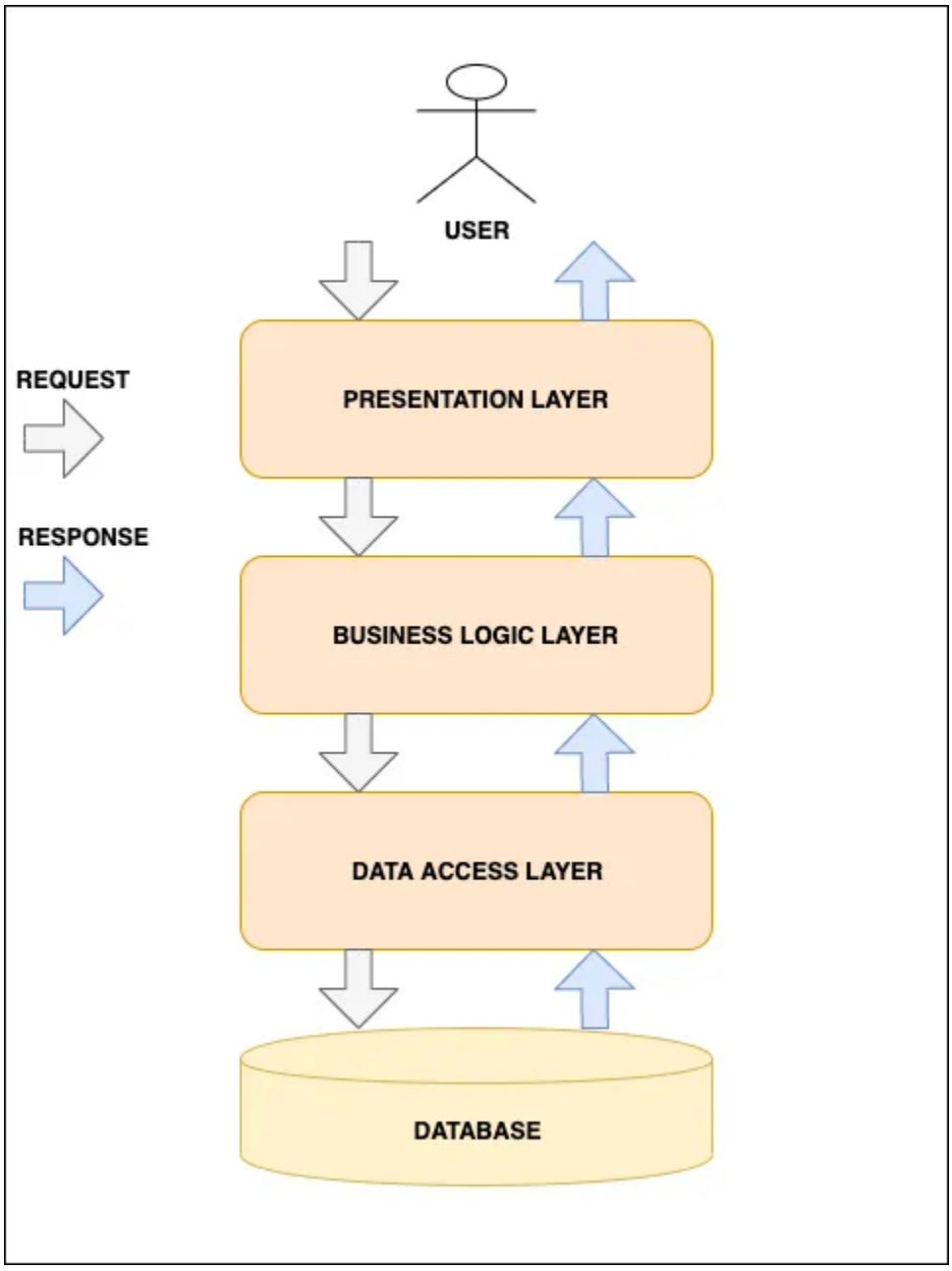
> ℹ️ Чистая архитектура позволяет уменьшить зависимости между слоями приложения. Каждый слой имеет свою ответственность и не зависит от других слоев. Это позволяет уменьшить связность между компонентами приложения и облегчить его сопровождение и изменение.

Архитектура традиционных приложений (N-Tier Architecture)

Большинство традиционных приложений .NET которые вы разрабатывали представлялись в виде одного проекта решения, который и является исполняемым файлом. (Проект Console, WPF, WinForms и т.д.) Однако, даже для таких небольших проектов, большинство разработчиков склоняется к логическому разделению проекта на отдельные слои.

Рассмотрим схему **N-Tier архитектуры** и разберем каждый из слоёв:

1. Слой представления “Presentation Layer” - Это слой отображения и взаимодействия с пользователем (То есть графический интерфейс с которым работает пользователь, например настольное приложение, мобильное приложение или веб-приложение)
2. Слой бизнес логики “Business Logic Layer” - Как следует из названия, содержит **бизнес**-**логику** приложения. Он отделяет UI/UX от вычислений, связанных с **бизнесом**. Это позволяет с легкостью изменять **логику** в зависимости от постоянно меняющихся **бизнес**-требований, никак не влияя на другие **слои**
3. Слой доступа к данным “Data Access Layer” - Это слой, который предоставляет упрощенный доступ к данным, хранимым в постоянном хранилище какого-либо типа, например как реляционная база данных, сторонний API и так далее. Таким образом приложение не зависит от конкретного источника данных и может быть легко заменён (Например с легкостью заменить базу данных SQL Server на Postgre SQL или любой другой источник данных)
4. Слой базы данных “Database” - Это непосредственно база данных (Источник данных), откуда берутся данные для приложения



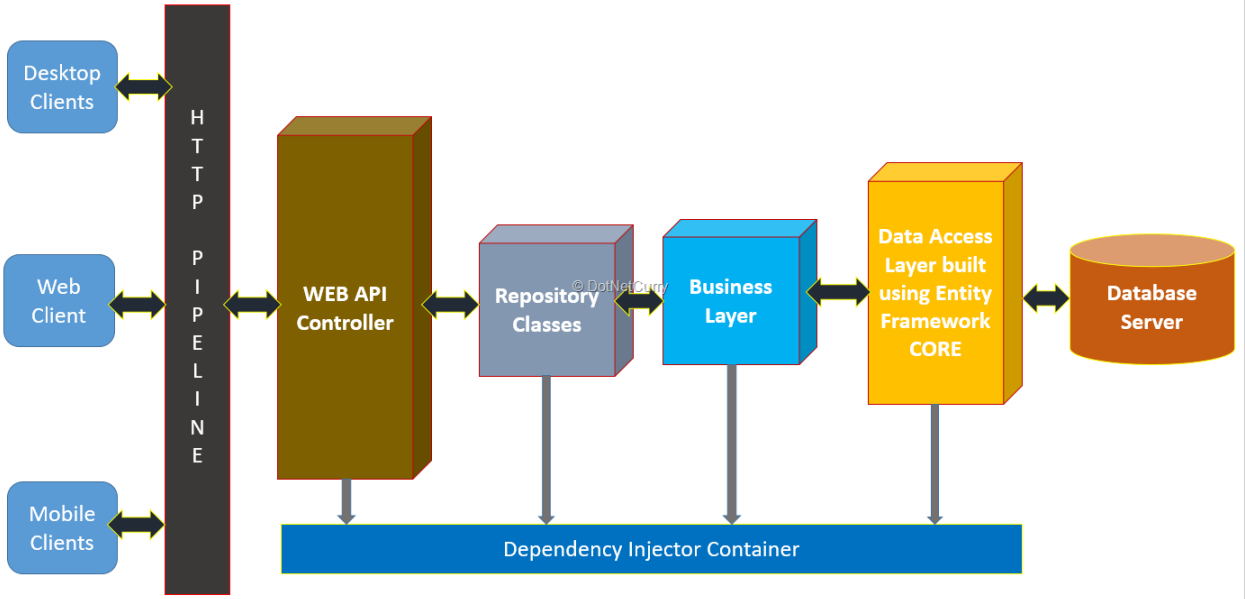
Таким образом, благодаря разделению приложения на отдельные логические слои достигается максимальная изолированность компонентов друг от друга и появляется удобная возможность для тестирования отдельных компонентов.

Рассмотрим схему к которой мы планомерно идём:

1. Мы создали API (Серверное приложение), который, в дальнейшем, будет предоставлять доступ к данным и решать некоторые бизнес-процессы

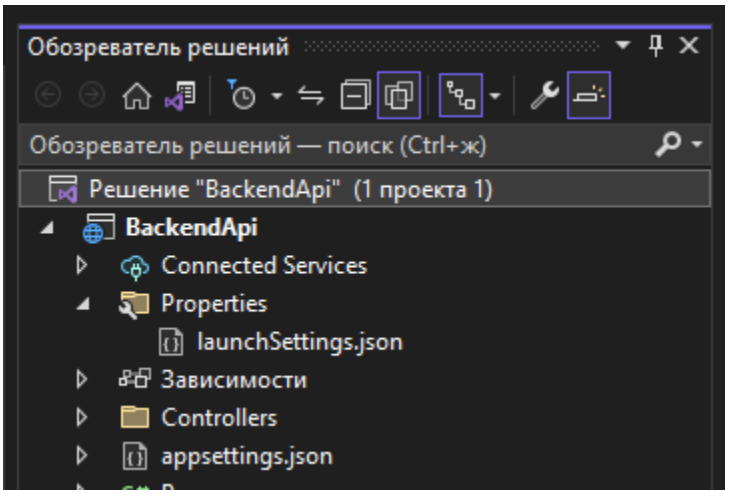
(Например: формировать заказ или выдавать доступный ассортимент). Данный API будет разделен на несколько логических слоев, которые будут связаны при помощи внедрения зависимостей (Dependency Injection)

2. Мы сформировали базу данных для серверного приложения, которое будет обеспечивать хранение данных
3. После полной реализации API мы сформируем один из клиентских приложений, который обеспечит пользователю возможность для взаимодействия с серверным приложением. Общение между клиентом и сервером будет обеспечивать протокол HTTP/HTTPS.

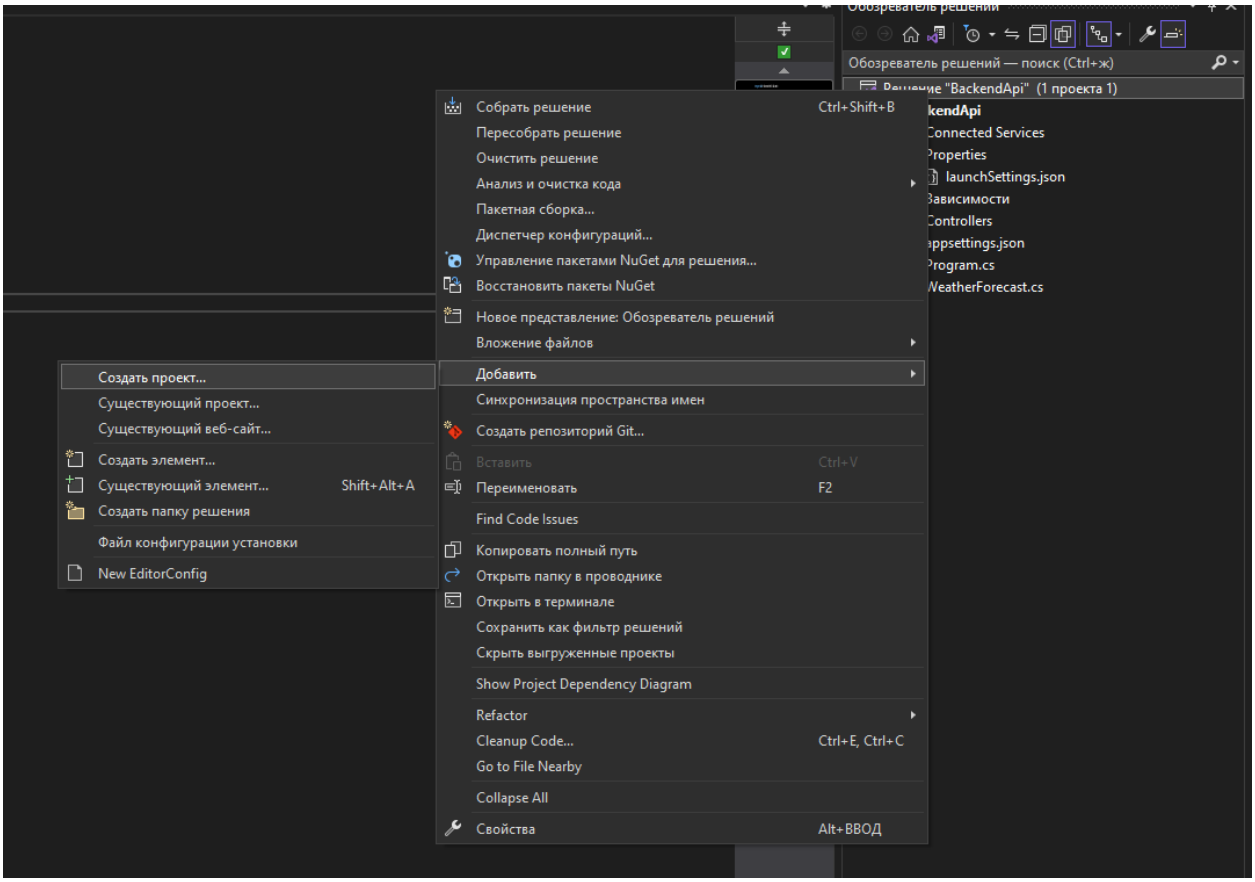


Формирование логических слоев серверного приложения

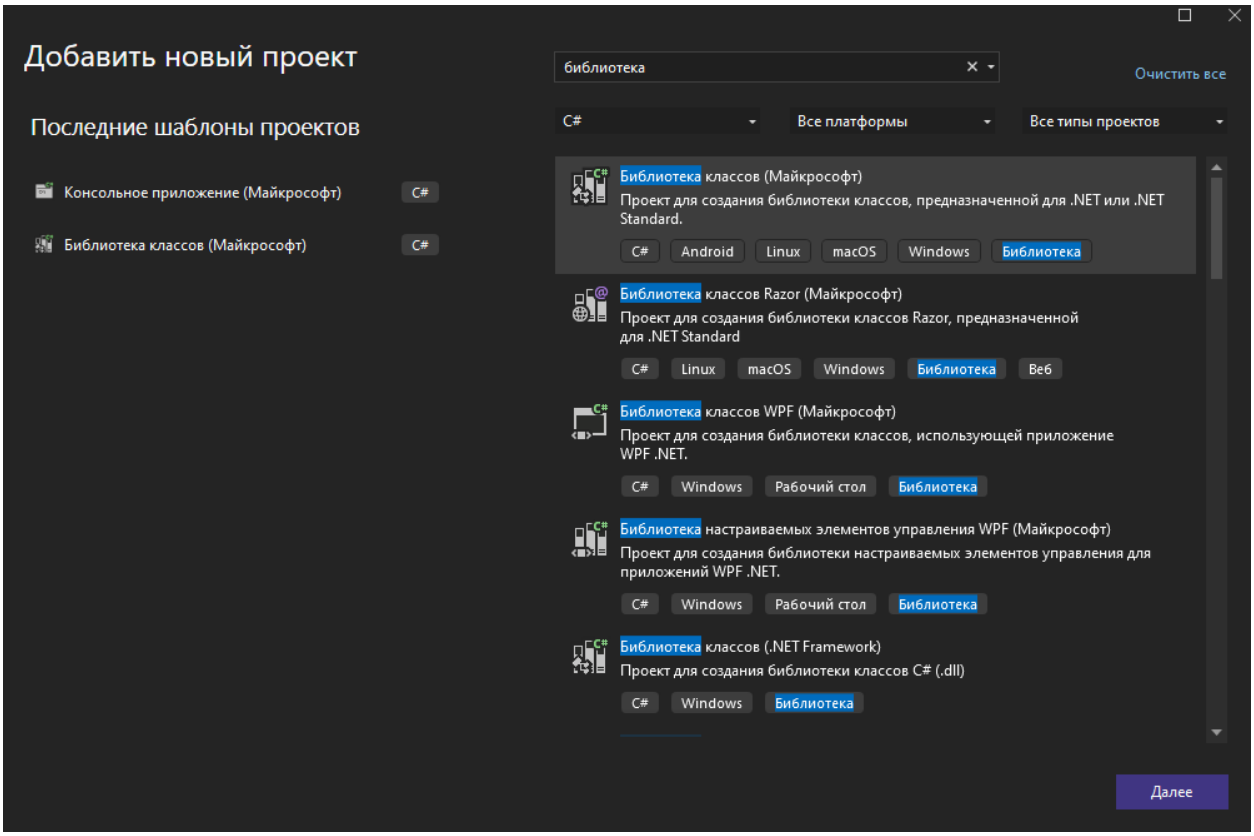
Откройте ваш проект и нажмите правой кнопкой мыши по решению.



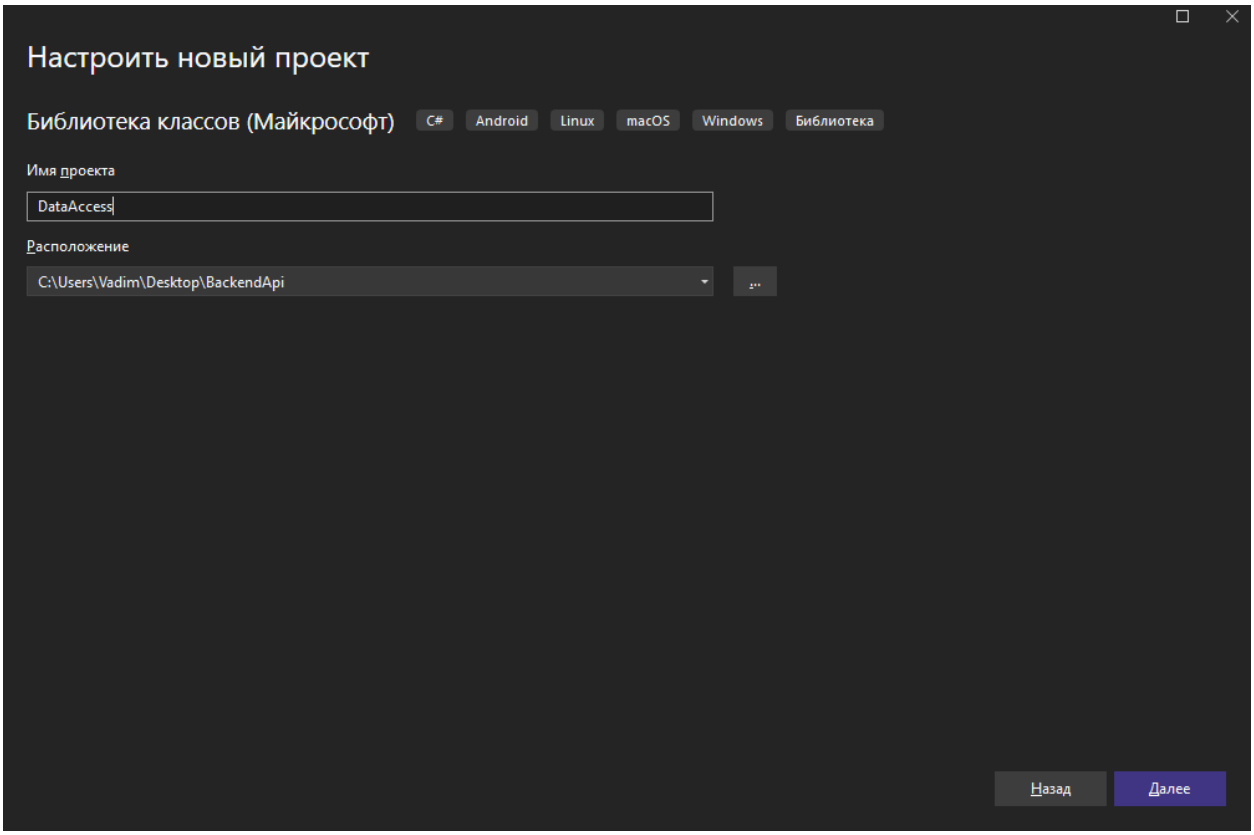
Выберите пункт “Добавить” и нажмите на “Создать проект…”



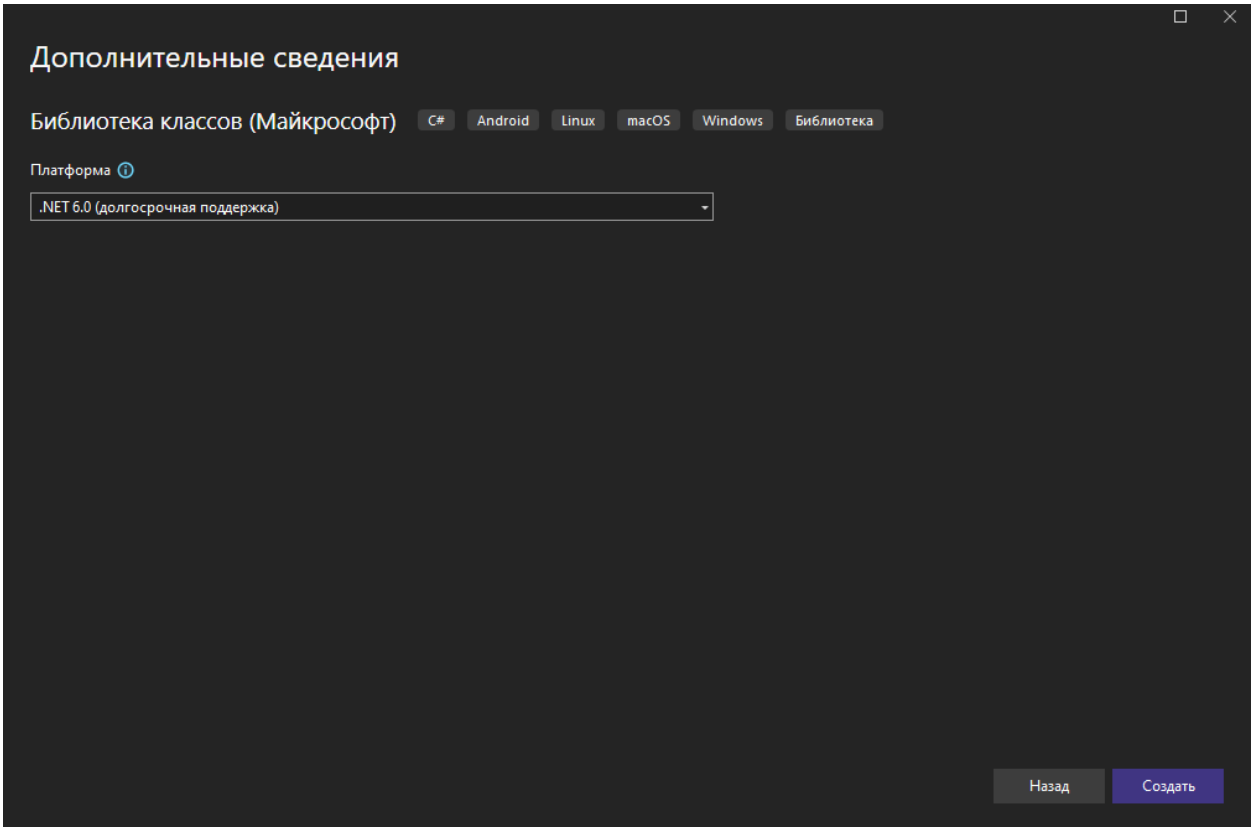
Выберите “Библиотека классов (Майкрософт)”



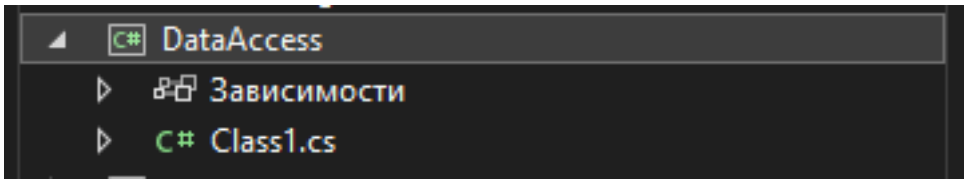
В качестве имени проекта укажите “DataAccess”



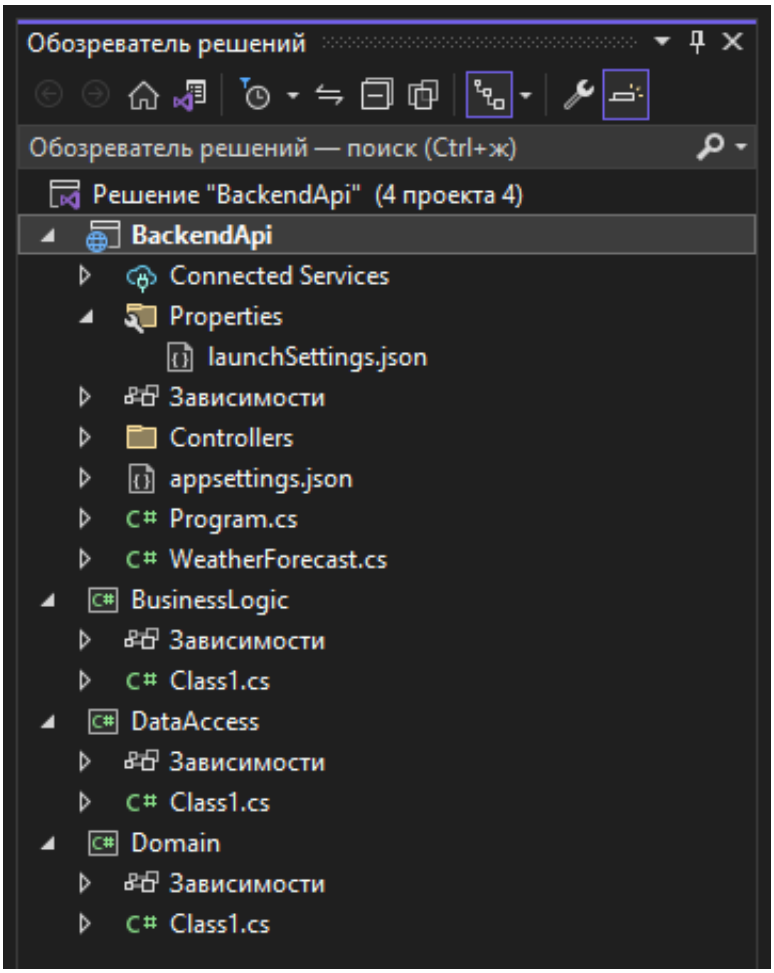
Платформу указывайте такую-же, какая была указана для проекта Web API



В решение добавился новый проект (В нашем случае, это слой для доступа к данным)



Добавьте в решение ещё два проекта: Domain и BusinessLogic



Миграция моделей данных в проект. Реконструирование.

В Entity Framework Core имеется механизм для генерации моделей базы данных на основе существующей базы данных.

Так как мы начали проектирование приложения именно с базы данных (Database First), то нам потребуются модели (классы), которые будут описывать таблицы базы данных для работы с данными.

Для работы с БД будут использоваться следующие пакеты:

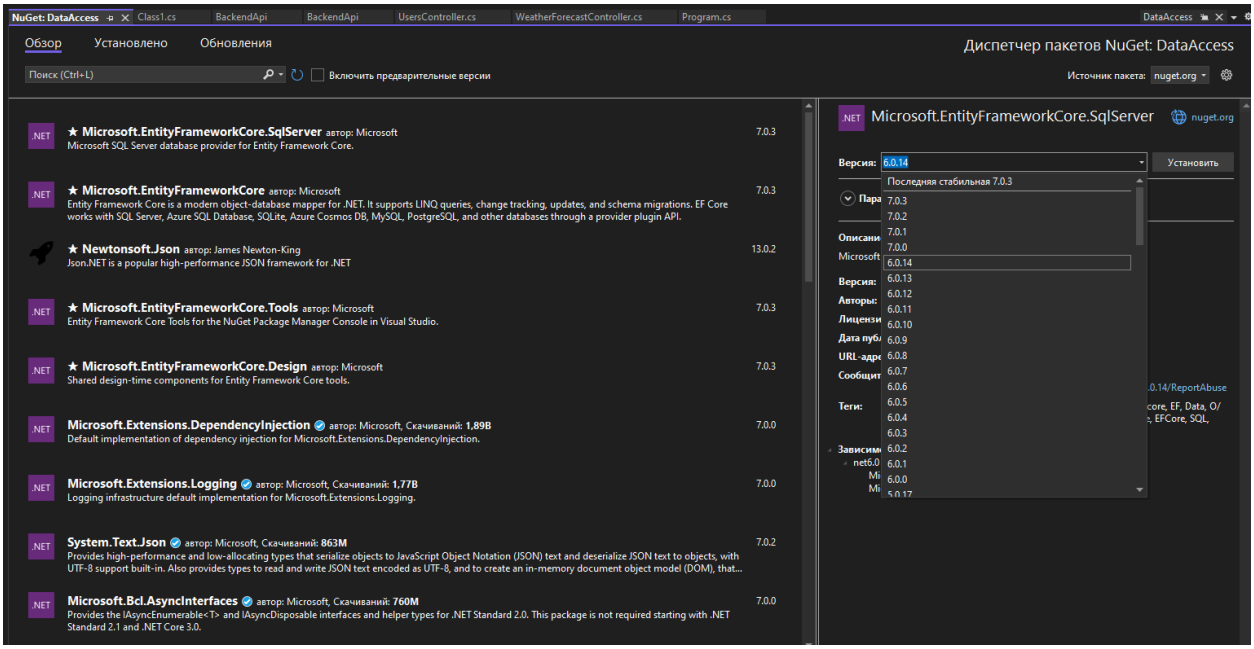
- **Microsoft.EntityFrameworkCore.SqlServer**
- **Microsoft.EntityFrameworkCore.Tools**

Нажмите ПКМ по проекту DataAccess.

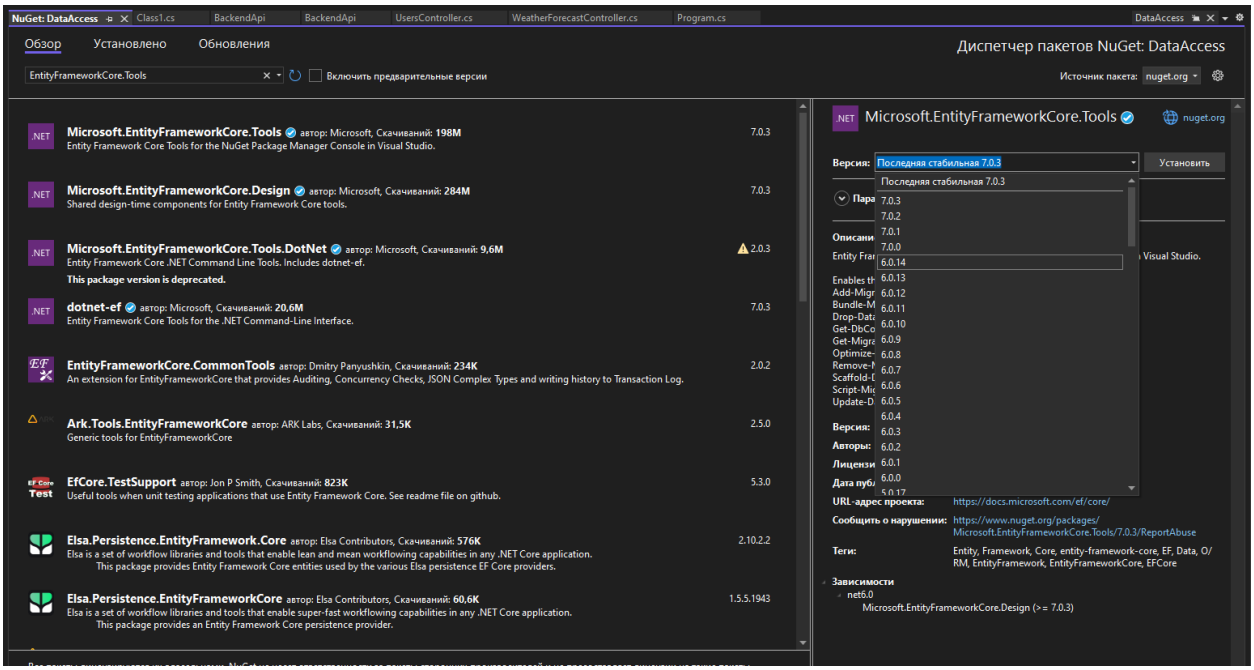
Найдите пакет **Microsoft.EntityFrameworkCore.SqlServer** и укажите подходящую версию для вашего проекта:

- Если ваша платформа .NET 6 - выбирайте версию, которая начинается на 6.
- Если ваша платформа .NET 5 - выбирайте версию, которая начинается на 5
- Если ваша платформа .Net Core 3.1 - выбирайте версию, которая начинается на 3.1

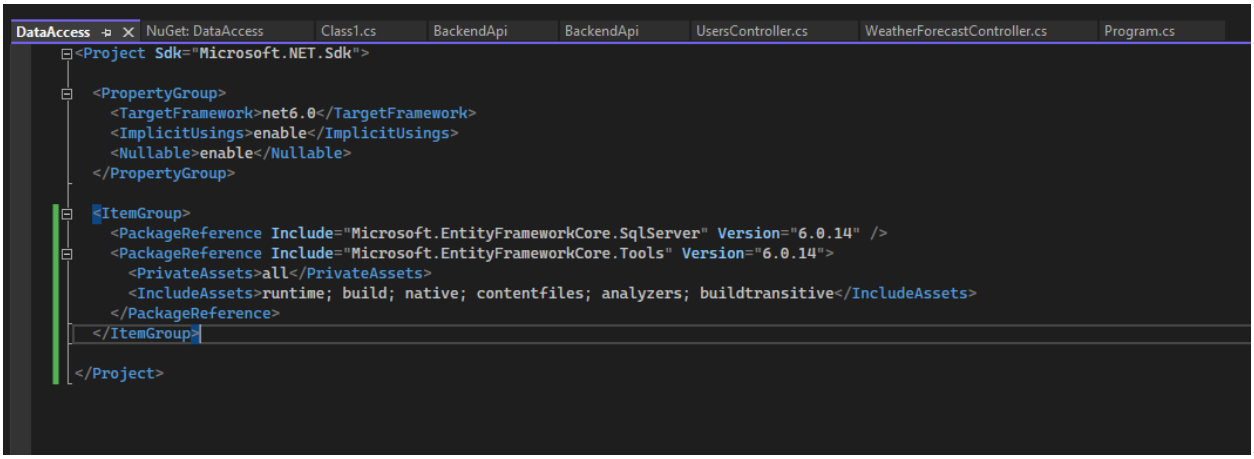
Установите выбранную версию пакета в ваш проект.



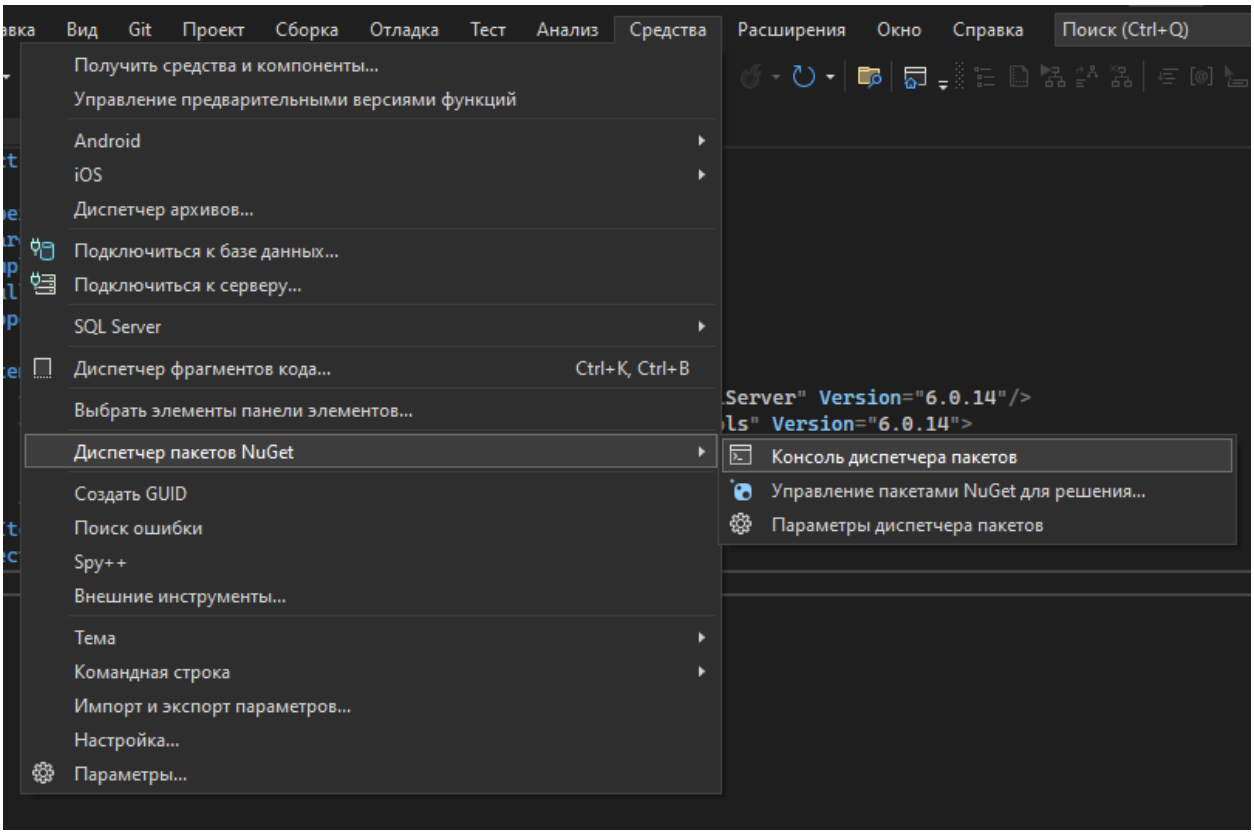
Найдите библиотеку **Microsoft.EntityFrameworkCore.Tools** и выполните такие же действия:



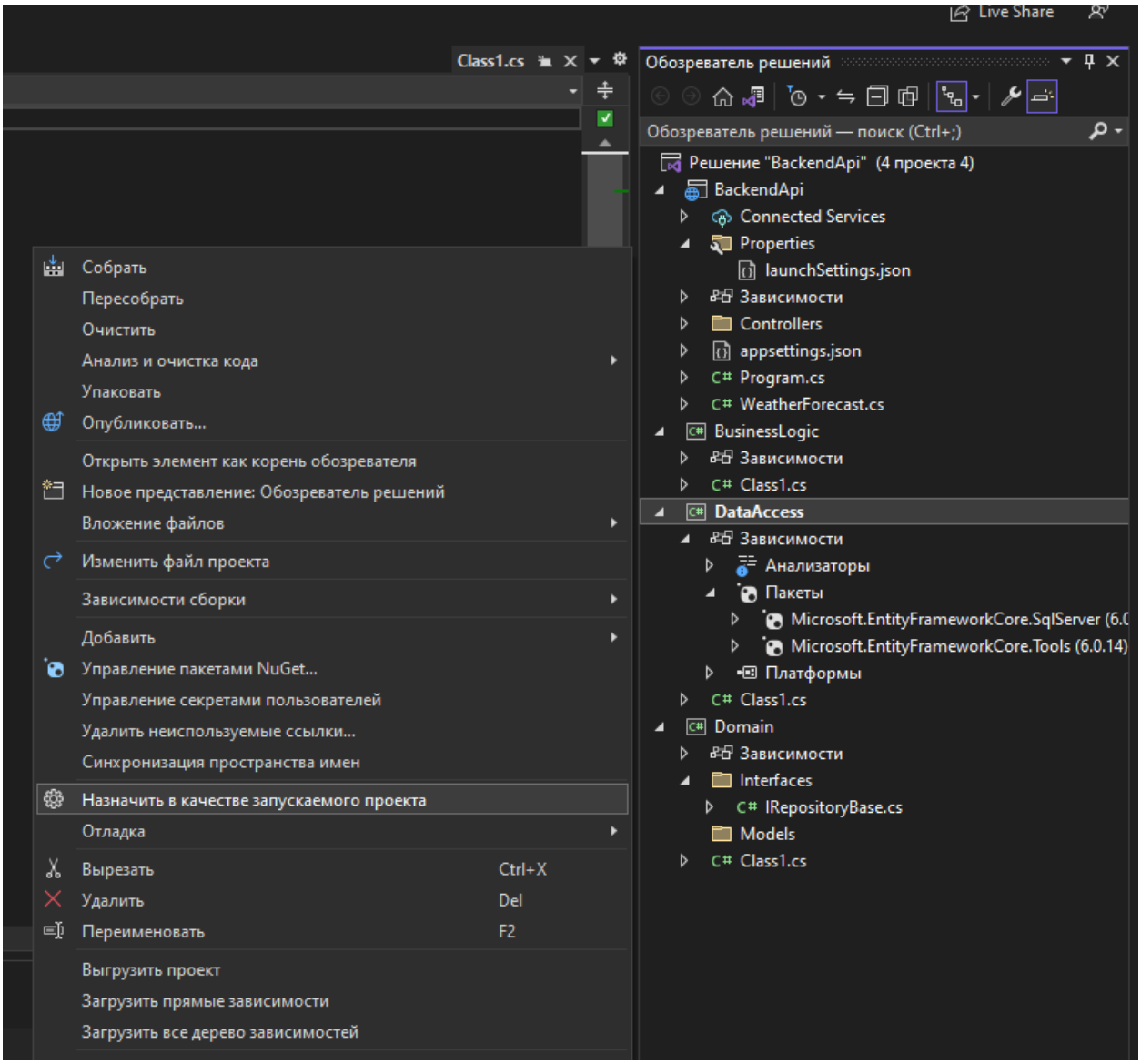
Проверьте, что пакеты были установлены в ваш проект



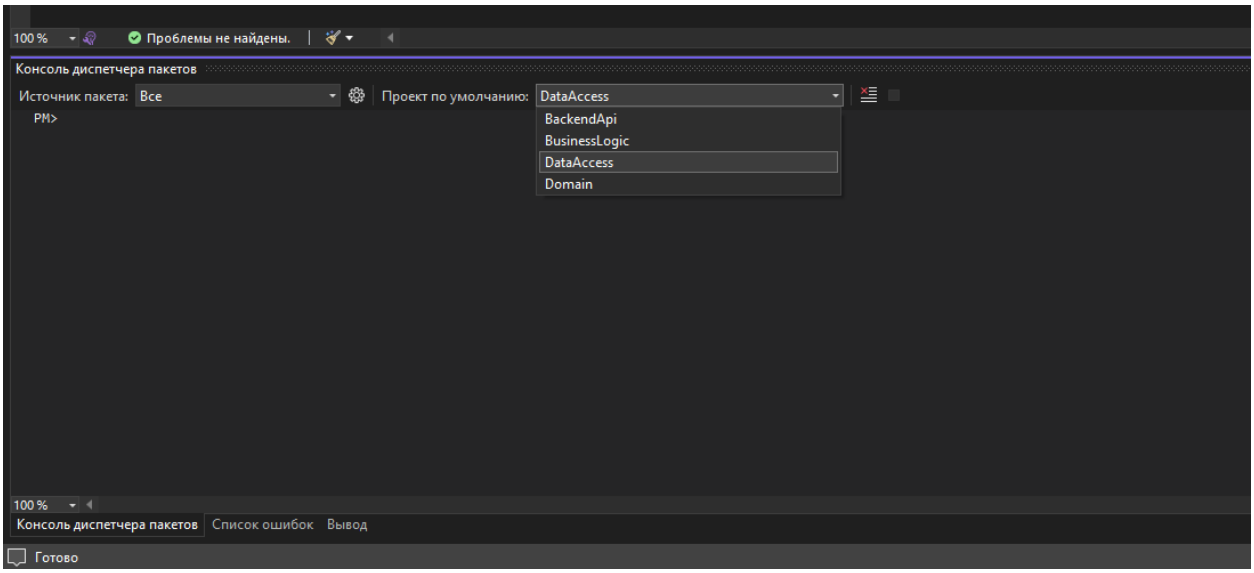
После установки пакетов **Microsoft.EntityFrameworkCore.SqlServer и Microsoft.EntityFrameworkCore.Tools**, откройте “Консоль диспетчера пакетов”.



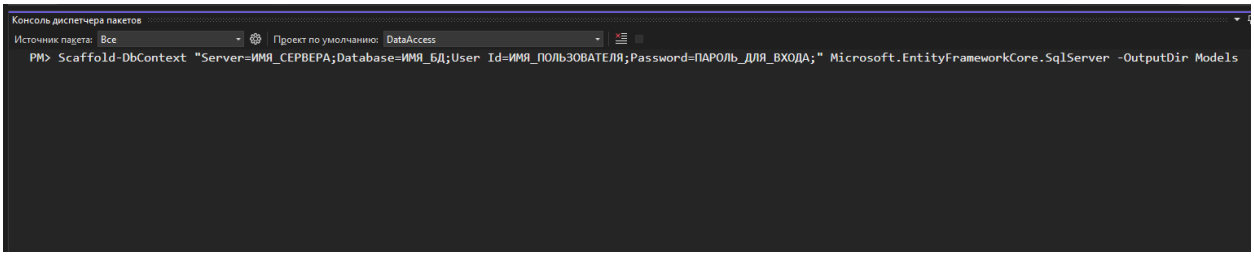
Укажите, что слой DataAccess будет выступать в качестве запускаемого проекта



В консоли диспетчера пакетов укажите DataAccess в качестве проекта по умолчанию

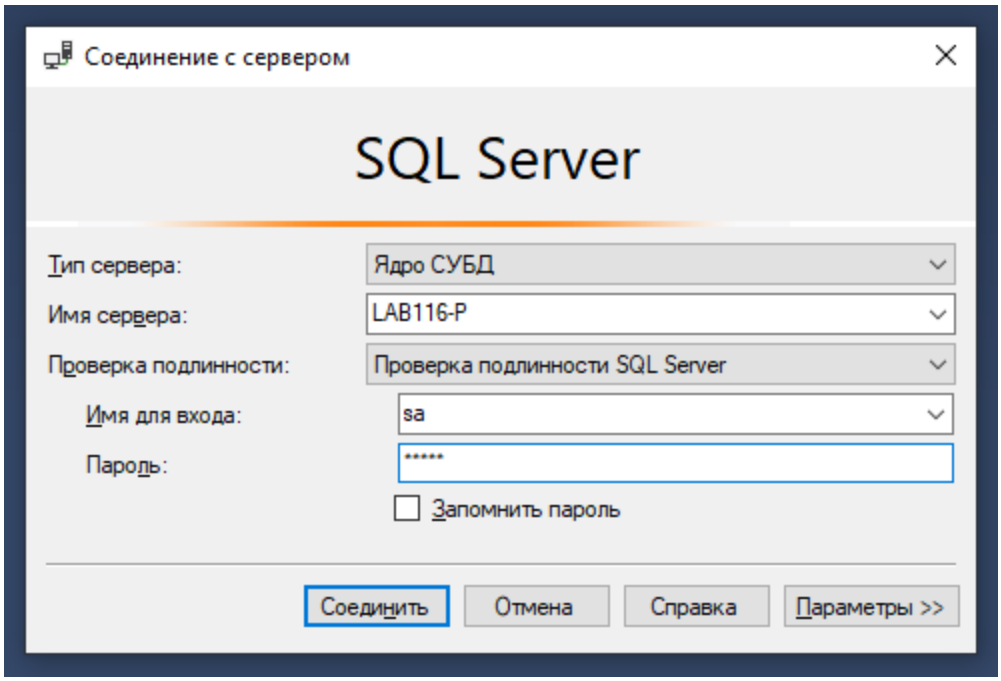


В данном окне напишите следующую команду и подставьте для полей Server, Database, User Id, Password соответствующие значения из



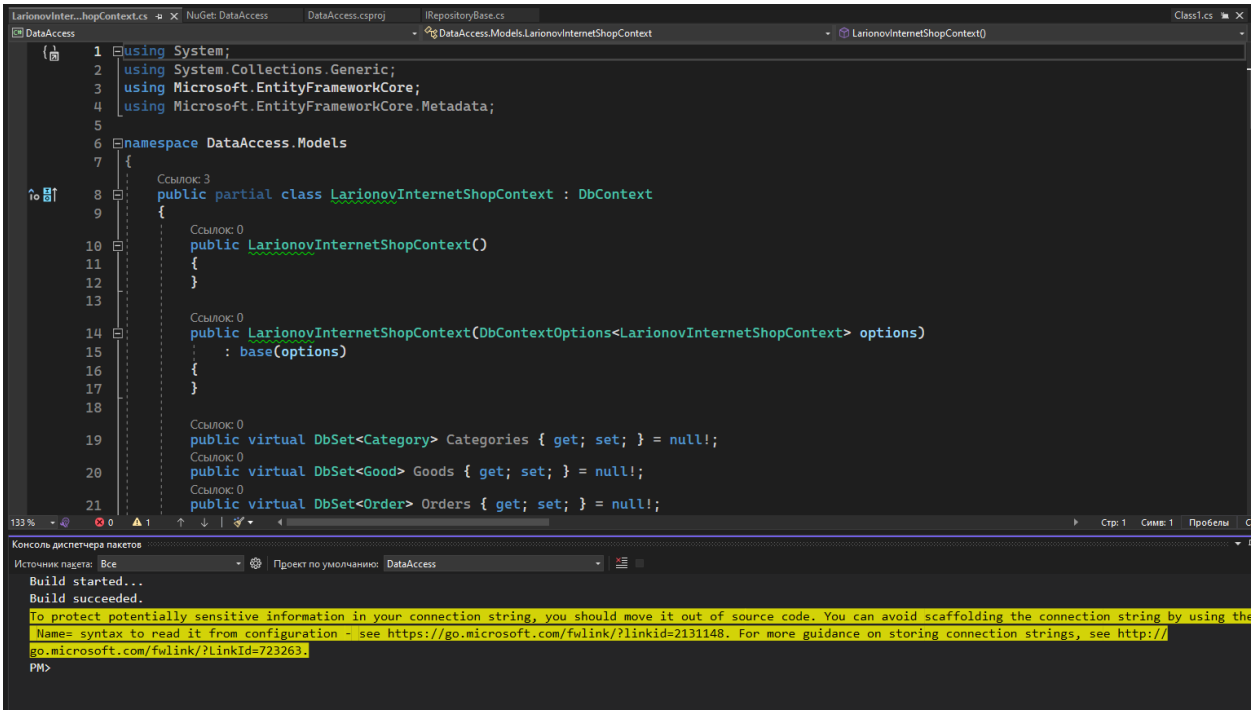

```
Scaffold-DbContext "Server=ИМЯ_СЕРВЕРА;Database=ИМЯ_БД;User Id=ИМЯ_ПОЛЬЗОВАТЕЛЯ;Password=ПАРОЛЬ_ДЛЯ_ВХОДА;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Подставьте для полей Server, Database, User Id, Password соответствующие значения из окна соединения с сервером.



Выполните команду, нажмите на кнопку “Enter”

В результате произойдёт сборка проекта и выполнение данной команды.



💡 **Если данная команда выдаёт ошибку, значит команда была написана неверно.**

В случае успеха - в проекте DataAccess появится папка “Models” и при её раскрытии будут сформированы модели (таблицы) из вашей базы данных, а также будет представлен класс Context для взаимодействия с БД