**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«РОССИЙСКАЯ АКАДЕМИЯ НАРОДНОГО ХОЗЯЙСТВА**

**И ГОСУДАРСТВЕННОЙ СЛУЖБЫ**

**ПРИ ПРЕЗИДЕНТЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**Смоленский филиал РАНХиГС**

Специальность:

09.02.07 Информационные системы и программирование

**ПРОЕКТНАЯ РАБОТА**

по дисциплине **«**Управление и автоматизация баз данных»

на тему: «Проектирование элементов системы автоматизации продаж музыкального оборудования»

**Автор:**

обучающийся

группы 37/11К-ИТО

очной формы обучения

Махницкий Д.С.

**Руководитель:**

преподаватель

Харламова О.Е.

Смоленск 2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc181971278)

[1 Информационное, алгоритмическое и программное обеспечение задачи автоматизации 4](#_Toc181971279)

[2 Тестирование элементов системы и формирование технологической среды 5](#_Toc181971280)

[3 Анализ проблем автоматизации процесса продаж музыкального оборудования и пути их решения 6](#_Toc181971281)

[ЗАКЛЮЧЕНИЕ 7](#_Toc181971282)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 8](#_Toc181971283)

[ПРИЛОЖЕНИЕ А 9](#_Toc181971284)

[ПРИЛОЖЕНИЕ Б 10](#_Toc181971285)

[ПРИЛОЖЕНИЕ В 11](#_Toc181971286)

# ВВЕДЕНИЕ

# 1 Информационное, алгоритмическое и программное обеспечение задачи автоматизации

Согласно предметной области (продажа товара в магазине музыкального оборудования), была смоделирована база данных (БД) с помощью библиотеки Microsoft.EntityFrameworkCore на языке С# и системы управления базами данных (СУБД) MySQL. Данная БД хранит информацию о товаре, который либо находится на складе, либо ожидает поставки, либо продан, либо зарезервирован, либо находится в электронной корзине (логическое место, куда продавец добавляет товары для совершения единой их продажи), либо ожидает оплаты (следующий этап после нахождения в корзине, но до получения статуса продано). На рисунках 1.1, 1.2 и 1.3 изображены концептуальная, логическая и физическая модели базы данных соответственно.

Изображение выглядит как текст, снимок экрана, диаграмма, число

Автоматически созданное описание

Рисунок 1.1 – Концептуальная модель БД в нотации IDEF1X

Изображение выглядит как текст, снимок экрана, диаграмма, число

Автоматически созданное описание

Рисунок 1.2 – Логическая модель БД в нотации Мартина

В логической модели (рисунок 1.2) можно заметить определенную симметрию – каждый вид музыкального товара, а именно аксессуары (accessories), аудио-оборудование (audio\_equipment\_units), нотные издания (sheet\_music\_editions) и музыкальные инструменты (musical\_instruments) имеет собственную таблицу для типов с шаблонным названием имя\_вида\_товара\_specific\_types, а так же свою связующую таблицу для отношения (тоже с шаблонным названием имя\_вида\_товара\_sale) многие-ко-многим между собой и таблицей «продажи» (sales).

Опишем преимущество использования отдельной таблицы со специфичными типами (подтипами) для каждого вида товара. Каждый вид товара имеет уникальные подтипы, которые, в данном случае, могут пересекаться с названиями других подтипов (например, данная структура позволят добавить в БД гитару (музыкальный инструмент) с подтипом «акустика», а также колонки (аудио-оборудование) с таким же подтипом, поддерживая ограничение уникальности названия для каждого вида товара. Из альтернатив данной реализации можно рассмотреть создание единой таблицы со всеми подтипами. В таком случае необходимо было бы создать вторую таблицу со всеми видами товара, и в каждой строке подтипа хранить информацию о виде товара (дискриминатор). Например, строка, которая в текущей реализации выглядит как отображено в таблице 1.1, в альтернативном варианте выглядела бы как в таблице 1.2.

Таблица 1.1 – Пример строки из таблицы musical\_instrument\_specific\_type

|  |  |
| --- | --- |
| specific\_type\_id | name |
| 1 | Акустика |

Таблица 1.2 – Пример строки из альтернативного варианта таблицы подтипов specific\_types

|  |  |  |
| --- | --- | --- |
| specific\_type\_id | name | kind\_of\_goods |
| 1 | Акустика | Музыкальные инструменты |

Таким образом, каждая строка должна была бы хранить лишнюю (с точки зрения текущей реализации) информацию (пренебрегая нормализацией). В текущей реализации вид товара «автоматически» хранится в таблице в соответствии с ее названием. Второй плюс данной реализации – более быстрый запрос подтипов товара определенного вида – достаточно выполнения следующей команды:

SELECT \*

FROM `musical\_instrument\_specific\_types`;

вместо:

SELECT \* `specific\_types`

WHERE `kind\_of\_goods` = ‘Музыкальные инструменты’;

Другими словами, в текущей реализации подтипы уже отфильтрованы, что увеличивает скорость выполнения запроса на получение всех подтипов определенного вида товара.

Для хранения информации о товаре разных типов используется подход Table Per Class (TPC) – для каждого класса (сущности) создается отдельная таблица со всеми свойствами (атрибутами). Объясняется это тем, что в данном случае ожидается, что поиск товара (относительно сложный по своей структуре запрос) будет осуществляться довольно часто, причем всегда по одному виду товара (с точки зрения предметной области нет смысла одновременно пытаться искать музыкальный инструмент и нотное издание (например, по фразе «синтезатор») – при поиске конечный пользователь всегда знает какой вид товара его интересует) и в результате, при поиске товара запрос так же, как в предыдущем случае с подтипами, не будет содержать дополнительной фильтрации по виду товара.

Оба сделанных выбора вариантов хранения информации в БД основаны на эмпирическом методе (так как данная работа – учебная), в реальной информационной системе (ИС) необходимо выполнить тесты для определения лучшего подхода.

Важная особенность данной БД заключается в том, что таблица sales (продажи) имеет отношение с таблицами товаров разного вида «многие-ко-многим», а таблица goods\_delivery (доставка товара) – «один-ко-многим». Рассмотрим два похожих по смыслу случая:

- Клиент магазина купил музыкальный инструмент, но позже вернул его. Затем этот инструмент купил другой клиент.

- Поставщик доставил нотное издание, которое через некоторое время купил определенный клиент, а затем это нотное издание «чудом» (крайне маловероятный случай) оказалось на складе у поставщика, который затем снова выполнит поставку товара с этим нотным изданием.

В результате определенный товар может побывать сразу в двух и более разных покупках и на практике это происходит регулярно. Возврат товара клиентом – обыденное дело. Также книга действительно может побывать в двух и более поставках товара, но вероятность данного случая настолько мала, что ей можно пренебречь. Соответственно, в первом случае реализовано отношение «многие-ко-многим», а во втором – «один-ко-многим».

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 1.3 – Физическая модель базы данных в нотации Мартина

Относительно физической модели БД можно сказать следующее:

- В качестве идентификатором используются глобально уникальные идентификаторы (GUID), длина которых – 36 символов (включая тире) поэтому id всех таблиц имеет тип CHAR(36).

- Атрибут soft\_deleted (досл. мягко удалено) отображает состояние логической удаленности строки (1 – удалено, 0 – не удалено). Данный атрибут отсутствует у связующих таблиц, так как в его наличии у этих таблиц нет смысла.

- Таблицы с информацией о товарах атрибуты с одинаковыми названиями и типами, что в дальнейшем будет отображено на диаграмме классов. При этом некоторые таблицы имеют дополнительный атрибуты. Например, таблицы musical\_instruments и sheet\_music\_edition имеют атрибут release\_year (год выпуска).

- Таблица sales имеет три атрибута типа DATETIME. Согласно предметной области, у продажи должна быть как минимум один из этих атрибутов. Например, зарезервированная продажа будет иметь значение только у атрибута reservation\_date (дата резервирования) и находиться в состоянии ожидания получения клиентом. Наличие значения только у одного из других двух атрибутов так же имеет смысл с точки зрения предметной области. В некоторых случаях продажа может иметь значение сразу у всех трех атрибутов (клиент зарезервировал товар, оплатил (и получил), а затем вернул). Целостность состояния данных трех атрибутов поддерживается с помощью триггеров на внесение и обновление строк.

Назначение ИС – автоматизация продаж музыкального оборудования. Конечные пользователи – персонал магазина, то есть ограниченный круг лиц (от четырех человек). Рассмотрим алгоритмы работы ИС. У каждой категории пользователей есть типичный для нее способ взаимодействия с системой, которым они пользуются большую часть времени:

- продавец ищет и резервирует товар, просматривает о нем информацию, оформляет продажи и возвраты,

- администратор добавляет, удаляет и редактирует существующих пользователей, просматривает отчеты и создает резервные копии БД,

- складской менеджер вносит информацию о новом товаре в ИС,

- консультант ищет товары и просматривает о них информацию.

В данном случае у консультанта меньше всего прав в данной ИС, а все остальные пользователи наделены правами консультанта, соответственно сначала рассмотрим его алгоритм работы (рисунок 1.4).

Изображение выглядит как текст, диаграмма, линия, чек

Автоматически созданное описание

Рисунок 1.4 – типичный способ использования ИС консультантом

Здесь пользователь вводит информацию, которая поможет найти желаемый товар, затем система обрабатывает данный запрос и выводит результаты. Если желаемый товар не был найден, рекомендуется повторить ввод данных для поиска, так как там могла быть ошибка пользователя (например, он выбрал вид «аудио-оборудование», но пытался найти товар по запросу «глюкофон», что является музыкальным инструментом).

Рассмотрим способ использования ИС продавцом (рисунок 1.5), алгоритм работы которого изображен в виде жизненного цикла определенной продажи определенному покупателю: сначала клиент может зарезервировать продажу, затем оплатить ее и получить товар, а затем может вернуть. Как указано в комментариях к блок-схеме, этап резервации и возврата может отсутствовать – например, клиент пришел в магазин, купил товар и больше не возвращался.

Изображение выглядит как диаграмма, текст, Технический чертеж, План

Автоматически созданное описание

Рисунок 1.5 - типичный способ использования ИС продавцом

Алгоритм работы складского менеджера (рисунок 1.6) достаточно тривиален – пользователь вводит данные о товаре, система выполняет валидацию и если она успешна то добавляет товар в ИС, иначе – отображает ошибки.

Изображение выглядит как текст, диаграмма, линия, шаблон

Автоматически созданное описание

Рисунок 1.6 - типичный способ использования ИС складским менеджером

Алгоритм использования ИС администратором не такой структурированный и последовательный, как предыдущие алгоритмы, поэтому его описание будет текстовым. Администратор управляет всеми пользователями в ИС, включая следующие действия:

- просмотр и редактирование полной информации о пользователе,

- добавление или удаление ролей пользователя,

- добавление или удаление пользователей.

Относительно отчетности администратору доступны следующие действия:

- генерация отчетов в соответствии с желаемыми параметрами,

- просмотр отчетов.

Действия администратора относительно резервного копирования БД:

- создание логической резервной копии БД с текстовым примечанием,

- восстановление базы данных из резервной копии.

Таким образом, совокупность обобщенно описанных алгоритмов представляет собой описание общего алгоритма работы ИС.

В качество сложного алгоритма ИС можно рассмотреть алгоритм, который оформляет продажу как проданную. Вводная информация об ИС: архитектура – n-уровневая (слоистая), используемый в приложении паттерн проектирования для обработки HTTP-запросов – Model View Controller (MVC). В качестве подготовки для выполнения работы данного алгоритма сначала необходимо добавить минимум одну единицу товара в корзину, как это изображено на рисунке 1.7.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 1.7 – Корзина с четырьмя добавленными единицами товара

После нажатия кнопки «Оформить продажу» пользователь переходит на страницу оплаты (рисунок 1.8), но перед переходом на эту страницу происходит обработка товара в корзине и создание новой продажи. Кнопку «Оформить продажу» обрабатывает метод CreateSaleAsNotSold контроллера SalesController (листинг 1.1). На странице оплаты используется latch (заглушка) вместо меню оплаты.

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Автоматически созданное описание

Рисунок 1.8 -

При нажатии на кнопку «Успешно оплачено», пользователь получает сообщение «Успешно оплачено» (рисунок 1.9), товар удаляется из корзины (рисунок 1.10), а в списке продаж появляется только что совершенная продажа (рисунок 1.11).

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, мультимедиа

Автоматически созданное описание

Рисунок 1.9 – Сообщение об успешной оплате

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.10 – Очищенная корзина после совершения продажи

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Автоматически созданное описание

Рисунок 1.11 – Совершенная продажа

Листинг 1.1

[HttpPost("/sale/arrange")]

public async Task<IActionResult> CreateSaleAsNotSold([FromForm] SalePaidBy? paidBy, [FromServices] ICreateSaleService createSaleService, [FromServices] ICartService cartService)

{

var goods = await cartService.GetGoodsFromCart(GoodsIdsAndKinds);

Guid? saleId = await createSaleService.CreateSaleAsNotPaid(goods, paidBy);

if (!createSaleService.HasErrors)

{

ClearCart();

return RedirectToAction("PayForSale", new { saleId });

}

else

return RedirectToAction("Cart", "Goods"); //следовало вывести ошибки, Но Это Потом (НЭП)

}

Данный метод получает товар из корзины в виде объектов, создает продажу с этими товарами (со статусом ожидает оплаты). При отсутствии ошибок (что проверяется с помощью свойства ICreateSaleService.HasErrors и объясняется далее) происходит очистка корзины, так как теперь товар привязан к определенной продаже, а затем происходит перенаправление на страницу оплаты (через метод GET, в качестве query-string параметра передается saleId с помощью анонимного объекта с использованием инициализатора проекций – свойство объекта получает имя текущей переменной (или свойства), переданной ему). Для выполнения описанных выше действий в используемом в данном листинге методе ICreateSaleService.CreateSaleAsNotPaid() применяется паттерн, описанный в книге Джона П Смита (John P Smith) «Asp.Net core in action» под названием BuzRunner [JOHN SMITH]. Данный паттерн упрощает обработку сложной бизнес-логики, но усложняет написание простой бизнес-логики, поэтому при разработке ИС он использовался не везде. Для упрощения разработки данной ИС паттерн был перенесен в отдельную библиотеку BizRunner и модифицирован. Итоговая диаграмма классов данного паттерна изображена на рисунке 1.12. Идея заключается в следующем: класс Runner (имеется в виду любой объект данного класса. Можно было создать абстрактный класс Runner, но в этом не было необходимости) вызывает метод Action() у переданного ему реализующий интерфейс IBizAction<in TIn, outTOut> объект, затем проверяет данный объект на наличие ошибок. Если ошибок нет, то Runner сохраняет все изменения, сделанные этим объектом. Метод Runner.Run() выглядит следующим образом:

public TOut Run(TIn argument)

{

var result = action.Action(argument);

if (!action.HasErrors)

context.SaveChanges();

return result;

}

Класс Runner создается один раз и затем используется несколько, в разных сервисах, получая разные объекты действий. Тем не менее, данный класс может иметь разные виды, например, RunnerTransact2WriteDb, который использует два объекта действий вместо одного и выполняет их в виде одной транзакции. В самом методе IBizAction.Action(argument) и упакована бизнес-логика.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

Рисунок 1.12 – Диаграмма классов библиотеки реализующей паттерн BizRunner

В качестве примера класса, реализующего интерфейс IBizAction<in TIn, out TOut>, можно привести класс UpdateUserAction, а точнее, его метод Action(TIn argument) (листинг 1.2).

Листинг 1.2

public async Task<string?> Action(UpdateUserDto dto)

{ // некоторые свойства можно было валидировать через js, НЭП

var user = await dbAccess.GetUserInfo(dto.Id);

if (user == null)

{

AddError("Пользователь с таким идентификатором не найден", nameof(user.Id));

return null;

}

if (string.IsNullOrWhiteSpace(user.UserName))

AddError("Имя пользователя не может быть пустым", nameof(user.UserName));

else

{

var normalizedUserName = dto.UserName!.ToUpper();

if (!await dbAccess.IsUniqueNormalizedUserName(normalizedUserName, user.Id))

AddError("Пользователь с таким именем уже существует", nameof(user.UserName));

else

{

user.UserName = dto.UserName;

user.NormalizedUserName = normalizedUserName;

}

}

if (dto.EmailConfirmed && string.IsNullOrWhiteSpace(dto.Email))

AddError("Электронная почта указана как подтврежденная, но ее значение отсутствует", nameof(user.EmailConfirmed), nameof(user.Email));

if (!string.IsNullOrWhiteSpace(dto.Email))

{

var normalizedEmail = dto.Email.ToUpper();

if (!await dbAccess.IsUniqueNormalizedEmail(normalizedEmail, user.Id))

AddError("Данная электронная почта уже используется", nameof(user.Email));

else

{

user.Email = dto.Email;

user.NormalizedEmail = normalizedEmail;

user.EmailConfirmed = dto.EmailConfirmed;

}

}

else

{

user.Email = null;

user.NormalizedEmail = null;

user.EmailConfirmed = false;

}

if (dto.PhoneNumber == null)

{

if (dto.PhoneNumberConfirmed)

AddError("Номер телефона указан как подтврежденный, но его значение отсутствует", nameof(user.PhoneNumberConfirmed), nameof(user.PhoneNumber));

}

else if (dto.PhoneNumber.Length != 11)

AddError("Номер телефона указан не полностью", nameof(user.PhoneNumber));

user.PhoneNumber = dto.PhoneNumber;

user.PhoneNumberConfirmed = dto.PhoneNumberConfirmed;

user.TwoFactorEnabled = dto.TwoFactorEnabled;

user.LockoutEnd = dto.LockoutEnd != null ? ((DateTime)dto.LockoutEnd).ToUniversalTime() : null;

user.LockoutEnabled = dto.LockoutEnabled;

return user.Id;

}

Продолжим рассмотрение алгоритма, который оформляет продажу. В листинге 1.3 отображены действия метода, вызываемого сервисом ICreateSaleService (который является Runner’ом). Преимущества описанного ранее паттерна BizRunner в том, что при обнаружении ошибки метод не возвращает значение сразу (кроме случаев, когда в этом есть смысл. Например, здесь если список товаров пуст, то имеет смысл сразу добавить ошибку и вернуть null, так как последующие проверки не добавят ошибок), а выполняет весь остальной код, собирая все остальные ошибки. Так же в данном паттерне плюсом будет упаковать логику взаимодействия с базой данных в отдельный слой чтобы сделать бизнес-логику более чистой и очевидной. Здесь доступ к базе данных осуществляется через readonly свойство класса SalesDbAccess dbAccess. Важный момент при использовании данного класса заключается в том, чтобы не вызывать метод context.SaveChanges(), так как он должен быть вызван внешним кодом (Runner’ом). В идеале, согласно SoC принципу, необходимо разделить доступ к базе данных для сложной и простой бизнес логики, вызывая метод SaveChanges() для простой логики и не вызывая для сложной (которая реализована с помощью BizRunner), что можно сделать, например, с помощью паттерна «Адаптер», разрешая для простой бизнес-логике выбирать между сохранить или нет, а для сложной – запретить данный выбор.

Продолжая говорить о листинге 1.3, метод Action в нем выполняет «сложную» валидацию, которая проверяет значения свойств (а не их корректность с точки зрения типов данных – валидация модели уже выполнена к этому моменту) и их целостность относительно предметной области. Так, например, если происходит оформление продажи с определенным списком товаров, то статус этих товаров должен быть «в корзине», они не должны быть логически удаленными из БД или товар не может быть оформлен на продажу если он уже относится к определенной продаже (не зарезервированной). При успешной валидации создается объект продажи, которому присваивается текущие дата и время, а объект dbAccess связывает эту продажу со списком товаров.

Листинг 1.3

public async Task<Guid?> Action(CreateSaleDto dto)

{

if (dto.GoodsForSale.Count == 0)

{

AddError("Список товаров пуст.");

return null;

}

foreach (var goodsUnit in dto.GoodsForSale)

{

ValidateGoodsUnit(goodsUnit);

goodsUnit.Status = GoodsStatus.AwaitingPayment;

}

if (HasErrors)

return null;

var sale = new Sale()

{

LocalSaleDate = DateTime.UtcNow,

Status = SaleStatus.YetNotPaid

};

dbAccess.CreateSale(sale, dto.GoodsForSale);

return sale.SaleId;

}

public void ValidateGoodsUnit(Goods goodsUnit)

{

if (goodsUnit.Status != GoodsStatus.InCart)

AddError("В корзину находится товар, статус которого не \"В корзине\"");

if (goodsUnit.SoftDeleted)

AddError("В корзину добавлен удаленный товар");

if (goodsUnit.Delivery?.LocalActualDeliveryDate is null || goodsUnit.ReceiptDate is null)

AddError("В корзину добавлен непоступивший на склад товар");

if (goodsUnit.Price <= 0)

AddError("В корзину добавлен товар с некорректной ценой. Цена не может быть меньше или равна 0");

// if goods unit has a sale, it must be returned. If goods unit doesn’t have sales, this loop won't be executed

foreach (var sale in goodsUnit.Sales)

if (sale.Status != SaleStatus.Returned)

AddError("В корзину добавлен товар, который входит в другую продажу и не может быть продан");

}

Метод ClearCart из листинга 1.1 является оберткой – он инкапсулирует способ очистки корзины, взаимодействие с которой сейчас реализовано через абстрактный класс CartViewerBaseController, который наследует от класса Controller (листинг 1.4). Данная реализация хранит в сессии строку с GUID всех единиц товара, находящихся сейчас в корзине, а так же их виды (GUID и вид разделены одним сепаратором, а их пары друг от друга – другим). В дальнейшем при модернизации корзины (например, добавление порядка, в котором товар был добавлен) необходимо будет изменить только класс CartViewerBaseController один раз и все его вызовы будут очищать корзину другим способом.

Взаимодействие (на самом деле, только его часть) с корзиной реализовано через наследование от абстрактный контроллер, а не через сервисный слой, так как ~~я не нашел другого способа~~  оно требует доступа к текущей сессии, что возможно только в слое приложения, а сервисный слой находится ниже слоя приложения, соответственно он «не знает» ни о классе HttpContext, ни об интерфейсе ISession, и, соответственно, он не может обрабатывать объекты их типов.

Листинг 1.4

public abstract class CartViewerBaseController : Controller

{

public string? GoodsIdsAndKindsInCart => HttpContext.Session.GetString(CommonNames.GoodsIdsInCartKey);

public string[] GoodsIdsAndKinds => GoodsIdsAndKindsInCart?.Split(CommonNames.GoodsIdSeparator, StringSplitOptions.RemoveEmptyEntries) ?? [];

public bool IsInCart(Guid goodsId) => GoodsIdsAndKindsInCart != null && GoodsIdsAndKinds!.Any(s => s.Contains(goodsId.ToString()));

public void SetNewCartValue(string newValue) => HttpContext.Session.SetString(CommonNames.GoodsIdsInCartKey, newValue);

public void ClearCart()

=> SetNewCartValue(string.Empty);

}

При успешной оплате сервер выполняет часть кода из листинга 1.5. Метод RegisterSaleAsSold (досл. зарегистрировать продажу как проданную) обрабатывает POST-запрос, который предполагается для отправки через AJAX, следующим образом: сначала производится попытка регистрации продажи с помощью сервиса saleService, вызываемого из IoC-контейнера с помощью атрибута [FromServices]. При любом исходе рассматриваемый метод возвращает текстовую информацию о результате HTTP-запроса: успех или неудача. В качестве альтернативы можно возвращать значение типа bool. При неудаче выполняется восстановление товара обратно в корзину, так как на данный момент в БД была сформирована новая продажа, включающая в себя весь товар из корзины, а из корзины этот товар был удален. Данный метод – первая ступень (слой) обработки HTTP-запроса, которая «направляет» полученные данные в следующий слой – сервисный (в данном случае представлен параметрами метода saleService и cartService).

Листинг 1.5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ContentResult> RegisterSaleAsSold(Guid saleId, SalePaidBy paidBy, [FromServices] IExistingSaleManagementService saleService, [FromServices] ICartService cartService, [FromServices] IGetGoodsUnitsRelatedToSaleService goodsService, [FromServices] IUpdateGoodsStatusService goodsStatusService, [FromServices] IMapKindOfGoodsService kindOfGoodsService)

{

string result;]

try

{

await saleService.RegisterSaleAsPaid(saleId, paidBy);

var goods = await goodsService.GetOrigGoodsUnitsRelatedToSale(saleId);

foreach(var goodsUnit in goods)

{

await goodsStatusService.UpdateGoodsStatus(goodsUnit.GoodsId, await kindOfGoodsService.GetGoodsKind(goodsUnit.GoodsId), GoodsStatus.Sold);

}

result = "Successfully registered";

}

catch

{

await RestoreCart(saleId, cartService);

await saleService.CancelSale(saleId);

result = "Failed to register";

}

return Content(result);

}

Следующий этап – метод RegisterSaleAsPaid сервиса с типом IExistingSaleManagementService (листинг 1.6). Сначала в данном методе у контекста базы данных запрашивается экземпляр продажи, идентификатор которой соответствует полученному параметру saleId, затем происходит проверка: оплачена ли продажа или нет. Если продажа уже оплачена, то генерируется исключение – методу был передан некорректный аргумент. Иными словами, происходит попытка зарегистрировать уже оплаченную продажу как снова оплаченную, что с точки зрения предметной области не имеет смысла.

Если продажа не была оплачена, то она помечается как оплаченная, после чего запись в контексте БД обновляется и изменения сохраняются. Если не выполнять метод Update(), то контекст не будет проверять изменения объекта sale, а если не выполнять метода SaveChanges() (в данном случае его асинхронную версию), то контекст не выполнит запросы к БД. Также важным замечанием относительно данного метода (а также всего приложения) является отсутствие обработки исключений в сервисном слое. Объясняется это тем, что исключения обрабатываются на стоящем выше уровне – контроллерами. ~~не уверен что это правильно но пока что это неплохо работает.~~  В данном случае, например, возможна ситуация, когда запись о продаже с полученным saleId не будет найдена в БД и метод SingleAsync() сгенерирует исключение.

Листинг 1.6

public async Task RegisterSaleAsPaid(Guid saleId, SalePaidBy paidBy)

{

var sale = await context.Sales.SingleAsync(sale => sale.SaleId == saleId);

if (sale.IsPaid)

throw new ArgumentException("attempt to register an already paid sale as a paid one");

sale.IsPaid = true;

sale.PaidBy = paidBy;

sale.Status = SaleStatus.Sold;

context.Update(sale);

await context.SaveChangesAsync();

}

Следующий методы для рассмотрения – RestoreCart (листинг 1.7) и CancelSale (листинг 1.8) – выполняются при генерации исключения в методе из листинга 1.5. Первый метод использует метод MoveGoodsBackToCart (переместить товар обратно в корзину) для вызова соответствующего метода у cartService, а затем устанавливает полученный список товаров как новое значение корзины.

Листинг 1.7

private async Task RestoreCart(Guid saleId, ICartService cartService)

{

string newCartContent = await cartService.MoveGoodsBackToCart(saleId);

SetNewCartValue(newCartContent);

}

Второй метод (CancelSale) находит необходимую продажу, и затем логически удаляет ее, следовательно, отмена покупки – это логическое ее удаление. В дальнейшем можно добавить функцию восстановления товара в корзине при проблемах с оплатой. Например, клиент выбрал оплату картой, но она оказалась заблокирована – продажа удаляется, а у клиента есть наличные средства, и он меняет способ оплаты. При данной реализации кассир в таком случае вынужден заполнять корзину снова.

Листинг 1.8

public async Task CancelSale(Guid saleId)

{

var sale = await context.Sales.SingleAsync(sale => sale.SaleId == saleId);

sale.SoftDeleted = true; // надо мягко удалить записи из соединяющих таблиц НЭП

await context.SaveChangesAsync();

}

Несмотря на то, что ранее было сказано, что корзина реализована через абстрактный котроллер, у записей товара в базе данных есть статус «в корзине», которое необходимо изменять соответствующе. Данная ответственность возложена на класс CartService (реализующий ICartService). В листинге 1.9 выполняется перемещение товара обратно в корзину, что происходит при безуспешной оплате в листинге 1.7. Сначала запрашивается список всех товаров, относящихся к продаже с идентификатором saleId, затем для каждого товара устанавливается статус «в корзине». Возвращаемое значение – строка с идентификаторами товаров и типами, которая затем установится как значение корзины для текущей сессии.

Листинг 1.9

public async Task<string> MoveGoodsBackToCart(Guid saleId)

{

var goods = await goodsRelatedToSaleService.GetOrigGoodsUnitsRelatedToSale(saleId);

foreach (var goodsUnit in goods)

{

goodsUnit.Status = GoodsStatus.InStock;

context.Update(goodsUnit);

}

context.SaveChanges();

return string.Join(GoodsIdSeparator, goods.Select(goodsUnit => new { goodsUnit.GoodsId, TypeName = goodsUnit.GetType().Name }));

}

Далее, в листинге 1.10, описывается способ получения списка всех товаров с помощью методов расширения Cast() и Concat() – метод Cast приводит объекты товаров разных типов к общему классу Goods, который наследуется ими всеми, а метод Concat объединяет все четыре коллекции товаров в одну.

Листинг 1.10

public async Task<List<Goods>> GetOrigGoodsUnitsRelatedToSale(Guid saleId)

{

var sale = await saleService.GetOriginalSale(saleId);

return sale.MusicalInstruments

.Cast<Goods>()

.Concat(sale.Accessories

.Cast<Goods>())

.Concat(sale.SheetMusicEditions

.Cast<Goods>())

.Concat(sale.AudioEquipmentUnits

.Cast<Goods>())

.ToList();

}

Несмотря на то, что все описанные выше методы используются в данном предложении во множественном числе, вместе они все еще составляют алгоритм работы одного и того же действия – оформления продажи. Тем не менее, данное действие разделено на два этапа: создание неоплаченной продажи и последующее регистрирование этой продажи как оплаченной (при отсутствии ошибок). В итоге это действие рассматривается как единое целое.

На рисунке 1.13 изображено дерево функций разрабатываемой ИС. Служебных функций всего три, основных – 13. На рисунке 1.14 – сценарий диалога с ИС для продавца.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Рисунок 1.13 – Дерево функций ИС

Изображение выглядит как текст, диаграмма, План, Параллельный

Автоматически созданное описание

Рисунок 1.14 – Сценарий диалога для продавца

На рисунке 1.15 изображена диаграмма классов для второго снизу слоя – доменной модели. Здесь важно заметить следующее: все виды товара наследуют от класса Goods, при этом у него есть свойство SpecificTypeId, а у всех классов товаров – свойство Specific Type. Суть в том, что класс Goods гарантирует, что у класса-наследника будет отношение со специфичным типом, для которого необходимо использовать собственный класс. Например, класс MusicalInstruments имеет навигационное свойство SpecificType типа MusicalInstrumentSpecificType, которое связано с внешним ключом SpecificTypeId. Перечисления в правом нижнем углу – это значения, которые могут принимать определенные атрибуты некоторых классов, которые есть на этой диаграмме. Интерфейс ISoftDeletable имеет свойство SoftDeleted.

Изображение выглядит как текст, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 1.15 – Диаграмма классов для слоя данных

Рисунок 1.16 демонстрирует наглядно слои приложения, которые в рамках языка C# являются проектами решения. Слой Common хранит:

- методы (иногда расширяющие), которые могут быть использованы во всех слоях, например, метод преобразования даты из локальной в UTC,

- общие названия, например, MusicalInstruments на Русском языке:

public const string MusicalInstruments = "Музыкальные инструменты";

- константные значения, например, максимальная длина описания товара: данная величина используется при валидации модели (при обработке метода POST, например) и при определении характеристик товара в слое данных,

- SQL-команды, которые создают триггеры, функции и т. д. Они могли быть и в слое данных, но они представляют собой строковые литералы и поэтому находятся в специальном классе SqlStatements,

- команды для командной строки для создания резервных копий базы данных. Тоже являются строковыми литералами.

Основная информация о слое данных изображена на диаграмме классов (рисунок 1.15). Слой DbAccess, как уже было ранее описано, изолирует доступ к БД для сложной логики. Слой ViewModels содержит модели представлений и некоторые DTO. Сервисный слой имеет свой тестовый проект с интеграционными тестами на СУБД SQLite. BasicAppConfiguration инкапсулирует создание построителя приложения с настройкой инъекции зависимостей, базы данных и общей конфигурации приложения (appSettings.json). Таким образом соблюдается принцип DRY (Don’t Repeat Yourself), настраивая приложение в одном месте, а затем используя эту настройку для приложений ASP.NET Core и WebApi. API, которое ранее не упоминалось, добавлено с целью реализации возможности резервации покупателями товара через приложения от third-party поставщиков. Данная функция позволит покупателям зарезервировать товар без необходимости связываться с магазином или посещать его.



Рисунок 1.16 – Диаграмма слоев приложения

Слой BusinessLogic манипулирует функциями слоя доступа к базе данных и независимым слоем, реализующим паттерн BizRunner – BizLogicBase. По факту этот слой хранит классы, которые реализуют интерфейс IBizAction, включая классы, чьи методы находятся в листингах 1.2 и 1.3. Приложение состоит из 12 слоев, около 17000 строк, что в итоге означает в среднем 1400 строки на слой, что звучит, как «процесс поддержания этого приложения относительно просто».

В результате, данный раздел подробно описал разработанную ИС со многих сторон, затрагивая технические тонкости и описывая принципы работы всего приложения. Написанная библиотека BizRunner может использоваться в дальнейшем при разработке других проектов на ASP.NET Core с использованием EntityFrameworkCore. Множественные диаграммы и схемы описывают поведение приложения наглядно, более доступно, а далее будут описаны тесты и их результаты. В приложении Б находится программный код определенных классов, исключая автоматически сгенерированные (например, миграции базы данных).

# 2 Тестирование элементов системы и формирование технологической среды

В качестве основной технологии тестирования выбрана C# библиотека Selenium с фреймворком для написания юнит-тестов MsTest. В совокупности они предоставляют тестировщику удобный способ автоматизации тест-кейсов для веб-сайта. С помощью Selenium можно выполнять действия на веб-сайте через браузер автоматически, такие как: нажатие кнопки, разворачивание окна, переключение вкладки, прокрутка страницы, ввод текста и множество других. Так же будут приведены обычные тесты сервисного слоя (с помощью MsTest). Обычно при написании тестов с помощью Selenium код читается достаточно хорошо, но его читабельность все еще можно улучшить с помощью паттерна Page Object Model Design

1. Обоснование выбора технологии тестирования, наиболее соответствующей особенностям разработанной системы, и ее характеристика.
2. Разработать тесты для спроектированного в ПО (объяснить выбор вида тестирования, почему применяются именно эти, а не другие тестовые комбинации пути прохождения вычислительного процесса).
3. Привести результаты тестирования в виде таблиц и экранных форм. Описание тестов, результатов тестирования и их анализ. В тестировании предусмотреть проверку выполнения требований по надежности и защите информации. Результаты тестирования рекомендуется оформлять в виде таблицы, в которой указываются тестовые воздействия и соответствующие реакции ИС. При целесообразности можно добавить экранные формы для иллюстрации результатов некоторых тестов.

# ЗАКЛЮЧЕНИЕ

Таким образом, все поставленные в работе задачи выполнены, и, соответственно, цель достигнута.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Lock A. ASP.NET Core in action : study man. – third edition. – Shelter Island : Manning, 2023, 983 p.
2. Smith J.P. Entity Framework Core in action : study man. – second edition. – Shelter Island : Manning, 2021, 626 p.
3. Giretti A. Coding clean, reliable, and safe REST APIs wit hASP.NET Core 8 : study man. – La Salle : Apress, 2023, 438 p.
4. Skeet J. C# in depth : study man. – fourth edition. – Greenwich : Manning, 2019, 528 p.
5. Thomas A.S. HTTP Essentials : study man. – New York : Wiley Computer Publishing, 2021, 336 p.
6. Barbettini N. The little ASP.NET Core book [electronic resource] // GitBook URL: <https://nbarbettini.gitbooks.io/little-asp-net-core-book/content/> (access date: 09.12.2024)
7. Helland A, Durano W.M., Chilberto J., Price E. ASP.NET Core 5 for beginners : study man. – Mumbai : Packt Publishing, 2020, 599 p.
8. Wenz C. ASP.NET Core security : study man. – Shelter Island : Manning, 2023, 623 p.
9. ф

# **ПРИЛОЖЕНИЕ А**

Экранные формы системы автоматизации

# **ПРИЛОЖЕНИЕ Б**

Программный код элементов системы автоматизации

|  |  |
| --- | --- |
| // Файл \WebApi\Extensions.cs  using DataLayer.SupportClasses;  namespace WebApi;  public static class Extensions  {  public static bool TryParse(this string kindOfGoods)  => Enum.TryParse(kindOfGoods, out KindOfGoods \_);  //public class KindOfGoodsConstraint<TEnum> : IRouteConstraint  // where TEnum : struct, Enum  //{  // public bool Match(HttpContext httpContext, IRouter route, string routeKey, RouteValueDictionary values, RouteDirection routeDirection)  // {  // var candidate = values[routeKey]?.ToString();  // return Enum.TryParse(candidate, true, out TEnum \_);  // }  //}  }  // Файл \WebApi\WebApiProgram.cs  using WebApi.RouteGroups;  using DbAccessLayer;  using ServiceLayer;  using DataLayer.Common;  using DataLayer.Models;  using Microsoft.AspNetCore.Identity;  using Microsoft.Extensions.DependencyInjection;  using BasicAppConfiguration;  var builder = BasicApp.CreateBuilder(args);  builder.Services.AddEndpointsApiExplorer();  builder.Services.AddSwaggerGen();  var app = builder.Build();  if (app.Environment.IsDevelopment())  {  app.UseSwagger();  app.UseSwaggerUI();  }  app.UseHttpsRedirection();  app.MapGroup("/api")  .MapApi();  app.Run();  // Файл \WebApi\RouteGroups\ApiRouteGroups.cs  using DataLayer.SupportClasses;  using Microsoft.AspNetCore.Mvc;  using ServiceLayer.GoodsServices;  using WebApi.Dto;  namespace WebApi.RouteGroups;  public static class ApiRouteGroups  {  // It forces server to use internet-connection, which is not written in the tt  // app.MapGet("/api/v1/goods/{kindOfGoods:KindOfGoods}/{goodsId:Guid}",  // ([FromRoute] Guid goodsId, [FromRoute] KindOfGoods kindOfGoods, [FromServices] IGetGoodsService service) =>  // {  // var goods = service.GetGoodsInfo(goodsId, kindOfGoods);  // if (service.)  //});  //app.MapPost("/api/reserve");  public static RouteGroupBuilder MapApi(this RouteGroupBuilder group)  {  group.MapGroup("/v1")  .RouteVersion1();  return group;  }  private static RouteGroupBuilder RouteVersion1(this RouteGroupBuilder group)  {  group.MapGet("/goods/{kindOfGoods}/{goodsId:Guid}",  ([FromRoute] Guid goodsId, [FromRoute] KindOfGoods kindOfGoods, [FromServices] IGetGoodsService service) =>  {  try  {  var goods = service.GetGoodsInfo(goodsId, kindOfGoods);  return Results.Json(goods);  }  catch  {  return Results.NotFound();  }  });  group.MapPost("/goods/reserve", ([FromBody] List<GoodsUnitForReservation> goods) =>  {  });  return group;  }  }  // Файл \WebApi\Dto\GoodsUnitForReservation.cs  using DataLayer.SupportClasses;  namespace WebApi.Dto;  public record class GoodsUnitForReservation(Guid goodsId, KindOfGoods kindOfGoods);  // Файл \ViewModelsLayer\GlobalUsings.cs  global using Common;  // Файл \ViewModelsLayer\Stock\AddGoodsToWarehouseDto.cs  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using ViewModelsLayer.Stock.CustomAttributes;  namespace ViewModelsLayer.Stock;  [DoesNotContainHtmlTags]  public record class AddGoodsToWarehouseDto(  [RussianRequired("Наименование")]  [RussianStringLength(ConstValues.GoodsNameMaxLength, "Наименование")]  string Name,  [RequiredWhenNewSpecificTypeAddingIsEqualTo(false, "Тип")]  [RussianStringLength(ConstValues.GoodsSpecificTypeMaxLength, "Тип")]  string? SpecificType,  [RussianRequired("Выполнить добавление нового типа")]  bool NewSpecificTypeIsBeingAdded,  [RequiredWhenNewSpecificTypeAddingIsEqualTo(true, "Название нового типа")]  [RussianStringLength(ConstValues.GoodsSpecificTypeMaxLength, "Название нового типа")]  string? NewSpecificType,  [RussianRequired("Цена")]  [Range(ConstValues.GoodsPriceMinValue, ConstValues.GoodsPriceMaxValue, ErrorMessage = "Значение поля \"Цена\" должно быть в диапазоне от {1} до {2}")]  int? Price,  [RussianRequired("Статус")]  GoodsStatus Status,  [RussianStringLength(ConstValues.GoodsDescriptionMaxLength, "Описание")]  string? Description,  [RussianRequired("Количество")]  [Range(ConstValues.GoodsPriceMinValue, ConstValues.GoodsPriceMaxValue, ErrorMessage = "Значение поля \"Количество\" должно быть в диапазоне от {1} до {2}")]  int? NumberOfUnits,  [Required(ErrorMessage = "Отсутствуют специфичные данные")]  GoodsKindSpecificDataDto GoodsKindSpecificDataDto);  // Файл \ViewModelsLayer\Stock\AddGoodsToWarehouseModel.cs  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ViewModelsLayer.Stock;  public record class AddGoodsToWarehouseModel(AddGoodsToWarehouseDto AddGoodsToWarehouseDto, Dictionary<KindOfGoods, List<string>> SpecificTypes, List<ValidationResult> Errors);  // Файл \ViewModelsLayer\Stock\GoodsKindSpecificDataDto.cs  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.Diagnostics.CodeAnalysis;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using ViewModelsLayer.Stock.CustomAttributes;  namespace ViewModelsLayer.Stock;  //[DoesNotContainHtmlTags]  public record class GoodsKindSpecificDataDto  (  [RussianRequired("Вид")]  KindOfGoods KindOfGoods,  // Accessory  [RussianStringLength(ConstValues.AccessoryColorMaxLength, "Цвет")]  [RequiredWhenKindOfGoodsIs(KindOfGoods.Accessories, "Цвет")]  // TODO to cyrillic  string? Color,  [RussianStringLength(ConstValues.AccessorySizeMaxLength, "Размер")]  [RequiredWhenKindOfGoodsIs(KindOfGoods.Accessories, "Размер")]  string? Size,  // Sheet music edition  [RussianStringLength(ConstValues.SheetMusicEditionAuthorMaxLength, "Автор")]  string? Author,  // Musical instrument  [RequiredWhenKindOfGoodsIs(KindOfGoods.MusicalInstruments, "Год выпуска")]  [RequiredWhenKindOfGoodsIs(KindOfGoods.SheetMusicEditions, "Год выпуска")]  int? ReleaseYear,  [RequiredWhenKindOfGoodsIs(KindOfGoods.MusicalInstruments, "Тип производителя")]  ManufacturerType? ManufacturerType,  [RequiredWhenKindOfGoodsIs(KindOfGoods.MusicalInstruments, "Производитель")]  [RussianStringLength(ConstValues.MusicalInstrumentManufacturerMaxLength, "Производитель")]  string? Manufacturer  );  // Файл \ViewModelsLayer\Stock\CustomAttributes\DoesNotContainHtmlTagsAttribute.cs  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Reflection;  using System.Text;  using System.Text.RegularExpressions;  using System.Threading.Tasks;  namespace ViewModelsLayer.Stock.CustomAttributes;  [AttributeUsage(AttributeTargets.Class)]  public class DoesNotContainHtmlTagsAttribute : ValidationAttribute  {  protected override ValidationResult? IsValid(object? value, ValidationContext validationContext)  {  var obj = value;  var objType = obj!.GetType();  var properties = objType.GetProperties()  .Where(property => property.PropertyType == typeof(string))  .ToList();  foreach (var property in properties)  {  var propertyValue = (string?)property.GetValue(obj);  if (propertyValue is null)  continue;  var regexp = new Regex("<[^>]\*>");  if (regexp.IsMatch(propertyValue))  return new ValidationResult(string.Format(CommonNames.HtmlTagsViolationMessageRu, property.Name));  }  return ValidationResult.Success;  }  }  // Файл \ViewModelsLayer\Stock\CustomAttributes\RequiredWhenKindOfGoodsIs.cs  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ViewModelsLayer.Stock.CustomAttributes;  [AttributeUsage(AttributeTargets.Parameter, AllowMultiple = true)]  public class RequiredWhenKindOfGoodsIsAttribute : RequiredAttribute  {  private readonly KindOfGoods targetKindOfGoods;  public RequiredWhenKindOfGoodsIsAttribute(KindOfGoods kindOfGoods, string parameterName)  {  this.targetKindOfGoods = kindOfGoods;  ErrorMessage = string.Format(CommonNames.FieldIsRequiredMessageRu, parameterName);  }  protected override ValidationResult? IsValid(object? value, ValidationContext validationContext)  {  var dto = (GoodsKindSpecificDataDto)validationContext.ObjectInstance;  if (dto.KindOfGoods == targetKindOfGoods)  // TODO difference?  return base.IsValid(value, validationContext);  else  return ValidationResult.Success;  }  }  // Файл \ViewModelsLayer\Stock\CustomAttributes\RequiredWhenNewSpecificTypeAddingIsEqualToAttribute.cs  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ViewModelsLayer.Stock.CustomAttributes;  public class RequiredWhenNewSpecificTypeAddingIsEqualToAttribute : RequiredAttribute  {  private readonly bool newSpecificTypeAdding;  public RequiredWhenNewSpecificTypeAddingIsEqualToAttribute(bool newSpecificTypeAdding, string propertyName)  {  ErrorMessage = string.Format(CommonNames.FieldIsRequiredMessageRu, propertyName);  this.newSpecificTypeAdding = newSpecificTypeAdding;  }  protected override ValidationResult? IsValid(object? value, ValidationContext validationContext)  {  AddGoodsToWarehouseDto dto = (AddGoodsToWarehouseDto)validationContext.ObjectInstance;  if (dto.NewSpecificTypeIsBeingAdded == newSpecificTypeAdding)  // default check  // TODO difference?  return base.IsValid(value)  ? ValidationResult.Success  : new ValidationResult(ErrorMessage);  else  // ignore  return ValidationResult.Success;  }  }  // Файл \ViewModelsLayer\Stock\CustomAttributes\RussianRequiredAttribute.cs  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ViewModelsLayer.Stock.CustomAttributes;  [AttributeUsage(AttributeTargets.Parameter)]  public class RussianRequiredAttribute : RequiredAttribute  {  public RussianRequiredAttribute(string parameterName)  {  ErrorMessage = string.Format(CommonNames.FieldIsRequiredMessageRu, parameterName);  }  //public override bool IsValid(object? value)  //=> value is not null;  }  // Файл \ViewModelsLayer\Stock\CustomAttributes\RussianStringLengthAttribute.cs  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ViewModelsLayer.Stock.CustomAttributes;  [AttributeUsage(AttributeTargets.Parameter)]  public class RussianStringLengthAttribute : StringLengthAttribute  {  public RussianStringLengthAttribute(int maximumLength, string fieldName) : base(maximumLength)  {  ErrorMessage = string.Format(CommonNames.MaxLengthViolationMessageRu, fieldName, maximumLength);  }  }  // Файл \ViewModelsLayer\Sales\SaleErrorModel.cs  using System;  using System.Collections.Generic;  using System.Collections.Immutable;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ViewModelsLayer.Sales;  public class SaleErrorModel(IImmutableList<ValidationResult> Errors);  // Файл \ViewModelsLayer\Sales\SaleSearchDto.cs  using DataLayer.Models;  using DataLayer.SupportClasses;  namespace ViewModelsLayer.Sales;  public record class SaleSearchDto(Guid SalesId, DateTime? ReservationDate, DateTime? SaleDate, DateTime? ReturningDate, SaleStatus Status, int Total, SalePaidBy PaidBy, List<string> BriefGoodsDescriptions);  // Файл \ViewModelsLayer\Sales\SalesFilterOptions.cs  using DataLayer.SupportClasses;  namespace ViewModelsLayer.Sales;  public record class SalesFilterOptions(DateTime? MinSaleDate, DateTime? MaxSaleDate, DateTime? MinReservationDate, DateTime? MaxReservationDate, DateTime? MinReturningDate, DateTime? MaxReturningDate, SaleStatus? Status, SalePaidBy? PaidBy);  // Файл \ViewModelsLayer\Sales\SalesOrderBy.cs  namespace ViewModelsLayer.Sales;  public enum SalesOrderBy  {  Relevance,  SaleDate,  ReservationDate,  ReturningDate,  GoodsUnitsCount  }  // Файл \ViewModelsLayer\Sales\SalesOrderByOptions.cs  namespace ViewModelsLayer.Sales;  public record class SalesOrderByOptions(SalesOrderBy OrderBy, bool AscendingOrder);  // Файл \ViewModelsLayer\Sales\SalesSearchModel.cs  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ViewModelsLayer.Sales;  public record class SalesSearchModel(string ResearchText, List<SaleSearchDto> Sales, int ResultsCount, SalesFilterOptions Filter, SalesOrderByOptions OrderBy);  // Файл \ViewModelsLayer\Goods\GoodsFilterOptions.cs  using DataLayer.SupportClasses;  namespace ViewModelsLayer.Goods;  /// <summary>  /// It filters goods, where <paramref name="FilterValue"/> is a value or comma-separated values (depends upon <paramref name="GoodsFilter"/> value).  /// Pass <paramref="FilterValue"/> as something like a string.Join([nameof(type1), nameof(typen)], ',').  /// </summary>  /// <param name="GoodsFilter"></param>  /// <param name="FilterValue"></param>  public record class GoodsFilterOptions(int? MinPrice, int? MaxPrice, DateTimeOffset? FromReceiptDate, DateTimeOffset? ToReceiptDate, KindOfGoods KindOfGoods, GoodsStatus Status);  // Файл \ViewModelsLayer\Goods\GoodsOrderBy.cs  namespace ViewModelsLayer.Goods;  public enum GoodsOrderBy  {  /// <summary>  /// Random sorting, strictly speaking.  /// </summary>  Relevance,  Price,  ReceiptDate,  }  // Файл \ViewModelsLayer\Goods\GoodsOrderByOptions.cs  namespace ViewModelsLayer.Goods;  public record class GoodsOrderByOptions(GoodsOrderBy OrderBy, bool AscendingOrder);  // Файл \ViewModelsLayer\Goods\GoodsSearchModel.cs  namespace ViewModelsLayer.Goods;  public class GoodsSearchModel  {  public string ResearchText { get; set; }  public List<GoodsUnitSearchDto> GoodsUnitModels { get; set; }  public GoodsOrderByOptions OrderBy { get; set; }  public GoodsFilterOptions Filter { get; set; }  #warning probably it's redundant thing here because view can work out this value by its own  public int ResultsCount { get; set; }  }  // Файл \ViewModelsLayer\Goods\GoodsUnitModel.cs  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ViewModelsLayer.Goods;  public record class GoodsUnitModel(Guid Guid, KindOfGoods KindOfGoods, string Name, int Price, GoodsStatus Status, string? Description);  // Файл \ViewModelsLayer\Goods\GoodsUnitSearchDto.cs  using DataLayer.SupportClasses;  namespace ViewModelsLayer.Goods;  public class GoodsUnitSearchDto  {  public string Id { get; set; }  public KindOfGoods KindOfGoods { get; set; }  public string Type { get; set; }  public string Name { get; set; }  public string? Description { get; set; }  public int Price { get; set; }  public GoodsStatus Status { get; set; }  public bool IsInCart { get; set; }  }  // Файл \ViewModelsLayer\Admin\BackupsModel.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ViewModelsLayer.Admin;  public record class BackupsModel(Dictionary<DateTime, string> BackupsDict);  // Файл \ServiceLayerTests\GlobalUsings.cs  global using Microsoft.VisualStudio.TestTools.UnitTesting;  global using Common;  // Файл \ServiceLayerTests\UnitTest1.cs  namespace ServiceLayerTests  {  [TestClass]  public class UnitTest1  {  [TestMethod]  public void TestMethod1()  {  }  }  }  // Файл \ServiceLayer\GlobalUsings.cs  global using Common;  // Файл \ServiceLayer\ServicesRegisterExtension.cs  using Microsoft.Extensions.DependencyInjection;  using NetCore.AutoRegisterDi;  namespace ServiceLayer;  public static class ServicesRegisterExtension  {  public static void RegisterServiceLayer(this IServiceCollection services)  {  services.RegisterAssemblyPublicNonGenericClasses()  .Where(@class => @class.Name.EndsWith("Service"))  .AsPublicImplementedInterfaces();  }  }  // Файл \ServiceLayer\StockServices\AddNewGoodsService.cs  using BizLogicBase.Validation;  using DataLayer.Common;  using DataLayer.Models;  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.Collections.Immutable;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using ViewModelsLayer.Stock;  namespace ServiceLayer.StockServices;  public interface IAddNewGoodsService : IErrorAdder  {  Task<List<Goods>> AddNewGoods(AddGoodsToWarehouseDto dto);  }  public class AddNewGoodsService(MusicalShopDbContext context, ISpecificTypeService specificTypesService) : ErrorAdder, IAddNewGoodsService  {  public override IImmutableList<ValidationResult> Errors  => base.Errors.Concat(specificTypesService.Errors).ToImmutableList();  public async Task<List<Goods>> AddNewGoods(AddGoodsToWarehouseDto dto)  {  SpecificType specificType;  try  {  specificType = dto.NewSpecificTypeIsBeingAdded  ? await specificTypesService.CreateSpecificType(dto.NewSpecificType, dto.GoodsKindSpecificDataDto.KindOfGoods)  : await specificTypesService.GetSpecificType(dto.SpecificType, dto.GoodsKindSpecificDataDto.KindOfGoods);  }  catch  {  AddError("Некорректное значение типа товара");  return null;  }  DateTimeOffset? receiptDate = default;  switch (dto.Status)  {  case GoodsStatus.InStock:  receiptDate = DateTimeOffset.UtcNow;  break;  case GoodsStatus.AwaitingDelivery:  receiptDate = null;  break;  default:  AddError("Вы не можете выбрать данный статус");  break;  }  if (HasErrors)  return [];  // Validated successfully  var result = new List<Goods>();  for (int i = 0; i < dto.NumberOfUnits; i++)  {  Goods goods = dto.GoodsKindSpecificDataDto.KindOfGoods switch  {  // TODO refactoring  KindOfGoods.MusicalInstruments => new MusicalInstrument  {  ReleaseYear = (int)dto.GoodsKindSpecificDataDto.ReleaseYear,  ManufacturerType = (ManufacturerType)dto.GoodsKindSpecificDataDto.ManufacturerType,  Manufacturer = dto.GoodsKindSpecificDataDto.Manufacturer  },  KindOfGoods.Accessories => new Accessory  {  Color = dto.GoodsKindSpecificDataDto.Color,  Size = dto.GoodsKindSpecificDataDto.Size  },  KindOfGoods.AudioEquipmentUnits => new AudioEquipmentUnit(),  KindOfGoods.SheetMusicEditions => new SheetMusicEdition  {  ReleaseYear = (int)dto.GoodsKindSpecificDataDto.ReleaseYear,  Author = dto.GoodsKindSpecificDataDto.Author  },  \_ => throw new Exception()  };  goods.Description = dto.Description;  goods.Name = dto.Name;  goods.Price = (int)dto.Price;  goods.ReceiptDate = receiptDate;  goods.SoftDeleted = false;  goods.Status = dto.Status;  goods.SpecificTypeId = specificType.SpecificTypeId;  // TODO specific type  //goods.SpecificType = specificTypeEntity;  result.Add(goods);  await context.AddAsync(goods);  }  await context.SaveChangesAsync();  return result;  }  }  // Файл \ServiceLayer\StockServices\SpecificTypeService.cs  using BizLogicBase.Validation;  using DataLayer.Common;  using DataLayer.SupportClasses;  using Microsoft.EntityFrameworkCore;  using ServiceLayer.GoodsServices;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ServiceLayer.StockServices;  public interface ISpecificTypeService : IErrorAdder  {  Task<SpecificType> GetSpecificType(string specificType, KindOfGoods kindOfGoods);  // TODO probably merge in one method because the first one is just specificTypes.Select(...)  Task<List<string>> ListTheSpecificTypes(KindOfGoods kindOfGoods);  Task<Dictionary<KindOfGoods, List<string>>> GetAllSpecificTypes();  Task<SpecificType> CreateSpecificType(string? newSpecificType, KindOfGoods kindOfGoods);  }  public class SpecificTypeService(MusicalShopDbContext context, IMapKindOfGoodsService kindOfGoodsMapper) : ErrorAdder, ISpecificTypeService  {  public async Task<SpecificType> GetSpecificType(string specificType, KindOfGoods kindOfGoods)  => kindOfGoodsMapper.MapToSpecificTypes(kindOfGoods).Single(st => st.Name.ToLower() == specificType.ToLower());  public async Task<List<string>> ListTheSpecificTypes(KindOfGoods kindOfGoods)  => await kindOfGoodsMapper.MapToSpecificTypes(kindOfGoods)  .Select(st => st.Name)  .ToListAsync();  public async Task<Dictionary<KindOfGoods, List<string>>> GetAllSpecificTypes()  {  Dictionary<KindOfGoods, List<string>> specificTypes = new();  foreach (var kindOfGoods in Enum.GetValues<KindOfGoods>())  specificTypes[kindOfGoods] = await ListTheSpecificTypes(kindOfGoods);  return specificTypes;  }  // TODO maybe create or update?  public async Task<SpecificType> CreateSpecificType(string newSpecificType, KindOfGoods kindOfGoods)  {  var correspondingSpecificTypes = kindOfGoodsMapper.MapToSpecificTypes(kindOfGoods);  // TODO should i listen to ide's advice and use another comparison here?  if (await correspondingSpecificTypes  .AnyAsync(specificType => specificType.Name.ToLower() == newSpecificType.ToLower()))  {  AddError($"Тип \"{newSpecificType}\" уже существует");  return null;  }  SpecificType newSpecificTypeEntity = kindOfGoodsMapper.CreateNewSpecificType(kindOfGoods);  newSpecificTypeEntity.Name = newSpecificType;  context.Add(newSpecificTypeEntity);  context.SaveChanges();  return newSpecificTypeEntity;  }  }  // Файл \ServiceLayer\SalesServices\CartService.cs  using DataLayer.Common;  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  using ServiceLayer.GoodsServices;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Net.Http;  using System.Text;  using System.Threading.Tasks;  using static Common.CommonNames;  namespace ServiceLayer.SalesServices;  public interface ICartService  {  Task<string> MoveGoodsBackToCart(Guid saleId);  Task<string?> AddToOrRemoveFromCart(Guid goodsId, bool isInCart, string? goodsIdsAndKinds);  Task<List<Goods>> GetGoodsFromCart(string[] cartContent);  // Raw logic methods  string CutGoodsId(string goodsIdAndKind);  KindOfGoods CutGoodsKind(string goodsIdAndKind);  }  #warning so where is services for adding to and removing from the cart  #warning UPD: what did i mean?  #warning UPD2: i meant that the cart is supposed to have methods like AddGoodsUnitInCart() and RemoveGoodsUnitFromCart()  public class CartService(MusicalShopDbContext context, IGetGoodsUnitsRelatedToSaleService goodsRelatedToSaleService, IGetGoodsService getGoodsService, IUpdateGoodsStatusService updateGoodsStatusService, IMapKindOfGoodsService kindOfGoodsMapper) : ICartService  {  public string CutGoodsId(string goodsIdAndKind) => goodsIdAndKind.Split(CommonNames.GoodsIdAndKindSeparator)[0];  public KindOfGoods CutGoodsKind(string goodsIdAndKind) => Enum.Parse<KindOfGoods>(goodsIdAndKind.Split(CommonNames.GoodsIdAndKindSeparator)[1])!;  public async Task<string> MoveGoodsBackToCart(Guid saleId)  {  var goods = await goodsRelatedToSaleService.GetOrigGoodsUnitsRelatedToSale(saleId);  foreach (var goodsUnit in goods)  {  goodsUnit.Status = GoodsStatus.InCart;  context.Update(goodsUnit);  }  context.SaveChanges();  // TODO change to kind of goods  return string.Join(GoodsIdSeparator, goods.Select(goodsUnit => new { goodsUnit.GoodsId, TypeName = goodsUnit.GetType().Name }));  }  // dirty stuff  public async Task<string> AddToOrRemoveFromCart(Guid goodsId, bool isInCart, string? goodsIdsAndKinds)  {  List<string> goodsIdsAndKindsList = goodsIdsAndKinds?.Split(GoodsIdSeparator, StringSplitOptions.RemoveEmptyEntries)  ?.ToList() ?? [];  var kindOfGoods = await kindOfGoodsMapper.GetGoodsKind(goodsId);  #warning is it validation? did i try to validate whether goods unit exists in db?  var goods = await getGoodsService.GetGoodsInfo(goodsId, kindOfGoods);  if (isInCart)  RemoveFromCart(goodsId, goodsIdsAndKindsList);  else  AddInCart(goodsId, kindOfGoods, goodsIdsAndKindsList);  // TODO check goods status  // what if this goods unit is already sold?  //if (goodsUnit.Status != GoodsStatus.InCart)  //{  // AddError("В корзине обнаружен товар, статус которого не \"В корзине\"");  //}  await updateGoodsStatusService.UpdateGoodsStatus(goods.GoodsId, kindOfGoods, isInCart ? GoodsStatus.InStock : GoodsStatus.InCart);  return string.Join(GoodsIdSeparator, goodsIdsAndKindsList);  }  private List<string> RemoveFromCart(Guid goodsId, List<string> goodsIdsAndKindsList)  {  foreach (var goodsIdAndKind in goodsIdsAndKindsList)  {  if (goodsIdAndKind.Contains(goodsId.ToString()))  {  goodsIdsAndKindsList.Remove(goodsIdAndKind);  return goodsIdsAndKindsList;  }  }  throw new Exception("removing from cart error");  }  private List<string> AddInCart(Guid goodsId, KindOfGoods kindOfGoods, List<string> goodsIdsAndKindsList)  {  foreach (var goodsIdAndKind in goodsIdsAndKindsList)  {  if (goodsIdAndKind.Contains(goodsId.ToString()))  throw new Exception("this goods is already in cart");  }  goodsIdsAndKindsList.Add($"{goodsId}{GoodsIdAndKindSeparator}{kindOfGoods}");  return goodsIdsAndKindsList;  }  public async Task<List<Goods>> GetGoodsFromCart(string[] cartContent)  {  List<Goods> goodsList = [];  foreach (var goodsIdAndType in cartContent)  {  goodsList.Add(await getGoodsService.GetGoodsInfo(Guid.Parse(CutGoodsId(goodsIdAndType)), CutGoodsKind(goodsIdAndType)));  }  return goodsList;  }  }  // Файл \ServiceLayer\SalesServices\CreateSaleService.cs  using BizLogicBase.Common;  using BizLogicBase.Validation;  using BusinessLogicLayer.Sales;  using BusinessLogicLayer.Sales.Dto;  using DataLayer.Common;  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  using DbAccessLayer;  using ServiceLayer.GoodsServices;  using System;  using System.Collections.Generic;  using System.Collections.Immutable;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ServiceLayer.SalesServices;  public interface ICreateSaleService : IErrorStorage  {  Task<Guid?> CreateSaleAsNotPaid(List<Goods> goods);  }  /// <summary>  /// This service is separated from SaleManagementService because it uses complicated business logic  /// </summary>  /// <param name="context"></param>  public class CreateSaleService(MusicalShopDbContext context) : ErrorStorage, ICreateSaleService  {  private readonly RunnerWriteDb<CreateSaleDto, Task<Guid?>> runner = new(context, new CreateSaleAsNotPaidAction(new SalesDbAccess(context)));  public override IImmutableList<ValidationResult> Errors => runner.Errors;  // interesting thing about DRN (don't return null), here it is an architectural decision. anyway either HasErrors is true and then returned value won't be used or HasErrors is false, hence the return value is not null  public async Task<Guid?> CreateSaleAsNotPaid(List<Goods> goodsList)  => await runner.Run(new CreateSaleDto(goodsList));  }  // Файл \ServiceLayer\SalesServices\ExistingSaleManagementService.cs  using DataLayer.Common;  using DataLayer.Models;  using DataLayer.SupportClasses;  using Microsoft.EntityFrameworkCore;  using ServiceLayer.GoodsServices;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ServiceLayer.SalesServices;  public interface IExistingSaleManagementService  {  Task RegisterSaleAsPaid(Guid saleId, SalePaidBy paidBy);  Task CancelSale(Guid saleId);  }  public class ExistingSaleManagementService(MusicalShopDbContext context) : IExistingSaleManagementService  {  public async Task CancelSale(Guid saleId)  {  var sale = await context.Sales.SingleAsync(sale => sale.SaleId == saleId);  sale.SoftDeleted = true;  // TODO also soft delte all goods\_sale linking tables  await context.SaveChangesAsync();  }  public async Task RegisterSaleAsPaid(Guid saleId, SalePaidBy paidBy)  {  var sale = await context.Sales.SingleAsync(sale => sale.SaleId == saleId);  if (sale.IsPaid)  throw new ArgumentException("attempt to register an already paid sale as a paid one");  // TODO via service??  sale.IsPaid = true;  sale.PaidBy = paidBy;  sale.Status = SaleStatus.Sold;  context.Update(sale);  await context.SaveChangesAsync();  }  }  // Файл \ServiceLayer\SalesServices\GetRelevantSalesService.cs  using DataLayer.Common;  using DataLayer.Models;  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  using Humanizer;  using Microsoft.EntityFrameworkCore;  using ServiceLayer.GoodsServices;  using ServiceLayer.SalesServices.QueryObjects;  using System;  using System.Collections.Generic;  using System.IO;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using ViewModelsLayer.Sales;  namespace ServiceLayer.SalesServices;  public interface IGetRelevantSalesService  {  Task<List<SaleSearchDto>> GetRelevantSales(string researchText, SalesFilterOptions filterOptions, SalesOrderByOptions orderByOptions);  }  public class GetRelevantSalesService(MusicalShopDbContext context, IGetSaleService getSaleService, IGetGoodsUnitsRelatedToSaleService goodsService) : IGetRelevantSalesService  {  public async Task<List<SaleSearchDto>> GetRelevantSales(string researchText, SalesFilterOptions filterOptions, SalesOrderByOptions orderByOptions)  {  List<Guid> saleIds = [.. context.SalesView  .AsNoTracking()  #warning add researchTextFilter  //.Where()  .FilterBy(filterOptions)  .OrderBy(orderByOptions)  .Select(sale => sale.SaleId)];  List<SaleSearchDto> result = [];  SaleView saleView;  foreach (Guid saleId in saleIds)  {  saleView = await getSaleService.GetSaleView(saleId);  List<Goods> relatedGoods = await goodsService.GetOrigGoodsUnitsRelatedToSale(saleId);  #warning not sure about it  List<string> briefGoodsDescriptions = [];  foreach(Goods goodsUnit in relatedGoods)  {  briefGoodsDescriptions.Add($"{goodsUnit.Name} {goodsUnit.Description}");  }  SaleSearchDto dto = new(saleView.SaleId, saleView.LocalReservationDate, saleView.LocalSaleDate, saleView.LocalReturningDate, saleView.Status, saleView.Total, saleView.PaidBy, briefGoodsDescriptions);  result.Add(dto);  }  return result;  }  }  // Файл \ServiceLayer\SalesServices\GetSaleService.cs  using DataLayer.Common;  using DataLayer.Models;  using Microsoft.EntityFrameworkCore;  using System;  using System.Collections.Generic;  using System.Collections.Immutable;  using System.Collections.ObjectModel;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using ViewModelsLayer.Sales;  namespace ServiceLayer.SalesServices;  public interface IGetSaleService  {  Task<Sale> GetOriginalSale(Guid saleId);  Task<SaleView> GetSaleView(Guid saleId);  #warning under question  //Task<SaleSearchDto> GetReadableSale(Guid saleId);  }  public class GetSaleService(MusicalShopDbContext context) : IGetSaleService  {  public async Task<Sale> GetOriginalSale(Guid saleId)  {  #warning is there any point to use find here?  return /\*await context.FindAsync<Sale>(saleId) ??\*/  await context.Sales.Include(s => s.MusicalInstruments)  .Include(s => s.Accessories)  .Include(s => s.SheetMusicEditions)  .Include(s => s.AudioEquipmentUnits)  .SingleAsync(s => s.SaleId == saleId);  }  public async Task<SaleView> GetSaleView(Guid saleId)  {  var saleView = await context.SalesView.SingleAsync(saleView => saleView.SaleId == saleId);  var sale = await GetOriginalSale(saleId);  //#warning whaaat's going on here?  // gotcha: saleView doesn't have relationships in database, so assign them manually  saleView.MusicalInstruments = [.. sale.MusicalInstruments];  saleView.Accessories = [.. sale.Accessories];  saleView.AudioEquipmentUnits = [.. sale.AudioEquipmentUnits];  saleView.SheetMusicEditions = [.. sale.SheetMusicEditions];  return saleView;  }  #warning under question  //public async Task<SaleSearchDto> GetReadableSale(Guid saleId)  //{  // var sale = await GetViewSale(saleId);  // return new SaleSearchDto(sale.SaleId, sale.LocalDate, sale.Status, sale.Total, sale.PaidBy);  //}  }  // Файл \ServiceLayer\SalesServices\QueryObjects\QueryObjectExtensions.cs  using DataLayer.Models;  using System;  using System.Collections.Generic;  using System.IO;  using System.Linq;  using System.Linq.Expressions;  using System.Reflection;  using System.Text;  using System.Threading.Tasks;  using ViewModelsLayer.Sales;  namespace ServiceLayer.SalesServices.QueryObjects;  public static class QueryObjectExtensions  {  public static IQueryable<T> Page<T>(this IQueryable<T> query, int pageNumber, int pageSize)  {  if (pageSize <= 0)  throw new ArgumentOutOfRangeException(nameof(pageSize), "pageSize cannot be zero or less than zero.");  if (pageNumber <= 0)  throw new ArgumentOutOfRangeException(nameof(pageSize), "pageNum cannot be zero or less than zero.");  if (pageNumber != 1)  query = query.Skip((pageNumber - 1) \* pageSize);  return query.Take(pageSize);  }  public static IQueryable<SaleView> FilterBy(this IQueryable<SaleView> query, SalesFilterOptions filterOptions)  {  string[] types = { "Sale", "Reservation", "Returning" };  string[] ranges = { "Min", "Max" };  foreach(string type in types)  foreach(string range in ranges)  {  PropertyInfo someDate = typeof(SalesFilterOptions).GetProperty($"{range}{type}Date")!;  var currentDate = (DateTime?)someDate.GetValue(filterOptions);  if (currentDate != null)  {  DateTimeOffset? newDateTimeOffset = currentDate.LocalToUniversal();  var parameter = Expression.Parameter(typeof(SaleView));  var property = Expression.Property(parameter, $"{type}Date");  var constantExpression = Expression.Constant(newDateTimeOffset, typeof(DateTimeOffset?));  var equalityExpression = Expression.GreaterThanOrEqual(property, constantExpression);  var predicate = (Expression<Func<SaleView, bool>>)Expression.Lambda(equalityExpression, parameter);  query = query.Where(predicate);    }  }  if (filterOptions.Status is not null)  query = query.Where(sale => sale.Status == filterOptions.Status);  if (filterOptions.PaidBy is not null)  query = query.Where(sale => sale.PaidBy == filterOptions.PaidBy);  return query;  }  public static IQueryable<SaleView> OrderBy(this IQueryable<SaleView> query, SalesOrderByOptions orderByOptions)  {  Expression<Func<SaleView, object>> selector = orderByOptions.OrderBy switch  {  SalesOrderBy.Relevance => sale => sale.SaleId,  SalesOrderBy.SaleDate => sale => sale.SaleDate,  SalesOrderBy.ReservationDate => sale => sale.ReservationDate,  SalesOrderBy.ReturningDate => sale => sale.ReturningDate,  SalesOrderBy.GoodsUnitsCount => sale => sale.GoodsUnitsCount,  \_ => throw new Exception()  };  return orderByOptions.AscendingOrder ? query.OrderBy(selector) : query.OrderByDescending(selector);  }  }  // Файл \ServiceLayer\GoodsServices\GetGoodsService.cs  using DataLayer.Common;  using DataLayer.Models;  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  using Microsoft.EntityFrameworkCore;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Reflection;  using System.Text;  using System.Threading.Tasks;  using ViewModelsLayer.Goods;  using static Common.ConstValues;  namespace ServiceLayer.GoodsServices;  public interface IGetGoodsService  {  Task<Goods> GetGoodsInfo(Guid goodsId, KindOfGoods kindOfGoods);  Task<GoodsUnitSearchDto> GetReadableGoodsInfo(Guid goodsId, KindOfGoods kindOfGoods);  //Task<Type> GetGoodsType(Guid goodsId);  }  public class GetGoodsService(MusicalShopDbContext context, IMapKindOfGoodsService kindOfGoodsMapper) : IGetGoodsService  {  public async Task<Goods> GetGoodsInfo(Guid id, KindOfGoods kindOfGoods)  {  IQueryable<Goods> goods = kindOfGoodsMapper.MapToSpecificGoods(kindOfGoods);  return await goods  // TODO refactoring  //.Include(g => g.SpecificType)  .SingleAsync(e => e.GoodsId == id)!;  }  public async Task<GoodsUnitSearchDto> GetReadableGoodsInfo(Guid id, KindOfGoods kindOfGoods)  {  Goods goods = await GetGoodsInfo(id, kindOfGoods);  GoodsUnitSearchDto dto = new()  {  Id = id.ToString(),  // TODO specific type  //Type = goods.SpecificType.Name,  Price = goods.Price,  KindOfGoods = kindOfGoods  };  switch (kindOfGoods)  {  case KindOfGoods.MusicalInstruments:  case KindOfGoods.SheetMusicEditions:  dynamic specificGoods = goods;  string from = kindOfGoods == KindOfGoods.MusicalInstruments ? specificGoods.Manufacturer : specificGoods.Author;  dto.Name = $"{specificGoods.Name} от \"{from}\"";  dto.Description = $"Год выпуска: {specificGoods.ReleaseYear} {goods.Description}";  break;  case KindOfGoods.Accessories:  var accessory = (Accessory)goods;  string color = accessory.Color.ToLower();  string size = accessory.Size.ToLower();  dto.Name = $"{accessory.Name}, {color}, {size}";  dto.Description = accessory.Description;  break;  case KindOfGoods.AudioEquipmentUnits:  var audioEquipmentUnit = (AudioEquipmentUnit)goods;  dto.Name = $"{audioEquipmentUnit.Name}";  dto.Description = audioEquipmentUnit.Description;  break;  default:  throw new ArgumentException("unknown type");  };  return dto;  }  //public async Task<Type> GetGoodsType(Guid goodsId)  //{  // throw new NotImplementedException();  //}    }  // Файл \ServiceLayer\GoodsServices\GetGoodsUnitsRelatedToSaleService.cs  using DataLayer.Common;  using DataLayer.Models;  using DataLayer.NotMapped;  using Microsoft.EntityFrameworkCore;  using ServiceLayer.SalesServices;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ServiceLayer.GoodsServices;  public interface IGetGoodsUnitsRelatedToSaleService  {  Task<List<Goods>> GetOrigGoodsUnitsRelatedToSale(Guid saleId);  }  public class GetGoodsUnitsRelatedToSaleService(IGetSaleService saleService) : IGetGoodsUnitsRelatedToSaleService  {  public async Task<List<Goods>> GetOrigGoodsUnitsRelatedToSale(Guid saleId)  {  var sale = await saleService.GetOriginalSale(saleId);  return sale.MusicalInstruments  .Cast<Goods>()  .Concat(sale.Accessories  .Cast<Goods>())  .Concat(sale.SheetMusicEditions  .Cast<Goods>())  .Concat(sale.AudioEquipmentUnits  .Cast<Goods>())  .ToList();  }  }  // Файл \ServiceLayer\GoodsServices\GetRelevantGoodsService.cs  using DataLayer.Common;  using static Common.ConstValues;  using DataLayer.Models;  using DataLayer.NotMapped;  using Microsoft.EntityFrameworkCore;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Reflection;  using DataLayer.SupportClasses;  using System.Net.Http;  using System.Collections;  using ViewModelsLayer.Goods;  using ServiceLayer.SalesServices.QueryObjects;  namespace ServiceLayer.GoodsServices;  public interface IGetRelevantGoodsService  {  // Kinda complex task to implement.  // Upd: nice. this method is absolutely useless because it is impossible to implement paging for a lightweight quantity of objects in memory. The problem here is that goods of different types are not binded, so it's impossible to know what place item A takes in paging without getting knowledge about others. It may be first and last by match, it's depend upon other items. So, to select little objects in memory won't work. Do anyone understand what did i write here?  // Consequently, the cause of problem here - i hadn't known what exactly do i implement because i didn't know how the app will look like at all on the whole. I should've sketch out a layout of website = this is the gist.  Task<List<Guid>> GetRelevantGoodsIds(string researchText, GoodsFilterOptions filterOptions, GoodsOrderByOptions orderByOptions, int page, int pageSize);  }  #warning i ain't wanna create the service for each action  public class GetRelevantGoodsService(MusicalShopDbContext context, IMapKindOfGoodsService kindOfGoodsMapper) : IGetRelevantGoodsService  {  // Kinda complex task to implement.  // Upd: nice. this method is absolutely useless because it is impossible to implement paging for a lightweight quantity of objects in memory. The problem here is that goods of different types are not binded, so it's impossible to know what place item A takes in paging without getting knowledge about others. It may be first and last by match, it's depend upon other items. So, to select little objects in memory won't work. Do anyone understand what did i write here?  // Consequently, the cause of problem here - i hadn't known what exactly do i implement because i didn't know how the app will look like at all on the whole. I should've sketch out a layout of website = this is the gist.  public async Task<List<Guid>> GetRelevantGoodsIds(string researchText, GoodsFilterOptions filterOptions, GoodsOrderByOptions orderByOptions, int page, int pageSize)  {  IQueryable<Goods> goods = kindOfGoodsMapper.MapToSpecificGoods(filterOptions.KindOfGoods);  #warning separate it in few methods  goods = goods.AsNoTracking();  #warning explicit load of specificType or something else  //goods = goods.Include(g => g.SpecificType);  //goods = goods.Include(g => g.Sales);  //goods = goods.Where(g => g.Sales)  goods = goods.Where(g => g.Status == filterOptions.Status);  goods = goods.Where(g => g.Description.Contains(researchText) || g.Name.Contains(researchText));// || g.SpecificType.Name.Contains(researchText));  if (filterOptions.MinPrice != null)  goods = goods.Where(g => g.Price >= filterOptions.MinPrice);  if (filterOptions.MaxPrice != null)  goods = goods.Where(g => g.Price <= filterOptions.MaxPrice);  if (filterOptions.FromReceiptDate != null)  goods = goods.Where(g => g.ReceiptDate >= filterOptions.FromReceiptDate);  if (filterOptions.ToReceiptDate != null)  goods = goods.Where(g => g.ReceiptDate <= filterOptions.ToReceiptDate);  switch (orderByOptions.OrderBy)  {  case GoodsOrderBy.Relevance:  goods = orderByOptions.AscendingOrder  ? goods.OrderBy(g => g.GoodsId)  : goods.OrderByDescending(g => g.GoodsId);  break;  case GoodsOrderBy.Price:  goods = orderByOptions.AscendingOrder  ? goods.OrderBy(g => g.Price)  : goods.OrderByDescending(g => g.Price);  break;  case GoodsOrderBy.ReceiptDate:  goods = orderByOptions.AscendingOrder  ? goods.OrderBy(g => g.ReceiptDate)  : goods.OrderByDescending(g => g.ReceiptDate);  break;  default:  throw new Exception();  }  goods = goods.Page(page, pageSize);  List<Guid> result = [];  foreach (var goodsUnit in goods)  {  result.Add(goodsUnit.GoodsId);  }  return result;  #region buried previous version  // var suitableGoods = new List<Goods>();  // switch (filterOptions.GoodsFilter)  // {  // // I added paging for each table here to reduce in-memory size of data.  // // Then I page them again. At first, there's collection of 15 + 15 + 15 + 15 = 60 rows,  // // than 15 rows paging happens again.  //#warning it can be implemented via generic avoiding copy-past  // case GoodsFilter.None:  // suitableGoods.AddRange(context.Accessories  // .Where(a => a.Description.Contains(researchText))  // .OrderGoodsBy<Goods>(orderByEnum)  //#warning i screw up with paging.  // .Page(page, pageSize)  // .ToList());  // suitableGoods.AddRange(context.MusicalInstruments  // .Where(a => a.Description.Contains(researchText))  // .OrderGoodsBy<Goods>(orderByEnum)  // .Page(page, pageSize)  // .ToList());  // suitableGoods.AddRange(context.AudioEquipmentUnits  // .Where(a => a.Description.Contains(researchText))  // .OrderGoodsBy<Goods>(orderByEnum)  // .Page(page, pageSize)  // .ToList());  // suitableGoods.AddRange(context.SheetMusicEditions  // .Where(a => a.Description.Contains(researchText))  // .OrderGoodsBy<Goods>(orderByEnum)  // .Page(page, pageSize)  // .ToList());  // break;  // case GoodsFilter.Type:  // string[] types = filterOptions.FilterValue.Split(',');  // if (types.Length == 0)  // throw new ArgumentException("goods filter value was empty");  // foreach (string type in types)  // {  // switch (type.ToLower())  // {  // case "accessories":  // suitableGoods.AddRange(context.Accessories  // .Where(a => a.Description.Contains(researchText))  // .OrderGoodsBy<Goods>(orderByEnum)  // .Page(page, pageSize)  // .ToList());  // break;  // case "musicalinstruments":  // suitableGoods.AddRange(context.MusicalInstruments  // .Where(a => a.Description.Contains(researchText))  // .OrderGoodsBy<Goods>(orderByEnum)  // .Page(page, pageSize)  // .ToList());  // break;  // case "audioequipmentunits":  // suitableGoods.AddRange(context.AudioEquipmentUnits  // .Where(a => a.Description.Contains(researchText))  // .OrderGoodsBy<Goods>(orderByEnum)  // .Page(page, pageSize)  // .ToList());  // break;  // case "sheetmusiceditions":  // suitableGoods.AddRange(context.SheetMusicEditions  // .Where(a => a.Description.Contains(researchText))  // .OrderGoodsBy<Goods>(orderByEnum)  // .Page(page, pageSize)  // .ToList());  // break;  // default:  // throw new ArgumentException("unknown goods filter value");  // }  // }  // break;  // case GoodsFilter.ReceiptDate:  // string[] dates = filterOptions.FilterValue.Split(',');  // DateTime fromDate = DateTime.Parse(dates[0]);  // DateTime toDate = DateTime.Parse(dates[1]);  // suitableGoods.AddRange(context.Accessories  // .Where(a => a.Description.Contains(researchText)  // && a.ReceiptDate >= fromDate  // && a.ReceiptDate <= toDate)  // .OrderGoodsBy<Goods>(orderByEnum)  // .Page(page, pageSize)  // .ToList());  // suitableGoods.AddRange(context.MusicalInstruments  // .Where(a => a.Description.Contains(researchText)  // && a.ReceiptDate >= fromDate  // && a.ReceiptDate <= toDate)  // .OrderGoodsBy<Goods>(orderByEnum)  // .Page(page, pageSize)  // .ToList());  // suitableGoods.AddRange(context.AudioEquipmentUnits  // .Where(a => a.Description.Contains(researchText)  // && a.ReceiptDate >= fromDate  // && a.ReceiptDate <= toDate)  // .OrderGoodsBy<Goods>(orderByEnum)  // .Page(page, pageSize)  // .ToList());  // suitableGoods.AddRange(context.SheetMusicEditions  // .Where(a => a.Description.Contains(researchText)  // && a.ReceiptDate >= fromDate  // && a.ReceiptDate <= toDate)  // .OrderGoodsBy<Goods>(orderByEnum)  // .Page(page, pageSize)  // .ToList());  // break;  // case GoodsFilter.ReleaseYear:  // string[] years = filterOptions.FilterValue.Split(',');  // int fromYear = int.Parse(years[0]);  // int toYear = int.Parse(years[0]);  // suitableGoods.AddRange(context.Accessories  // .Where(a => a.Description.Contains(researchText)  // && a.ReceiptDate != null  // && a.ReceiptDate.Value.Year >= fromYear  // && a.ReceiptDate.Value.Year <= toYear)  // .OrderGoodsBy<Goods>(orderByEnum)  // .Page(page, pageSize)  // .ToList());  // suitableGoods.AddRange(context.MusicalInstruments  // .Where(a => a.Description.Contains(researchText)  // && a.ReceiptDate != null  // && a.ReceiptDate.Value.Year >= fromYear  // && a.ReceiptDate.Value.Year <= toYear)  // .OrderGoodsBy<Goods>(orderByEnum)  // .Page(page, pageSize)  // .ToList());  // suitableGoods.AddRange(context.AudioEquipmentUnits  // .Where(a => a.Description.Contains(researchText)  // && a.ReceiptDate != null  // && a.ReceiptDate.Value.Year >= fromYear  // && a.ReceiptDate.Value.Year <= toYear)  // .OrderGoodsBy<Goods>(orderByEnum)  // .Page(page, pageSize)  // .ToList());  // suitableGoods.AddRange(context.SheetMusicEditions  // .Where(a => a.Description.Contains(researchText)  // && a.ReceiptDate != null  // && a.ReceiptDate.Value.Year >= fromYear  // && a.ReceiptDate.Value.Year <= toYear)  // .OrderGoodsBy<Goods>(orderByEnum)  // .Page(page, pageSize)  // .ToList());  // break;  // default:  // throw new Exception();  // }  // switch (orderByEnum)  // {  // case GoodsOrderBy.Relevance:  //#warning how to filter by relevance  // //result.OrderBy(goods => )  // break;  // case GoodsOrderBy.PriceAscending:  // suitableGoods = [.. suitableGoods.OrderBy(goods => goods.Price)];  // break;  // case GoodsOrderBy.PriceDescending:  // suitableGoods = [.. suitableGoods.OrderByDescending(goods => goods.Price)];  // break;  // case GoodsOrderBy.ReceiptDateAscending:  // suitableGoods = [.. suitableGoods.OrderBy(goods => goods.ReceiptDate)];  // break;  // case GoodsOrderBy.ReceiptDateDescending:  // suitableGoods = [.. suitableGoods.OrderByDescending(goods => goods.ReceiptDate)];  // break;  // default:  // throw new Exception();  // }  // var result = new List<KeyValuePair<string, Type>>();  // for (int i = 0; i < 15; i++)  // {  // result.Add(new KeyValuePair<string, Type>(suitableGoods[i].GoodsId.ToString(), suitableGoods[i].GetType()));  // }  // return result;  #endregion  }  }  // Файл \ServiceLayer\GoodsServices\MapKindOfGoodsService.cs  using DataLayer.Common;  using DataLayer.Models.SpecificTypes;  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ServiceLayer.GoodsServices;  public interface IMapKindOfGoodsService  {  IQueryable<Goods> MapToSpecificGoods(KindOfGoods kindOfGoods);  SpecificType CreateNewSpecificType(KindOfGoods kindOfGoods);  Type MapToType(KindOfGoods kindOfGoods);  IQueryable<SpecificType> MapToSpecificTypes(KindOfGoods kindOfGoods);  Task<KindOfGoods> GetGoodsKind(Guid goodsId);  }  public class MapKindOfGoodsService(MusicalShopDbContext context) : IMapKindOfGoodsService  {  public IQueryable<Goods> MapToSpecificGoods(KindOfGoods kindOfGoods) => kindOfGoods switch  {  KindOfGoods.Accessories => context.Accessories,  KindOfGoods.AudioEquipmentUnits => context.AudioEquipmentUnits,  KindOfGoods.MusicalInstruments => context.MusicalInstruments,  KindOfGoods.SheetMusicEditions => context.SheetMusicEditions,  \_ => throw new ArgumentException()  };  public IQueryable<SpecificType> MapToSpecificTypes(KindOfGoods kindOfGoods) => kindOfGoods switch  {  KindOfGoods.MusicalInstruments => context.MusicalInstrumentSpecificTypes,  KindOfGoods.Accessories => context.AccessorySpecificTypes,  KindOfGoods.AudioEquipmentUnits => context.AudioEquipmentUnitSpecificTypes,  KindOfGoods.SheetMusicEditions => context.SheetMusicEditionSpecificTypes,  \_ => throw new ArgumentException()  };  public Type MapToType(KindOfGoods kindOfGoods) => kindOfGoods switch  {  KindOfGoods.MusicalInstruments => typeof(MusicalInstrumentSpecificType),  KindOfGoods.Accessories => typeof(AccessorySpecificType),  KindOfGoods.AudioEquipmentUnits => typeof(AudioEquipmentUnitSpecificType),  KindOfGoods.SheetMusicEditions => typeof(SheetMusicEditionSpecificType),  \_ => throw new ArgumentException()  };  public SpecificType CreateNewSpecificType(KindOfGoods kindOfGoods)  => (SpecificType)Activator.CreateInstance(MapToType(kindOfGoods))!;  public async Task<KindOfGoods> GetGoodsKind(Guid goodsId)  {  IQueryable<Goods> goods;  foreach (var kindOfGoods in Enum.GetValues<KindOfGoods>())  {  goods = MapToSpecificGoods(kindOfGoods);  if (goods.Any(g => g.GoodsId == goodsId))  return kindOfGoods;  }  throw new ArgumentException();  }  }  // Файл \ServiceLayer\GoodsServices\UpdateGoodsStatusService.cs  using DataLayer.Common;  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ServiceLayer.GoodsServices;  public interface IUpdateGoodsStatusService  {  Task<Guid> UpdateGoodsStatus(Guid goodsId, KindOfGoods kindOfGoods, GoodsStatus status);  }  public class UpdateGoodsStatusService(IGetGoodsService getGoodsService, MusicalShopDbContext context) : IUpdateGoodsStatusService  {  public async Task<Guid> UpdateGoodsStatus(Guid goodsId, KindOfGoods kindOfGoods, GoodsStatus status)  {  Goods goods = await getGoodsService.GetGoodsInfo(goodsId, kindOfGoods);  goods.Status = status;  context.Update(goods);  context.SaveChanges();  return goods.GoodsId;  }  }  // Файл \ServiceLayer\GoodsServices\Extensions\GoodsExtensions.cs  using DataLayer.NotMapped;  using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Reflection;  using System.Runtime.Intrinsics.X86;  using System.Text;  using System.Threading.Tasks;  using ViewModelsLayer.Goods;  using static Microsoft.EntityFrameworkCore.DbLoggerCategory;  namespace ServiceLayer.GoodsServices.Extensions;  public static class GoodsExtensions  {  #warning i'm not sure about it  public static /\*IQueryable<T>\*/ void OrderGoodsBy<T>(this IQueryable<T> goods, GoodsOrderByOptions goodsOrderByOptions)  where T : Goods  {  //return goodsOrderBy switch  //{  // GoodsOrderBy.Relevance => goods,  // GoodsOrderBy.PriceAscending => goods.OrderBy(g => g.Price),  // GoodsOrderBy.PriceDescending => goods.OrderByDescending(g => g.Price),  // GoodsOrderBy.ReceiptDateAscending => goods.OrderBy(g => g.ReceiptDate),  // GoodsOrderBy.ReceiptDateDescending => goods.OrderByDescending(g => g.ReceiptDate),  // \_ => throw new ArgumentOutOfRangeException(  // nameof(GoodsOrderBy), goodsOrderBy, null),  //};  }  }  // Файл \ServiceLayer\AdminServices\AddUserService.cs  using BizLogicBase.Common;  using BizLogicBase.Validation;  using BusinessLogicLayer.Admin.Dto;  using BusinessLogicLayer.Admin;  using DataLayer.Common;  using DataLayer.Models;  using DbAccessLayer.Admin;  using Microsoft.AspNetCore.Identity;  using System;  using System.Collections.Generic;  using System.Collections.Immutable;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ServiceLayer.AdminServices  {  class AddUserService(MusicalShopDbContext context, UserManager<IdentityUser> userManager) : ErrorStorage  {  private readonly RunnerWriteDb<NewUserDto, Task<string?>> \_runner = new(context, new AddUserAction(new(context, userManager)));  public override IImmutableList<ValidationResult> Errors => \_runner.Errors;  public async Task<string?> Add(string? userName, string? email, bool emailConfirmed, string? phoneNumber, bool phoneNumberConfirmed, string password)  => await \_runner.Run(new NewUserDto(userName, email, emailConfirmed, phoneNumber, phoneNumberConfirmed, password));  }  }  // Файл \ServiceLayer\AdminServices\BackupService.cs  using BizLogicBase.Validation;  using Humanizer.Configuration;  using Microsoft.Extensions.Configuration;  using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace ServiceLayer.AdminServices;  public interface IBackupService : IErrorAdder  {  Task<string> CreateBackup(string note);  Task<Dictionary<DateTime, string>> GetBackups();  Task ApplyRestoreFromBackup(DateTime backupDateTime);  }  public class BackupService : ErrorAdder, IBackupService  {  private readonly IConfigurationSection \_backupInfo;  private readonly string \_backupsDirectoryName;  private readonly DirectoryInfo \_backupsDirectoryInfo;  private IConfigurationSection UserCredentials => \_backupInfo.GetRequiredSection("UserCredentials");  private string UserName => (string)UserCredentials.GetValue(typeof(string), "Name")!;  private string UserPassword => (string)UserCredentials.GetValue(typeof(string), "Password")!;  private string DbName => (string)\_backupInfo.GetValue(typeof(string), "Database")!;  public BackupService(IConfiguration configuration)  {  \_backupInfo = configuration.GetRequiredSection("BackupData");  \_backupsDirectoryName = (string)\_backupInfo.GetValue(typeof(string), "Directory")!;  \_backupsDirectoryInfo = new DirectoryInfo(\_backupsDirectoryName);  }  #warning validate note length and its characters like :?\*<>.,"  public async Task<string> CreateBackup(string note)  {  string hostName = (string)\_backupInfo.GetValue(typeof(string), "Host")!;  EnsureDirectory();  string fileName = $"{DateTimeOffset.UtcNow.ToString(ConstValues.BackupDateTimeFormat)}{note}.sql";  string fullFileName = Path.Combine(\_backupsDirectoryName, fileName);  ProcessStartInfo processInfo = new("mysqldump", string.Format(Cmd.MysqldumpArguments, UserName, UserPassword, hostName, DbName, fullFileName))  {  UseShellExecute = false,  //RedirectStandardError = true,  //RedirectStandardOutput = true  };  var process = Process.Start(processInfo)!;  //Debug.WriteLine(process!.StandardOutput.ReadToEnd());  //Debug.WriteLine(process!.StandardError.ReadToEnd());  await process.WaitForExitAsync();  return fullFileName;  }  private void EnsureDirectory()  {  if (!\_backupsDirectoryInfo.Exists)  \_backupsDirectoryInfo.Create();  }  public async Task<Dictionary<DateTime, string>> GetBackups()  {  if (!\_backupsDirectoryInfo.Exists)  return [];  Dictionary<DateTime, string> result = [];  DateTime dateTime;  const string format = ConstValues.BackupDateTimeFormat;  foreach (var fileInfo in \_backupsDirectoryInfo.GetFiles())  {  dateTime = DateTime.ParseExact(fileInfo.Name.AsSpan(0, format.Length), format, null);  result[dateTime] = fileInfo.Name.AsSpan(format.Length, fileInfo.Name.Length - ".sql".Length - format.Length).ToString();  }  return result;  }  public async Task ApplyRestoreFromBackup(DateTime backupDateTime)  {  // checking  if (!\_backupsDirectoryInfo.Exists)  {  AddError("Резервные копии не найдены.");  return;  }  string formattedDateTime = backupDateTime.ToString(ConstValues.BackupDateTimeFormat);  var file = \_backupsDirectoryInfo.GetFiles()  .Where(file => file.Name.StartsWith(formattedDateTime))  .SingleOrDefault();  if(file == null)  {  AddError("Резервная копия с данной датой не найдена");  return;  }  #warning ask user permission to database dropping somewhere here  string commandToExecute = string.Format(SqlStatements.RestoreDatabaseFromBackup, DbName, file.FullName.Replace('\\', '/'));  ProcessStartInfo processStartInfo = new()  {  FileName = "mysql",  Arguments = string.Format(Cmd.MysqRestoreBackupArgs, UserName, UserPassword, commandToExecute),  UseShellExecute = false,  RedirectStandardOutput = true,  };  var process = Process.Start(processStartInfo)!;  Debug.WriteLine(process.StandardOutput.ReadToEnd());  await process.WaitForExitAsync();  }  }  // Файл \ServiceLayer\AdminServices\GetUserService.cs  using DataLayer.Common;  using DataLayer.Models;  using DbAccessLayer.Admin;  using Microsoft.AspNetCore.Identity;  using Microsoft.EntityFrameworkCore;  namespace ServiceLayer.AdminServices;  public interface IGetUserService  {  Task<IdentityUser?> GetUserInfo(Guid userId);  }  public class GetUserService(MusicalShopDbContext context, UserManager<IdentityUser> userManager) : IGetUserService  {  private readonly UserDbAccess dbAccess = new(context, userManager);  public async Task<IdentityUser?> GetUserInfo(Guid userId)  {  return await dbAccess.GetUserInfo(userId);  }  }  // Файл \ServiceLayer\AdminServices\UpdateUserService.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using DataLayer.Models;  using DataLayer.Common;  using DbAccessLayer.Admin;  using BizLogicBase.Validation;  using System.Collections.Immutable;  using System.ComponentModel.DataAnnotations;  using BizLogicBase.Common;  using BusinessLogicLayer.Admin;  using Microsoft.AspNetCore.Identity;  using BusinessLogicLayer.Admin.Dto;  namespace ServiceLayer.AdminServices;  public interface IUpdateUserService : IErrorStorage  {  Task<string?> UpdateUser(UpdateUserDto dto);  }  public class UpdateUserService(MusicalShopDbContext context, UserManager<IdentityUser> userManager) : ErrorStorage, IUpdateUserService  {  private readonly RunnerWriteDb<UpdateUserDto, Task<string?>> \_runner = new(context, new UpdateUserAction(new(context, userManager)));  public override IImmutableList<ValidationResult> Errors => \_runner.Errors;  #warning actually here must not be dto  public async Task<string?> UpdateUser(UpdateUserDto dto) => await \_runner.Run(dto);  }  // Файл \SeleniumTests\UnitTest1.cs  namespace SeleniumTests;  [TestClass]  public class UnitTest1  {  [TestMethod]  public void TestMethod1()  {  }  }  // Файл \MusicalShopApp\AppProgram.cs  using DataLayer.Common;  using DataLayer.Models;  using Microsoft.AspNetCore.Identity;  using Microsoft.EntityFrameworkCore;  using DbAccessLayer;  using ServiceLayer;  using Microsoft.AspNetCore.Mvc.Rendering;  using BasicAppConfiguration;  using Microsoft.Extensions.Caching.Distributed;  using Microsoft.Extensions.Configuration;  using Microsoft.Extensions.DependencyInjection;  using ServiceLayer.AdminServices;  using ServiceLayer.GoodsServices;  using NetCore.AutoRegisterDi;  using System.Reflection;  using DbAccessLayer.Admin;  using Microsoft.AspNetCore.Mvc;  using DataLayer.SupportClasses;  using Microsoft.AspNetCore.Builder;  var builder = BasicApp.CreateBuilder(args);  builder.Services.AddControllersWithViews()  .AddViewOptions(options =>  {  options.HtmlHelperOptions.FormInputRenderMode = FormInputRenderMode.AlwaysUseCurrentCulture;  });  builder.Services.AddDistributedMemoryCache();  builder.Services.AddSession(options =>  {  options.IdleTimeout = TimeSpan.FromMinutes(30);  options.Cookie.HttpOnly = true;  options.Cookie.IsEssential = true;  });  var app = builder.Build();  using (var scope = app.Services.CreateAsyncScope())  {  await DataSeeding.SeedAsync(scope.ServiceProvider);  }  // Configure the HTTP request pipeline.  if (app.Environment.IsDevelopment())  {  app.UseMigrationsEndPoint();  }  else  {  app.UseExceptionHandler("/Goods/Error");  // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.  app.UseHsts();  }  app.UseHttpsRedirection();  app.UseStaticFiles();  app.UseRouting();  app.UseAuthorization();  app.UseSession();  app.MapControllerRoute(  name: "default",  pattern: "{controller=Goods}/{action=Index}/{id?}");  app.MapRazorPages();  app.Run();  // Файл \MusicalShopApp\GlobalUsings.cs  global using Common;  // Файл \MusicalShopApp\Views\Shared\PartialNavigationItemModel.cs  namespace MusicalShopApp.Views.Shared;  public record class PartialNavigationItemModel(string Role, string Controller, string Action, string Title);  // Файл \MusicalShopApp\Views\Shared\CustomTagHelpers\CustomAttributeTagHelper.cs  using Microsoft.AspNetCore.Razor.TagHelpers;  namespace MusicalShopApp.Views.Shared.CustomTagHelpers;  [HtmlTargetElement(Attributes = "custom-attributes")]  public class CustomAttributeTagHelper : TagHelper  {  #warning whaaat  public Dictionary<string, string> CustomAttributes { get; set; }  public override void Process(TagHelperContext context, TagHelperOutput output)  {  if (CustomAttributes != null)  foreach (var pair in CustomAttributes)  if (!output.Attributes.ContainsName(pair.Key))  output.Attributes.Add(pair.Key, pair.Value);  }  }  // Файл \MusicalShopApp\Models\ErrorViewModel.cs  namespace MusicalShopApp.Models;  public class ErrorViewModel  {  public string? RequestId { get; set; }  public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);  }  // Файл \MusicalShopApp\Controllers\AdminController.cs  using BusinessLogicLayer.Admin.Dto;  using DataLayer.Models;  using Microsoft.AspNetCore.Authorization;  using Microsoft.AspNetCore.Identity;  using Microsoft.AspNetCore.Mvc;  using Microsoft.AspNetCore.Mvc.ModelBinding;  using ServiceLayer.AdminServices;  using ViewModelsLayer.Admin;  namespace MusicalShopApp.Controllers  {  [Authorize(Roles = CommonNames.AdminRole)]  public class AdminController : Controller  {  [HttpGet]  public IActionResult Users()  {  return View();  }  [HttpGet(template: "/Admin/Users/{userId:Guid}", Name = "SpecificUser")]  public async Task<IActionResult> SpecificUser(Guid userId, [FromServices] IGetUserService service, [FromQuery] string? errors)  {  var user = await service.GetUserInfo(userId);  ViewBag.Errors = errors;  return View(user);  }  [HttpGet]  public async Task<IActionResult> Backups([FromServices] IBackupService service)  {  try  {  // TODO handle in  var dict = await service.GetBackups();  return View(new BackupsModel(dict));  }  catch  {  #warning handle errors  return View(null);  }  }  [HttpPost]  [ValidateAntiForgeryToken]  public async Task<ContentResult> CreateBackup([FromForm] string note, [FromServices] IBackupService service)  {  if(!ModelState.IsValid)  {  }  else  {  return Content("");  }  try  {  string fullFileName = await service.CreateBackup(note);  return Content($"Успешно. Название файла: {fullFileName}");  }  catch  {  #warning what error?  return Content("Ошибка.");  }  }  [HttpPost]  [ValidateAntiForgeryToken]  public async Task<ContentResult> RestoreBackup([FromForm] DateTime? dateTime, [FromServices] IBackupService service)  {  if (dateTime is null)  return Content("Укажите резервную копию для восстановления");  await service.ApplyRestoreFromBackup((DateTime)dateTime);  if (service.HasErrors)  #warning i don't think it's correctly. The view details in the controller?  return Content(string.Join("<br>", service.Errors));  return Content("База данных восстановлена");  }  [HttpPost]  [ValidateAntiForgeryToken]  #warning rename to EditUsers  public async Task<IActionResult> Edit(UpdateUserDto dto, [FromServices] IUpdateUserService service)  {  await service.UpdateUser(dto);  return RedirectToAction("Users", new RouteValueDictionary(  #warning workaround  new { userId = dto.Id, errors = string.Join("SEP", service.Errors)}));  }  }  }  // Файл \MusicalShopApp\Controllers\GoodsController.cs  using System;  using System.Diagnostics;  using System.Reflection;  using System.Text;  using DataLayer.Common;  using DataLayer.Models;  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  using Microsoft.AspNetCore.Authorization;  using Microsoft.AspNetCore.Mvc;  using Microsoft.IdentityModel.Tokens;  using MusicalShopApp.Controllers.BaseControllers;  using MusicalShopApp.Models;  using ServiceLayer.GoodsServices;  using ServiceLayer.GoodsServices.Extensions;  using ServiceLayer.SalesServices;  using ViewModelsLayer;  using ViewModelsLayer.Goods;  namespace MusicalShopApp.Controllers;  [Authorize(Roles = $"{CommonNames.AdminRole},{CommonNames.StockManagerRole},{CommonNames.SellerRole},{CommonNames.ConsultantRole}")]  public class GoodsController : CartViewerBaseController  {  private readonly ILogger<GoodsController> \_logger;  public GoodsController(ILogger<GoodsController> logger)  {  \_logger = logger;  }  public IActionResult Index()  {  #warning if i don't need it, delete view and this method and then change the url for ������ in \_Layout  return RedirectToAction("Search");  }  public IActionResult Privacy()  {  return View();  }  [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]  public IActionResult Error()  {  return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });  }  //[HttpPost]  //[ValidateAntiForgeryToken]  //public async Task<IActionResult> Search(GoodsSearchModel model)  //{  // //GoodsSearchModel model = new()  // //{  // // Filter = new GoodsFilterOptions(null, null, null, null, KindOfGoods: KindOfGoods.MusicalInstruments,  // // Status: GoodsStatus.InStock),  // // GoodsUnitModels = [],  // // OrderBy = new GoodsOrderByOptions(GoodsOrderBy.Relevance, true),  // // ResearchText = string.Empty  // //};  // return View(model);  //}  #warning what the hell is going on here? i have not been seeing this method for just about 1 week and now it looks like insane one  [HttpGet]  public async Task<IActionResult> Search([FromServices] IGetRelevantGoodsService getRelevantGoodsService, [FromServices] IGetGoodsService getGoodsService, [FromQuery] int? minPrice, [FromQuery] int? maxPrice, [FromQuery] DateTime? fromReceiptDate, [FromQuery] DateTime? toReceiptDate, [FromQuery] KindOfGoods kindOfGoods, [FromQuery] GoodsOrderBy orderBy, [FromQuery] GoodsStatus status, [FromQuery] bool ascendingOrder, [FromQuery] string q = "", [FromQuery] int page = 1, [FromQuery] int pageSize = 15)  {  var filterOptions = new GoodsFilterOptions(minPrice, maxPrice, fromReceiptDate.LocalToUniversal(), toReceiptDate.LocalToUniversal(), kindOfGoods, status);  var orderByOptions = new GoodsOrderByOptions(orderBy, ascendingOrder);  #warning what about query object pattern here?  var goodsIds = await getRelevantGoodsService.GetRelevantGoodsIds(q, filterOptions, orderByOptions, page, pageSize);  List<GoodsUnitSearchDto> goodsUnitModels = new();  foreach (var goodsId in goodsIds)  {  try  {  var goodsUnitSearchDto = await getGoodsService.GetReadableGoodsInfo(goodsId, kindOfGoods);  goodsUnitSearchDto.IsInCart = IsInCart(goodsId);  goodsUnitModels.Add(goodsUnitSearchDto);  }  catch  {  \_logger.LogWarning("unknown goods id in cart: {goodsId}", goodsId);  }  }  var goodsSearchModel = new GoodsSearchModel  {  ResearchText = q,  Filter = filterOptions,  OrderBy = orderByOptions,  GoodsUnitModels = goodsUnitModels,  ResultsCount = goodsUnitModels.Count()  };  ViewBag.Session = GoodsIdsAndKindsInCart;  return View(goodsSearchModel);  }  [HttpPost]  [Authorize(Roles = CommonNames.SellerRole)]  [ValidateAntiForgeryToken]  public async Task<ContentResult> AddToOrRemoveFromCart(Guid goodsId, bool isInCart, [FromServices] ICartService cartService)  {  // DRN violation  string? newGoodsIdsAndTypes = await cartService.AddToOrRemoveFromCart(goodsId, isInCart, GoodsIdsAndKindsInCart);  if (newGoodsIdsAndTypes == null)  return Content("failed");  SetNewCartValue(newGoodsIdsAndTypes);  ViewBag.Session = GoodsIdsAndKindsInCart;  return Content("success");  }  [HttpGet]  [Authorize(Roles = CommonNames.SellerRole)]  public async Task<IActionResult> Cart([FromServices] IGetGoodsService getGoodsService, [FromServices] ICartService cartService)  {  #warning i'm confused around the whole this cart stuff. i stopped understanding what's going on here  List<GoodsUnitSearchDto> GoodsUnitModels = new();  if (!string.IsNullOrEmpty(GoodsIdsAndKindsInCart))  {  foreach (var goodsIdAndType in GoodsIdsAndKinds!)  {  Guid goodsId = Guid.Parse(cartService.CutGoodsId(goodsIdAndType));  var goodsInfo = await getGoodsService.GetReadableGoodsInfo(goodsId, cartService.CutGoodsKind(goodsIdAndType));  goodsInfo.IsInCart = IsInCart(goodsId);  GoodsUnitModels.Add(goodsInfo);  //try  //{  // var goodsInfo = await service.GeitReadableGoodsInfo(goodsId, CutGoodsKind(goodsIdAndType));  // goodsInfo.IsInCart = IsInCart(goodsId);  // GoodsUnitModels.Add(goodsInfo);  //}  //catch  //{  // \_logger.LogError("unknown goods id in cart: {goodsId}", goodsId);  //}  }  }  ViewBag.Session = GoodsIdsAndKindsInCart;  return View(GoodsUnitModels);  }  [HttpGet("/goods/{kindOfGoods}/{goodsId}")]  public async Task<IActionResult> GoodsUnit([FromRoute] KindOfGoods kindOfGoods, [FromRoute] Guid goodsId, [FromServices] IGetGoodsService service)  {  var goods = await service.GetGoodsInfo(goodsId, kindOfGoods);  var goodsModel = new GoodsUnitModel(goods.GoodsId, kindOfGoods, goods.Name, goods.Price, goods.Status, goods.Description);  return View(goodsModel);  }  }  // Файл \MusicalShopApp\Controllers\PurchaseReturnController.cs  using Microsoft.AspNetCore.Authorization;  using Microsoft.AspNetCore.Mvc;  namespace MusicalShopApp.Controllers  {  [Authorize(Roles = CommonNames.SellerRole)]  public class PurchaseReturnController : Controller  {  public IActionResult Index()  {  return View();  }  }  }  // Файл \MusicalShopApp\Controllers\ReportsController.cs  using Microsoft.AspNetCore.Authorization;  using Microsoft.AspNetCore.Mvc;  namespace MusicalShopApp.Controllers  {  [Authorize(Roles = CommonNames.AdminRole)]  public class ReportsController : Controller  {  public IActionResult Index()  {  return View();  }  }  }  // Файл \MusicalShopApp\Controllers\SalesController.cs  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  using Microsoft.AspNetCore.Authorization;  using Microsoft.AspNetCore.Mvc;  using MusicalShopApp.Controllers.BaseControllers;  using ServiceLayer.GoodsServices;  using ServiceLayer.SalesServices;  using ViewModelsLayer.Sales;  namespace MusicalShopApp.Controllers;  [Authorize(Roles = CommonNames.SellerRole)]  public class SalesController : CartViewerBaseController  {  [HttpGet]  public async Task<IActionResult> Search(string q, [FromServices] IGetRelevantSalesService service, DateTime? minSaleDate, DateTime? maxSaleDate, DateTime? minReservationDate, DateTime? maxReservationDate, DateTime? minReturningDate, DateTime? maxReturningDate, SalePaidBy? paidBy, SaleStatus? status, SalesOrderBy orderBy=SalesOrderBy.Relevance, bool orderByAscending=true)  {  var filterOptions = new SalesFilterOptions(minSaleDate, maxSaleDate, minReservationDate, maxReservationDate, minReturningDate, maxReturningDate, status, paidBy);  var orderByOptions = new SalesOrderByOptions(orderBy, orderByAscending);  List<SaleSearchDto> list = await service.GetRelevantSales(q, filterOptions, orderByOptions);  return View(new SalesSearchModel(q, list, list.Count, filterOptions, orderByOptions));  }  [HttpPost("/sale/arrange")]  public async Task<IActionResult> CreateSaleAsNotSold([FromServices] ICreateSaleService createSaleService, [FromServices] ICartService cartService)  {  var goods = await cartService.GetGoodsFromCart(GoodsIdsAndKinds);  Guid? saleId = await createSaleService.CreateSaleAsNotPaid(goods);  if (!createSaleService.HasErrors)  {  ClearCart();  return RedirectToAction("PayForSale", new { saleId });  }  else  return RedirectToAction("Cart", "Goods");//, new SaleErrorModel(service.Errors));  }  [HttpPost("/sale/reserve")]  [ValidateAntiForgeryToken]  public async Task<IActionResult> Reserve([FromServices] ICartService cartService)  {  return View();  }  [HttpGet]  public async Task<IActionResult> PayForSale([FromQuery] Guid saleId)  {  return View(saleId);  }  /// <summary>  /// Payment was successful.  /// </summary>  /// <param name="saleId"></param>  /// <returns></returns>  // TODO encapsulate the goods status updating  [HttpPost]  [ValidateAntiForgeryToken]  public async Task<ContentResult> RegisterSaleAsSold(Guid saleId, SalePaidBy paidBy, [FromServices] IExistingSaleManagementService saleService, [FromServices] ICartService cartService, [FromServices] IGetGoodsUnitsRelatedToSaleService goodsService, [FromServices] IUpdateGoodsStatusService goodsStatusService, [FromServices] IMapKindOfGoodsService kindOfGoodsService)  {  string result;  try  {  await saleService.RegisterSaleAsPaid(saleId, paidBy);  var goods = await goodsService.GetOrigGoodsUnitsRelatedToSale(saleId);  foreach(var goodsUnit in goods)  {  await goodsStatusService.UpdateGoodsStatus(goodsUnit.GoodsId, await kindOfGoodsService.GetGoodsKind(goodsUnit.GoodsId), GoodsStatus.Sold);  }  result = "Successfully registered";  }  catch  {  await RestoreCart(saleId, cartService);  await saleService.CancelSale(saleId);  result = "Failed to register";  }  return Content(result);  }  private async Task RestoreCart(Guid saleId, ICartService cartService)  {  string newCartContent = await cartService.MoveGoodsBackToCart(saleId);  SetNewCartValue(newCartContent);  }  /// <summary>  /// Something went wrong during the payment.  /// </summary>  /// <param name="saleId"></param>  /// <returns></returns>  [HttpPost]  [ValidateAntiForgeryToken]  public async Task<ContentResult> SaleCancelling([FromForm] Guid saleId, [FromServices] IExistingSaleManagementService service, [FromServices] ICartService cartService)  {  string result;  try  {  // TODO something is wrong here  await RestoreCart(saleId, cartService);  await service.CancelSale(saleId);  result = "Successfully cancelled";  }  catch  {  result = "Failed to cancel";  }  return Content(result);  }  } | // Файл \MusicalShopApp\Controllers\StockController.cs  using BizLogicBase.Validation;  using DataLayer.SupportClasses;  using Microsoft.AspNetCore.Mvc;  using Microsoft.AspNetCore.Mvc.ModelBinding;  using Microsoft.AspNetCore.Mvc.Rendering;  using ServiceLayer.StockServices;  using ViewModelsLayer.Stock;  namespace MusicalShopApp.Controllers;  public class StockController : Controller  {  [HttpGet]  public async Task<IActionResult> AddGoodsToWarehouse([FromServices] ISpecificTypeService specificTypesService)  {  var specificTypes = await specificTypesService.GetAllSpecificTypes();  GoodsKindSpecificDataDto defaultSpecificData = new(KindOfGoods.MusicalInstruments, default, default, default, default, default, default);  var defaultDto = new AddGoodsToWarehouseDto(default!, default!, false, default, default, GoodsStatus.InStock, default, default, defaultSpecificData);  return View(new AddGoodsToWarehouseModel(defaultDto, specificTypes, []));  }  [HttpPost]  [ValidateAntiForgeryToken]  public async Task<ContentResult> AddGoodsToWarehouse([FromServices] IAddNewGoodsService addNewGoodsService, AddGoodsToWarehouseDto addGoodsToWarehouseDto)  {  if (!ModelState.IsValid)  {  string errors = string.Empty;  foreach (var error in ModelState.Values.SelectMany(modelEntry => modelEntry.Errors.Select(e => e.ErrorMessage)))  {  errors += error + "<br>";  }  return Content(errors);  }  // TODO what data? how to get errors of model?  await addNewGoodsService.AddNewGoods(addGoodsToWarehouseDto);  string result = addNewGoodsService.HasErrors  ? string.Join("\r\n", addNewGoodsService.Errors)  : "Товар успешно добавлен";  return Content(result);  }  }  // Файл \MusicalShopApp\Controllers\BaseControllers\CartViewerBaseController.cs  using DataLayer.SupportClasses;  using Microsoft.AspNetCore.Mvc;  namespace MusicalShopApp.Controllers.BaseControllers;  public abstract class CartViewerBaseController : Controller  {  public string? GoodsIdsAndKindsInCart => HttpContext.Session.GetString(CommonNames.GoodsIdsInCartKey);  public string[] GoodsIdsAndKinds => GoodsIdsAndKindsInCart?.Split(CommonNames.GoodsIdSeparator, StringSplitOptions.RemoveEmptyEntries) ?? [];  public bool IsInCart(Guid goodsId) => GoodsIdsAndKindsInCart != null && GoodsIdsAndKinds!.Any(s => s.Contains(goodsId.ToString()));  public void SetNewCartValue(string newValue) => HttpContext.Session.SetString(CommonNames.GoodsIdsInCartKey, newValue);  public void ClearCart()  => SetNewCartValue(string.Empty);  }  // Файл \MusicalShopApp\Areas\Identity\Pages\Account\Login.cshtml.cs  // Licensed to the .NET Foundation under one or more agreements.  // The .NET Foundation licenses this file to you under the MIT license.  #nullable disable  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Threading.Tasks;  using Microsoft.AspNetCore.Authorization;  using Microsoft.AspNetCore.Authentication;  using Microsoft.AspNetCore.Identity;  using Microsoft.AspNetCore.Identity.UI.Services;  using Microsoft.AspNetCore.Mvc;  using Microsoft.AspNetCore.Mvc.RazorPages;  using Microsoft.Extensions.Logging;  using DataLayer.Models;  namespace MusicalShopApp.Areas.Identity.Pages.Account;  public class LoginModel : PageModel  {  private readonly SignInManager<IdentityUser> \_signInManager;  private readonly ILogger<LoginModel> \_logger;  public LoginModel(SignInManager<IdentityUser> signInManager, ILogger<LoginModel> logger)  {  \_signInManager = signInManager;  \_logger = logger;  }  /// <summary>  /// This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used  /// directly from your code. This API may change or be removed in future releases.  /// </summary>  [BindProperty]  public InputModel Input { get; set; }  /// <summary>  /// This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used  /// directly from your code. This API may change or be removed in future releases.  /// </summary>  public IList<AuthenticationScheme> ExternalLogins { get; set; }  /// <summary>  /// This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used  /// directly from your code. This API may change or be removed in future releases.  /// </summary>  public string ReturnUrl { get; set; }  /// <summary>  /// This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used  /// directly from your code. This API may change or be removed in future releases.  /// </summary>  [TempData]  public string ErrorMessage { get; set; }  /// <summary>  /// This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used  /// directly from your code. This API may change or be removed in future releases.  /// </summary>  public class InputModel  {  /// <summary>  /// This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used  /// directly from your code. This API may change or be removed in future releases.  /// </summary>  [Required]  [EmailAddress]  public string Email { get; set; }  /// <summary>  /// This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used  /// directly from your code. This API may change or be removed in future releases.  /// </summary>  [Required]  [DataType(DataType.Password)]  public string Password { get; set; }  /// <summary>  /// This API supports the ASP.NET Core Identity default UI infrastructure and is not intended to be used  /// directly from your code. This API may change or be removed in future releases.  /// </summary>  [Display(Name = "Remember me?")]  public bool RememberMe { get; set; }  }  public async Task OnGetAsync(string returnUrl = null)  {  if (!string.IsNullOrEmpty(ErrorMessage))  {  ModelState.AddModelError(string.Empty, ErrorMessage);  }  returnUrl ??= Url.Content("~/");  // Clear the existing external cookie to ensure a clean login process  await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);  ExternalLogins = (await \_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();  ReturnUrl = returnUrl;  }  public async Task<IActionResult> OnPostAsync(string returnUrl = null)  {  returnUrl ??= Url.Content("~/");  ExternalLogins = (await \_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();  if (ModelState.IsValid)  {  // This doesn't count login failures towards account lockout  // To enable password failures to trigger account lockout, set lockoutOnFailure: true  var result = await \_signInManager.PasswordSignInAsync(Input.Email, Input.Password, Input.RememberMe, lockoutOnFailure: false);  if (result.Succeeded)  {  \_logger.LogInformation("User logged in.");  return LocalRedirect(returnUrl);  }  if (result.RequiresTwoFactor)  {  return RedirectToPage("./LoginWith2fa", new { ReturnUrl = returnUrl, RememberMe = Input.RememberMe });  }  if (result.IsLockedOut)  {  \_logger.LogWarning("User account locked out.");  return RedirectToPage("./Lockout");  }  else  {  ModelState.AddModelError(string.Empty, "Invalid login attempt.");  return Page();  }  }  // If we got this far, something failed, redisplay form  return Page();  }  }  // Файл \DbAccessLayer\GlobalUsings.cs  global using Common;  // Файл \DbAccessLayer\GoodsDbAccess.cs  using DataLayer.Common;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DbAccessLayer;  public class GoodsDbAccess(MusicalShopDbContext context)  {  }  // Файл \DbAccessLayer\SalesDbAccess.cs  using DataLayer.Common;  using DataLayer.Models;  using DataLayer.NotMapped;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DbAccessLayer;  public class SalesDbAccess(MusicalShopDbContext context)  {  public void CreateSaleAndUpdateGoods(Sale sale, List<Goods> goods)  {  foreach(var goodsUnit in goods)  {  goodsUnit.Sales.Add(sale);  context.Update(goodsUnit);  #warning what if goods were passed here with edited values except goods.Status? UPD: okey, here it's safety, but probably dto with restricted data for update would be better. after all, that's a purpose of dto  #warning do i need it at all? what if the latest line of this method may execute updating?  //context.Update(goodsUnit);  }  context.Add(sale);  }  }  // Файл \DbAccessLayer\ServicesRegisterExtension.cs  using Microsoft.Extensions.DependencyInjection;  using NetCore.AutoRegisterDi;  namespace DbAccessLayer;  public static class ServicesRegisterExtension  {  public static void RegisterDbAccessLayer(this IServiceCollection services)  {  services.RegisterAssemblyPublicNonGenericClasses()  .Where(@class => @class.Name.EndsWith("DbAccess"))  .AsPublicImplementedInterfaces();  }  }  // Файл \DbAccessLayer\Admin\UserDbAccess.cs  using DataLayer.Common;  using DataLayer.Models;  using Microsoft.AspNetCore.Identity;  using Microsoft.EntityFrameworkCore;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DbAccessLayer.Admin;  public interface IUserDbAccess  {  Task<IdentityUser?> GetUserInfo(Guid userId);  Task<bool> IsUniqueNormalizedUserName(string normalizedUserName, string userId);  Task<bool> IsUniqueNormalizedUserName(string normalizedUserName);  Task<bool> IsUniqueNormalizedEmail(string normalizedEmail, string userId);  Task<string?> AddUser(IdentityUser newUser, string password);  }  public class UserDbAccess(MusicalShopDbContext context, UserManager<IdentityUser> userManager) : IUserDbAccess  {  #warning don't return null...  public async Task<IdentityUser?> GetUserInfo(Guid userId)  {  return await context.Users.Where(u => u.Id == userId.ToString())  .SingleOrDefaultAsync();  }  public async Task<bool> IsUniqueNormalizedUserName(string normalizedUserName, string userId) =>  !await context.Users.AnyAsync(u => u.NormalizedUserName == normalizedUserName  && u.Id != userId);  public async Task<bool> IsUniqueNormalizedUserName(string normalizedUserName) =>  await IsUniqueNormalizedUserName(normalizedUserName, CommonNames.NotExistingGuid);  public async Task<bool> IsUniqueNormalizedEmail(string normalizedEmail, string userId)  {  return !await context.Users.AnyAsync(u => u.NormalizedEmail == normalizedEmail && u.Id != userId);  }  public async Task<string?> AddUser(IdentityUser newUser, string password)  {  var result = await userManager.CreateAsync(newUser, password);  if (result.Errors.Any())  return newUser.Id;  else  return null;  }  }  // Файл \DataLayerTests\UnitTest1.cs  using DataLayer.Common;  namespace DataLayerTests;  [TestClass]  public class UnitTest1  {  public MusicalShopDbContext context;  #warning what did i want to test here?  #warning UPD: the same question a month ago. so funny that i asked it again.  [TestInitialize]  public void SetUp()  {  context = new MusicalShopDbContext();  }  [TestMethod]  public void TestMethod1()  {    }  }  // Файл \DataLayer\GlobalUsings.cs  global using Common;  // Файл \DataLayer\UNDER QUESTION Factory\AbstractFactory.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Factory;  public class AbstractFactory  {  }  // Файл \DataLayer\SupportClasses\Extensions.cs  namespace DataLayer.SupportClasses;  public static class Extensions  {  public static string ToFormattedText(this GoodsStatus goodsStatus) =>  goodsStatus switch  {  GoodsStatus.InStock => "На складе",  GoodsStatus.AwaitingDelivery => "Ожидает поставки",  GoodsStatus.Sold => "Продано",  GoodsStatus.Reserved => "Зарезервировано"  };  //public static bool TryParse(this string kindOfGoods, out KindOfGoods result) => Enum.TryParse<KindOfGoods>(kindOfGoods, out result);  }  // Файл \DataLayer\SupportClasses\GoodsStatus.cs  using System.ComponentModel;  public enum GoodsStatus  {  InStock,  InCart,  AwaitingDelivery,  Sold,  Reserved,  AwaitingPayment  }  // Файл \DataLayer\SupportClasses\ISoftDeletable.cs  namespace DataLayer.SupportClasses;  public interface ISoftDeletable  {  public bool SoftDeleted { get; set; }  }  // Файл \DataLayer\SupportClasses\KindOfGoods.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.SupportClasses;  public enum KindOfGoods  {  MusicalInstruments,  Accessories,  AudioEquipmentUnits,  SheetMusicEditions  }  // Файл \DataLayer\SupportClasses\ManufacturerType.cs  namespace DataLayer.SupportClasses;  public enum ManufacturerType  {  Factory,  Master  }  // Файл \DataLayer\SupportClasses\SalePaidBy.cs  namespace DataLayer.SupportClasses;  public enum SalePaidBy  {  Cash,  BankCard,  }  // Файл \DataLayer\SupportClasses\SaleStatus.cs  namespace DataLayer.SupportClasses;  public enum SaleStatus  {  Sold,  Reserved,  Returned,  YetNotPaid  }  // Файл \DataLayer\SupportClasses\SpecificType.cs  using System.ComponentModel.DataAnnotations;  using System.ComponentModel.DataAnnotations.Schema;  namespace DataLayer.SupportClasses;  [NotMapped]  public abstract class SpecificType  {  [Key]  public Guid SpecificTypeId { get; set; }  [Required]  public string Name { get; set; }  }  // Файл \DataLayer\NotMapped\Goods.cs  using DataLayer.Models;  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.ComponentModel.DataAnnotations.Schema;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.NotMapped;  //[NotMapped] // Apparently ef core ignores types which are inhereted by other types and doesn't model them even without this attribute. But I wrote it here just for clarity.  public abstract class Goods : ISoftDeletable  {  // TODO give the opportunity to reserve the goods for consultant  // TODO add consultant  [Key]  public Guid GoodsId { get; set; }  [Required]  [MaxLength(ConstValues.GoodsNameMaxLength)]  public string Name { get; set; }  public bool SoftDeleted { get; set; }  [Range(ConstValues.GoodsPriceMinValue, ConstValues.GoodsPriceMaxValue)]  public int Price { get; set; }  public GoodsStatus Status { get; set; }  [MaxLength(ConstValues.GoodsDescriptionMaxLength)]  public string? Description { get; set; }  // relationships  public virtual ICollection<Sale> Sales { get; set; } = [];  public Guid SpecificTypeId { get; set; }  //public abstract SpecificType SpecificType { get; set; }  public DateTimeOffset? ReceiptDate { get; set; }  public Guid? DeliveryId { get; set; }  [ForeignKey(nameof(Goods.DeliveryId))]  public virtual GoodsDelivery? Delivery { get; set; }  }  // Файл \DataLayer\Models\Accessory.cs  using DataLayer.Models.SpecificTypes;  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.ComponentModel.DataAnnotations.Schema;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Models;  public class Accessory : Goods  {  [MaxLength(ConstValues.AccessoryColorMaxLength)]  public string Color { get; set; }  [MaxLength(ConstValues.AccessorySizeMaxLength)]  public string Size { get; set; }  public AccessorySpecificType SpecificType { get; set; }  }  // Файл \DataLayer\Models\AudioEquipmentUnit.cs  using DataLayer.Models.SpecificTypes;  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations.Schema;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Models;  public class AudioEquipmentUnit : Goods  {  //[Column("audio\_equipment\_unit\_id")]  //public Guid AudioEquipmentUnitId { get; set; }  public AudioEquipmentUnitSpecificType SpecificType { get; set; }  }  // Файл \DataLayer\Models\GoodsDelivery.cs  using DataLayer.NotMapped;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations.Schema;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Models;  public class GoodsDelivery  {  public Guid GoodsDeliveryId { get;set; }  public DateTimeOffset? ExpectedDeliveryDate { get; set; }  [NotMapped]  public DateTime? LocalExpectedDeliveryDate  {  get => ExpectedDeliveryDate?.LocalDateTime;  set  {  ExpectedDeliveryDate = value.LocalToUniversal();  }  }  public DateTimeOffset? ActualDeliveryDate { get; set; }  [NotMapped]  public DateTime? LocalActualDeliveryDate  {  get => ActualDeliveryDate?.LocalDateTime;  set  {  ActualDeliveryDate = value.LocalToUniversal();  }  }  [InverseProperty(nameof(Goods.Delivery))]  public virtual ICollection<MusicalInstrument> MusicalInstruments { get; set; } = [];  [InverseProperty(nameof(Goods.Delivery))]  public virtual ICollection<Accessory> Accessories { get; set; } = [];  [InverseProperty(nameof(Goods.Delivery))]  public virtual ICollection<AudioEquipmentUnit> AudioEquipmentUnits { get; set; } = [];  [InverseProperty(nameof(Goods.Delivery))]  public virtual ICollection<SheetMusicEdition> SheetMusicEditions { get; set; } = [];  }  // Файл \DataLayer\Models\MusicalInstrument.cs  using DataLayer.Models.SpecificTypes;  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.ComponentModel.DataAnnotations.Schema;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Models;  public class MusicalInstrument : Goods  {  public int ReleaseYear { get; set; }  [MaxLength(ConstValues.MusicalInstrumentManufacturerMaxLength)]  public string Manufacturer { get; set; }  public ManufacturerType ManufacturerType { get; set; }  public MusicalInstrumentSpecificType SpecificType { get; set; }  }  // Файл \DataLayer\Models\Sale.cs  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.ComponentModel.DataAnnotations.Schema;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using DataLayer.SupportClasses;  namespace DataLayer.Models;  [Table("sales")]  public class Sale : ISoftDeletable  {  public Guid SaleId { get; set; }  public DateTimeOffset? SaleDate { get; set; }  public DateTimeOffset? ReservationDate { get; set; }  public DateTimeOffset? ReturningDate { get; set; }  #warning is it working correctly?  [NotMapped]  public DateTime? LocalSaleDate  {  get => SaleDate?.LocalDateTime;  set  {  SaleDate = value.LocalToUniversal();  }  }  [NotMapped]  public DateTime? LocalReservationDate  {  get => ReservationDate?.LocalDateTime;  set  {  ReservationDate = value.LocalToUniversal();  }  }  [NotMapped]  public DateTime? LocalReturningDate  {  get => ReturningDate?.LocalDateTime;  set  {  ReturningDate = value.LocalToUniversal();  }  }  #warning how to add references to several tables like ICollection<Goods> Answer: view. Upd: bad answer.  [Required]  public SaleStatus Status { get; set; }  [Required]  public bool IsPaid { get; set; }  public SalePaidBy? PaidBy { get; set; }  public bool SoftDeleted { get; set; }  // relationships  public virtual ICollection<MusicalInstrument> MusicalInstruments { get; set; } = [];  public virtual ICollection<Accessory> Accessories { get; set; } = [];  public virtual ICollection<AudioEquipmentUnit> AudioEquipmentUnits { get; set; } = [];  public virtual ICollection<SheetMusicEdition> SheetMusicEditions { get; set; } = [];  }  // Файл \DataLayer\Models\SaleView.cs  using DataLayer.SupportClasses;  using Microsoft.EntityFrameworkCore;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.ComponentModel.DataAnnotations.Schema;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Models;  [Keyless]  public class SaleView  {  [Column("sale\_id")]  public Guid SaleId { get; set; }  [Column("sale\_date")]  public DateTimeOffset? SaleDate { get; set; }  [Column("reservation\_date")]  public DateTimeOffset? ReservationDate { get; set; }  [Column("returning\_date")]  public DateTimeOffset? ReturningDate { get; set; }  [NotMapped]  public DateTime? LocalSaleDate => SaleDate?.LocalDateTime;  [NotMapped]  public DateTime? LocalReservationDate => ReservationDate?.LocalDateTime;  [NotMapped]  public DateTime? LocalReturningDate => ReturningDate?.LocalDateTime;  [Required]  public int Total { get; set; }  #warning how to add refers to several tables like ICollection<Goods> Answer: view  [Required]  public SaleStatus Status { get; set; }  [Required]  [Column("paid\_by")]  public SalePaidBy PaidBy { get; set; }  [Required]  [Column("goods\_units\_count")]  public int GoodsUnitsCount { get; set; }  [Required]  [Column("is\_paid")]  public bool IsPaid { get; set; }  [NotMapped]  public IReadOnlyCollection<MusicalInstrument> MusicalInstruments { get; set; } = [];  [NotMapped]  public IReadOnlyCollection<Accessory> Accessories { get; set; } = [];  [NotMapped]  public IReadOnlyCollection<AudioEquipmentUnit> AudioEquipmentUnits { get; set; } = [];  [NotMapped]  public IReadOnlyCollection<SheetMusicEdition> SheetMusicEditions { get; set; } = [];  }  // Файл \DataLayer\Models\SheetMusicEdition.cs  using DataLayer.Models.SpecificTypes;  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.ComponentModel.DataAnnotations.Schema;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Models;  public class SheetMusicEdition : Goods  {  [MaxLength(ConstValues.SheetMusicEditionAuthorMaxLength)]  public string? Author { get; set; }  public int ReleaseYear { get; set; }  public SheetMusicEditionSpecificType SpecificType { get; set; }  }  // Файл \DataLayer\Models\SpecificTypes\AccessorySpecificType.cs  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations.Schema;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Models.SpecificTypes;  [Table("accessory\_specific\_types")]  public class AccessorySpecificType : SpecificType  {  }  // Файл \DataLayer\Models\SpecificTypes\AudioEquipmentUnitSpecificType.cs  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations.Schema;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Models.SpecificTypes;  [Table("audio\_equipment\_unit\_specific\_types")]  public class AudioEquipmentUnitSpecificType : SpecificType  {  }  // Файл \DataLayer\Models\SpecificTypes\MusicalInstrumentSpecificType.cs  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations.Schema;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Models.SpecificTypes;  [Table("musical\_instrument\_specific\_types")]  public class MusicalInstrumentSpecificType : SpecificType  {  }  // Файл \DataLayer\Models\SpecificTypes\SheetMusicEditionSpecificType.cs  using DataLayer.SupportClasses;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations.Schema;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Models.SpecificTypes;  [Table("sheet\_music\_edition\_specific\_types")]  public class SheetMusicEditionSpecificType : SpecificType  {  }  // Файл \DataLayer\Models\LinkingTables\AccessorySale.cs  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations.Schema;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Models.LinkingTables;  #warning how to make it automatically  [Table("accessory\_sale")]  public class AccessorySale  {  [Required]  public Guid AccessoryId { get; set; }  public virtual Accessory Accessory { get; set; }  [Required]  public Guid SaleId { get; set; }  public virtual Sale Sale { get; set; }  }  // Файл \DataLayer\Models\LinkingTables\AudioEquipmentUnitSale.cs  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations.Schema;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Models.LinkingTables;  [Table("audio\_equipment\_unit\_sale")]  public class AudioEquipmentUnitSale  {  [Required]  public Guid AudioEquipmentUnitId { get; set; }  public virtual AudioEquipmentUnit AudioEquipmentUnit { get; set; }  [Required]  public Guid SaleId { get; set; }  public virtual Sale Sale { get; set; }  }  // Файл \DataLayer\Models\LinkingTables\GoodsSale.cs  using DataLayer.NotMapped;  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations;  using System.ComponentModel.DataAnnotations.Schema;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Models.LinkingTables;  #warning try to use it instead of several linking tables  //public class GoodsSale  //{  // [Required]  // public Guid GoodsId { get; set; }  // public Goods GoodsUnit { get; set; }  // [Required]  // public Guid SaleId { get; set; }  // public Sale Sale { get; set; }  //}  // Файл \DataLayer\Models\LinkingTables\MusicalInstrumentSale.cs  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations.Schema;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Models.LinkingTables;  [Table("musical\_instrument\_sale")]  public class MusicalInstrumentSale  {  [Required]  public Guid MusicalInstrumentId { get; set; }  public virtual MusicalInstrument MusicalInstrument { get; set; }  [Required]  public Guid SaleId { get; set; }  public virtual Sale Sale { get; set; }  }  // Файл \DataLayer\Models\LinkingTables\SheetMusicEditionSale.cs  using System;  using System.Collections.Generic;  using System.ComponentModel.DataAnnotations.Schema;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Models.LinkingTables;  [Table("sheet\_music\_edition\_sale")]  public class SheetMusicEditionSale  {  [Required]  public Guid SheetMusicEditionId { get; set; }  public virtual SheetMusicEdition SheetMusicEdition { get; set; }  [Required]  public Guid SaleId { get; set; }  public virtual Sale Sale { get; set; }  }  // Файл \DataLayer\Common\DataSeeding.cs  using DataLayer.Models;  using DataLayer.Models.SpecificTypes;  using DataLayer.SupportClasses;  using Microsoft.AspNetCore.Identity;  using Microsoft.EntityFrameworkCore;  using Microsoft.Extensions.Configuration;  using Microsoft.Extensions.DependencyInjection;  using System;  using System.Collections.Generic;  using System.Diagnostics.Metrics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Common;  public class DataSeeding  {  #warning probably use factory here  private static Sale[] Sales = [  new()  {  SaleId = Guid.Parse("01937ce5-61c0-4eaf-8580-aeeb653b2191"),  Status = SaleStatus.Sold,  LocalSaleDate = new DateTime(2023, 10, 1, 10, 49, 07),  PaidBy = SalePaidBy.BankCard,  IsPaid = true  },  new()  {  SaleId = Guid.Parse("0d9ee495-b44c-4465-ba45-6fa6ff562579"),  Status = SaleStatus.Sold,  LocalSaleDate = new DateTime(2023, 1, 19, 17, 19, 11),  PaidBy = SalePaidBy.Cash,  IsPaid = true  },  new()  {  SaleId = Guid.Parse("e4c58ecd-e871-4579-b7bc-58563b67299a"),  Status = SaleStatus.Sold,  LocalSaleDate = new DateTime(2023, 5, 28, 15, 50, 01),  PaidBy = SalePaidBy.BankCard,  IsPaid = true  },  new()  {  SaleId = Guid.Parse("286b3185-c983-4339-86c9-b12fc8fac5e2"),  Status = SaleStatus.Returned,  LocalSaleDate = new DateTime(2023, 3, 1, 13, 20, 35),  PaidBy = SalePaidBy.Cash,  IsPaid = true  },  new()  {  SaleId = Guid.Parse("bf1cf858-f491-4234-9331-0b4abef9f0e8"),  Status = SaleStatus.Reserved,  LocalSaleDate = new DateTime(2023, 7, 9, 10, 11, 35),  PaidBy = SalePaidBy.BankCard,  IsPaid = true  },  ];  #warning seed another data like types of musical instruments like guitars  public static async Task SeedAsync(IServiceProvider services)  {  var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();  await EnsureRolesAsync(roleManager);  var userManager = services.GetRequiredService<UserManager<IdentityUser>>();  await EnsureUsersAsync(userManager, services);  // Unneccesary stuff for app, but it provides visibility for test-review  var context = services.GetRequiredService<MusicalShopDbContext>();  await EnsureMusicalInstrumentsAndTheirTypes(context);  await EnsureAccessoriesAndTheirTypes(context);  await EnsureAudioEquipmentUnitsAndTheirTypes(context);  await EnsureSheetMusicEditionsAndTheirTypes(context);  }  private async static Task EnsureRolesAsync(RoleManager<IdentityRole> roleManager)  {  string[] roles = { CommonNames.AdminRole, CommonNames.SellerRole, CommonNames.ConsultantRole, CommonNames.StockManagerRole };  for (int i = 0; i < 4; i++)  {  var roleExists = await roleManager.RoleExistsAsync(roles[i]);  if (!roleExists)  {  await roleManager.CreateAsync(new IdentityRole { Name = roles[i] });  }  }  }  private async static Task EnsureUsersAsync(UserManager<IdentityUser> userManager, IServiceProvider services)  {  var configuration = services.GetRequiredService<IConfiguration>();  var passwordsSection = configuration.GetSection("DefaultPasswords");  #warning it could have been implemented more, more simply.  string[] emails = [CommonNames.DefaultAdminEmail, CommonNames.DefaultSellerEmail, CommonNames.DefaultConsultantEmail, CommonNames.DefaultStockManagerEmail];  string[] roleNames = [CommonNames.AdminRole, CommonNames.SellerRole, CommonNames.ConsultantRole, CommonNames.StockManagerRole];  for (int i = 0; i < emails.Length; i++)  {  var defaultUser = await userManager.Users  .Where(x => x.UserName == emails[i])  .SingleOrDefaultAsync();  if (defaultUser == null)  {  var user = new IdentityUser { UserName = emails[i], Email = emails[i], EmailConfirmed = true };  await userManager.CreateAsync(user, passwordsSection.GetValue<string>(roleNames[i])!);  await userManager.AddToRoleAsync(user, roleNames[i]);  }  }  return;  }  private async static Task EnsureMusicalInstrumentsAndTheirTypes(MusicalShopDbContext context)  {  if (context.MusicalInstruments.SingleOrDefault(mi => mi.GoodsId == Guid.Parse("05812ce5-61c0-4eaf-8580-aeeb653b2191")) == null)  {  var acousticGuitarType = new MusicalInstrumentSpecificType { Name = "Акустическая гитара" };  var drumsType = new MusicalInstrumentSpecificType { Name = "Барабанная установка" };  var fluteType = new MusicalInstrumentSpecificType { Name = "Флейта" };  var xylophoneType = new MusicalInstrumentSpecificType { Name = "Ксилофон" };  var synthesizerType = new MusicalInstrumentSpecificType { Name = "Синтезатор" };  var instruments = new List<MusicalInstrument>  {  new()  {  GoodsId = Guid.Parse("05812ce5-61c0-4eaf-8580-aeeb653b2191"),  Description = "Акустическая гитара, с вырезом, санберст, Foix",  Name = "FFG-3860C-SB",  ReceiptDate = new DateTimeOffset(new DateTime(2023, 10, 12, 10, 20, 35)),  Manufacturer = "John Spelberg",  ManufacturerType = ManufacturerType.Master,  Price = 9599,  ReleaseYear = 2023,  Status = GoodsStatus.InStock,  SpecificType = acousticGuitarType  },  new()  {  GoodsId = Guid.Parse("9384b7c1-6727-4dd0-88cc-7e1a1d9062cb"),  Description = " Акустическая гитара, без выреза, без санберста, Hoix",  Name = "Kolenval-SB-SUNBRESTLESS",  ReceiptDate = new DateTimeOffset(new DateTime(2024, 10, 12, 10, 20, 35)),  Manufacturer = "John Spelberg",  ManufacturerType = ManufacturerType.Master,  Price = 7499,  ReleaseYear = 2023,  Status = GoodsStatus.Reserved,  SpecificType = acousticGuitarType  },  new()  {  GoodsId = Guid.Parse("148b3083-f123-44df-8fce-af4c7014ac31"),  Description = "Акустическая гитара, без порожков, Nice",  Name = "L529-9X8823L-CCAS3-AR-2IC-3H-1SAEID50-BEZ-POROJKOF-OTVALILIS':(",  ReceiptDate = new DateTimeOffset(new DateTime(2022, 1, 10, 1, 20, 35)),  #warning palindrome  Manufacturer = "Завод гитар имени Инемиратигдова З.",  ManufacturerType = ManufacturerType.Factory,  Price = 3499,  ReleaseYear = 2022,  Status = GoodsStatus.InStock,  SpecificType = acousticGuitarType  },  new()  {  GoodsId = Guid.Parse("0e05ca0d-7e34-4b65-a5fc-3e7b69194390"),  Description = "Акустическая гитара, без розетки, санберст отсутствует, полые порожки, струны из дерева, ж/б гриф, встроенная когтеточка, удобная лежанка и автокормушка с функцией будильника.",  Name = "APPOLON-19-SUNBURSTLESS",  ReceiptDate = null,  Manufacturer = "Завод гитар для котов",  ManufacturerType = ManufacturerType.Factory,  Price = 2399,  ReleaseYear = 2021,  Status = GoodsStatus.AwaitingDelivery,  SpecificType = acousticGuitarType  },  new()  {  GoodsId = Guid.Parse("32eb21b6-6d88-42aa-9326-114297689a59"),  Description = "Акустическая гитара, без выреза, артишок, Belucci",  Name = "LDPWD",  ReceiptDate = null,  Manufacturer = "Завод собачьих гитар",  ManufacturerType = ManufacturerType.Factory,  Price = 2399,  ReleaseYear = 2020,  Status = GoodsStatus.AwaitingDelivery,  SpecificType = acousticGuitarType  },  new()  {  GoodsId = Guid.Parse("6f5c6af2-6fb7-4cf2-8730-5e365d2c1032"),  Description = "Акустическая гитара, без выреза, но с вырезом, проивзедено в США",  Name = "MyFirstGuitar",  ReceiptDate = new DateTimeOffset(new DateTime(2022, 1, 10, 1, 20, 35)),  Manufacturer = "John Maloe",  ManufacturerType = ManufacturerType.Master,  Price = 88990,  ReleaseYear = 2024,  Status = GoodsStatus.InStock,  SpecificType = acousticGuitarType  },  new()  {  GoodsId = Guid.Parse("2b14e3ec-6af4-4094-ba4b-255933603cc9"),  Description = "\*Барабанная дробь\*... Барабанная установка \"Барабанная мечта\" - барабанный рай барабанного любителя.",  Name = "Стукач",  ReceiptDate = new DateTimeOffset(new DateTime(2024, 1, 10, 1, 20, 35)),  Manufacturer = "Барабанный лидер",  ManufacturerType = ManufacturerType.Factory,  Price = 14900,  ReleaseYear = 2019,  Status = GoodsStatus.AwaitingDelivery,  SpecificType = drumsType  },  new()  {  GoodsId = Guid.Parse("a07319f6-4944-4f06-bb1c-b77c27e73b1d"),  Description = "Многослойные барабаны позволят слышать себя непревзойденно.",  Name = "Knocker-knocker",  ReceiptDate = new DateTimeOffset(new DateTime(2024, 1, 10, 1, 20, 35)),  Manufacturer = "Барабань-ка",  ManufacturerType = ManufacturerType.Factory,  Price = 1900,  ReleaseYear = 2019,  Status = GoodsStatus.InStock,  SpecificType = drumsType  },  new()  {  GoodsId = Guid.Parse("ee1c5679-2018-4192-8e02-6efed0ef8c5a"),  Description = "Народный духовой инструмент",  Name = "[oOo]",  ReceiptDate = new DateTimeOffset(new DateTime(2024, 1, 10, 1, 20, 35)),  Manufacturer = "Завод барабанных флейт имени Дыхалова",  ManufacturerType = ManufacturerType.Factory,  Price = 299,  ReleaseYear = 2023,  Status = GoodsStatus.Sold,  Sales = [Sales[0]],  SpecificType = fluteType  },  new()  {  GoodsId = Guid.Parse("9d7e4b3b-cbaa-4327-af1c-1ea3e232d68a"),  Description = "Компактное пианино, 3 режима, подсветка, присутствует нейросеть, позволяющая схватывать колебания головного мозга с целью воспроизведения желаемой мелодии. Сделано в СССР",  Name = "Sntzr-1937",  ReceiptDate = new DateTimeOffset(new DateTime(2023, 1, 10, 1, 20, 35)),  Manufacturer = "Steve Pianoe",  ManufacturerType = ManufacturerType.Master,  Price = 11590,  ReleaseYear = 1975,  Status = GoodsStatus.InStock,  SpecificType = synthesizerType  },  new()  {  GoodsId = Guid.Parse("d7b2ff21-cc80-41fa-bef5-3ca93c5ec4fa"),  Description = "Данный синтезатор изготовлен из нержавеющего пластика, слоновьего зуба и экранированного хлеба. Корпус выполнен в командной строке.",  Name = "Bearded?Bear?Beer?Breed?Bread?",  ReceiptDate = new DateTimeOffset(new DateTime(2023, 1, 10, 1, 20, 35)),  Manufacturer = "Синтезаторы? Производим.",  ManufacturerType = ManufacturerType.Factory,  Price = 2390,  ReleaseYear = 2022,  Status = GoodsStatus.Sold,  Sales = [Sales[3]],  SpecificType = synthesizerType  },  };  context.AddRange(instruments);  context.SaveChanges();  }  }  private async static Task EnsureAccessoriesAndTheirTypes(MusicalShopDbContext context)  {  if (context.Accessories.SingleOrDefault(mi => mi.GoodsId == Guid.Parse("05812ce5-61c0-4eaf-1937-aeeb653b2191")) == null)  {  var chairType = new AccessorySpecificType { Name = "Табуретка регулируемая" };  var keychainType = new AccessorySpecificType { Name = "Брелок" };  var accessories = new List<Accessory>  {  new()  {  GoodsId = Guid.Parse("05812ce5-61c0-4eaf-1937-aeeb653b2191"),  Description = "Круглая табуретка",  Name = "Табуретка",  ReceiptDate = new DateTimeOffset(new DateTime(2023, 10, 12, 10, 20, 35)),  Price = 599,  Status = GoodsStatus.Sold,  SpecificType = chairType,  Color = "Прозрачный",  Sales = [Sales[0]],  Size = "регулировка высоты от 10 до 150 см, 50см радиус седла"  },  new()  {  GoodsId = Guid.Parse("bf73bc1d-5d82-460b-9cf4-cd08e117face"),  Description = "Брелок с граммофоном отлично смотрится на архивных вещах",  Name = "Брелок с граммофоном",  ReceiptDate = new DateTimeOffset(new DateTime(2023, 10, 12, 10, 20, 35)),  Price = 99,  Status = GoodsStatus.Sold,  SpecificType = keychainType,  Sales = [Sales[3]],  Color = "Черно-желтый",  Size = "20см x 0.5см x 3см"  },  new()  {  GoodsId = Guid.Parse("6d4e31a0-9809-44c8-810f-7e0c4f435e03"),  Description = "Набор 3 в 1: пюпитр и каподастр",  Name = "Стартующий гитарист",  ReceiptDate = new DateTimeOffset(new DateTime(2023, 10, 12, 10, 20, 35)),  Price = 699,  Status = GoodsStatus.Sold,  SpecificType = keychainType,  Sales = [Sales[4]],  Color = "Черно-рыжий",  Size = "Высота пюпитра: 30-200см. Каподастр 13см x 1см x 12 см"  },  };  context.AddRange(accessories);  context.SaveChanges();  }  }  private static async Task EnsureAudioEquipmentUnitsAndTheirTypes(MusicalShopDbContext context)  {  #warning not implemented  return;  }  private static async Task EnsureSheetMusicEditionsAndTheirTypes(MusicalShopDbContext context)  {  #warning not implemented  return;  }  }  // Файл \DataLayer\Common\MusicalShopDbContext.cs  using DataLayer.Models;  using DataLayer.Models.LinkingTables;  using DataLayer.Models.SpecificTypes;  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  using EFCore.NamingConventions;  using Microsoft.AspNetCore.Identity;  using Microsoft.AspNetCore.Identity.EntityFrameworkCore;  using Microsoft.EntityFrameworkCore;  using Microsoft.EntityFrameworkCore.Metadata;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Linq.Expressions;  using System.Reflection;  using System.Text;  using System.Threading.Tasks;  namespace DataLayer.Common;  public partial class MusicalShopDbContext : IdentityDbContext<IdentityUser>  {  public virtual DbSet<MusicalInstrument> MusicalInstruments { get; set; }  public virtual DbSet<Accessory> Accessories { get; set; }  public virtual DbSet<AudioEquipmentUnit> AudioEquipmentUnits { get; set; }  public virtual DbSet<SheetMusicEdition> SheetMusicEditions { get; set; }  public virtual DbSet<Sale> Sales { get; set; }  public virtual DbSet<SaleView> SalesView { get; set; }  // linking tables  public virtual DbSet<MusicalInstrumentSale> MusicalInstrumentSale { get; set; }  public virtual DbSet<AccessorySale> AccessorySale { get; set; }  public virtual DbSet<AudioEquipmentUnitSale> AudioEquipmentUnitSale { get; set; }  public virtual DbSet<SheetMusicEditionSale> SheetMusicEditionSale { get; set; }  // specific types  public virtual DbSet<MusicalInstrumentSpecificType> MusicalInstrumentSpecificTypes { get; set; }  public virtual DbSet<AccessorySpecificType> AccessorySpecificTypes { get; set; }  public virtual DbSet<AudioEquipmentUnitSpecificType> AudioEquipmentUnitSpecificTypes { get; set; }  public virtual DbSet<SheetMusicEditionSpecificType> SheetMusicEditionSpecificTypes { get; set; }  public MusicalShopDbContext(DbContextOptions<MusicalShopDbContext> options)  : base(options)  {  //RebuildDb();  }  public MusicalShopDbContext() : base()  {  //RebuildDb();  }  private void RebuildDb()  {  Database.EnsureDeleted();  Database.EnsureCreated();  }  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)  {  #warning how to change the user here corresponding to the current user? or i don't need it? because there (on server) will be authentication  optionsBuilder.UseSnakeCaseNamingConvention();  if (optionsBuilder.IsConfigured)  return;  optionsBuilder.UseMySql("database=musical\_shop;server=localhost;port=3306;user=root;password=password;", ServerVersion.Parse("8.0.39"));  }  protected override void ConfigureConventions(ModelConfigurationBuilder configurationBuilder)  {  configurationBuilder.Properties<GoodsStatus>()  .HaveConversion<string>();  configurationBuilder.Properties<SaleStatus>()  .HaveConversion<string>();  configurationBuilder.Properties<SalePaidBy>()  .HaveConversion<string>();  }  protected override void OnModelCreating(ModelBuilder modelBuilder)  {  #warning if i don't have a personal life, i could rename everything like here  //modelBuilder.Entity<IdentityUser>().ToTable("identity\_users").Property(u => u.UserId).HasColumnName("user\_id");  modelBuilder.Entity<SaleView>(entity =>  {  entity.ToView("sales\_view");  });  //modelBuilder.Entity<SpecificType>(entity =>  //{  // entity.HasIndex(e => e.Name)  // .IsUnique();  //});  modelBuilder.Entity<AccessorySpecificType>();  modelBuilder.Entity<MusicalInstrumentSpecificType>();  modelBuilder.Entity<AudioEquipmentUnitSpecificType>();  modelBuilder.Entity<SheetMusicEditionSpecificType>();  #region many-to-many sales and goods  #warning well, i'm able to implement it, but i wanna something like single linking table like goods\_sale  modelBuilder.Entity<MusicalInstrument>(entity =>  {  entity.HasMany(mi => mi.Sales)  .WithMany(sale => sale.MusicalInstruments)  .UsingEntity<MusicalInstrumentSale>(linkingTable =>  {  linkingTable.HasKey(lt => new { lt.SaleId, lt.MusicalInstrumentId });  linkingTable.HasOne(e => e.MusicalInstrument)  .WithMany()  .HasForeignKey(e => e.MusicalInstrumentId)  .HasConstraintName("FK\_sale\_musical\_instrument\_id");  linkingTable.HasOne(e => e.Sale)  .WithMany()  .HasForeignKey(e => e.SaleId)  .HasConstraintName("FK\_sale\_musical\_instrument\_sale\_id");  });  #warning implement via reflection foreach (var goodsUnit in goods)  entity.HasOne(a => a.SpecificType)  .WithMany()  .HasForeignKey(a => a.SpecificTypeId);  });  modelBuilder.Entity<Accessory>(entity =>  {  entity.HasMany(a => a.Sales)  .WithMany(sale => sale.Accessories)  .UsingEntity<AccessorySale>(linkingTable =>  {  linkingTable.HasKey(lt => new { lt.SaleId, lt.AccessoryId });  linkingTable.HasOne(e => e.Accessory)  .WithMany()  .HasForeignKey(e => e.AccessoryId)  .HasConstraintName("FK\_sale\_accessory\_id");  linkingTable.HasOne(e => e.Sale)  .WithMany()  .HasForeignKey(e => e.SaleId)  .HasConstraintName("FK\_sale\_accessory\_sale\_id");  });  entity.HasOne(a => a.SpecificType)  .WithMany()  .HasForeignKey(a => a.SpecificTypeId);  });  modelBuilder.Entity<AudioEquipmentUnit>(entity =>  {  entity.HasMany(aeu => aeu.Sales)  .WithMany(sale => sale.AudioEquipmentUnits)  .UsingEntity<AudioEquipmentUnitSale>(linkingTable =>  {  linkingTable.HasKey(lt => new { lt.SaleId, lt.AudioEquipmentUnitId });  linkingTable.HasOne(e => e.AudioEquipmentUnit)  .WithMany()  .HasForeignKey(e => e.AudioEquipmentUnitId)  .HasConstraintName("FK\_sale\_aeu\_id");  linkingTable.HasOne(e => e.Sale)  .WithMany()  .HasForeignKey(e => e.SaleId)  .HasConstraintName("FK\_sale\_aeu\_sale\_id");  });  entity.HasOne(a => a.SpecificType)  .WithMany()  .HasForeignKey(a => a.SpecificTypeId);  });  modelBuilder.Entity<SheetMusicEdition>(entity =>  {  entity.HasMany(sme => sme.Sales)  .WithMany(sale => sale.SheetMusicEditions)  .UsingEntity<SheetMusicEditionSale>(linkingTable =>  {  linkingTable.HasKey(lt => new { lt.SaleId, lt.SheetMusicEditionId });  linkingTable.HasOne(e => e.SheetMusicEdition)  .WithMany()  .HasForeignKey(e => e.SheetMusicEditionId)  .HasConstraintName("FK\_sale\_sme\_id");  linkingTable.HasOne(e => e.Sale)  .WithMany()  .HasForeignKey(e => e.SaleId)  .HasConstraintName("FK\_sale\_sme\_sale\_id");  });  entity.HasOne(a => a.SpecificType)  .WithMany()  .HasForeignKey(a => a.SpecificTypeId);  });  #endregion  base.OnModelCreating(modelBuilder);  //var entities = modelBuilder.Model.GetEntityTypes();  //foreach (var entity in entities)  //{  // ConfigurePropertiesViaFluentApi(entity);  //}  }  private void ConfigurePropertiesViaFluentApi(IMutableEntityType entity)  {  var properties = entity.GetProperties();  foreach (var property in properties)  {  switch (property.Name)  {  case nameof(ISoftDeletable.SoftDeleted):  {  #warning i'm not sure it's working  var methodToCall = typeof(MusicalShopDbContext)  .GetMethod("GetSoftDeletedFilter", BindingFlags.NonPublic | BindingFlags.Static)!  .MakeGenericMethod(entity.ClrType);  var filter = methodToCall.Invoke(null, []);  entity.SetQueryFilter((LambdaExpression)filter!);  break;  }  case nameof(Goods.Price):  {  #warning how to do it (now i've implemented it via workaround)  //priceProperty.;  break;  }  default:  break;  }  }  }  private static Expression<Func<TEntity, bool>> GetSoftDeletedFilter<TEntity>()  where TEntity : class, ISoftDeletable  {  Expression<Func<TEntity, bool>> filter = e => !e.SoftDeleted;  return filter;  }  }  // Файл \ConstNames\Cmd.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Common;  public class Cmd  {  public const string MysqldumpArguments = "-u{0} -p{1} -h {2} --ignore-table={3}.\_\_efmigrationshistory --ignore-table={3}.aspnetroleclaims --ignore-table={3}.aspnetroles --ignore-table={3}.aspnetuserclaims --ignore-table={3}.aspnetuserlogins --ignore-table={3}.aspnetuserroles --ignore-table={3}.aspnetusers --ignore-table={3}.aspnetusertokens --add-drop-table --routines --databases {3} --result-file=\"{4}\"";  public const string MysqRestoreBackupArgs = "-u{0} -p{1} --execute=\"{2}\"";  }  // Файл \ConstNames\CommonExtensions.cs  using System;  using System.Collections;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Reflection;  using System.Runtime.Intrinsics.X86;  using System.Security.Principal;  using System.Text;  using System.Threading.Tasks;  namespace Common;  public static class CommonExtensions  {  public static string Humanize(this bool value)  {  return value ? "Да" : "Нет";  }  public static string ToStringOrDefaultValue<T>(this T value)  {  return value?.ToString() ?? "-";  }  public static string ToMoney(this int number) =>  number.ToString() + " руб.";  public static DateTimeOffset? LocalToUniversal(this DateTime? dateTime) => dateTime is null ? null : (DateTimeOffset?)new DateTimeOffset((DateTime)dateTime).ToUniversalTime();  public static string NameToLowerMysql(this Type type)  {  string typeName = type.Name;  var stringBuilder = new StringBuilder();  stringBuilder.Append(char.ToLower(typeName[0]));  for(int i = 1; i < typeName.Length; i++)  {  if (char.IsUpper(typeName[i]))  stringBuilder.Append("\_");  stringBuilder.Append(typeName[i]);  }  return stringBuilder.ToString();  }  }  // Файл \ConstNames\CommonNames.cs  namespace Common;  public static class CommonNames  {  public const string DefaultAdminEmail = "admin@koshka.prosrochka";  public const string DefaultSellerEmail = "seller@koshka.prosrochka";  public const string DefaultConsultantEmail = "consultant@koshka.prosrochka";  public const string DefaultStockManagerEmail = "stockmanager@koshka.prosrochka";  public const string AdminRole = "Администратор";  public const string SellerRole = "Продавец";  public const string ConsultantRole = "Консультант";  public const string StockManagerRole = "Менеджер складского учета";  public const string NotExistingGuid = "";  public const string GoodsIdsInCartKey = "cart-with-goods";  public const char GoodsIdSeparator = ',';  public const char GoodsIdAndKindSeparator = '|';  public const string MusicalInstruments = "Музыкальные инструменты";  public const string AudioEquipmentUnits = "Аудиооборудование";  public const string Accessories = "Аксессуары";  public const string SheetMusicEditions = "Нотные издания";  public const string FieldIsRequiredMessageRu = "Значение поля \"{0}\" не может быть пустым";  public const string MaxLengthViolationMessageRu = "Значение поля \"{0}\" не может быть длиннее {1} символов";  public const string HtmlTagsViolationMessageRu = "Значение поля \"{0}\" не может содержать HTML-теги";  }  // Файл \ConstNames\ConstValues.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Common;  public class ConstValues  {  public const string BackupDateTimeFormat = "yyyy\_MM\_dd HH\_mm\_ss";  public const int GoodsDescriptionMaxLength = 500;  public const int GoodsNameMaxLength = 100;  public const int GoodsPriceMinValue = 1;  public const int GoodsPriceMaxValue = int.MaxValue;  public const int GoodsSpecificTypeMaxLength = 100;  public const int AccessoryColorMaxLength = 100;  public const int AccessorySizeMaxLength = 200;  public const int SheetMusicEditionAuthorMaxLength = 100;  public const int MusicalInstrumentManufacturerMaxLength = 100;  }  // Файл \ConstNames\SqlStatements.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Common;  public class SqlStatements  {  #region Migrations statements  public const string DropSalesView = @"DROP VIEW `sales\_view`;";  public const string DropTotalPriceFunction = @"DROP FUNCTION `total\_price`;";  public const string DropTotalGoodsUnitsCountFunction = @"DROP FUNCTION `total\_price`;";  public const string CreateSalesViewV1 =  @"CREATE VIEW `sales\_view` AS  SELECT `sale\_id`, `sale\_date`, `reservation\_date`, `returning\_date`, `status`, total\_price(sale\_id) AS `total`, `paid\_by`, total\_goods\_units\_count(sale\_id) as `goods\_units\_count`, `is\_paid`  FROM `sales`;";  public const string CreateTotalGoodsUnitsCountV1 =  @"CREATE FUNCTION total\_goods\_units\_count(sale\_id char(36))  RETURNS INT  NOT DETERMINISTIC  READS SQL DATA  BEGIN  DECLARE total INT;  SET total = IFNULL((SELECT COUNT(\*) FROM musical\_instrument\_sale AS mis WHERE mis.sale\_id = sale\_id), 0);  SET total = total + IFNULL((SELECT COUNT(\*) FROM accessory\_sale AS acs WHERE acs.sale\_id = sale\_id), 0);  SET total = total + IFNULL((SELECT COUNT(\*) FROM sheet\_music\_edition\_sale AS smes WHERE smes.sale\_id = sale\_id), 0);  SET total = total + IFNULL((SELECT COUNT(\*) FROM audio\_equipment\_unit\_sale AS aeus WHERE aeus.sale\_id = sale\_id), 0);  RETURN total;  END;";  public const string CreateTotalPriceV1 = @"CREATE FUNCTION total\_price(sale\_id char(36))  RETURNS INT  NOT DETERMINISTIC  READS SQL DATA  BEGIN  DECLARE total INT;  SET total = IFNULL((SELECT SUM(`mi`.`price`) FROM `musical\_instrument\_sale` AS `mis` LEFT JOIN `musical\_instruments` AS `mi` ON `mi`.`goods\_id` = `mis`.`musical\_instrument\_id` WHERE `mis`.`sale\_id` = `sale\_id`), 0);  SET total = total + IFNULL((SELECT SUM(`goods`.`price`) FROM `accessory\_sale` AS `linking\_table` LEFT JOIN `accessories` AS `goods` ON `goods`.`goods\_id` = `linking\_table`.`accessory\_id` WHERE `linking\_table`.`sale\_id` = `sale\_id`), 0);  SET total = total + IFNULL((SELECT SUM(`goods`.`price`) FROM `sheet\_music\_edition\_sale` AS `linking\_table` LEFT JOIN `sheet\_music\_editions` AS `goods` ON `goods`.`goods\_id` = `linking\_table`.`sheet\_music\_edition\_id` WHERE `linking\_table`.`sale\_id` = `sale\_id`), 0);  SET total = total + IFNULL((SELECT SUM(`goods`.`price`) FROM `audio\_equipment\_unit\_sale` AS `linking\_table` LEFT JOIN `audio\_equipment\_units` AS `goods` ON `goods`.`goods\_id` = `linking\_table`.`audio\_equipment\_unit\_id` WHERE `linking\_table`.`sale\_id` = `sale\_id`), 0);  RETURN total;  END;";  #endregion  #region Backup statements  public const string RestoreDatabaseFromBackup = "SOURCE {1}; ";  #endregion  #region Triggers  public const string SalesUpdateTriggerDrop = @"DROP TRIGGER IF EXISTS `sale\_paid\_by\_check\_UPD`;";  public const string SalesUpdateTriggerCreateV1 = @"  #CREATE TRIGGER `sale\_paid\_by\_check\_INS`  #BEFORE INSERT ON `sales`  #FOR EACH ROW  #BEGIN  # IF(!is\_correct\_sale\_paid\_by(new.paid\_by, new.is\_paid)) THEN  # SIGNAL SQLSTATE '40000';  # END IF;  #END;";  public const string SalesInsertTriggerDrop = @"DROP TRIGGER IF EXISTS `sale\_paid\_by\_check\_INS`;";  // TODO add another check for status and uncomment  public const string SalesInsertTriggerCreateV1 = @"  #CREATE TRIGGER `sale\_paid\_by\_check\_UPD`  #BEFORE UPDATE ON `sales`  #FOR EACH ROW  #BEGIN  # IF(!is\_correct\_sale\_paid\_by(new.paid\_by, new.is\_paid)) THEN  # SIGNAL SQLSTATE '40000';  # END IF;  #END;";  public const string IsCorrectSalePaidByFuncDrop = @"DROP FUNCTION IF EXISTS `is\_correct\_sale\_paid\_by`;";  public const string IsCorrectSalePaidByFuncCreateV1 = @"  #CREATE FUNCTION is\_correct\_sale\_paid\_by(paid\_by VARCHAR(8), is\_paid BOOL)  #RETURNS BOOL  #DETERMINISTIC  #CONTAINS SQL  #SQL SECURITY DEFINER  #BEGIN  # IF(is\_paid = true) THEN  # RETURN paid\_by IS NOT NULL;  # ELSE  # RETURN paid\_by IS NULL;  # END IF;  #END;";  #endregion  }  // Файл \ConsoleApp latch\Program.cs  // See https://aka.ms/new-console-template for more information  Console.WriteLine("Hello, World!");  // Файл \BusinessLogic\GlobalUsings.cs  global using Common;  // Файл \BusinessLogic\Sales\CreateSaleAsNotPaidAction.cs  using BizLogicBase.Common;  using BizLogicBase.Validation;  using BusinessLogicLayer.Sales.Dto;  using DataLayer.Common;  using DataLayer.Models;  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  using DbAccessLayer;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace BusinessLogicLayer.Sales;  public class CreateSaleAsNotPaidAction(SalesDbAccess dbAccess) : ErrorAdder, IBizAction<CreateSaleDto, Task<Guid?>>  {  /// <summary>  /// When <paramref name="dto"/>.PaidBy is <c>null</c>, this method treat sale as a reservation, otherwise as a sale.  /// </summary>  /// <param name="dto"></param>  /// <returns></returns>  public async Task<Guid?> Action(CreateSaleDto dto)  {  if (dto.GoodsForSale.Count == 0)  {  AddError("Список товаров пуст.");  return null;  }  foreach (var goodsUnit in dto.GoodsForSale)  {  ValidateGoodsUnit(goodsUnit);  goodsUnit.Status = GoodsStatus.AwaitingPayment;  }  // TODO fix bug when goods is in cart but not in session cart  //foreach (var goods in dto.GoodsForSale)  //{  // goods.Status = GoodsStatus.InCart;  //}  if (HasErrors)  return null;  var sale = new Sale()  {  LocalSaleDate = DateTime.UtcNow,  Status = SaleStatus.YetNotPaid  };  dbAccess.CreateSaleAndUpdateGoods(sale, dto.GoodsForSale);  return sale.SaleId;  }  public void ValidateGoodsUnit(Goods goodsUnit)  {  if (goodsUnit.Status != GoodsStatus.InCart)  AddError("В корзину находится товар, статус которого не \"В корзине\"");  if (goodsUnit.SoftDeleted)  AddError("В корзину добавлен удаленный товар");  // TODO load delivery here  // TODO uncomment  //if (goodsUnit.Delivery?.LocalActualDeliveryDate is null || goodsUnit.ReceiptDate is null)  // AddError("В корзину добавлен непоступивший на склад товар");  if (goodsUnit.Price <= 0)  AddError("В корзину добавлен товар с некорректной ценой. Цена не может быть меньше или равна 0");  // if goods unit has a sale, it must be returned. otherwise this loop won't be executed  // TODO handle reserved sale.  foreach (var sale in goodsUnit.Sales)  if (sale.Status != SaleStatus.Returned)  AddError("В корзину добавлен товар, который входит в другую продажу и не может быть продан");  }  }  // Файл \BusinessLogic\Sales\Dto\CreateSaleDto.cs  using DataLayer.NotMapped;  using DataLayer.SupportClasses;  namespace BusinessLogicLayer.Sales.Dto;  public record class CreateSaleDto(List<Goods> GoodsForSale);  // Файл \BusinessLogic\Admin\AddUserAction.cs  using BizLogicBase.Common;  using BizLogicBase.Validation;  using BusinessLogicLayer.Admin.Dto;  using DataLayer.Models;  using DbAccessLayer.Admin;  using Microsoft.AspNetCore.Identity;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace BusinessLogicLayer.Admin;  #warning this is a CRUD, so it should be in service layer  public class AddUserAction(UserDbAccess dbAccess) : ErrorAdder, IBizAction<NewUserDto, Task<string?>>  {  public async Task<string?> Action(NewUserDto dto)  {  var newUser = new IdentityUser();  if (string.IsNullOrWhiteSpace(dto.UserName))  AddError("Имя пользователя не может быть пустым");  else  {  var normalizedUserName = dto.UserName.ToUpper();  if (!await dbAccess.IsUniqueNormalizedUserName(normalizedUserName))  AddError("Данное имя пользователя уже используется", nameof(IdentityUser.UserName));  else  {  newUser.UserName = dto.UserName;  newUser.NormalizedUserName = normalizedUserName;  }  }  if (string.IsNullOrWhiteSpace(dto.Password))  AddError("Пароль не может быть пустым");  newUser.Email = dto.Email;  newUser.EmailConfirmed = dto.EmailConfirmed;  newUser.PhoneNumber = dto.PhoneNumber;  newUser.PhoneNumberConfirmed = dto.PhoneNumberConfirmed;  return await dbAccess.AddUser(newUser, dto.Password);  }  }  // Файл \BusinessLogic\Admin\UpdateUserAction.cs  using System;  using System.Collections.Generic;  using System.Collections.Immutable;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using BizLogicBase;  using BizLogicBase.Common;  using BizLogicBase.Validation;  using BusinessLogicLayer.Admin.Dto;  using DataLayer.Models;  using DbAccessLayer.Admin;  namespace BusinessLogicLayer.Admin;  public class UpdateUserAction(UserDbAccess dbAccess) : ErrorAdder, IBizAction<UpdateUserDto, Task<string?>>  {  public async Task<string?> Action(UpdateUserDto dto)  {  var user = await dbAccess.GetUserInfo(dto.Id);  if (user == null)  {  AddError("Пользователь с таким идентификатором не найден", nameof(user.Id));  return null;  }  if (string.IsNullOrWhiteSpace(user.UserName))  AddError("Имя пользователя не может быть пустым", nameof(user.UserName));  else  {  var normalizedUserName = dto.UserName!.ToUpper();  if (!await dbAccess.IsUniqueNormalizedUserName(normalizedUserName, user.Id))  AddError("Пользователь с таким именем уже существует", nameof(user.UserName));  else  {  user.UserName = dto.UserName;  user.NormalizedUserName = normalizedUserName;  }  }  if (dto.EmailConfirmed && string.IsNullOrWhiteSpace(dto.Email))  AddError("Электронная почта указана как подтврежденная, но ее значение отсутствует", nameof(user.EmailConfirmed), nameof(user.Email));  if (!string.IsNullOrWhiteSpace(dto.Email))  {  var normalizedEmail = dto.Email.ToUpper();  if (!await dbAccess.IsUniqueNormalizedEmail(normalizedEmail, user.Id))  AddError("Данная электронная почта уже используется", nameof(user.Email));  else  {  #warning js to validate  user.Email = dto.Email;  user.NormalizedEmail = normalizedEmail;  user.EmailConfirmed = dto.EmailConfirmed;  }  }  else  {  user.Email = null;  user.NormalizedEmail = null;  user.EmailConfirmed = false;  }  if (dto.PhoneNumber == null)  {  if (dto.PhoneNumberConfirmed)  AddError("Номер телефона указан как подтврежденный, но его значение отсутствует", nameof(user.PhoneNumberConfirmed), nameof(user.PhoneNumber));  }  else if (dto.PhoneNumber.Length != 11)  AddError("Номер телефона указан не полностью", nameof(user.PhoneNumber));  user.PhoneNumber = dto.PhoneNumber;  user.PhoneNumberConfirmed = dto.PhoneNumberConfirmed;  user.TwoFactorEnabled = dto.TwoFactorEnabled;  user.LockoutEnd = dto.LockoutEnd != null ? ((DateTime)dto.LockoutEnd).ToUniversalTime() : null;  user.LockoutEnabled = dto.LockoutEnabled;  #warning how it's working?  return user.Id;  }  }  // Файл \BusinessLogic\Admin\Dto\NewUserDto.cs  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace BusinessLogicLayer.Admin.Dto;  #warning i think this is inappropriate place for dto  public record class NewUserDto(string? UserName, string? Email, bool EmailConfirmed, string? PhoneNumber, bool PhoneNumberConfirmed, string Password);  // Файл \BusinessLogic\Admin\Dto\UpdateUserDto.cs  namespace BusinessLogicLayer.Admin.Dto;  #warning i think this is inappropriate place for dto  public class UpdateUserDto  {  public Guid Id { get; set; }  public string? UserName { get; set; }  public bool EmailConfirmed { get; set; }  public string? Email { get; set; }  public string? PhoneNumber { get; set; }  public bool PhoneNumberConfirmed { get; set; }  public bool TwoFactorEnabled { get; set; }  public DateTime? LockoutEnd { get; set; }  public bool LockoutEnabled { get; set; }  }  // Файл \BizLogicBase\Validation\ErrorAdder.cs  using System.Collections.Immutable;  using System.ComponentModel.DataAnnotations;  namespace BizLogicBase.Validation;  public abstract class ErrorAdder : ErrorStorage, IErrorAdder  {  private readonly List<ValidationResult> \_errors = [];  public override IImmutableList<ValidationResult> Errors => \_errors.ToImmutableList();  public void AddError(string errorMessage, params string[] propertyNames)  {  \_errors.Add(new ValidationResult(errorMessage, propertyNames));  }  }  // Файл \BizLogicBase\Validation\ErrorStorage.cs  using System.Collections.Immutable;  using System.ComponentModel.DataAnnotations;  namespace BizLogicBase.Validation;  public abstract class ErrorStorage : IErrorStorage  {  public virtual IImmutableList<ValidationResult> Errors { get; protected set; } = new List<ValidationResult>().ToImmutableList();  public virtual bool HasErrors => Errors.Any();  }  // Файл \BizLogicBase\Validation\IErrorAdder.cs  namespace BizLogicBase.Validation;  public interface IErrorAdder : IErrorStorage  {  void AddError(string errorMessage, params string[] propertyNames);  }  // Файл \BizLogicBase\Validation\IErrorStorage.cs  using System.Collections.Immutable;  using System.ComponentModel.DataAnnotations;  namespace BizLogicBase.Validation;  public interface IErrorStorage  {  IImmutableList<ValidationResult> Errors { get; }  bool HasErrors { get; }  }  // Файл \BizLogicBase\Common\IBizAction.cs  using System.Collections.Immutable;  using System.ComponentModel.DataAnnotations;  namespace BizLogicBase.Common;  public interface IBizAction<in TIn, out TOut>  {  IImmutableList<ValidationResult> Errors { get; }  bool HasErrors { get; }  TOut Action(TIn argument);  }  // Файл \BizLogicBase\Common\RunnerTransact2WriteDb.cs  using BizLogicBase.Validation;  using Microsoft.EntityFrameworkCore;  namespace BizLogicBase.Common;  public class RunnerTransact2WriteDb<TIn, TPass, TOut>(IBizAction<TIn, TPass> actionPart1, IBizAction<TPass, TOut> actionPart2, DbContext context) : ErrorStorage  where TPass : class?  where TOut : class?  {  private readonly IBizAction<TIn, TPass> actionPart1 = actionPart1;  private readonly IBizAction<TPass, TOut> actionPart2 = actionPart2;  private readonly DbContext context = context;  public TOut Run(TIn argument)  {  using (var transaction = context.Database.BeginTransaction())  {  var pass1 = RunPart(actionPart1, argument);  if (HasErrors)  return null!;  var result = RunPart(actionPart2, pass1);  if (!HasErrors)  transaction.Commit();  return result;  }  }  private TPartOut RunPart<TPartIn, TPartOut>(IBizAction<TPartIn, TPartOut> actionPart, TPartIn argument)  where TPartOut : class?  {  var result = actionPart.Action(argument);  Errors = actionPart.Errors;  if (!HasErrors)  context.SaveChanges();  return result;  }  }  // Файл \BizLogicBase\Common\RunnerTransact3WriteDb.cs  using BizLogicBase.Validation;  using Microsoft.EntityFrameworkCore;  namespace BizLogicBase.Common;  public class RunnerTransact3WriteDb<TIn, TPass1, TPass2, TOut>(IBizAction<TIn, TPass1> actionPart1, IBizAction<TPass1, TPass2> actionPart2, IBizAction<TPass2, TOut> actionPart3, DbContext context) : ErrorStorage  where TPass1 : class?  where TPass2 : class?  where TOut : class?  {  private readonly IBizAction<TIn, TPass1> actionPart1 = actionPart1;  private readonly IBizAction<TPass1, TPass2> actionPart2 = actionPart2;  private readonly IBizAction<TPass2, TOut> actionPart3 = actionPart3;  private readonly DbContext context = context;  public TOut Run(TIn argument)  {  using (var transaction = context.Database.BeginTransaction())  {  TPass1 pass1 = RunPart(actionPart1, argument);  if (HasErrors)  return null!;  TPass2 pass2 = RunPart(actionPart2, pass1);  if (HasErrors)  return null!;  TOut result = RunPart(actionPart3, pass2);  if (!HasErrors)  transaction.Commit();  return result;  }  }  private TPartOut RunPart<TPartIn, TPartOut>(IBizAction<TPartIn, TPartOut> actionPart, TPartIn argument)  {  TPartOut result = actionPart.Action(argument);  Errors = actionPart.Errors;  if (!HasErrors)  {  context.SaveChanges();  }  return result;  }  }  // Файл \BizLogicBase\Common\RunnerWriteDb.cs  using BizLogicBase.Validation;  using Microsoft.EntityFrameworkCore;  using System;  using System.Collections.Generic;  using System.Collections.Immutable;  using System.ComponentModel.DataAnnotations;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace BizLogicBase.Common;  public class RunnerWriteDb<TIn, TOut>(DbContext context, IBizAction<TIn, TOut> action) : ErrorStorage  where TOut : class?  {  public override IImmutableList<ValidationResult> Errors => action.Errors;  public TOut Run(TIn argument)  {  var result = action.Action(argument);  if (!action.HasErrors)  context.SaveChanges();  return result;  }  }  // Файл \BizDbAccess\Class1.cs  namespace BizDbAccess;  public class Class1  {  }  // Файл \BaseAppConfiguration\BasicApp.cs  using DataLayer.Common;  using Microsoft.AspNetCore.Identity;  using Microsoft.EntityFrameworkCore;  using ServiceLayer;  using DbAccessLayer;  using System.Reflection;  using System.Diagnostics;  namespace BasicAppConfiguration;  /// <summary>  /// This class aids to solve the DRY problem when i had to configure identity+services+DbContext in both WebApi and MusicalShopApp. I avoid the copy+past stuff in this way, which is considered to be a good manners...  /// </summary>  public static class BasicApp  {  private static string AppSettingsFilePath  {  get  {  //"../appsettings.json"  var currentDirectory = new DirectoryInfo(Environment.CurrentDirectory);  DirectoryInfo parent = currentDirectory;  do  {  parent = parent.Parent!;  }  while (parent is not null && parent.Name.ToLower() != "diploma (musical shop)");  return parent is null  ? throw new Exception("Parent was null")  : Path.Combine(parent.FullName, "BaseAppConfiguration", "appsettings.json");  }  }  public static WebApplicationBuilder CreateBuilder(string[] args)  {  #warning rub it off  const bool USE\_MYSQL = true;  const bool USE\_SQL\_SERVER = false;  const bool USE\_SQLITE = false;  var builder = WebApplication.CreateBuilder(args);  builder.Configuration.AddJsonFile(AppSettingsFilePath);  var connectionString = builder.Configuration.GetConnectionString(USE\_MYSQL ? "MySql" : USE\_SQL\_SERVER ? "SqlServer" : USE\_SQLITE ? "Sqlite" : "") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");  builder.Services.AddDbContext<MusicalShopDbContext>(options =>  {  if (USE\_MYSQL)  options.UseMySql(connectionString, ServerVersion.Parse("8.0.39"));  else if (USE\_SQL\_SERVER)  options.UseSqlServer(connectionString);  else  options.UseSqlite(connectionString);  });  builder.Services.AddDatabaseDeveloperPageExceptionFilter();  builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)  .AddRoles<IdentityRole>()  .AddRoleManager<RoleManager<IdentityRole>>()  .AddEntityFrameworkStores<MusicalShopDbContext>();  builder.Services.RegisterServiceLayer();  builder.Services.RegisterDbAccessLayer();  return builder;  }  }  // Файл \BaseAppConfiguration\Program.cs  namespace BasicAppConfiguration;  public class Program  {  public static void Main(string[] args)  =>  throw new NotImplementedException();  } |

# ПРИЛОЖЕНИЕ В

Тест-кейсы для тестирования программного обеспечения