**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«РОССИЙСКАЯ АКАДЕМИЯ НАРОДНОГО ХОЗЯЙСТВА И**

**ГОСУДАРСТВЕННОЙ СЛУЖБЫ**

**при ПРЕЗИДЕНТЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

Смоленский филиал РАНХиГС

**ОТЧЕТ**

по производственной практике (по профилю специальности)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студента | 37/11-К/ИТО | группы | 3 | курса |

|  |
| --- |
| Махницкого Дмитрия Сергеевича |
| (Фамилия Имя Отчество) |

|  |  |  |
| --- | --- | --- |
| по специальности: | 09.02.07 Информационные системы и | |
| программирование | |

|  |  |
| --- | --- |
| Наименование профессионального модуля: | ПМ.01 Разработка модулей |
| программного обеспечения для компьютерных систем | |

|  |
| --- |
| ООО «Твинс» |
| (место прохождения практики) |

Сроки прохождения практики: с «22» ноября 2024 г. по «06» декабря 2024 г.

|  |  |  |  |
| --- | --- | --- | --- |
| Руководитель практики от Филиала |  |  | Гавриленкова А.А. |
|  | (подпись) |  | (ФИО руководителя) |
| Руководитель практики от организации |  |  | Соколов А.В. |
|  | (подпись) |  | (ФИО руководителя) |
|  | М.П. |  |  |

Смоленск, 2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc184218537)

[1 ТЕХНИКО-ЭКОНОМИЧЕСКАЯ ХАРАКТЕРИСТИКА ООО «ТВИНС» 5](#_Toc184218538)

[2 АНАЛИЗ БИЗНЕС-ПРОЦЕССА «ОФОРМЛЕНИЕ КОНТРАКТОВ С НОВЫМИ КЛИЕНТАМИ» В ООО «ТВИНС» 9](#_Toc184218539)

[3 РЕКОМЕНДАЦИИ ПО СОВЕРШЕНСТВОВАНИЮ БИЗНЕС-ПРОЦЕССА «ОФОРМЛЕНИЕ КОНТРАКТОВ С НОВЫМИ КЛИЕНТАМИ» В ООО «ТВИНС» 13](#_Toc184218540)

[ЗАКЛЮЧЕНИЕ 14](#_Toc184218541)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 15](#_Toc184218542)

[ПРИЛОЖЕНИЕ А Формулы для проведения количественного анализа 16](#_Toc184218543)

[ПРИЛОЖЕНИЕ Б Программный код 17](#_Toc184218544)

ВВЕДЕНИЕ

Целью проведения производственной практики (по профилю специальности) являются ознакомление с основными видами и задачами будущей профессиональной деятельности, а также систематизация, обобщение, закрепление и углубление теоретических знаний и умений, приобретенных при освоении основной образовательной программы.

Для достижения поставленной цели необходимо решить следующие задачи:

- составить технико-экономическую характеристику ООО «Твинс»;

- провести анализ бизнес-процесса «Оформление контрактов с новыми клиентами» в ООО «Твинс»;

- разработать рекомендации по совершенствованию бизнес-процесса «Оформление контрактов с новыми клиентами» в ООО «Твинс».

Тема индивидуального задания: «Анализ бизнес-процесса «Оформление контрактов с новыми клиентами»».

Актуальность темы индивидуального задания обосновывается тем, что оформление контрактов с новыми клиентами – это часть деятельности любой компании. Без оформления контрактов с клиентами компания не сможет функционировать на рынке и быть конкурентноспособной.

Объект исследования: ООО «Твинс».

Предмет исследования: бизнес-процесс «Оформление контрактов с новыми клиентами» в ООО «Твинс».

В числе информационных источников исследования использованы:

- научные источники: данные и сведения из учебников, монографий, учебных пособий;

- открытые Интернет-источники: веб-сайт организации ООО «Твинс».

Методы исследования, используемые в данной работе: теоретический анализ источников, метод изучения, метод моделирования, математические методы.

1 ТЕХНИКО-ЭКОНОМИЧЕСКАЯ ХАРАКТЕРИСТИКА ООО «ТВИНС»

Основное направление деятельности организации ООО «Твинс» (также зарегистрированной под товарным знаком «WebCanape») – разработка веб-сайтов для малого и среднего бизнеса [1]. Основной вид деятельности согласно коду ОКВЭД-2: разработка компьютерного программного обеспечения (62.01). Далее перечислены различные аспекты информации об организации:

- ИНН: 6731067957;

- КПП 673201001;

- ОГРН 1086731003732;

- дата образования 12.03.2008;

- юридический адрес: 214000, Смоленская Область, г. Смоленск, ул. Карла Маркса, д. 12;

- руководитель: Богомолов Алексей Васильевич (Директор)

- дата аккредитации: 25.02.2013;

- количество сотрудников: 65;

- номер и дата регистрации товарного знака «WebCanape»: 539456 от 10.04.2015.

Направления деятельности ООО «Твинс», осуществляемые в области информационных технологий, в части оказываемых услуг и разрабатываемых продуктов перечислены следующим образом: разработка, адаптация и модернизация программного обеспечения для ЭВМ и программирование баз данных с целью создания сайтов и веб приложений, поискового продвижения сайтов, оказания услуг по технической поддержке и модернизации сайтов, а так же с целью размещения обслуживания сайтов на сервере.

Миссия компании – помогать развивать бизнес, используя комплексно современные эффективные IT-решения, инструменты веб-разработки и интернет-маркетинга, а главная цель – удовлетворение требований и ожиданий заказчиков (клиентов) посредством предоставления им услуг по разработке и поисковому продвижению (SEO) сайтов, веб-приложений, on-line-сервисов, по интернет-маркетингу и автоматизации их бизнес-процессов.

На рисунке 1.1 изображена организационная структура ООО «Твинс», составленная в соответствии с информацией на сайте организации [2].

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

Рисунок 1.1 – Организационная структура ООО «Твинс»

Согласно данной структуре, организация состоит из 6 отделов, у каждого из которых есть свой руководитель. Также руководителями разработки программного обеспечения (ПО) управляют директор информационной безопасности (ИБ) и руководитель разработки сайтов. Директор ИБ поддерживает требуемый уровень безопасности согласно бизнес-требованиям клиента и предметной области, а руководитель разработки сайтов является связующим звеном между руководителями Frontend и Backend разработки. Коммерческий директор и директор по маркетингу управляют соответствующими руководителями двух разных, но близких по назначению отделов.

В таблице 1.1 перечислены бизнес-процессы, происходящие в ООО «Твинс».

Таблица 1.1 – Бизнес-процессы ООО «Твинс»

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Процесс | Тип процесса | Краткая характеристика процесса |
| 1 | Оформление контрактов с новыми клиентами | Управляющий | Процесс характеризует процесс заключения договоров на разработку веб-сайтов с новыми клиентами; клиент процесса: новые клиенты, желающие оформить контракт на разработку сайта; входы процесса: запрос клиента на разработку/поддержку/модернизацию/  продвижение сайта, условия сотрудничества, платежные реквизиты; выходы процесса: подписанный контракт, согласованные условия сотрудничества, чек об успешной оплате; ресурсы процесса: клиент, сотрудники отдела продаж, юристы; управление процесса: ГК РФ. |
| 2 | Анализ требований заказчика | Процесс развития | Процесс характеризует определение потребностей заказчика и последующее их преобразование в техническое задание; клиент процесса: менеджер проекта, заказчик; входы процесса: запрос клиента, потребности клиента; выходы процесса: план проекта, техническое задание; ресурсы процесса: бизнес-аналитики, менеджер проекта; управление процессом: отсутствует. |
| 3 | Разработка веб-сайтов | Основной | Процесс характеризует создание качественных веб-сайтов для малого и среднего бизнеса; клиент процесса: заказчики веб-сайтов; входы процесса: требования заказчика, дизайн-макеты, контент; выходы процесса: готовый веб-сайт, который соответствует требованиям заказчика; ресурсы процесса: веб-разработчики, дизайнеры, менеджеры проекта, программно-аппаратные средства; управление процесса: документация проекта. |
| 4 | Тестирование веб-сайтов | Обеспечивающий | Процесс характеризует проверку функциональности и корректности работы веб-сайтов перед их запуском; клиент процесса: веб-разработчики и заказчики; входы процесса: разработанный веб-сайт; выходы процесса: отчет о тестировании; ресурсы процесса: тестировщики, программно-аппаратные средства; управление процессом: план тестирования. |
| 5 | Обновление и поддержка веб-сайтов | Управляющий | Процесс характеризует обновление информации, исправление ошибок и поддержку веб-сайтов после их запуска; клиент процесса: заказчик; входы процесса: запросы на обновление, отчеты об ошибках; выходы процесса: обновленный веб-сайт, исправленные ошибки; ресурсы процесса: веб-разработчики, доступ к серверу; управление процессом: документация проекта. |

Продолжение таблицы 1.1

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | Процесс продвижения веб-сайта в поисковых системах (SEO) | Основной | Процесс характеризует оптимизацию контента и структуры веб-сайта для повышения его позиций в поисковых системах; клиент процесса: маркетологи, специалисты по SEO; входы процесса: цели продвижения, отчеты о поисковой выдаче; выходы процесса: выполненные цели, улучшенная видимость в поисковых системах; ресурсы процесса: SEO-специалисты, аналитические инструменты. управление: запрос клиента. |

Таким образом, описанные ранее характеристики рассматриваемой организации в дальнейшем позволят провести анализ бизнес-процесса оформления новыми клиентами контракта на разработку веб-сайта и выявить его «узкие» места.

2 АНАЛИЗ БИЗНЕС-ПРОЦЕССА «ОФОРМЛЕНИЕ КОНТРАКТОВ С НОВЫМИ КЛИЕНТАМИ» В ООО «ТВИНС»

Рассмотренная ранее IT-организация ООО «Твинс» имеет неавтоматизированный бизнес-процесс заключения контрактов с новыми клиентами. Данный процесс включает в себя следующие этапы [3,4,5,6]:

- поступление запроса от нового клиента на заключение контракта;

- проведение встречи с клиентом для обсуждения условий контракта и предоставления необходимой информации о продуктах и услугах компании;

- подготовка коммерческого предложения с учетом требований клиента и внутренних возможностей компании [7];

- переговоры с клиентом по условиям контракта и внесение необходимых изменений в предложение;

- подписание контракта и начало выполнения условий соглашения.

Анализ данного процесса позволяет выявить проблемы и «узкие» места, которые могут замедлять процесс заключение контрактов и ухудшать качество обслуживания клиентов. В данном случае основные проблемы могут быть связаны с неэффективным взаимодействием между отделом продаж и юристами, а также с отсутствием автоматизированных инструментов для подготовки и управления контрактами [8].

Для улучшения данного процесса необходимо внедрить систему управления контрактами [9,10], которая позволит хранить и обновлять информацию о клиентах и контрактах, автоматизировать процесс подписания документов и уведомлять сотрудников о необходимых действиях. Также стоит оптимизировать коммуникацию между отделами и внедрить единые стандарты оформления контрактов [11].

На рисунке 2.1 изображена модель AS-IS бизнес-процесса «Оформление контрактов с новыми клиентами» в ООО «Твинс», а на рисунке 2.2 – ее декомпозиция.

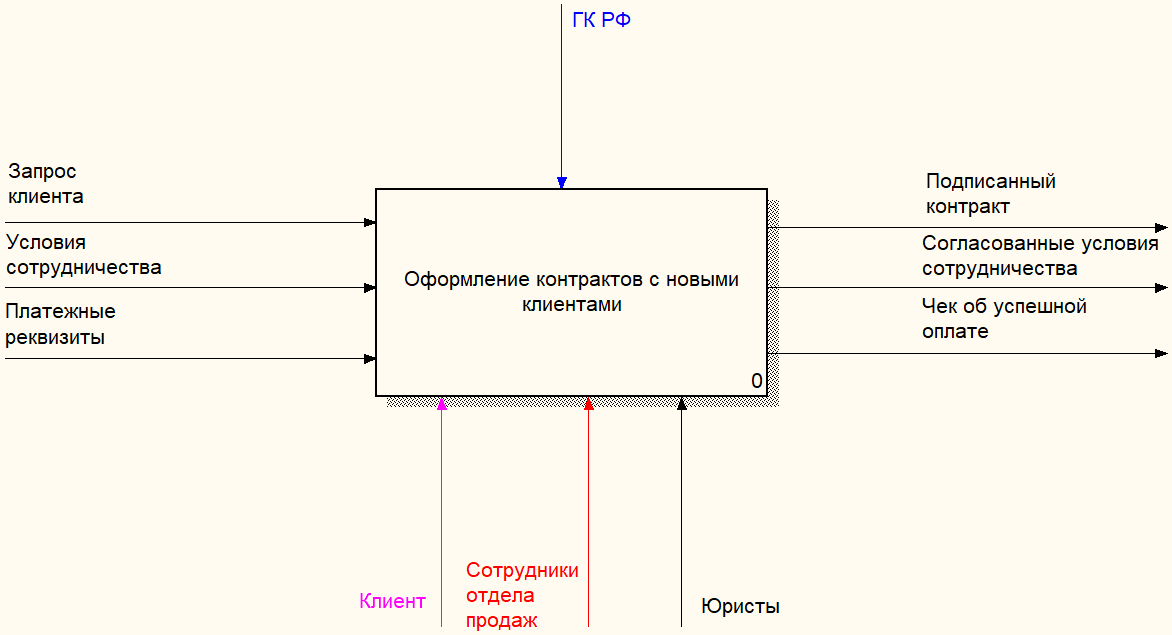


Рисунок 2.1 – Модель AS-IS бизнес-процесса «Оформление контрактов с новыми клиентами»

Данная модель (рисунок 2.1) демонстрирует, что сам процесс оформления контракта в ООО «Твинс» регулируется Гражданским Кодексом Российской Федерации (ГК РФ), в качестве входа использует различную информацию клиента, в качестве ресурса использует своих сотрудников, юристов (юристы могут быть наняты как клиентом, так и организацией) и самого клиента. Последний используется в качестве ресурса со следующим объяснением: клиент принимает активное участие в согласовании оформления контракта и прикладывает свои ресурсы так как он сам заинтересован в том, чтобы оформить контракт на выгодных для него условиях. Другими словами, клиент пытается «выторговать» лучшие для него условия, что требует вложения определенных сил. На выходе в данном процессе получается подписанный, оплаченный контракт с условиями, на которые согласны обе стороны.

Изображение выглядит как текст, диаграмма, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.2 – Декомпозиция модели AS-IS бизнес-процесса «Оформление контрактов с новыми клиентами»

Декомпозиция ранее описанной модели (рисунок 2.2) описывает следующие аспекты бизнес-процесса оформления контракта: на первой стадии происходит преобразование запроса клиента в форму формализованного документа, который четко описывает все обязанности обоих сторон, далее происходит окончательное согласование (с возможными изменениями) и подпись данного контракта, после чего клиент посредством указанных платежных реквизитов производит оплату. Здесь юристы участвуют в первых двух этапах, поставляя свою помощь обоим сторонам и отсутствуют на этапе оплаты услуг, так как контракт к этому моменту уже подписан и их помощь больше не нужна. На всех этапах действия совершаются строго в соответствии с ГК РФ.

Результаты количественного анализа модели AS-IS представлены в таблицах 2.1-2.3.

Таблица 2.1 – Расчет коэффициента уровня для модели AS-IS бизнес-процесса ООО «Твинс»

|  |  |  |  |
| --- | --- | --- | --- |
| Уровень | Значение N | Значение L | Значение Кур |
| 0 | 1 | 0 | – |
| 1 | 3 | 1 | 3,00 |

Коэффициент уровня 3,0 говорит о небольшом превышении нормы, но вписывается в рамки.

Таблица 2.2 – Расчет коэффициента сбалансированности для модели AS-IS бизнес-процесса ООО «Твинс»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Уровень | Значение N | Значение ΣAi | Значение max(Ai) | Значение Кb |
| 0 | 1 | 10 | 10 | 0,00 |
| 1 | 3 | 25 | 10 | 1,67 |

Коэффициент сбалансированности 1,67 достаточно близок к нулю, что говорит о сбалансированности в рамках нормы.

Таблица 2.3 – Расчет коэффициента применения элементарных функций для модели AS-IS бизнес-процесса ООО «Твинс»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Уровень | Значение N | Значение Nэл.ф. | Значение L | Значение Кэл.ф. |
| 0 | 1 | 0 | 0 | 0,00 |
| 1 | 3 | 1 | 1 | 0,33 |

Коэффициент элементарных функций 0,33 в рамках нормы.

Среди «узких» мест в описанной ранее модели AS-IS бизнес-процесса «Оформление контрактов с новыми клиентами» (рисунки 2.1-2.2) можно выделить следующие:

- запрос клиента может быть предоставлен в размытой формулировке, не иметь четких критериев;

- есть вероятность, что предлагаемые условия сотрудничества клиента могут быть невыгодными для организации, из-за чего обе стороны потеряют время и ресурсы;

- подготовка контракта может занять много времени;

- необходимо затратить денежные средства на найм юристов;

- согласование контракта так же может занять много времени;

- сотрудники организации сопровождают все этапы данного процесса.

3 РЕКОМЕНДАЦИИ ПО СОВЕРШЕНСТВОВАНИЮ БИЗНЕС-ПРОЦЕССА «ОФОРМЛЕНИЕ КОНТРАКТОВ С НОВЫМИ КЛИЕНТАМИ» В ООО «ТВИНС»

В качестве решения проблемы наличия «узких» мест в описанной ранее модели AS-IS предлагается автоматизация рассматриваемого процесса с применением искусственного интеллекта (ИИ), что описывается далее [12,13,14,15].

Необходимо разработать информационную систему (ИС), которая будет хранить текущий, регулярно обновляемый «образ поведения» организации относительно оформления контрактов с новыми клиентами. Данная ИС на базе ИИ сначала обучается поведению сотрудников на этапе оформления контрактов до удовлетворительного уровня, чтобы быть способной проводить консультации с реальными клиентами и подготавливать все обязательные документы с уже согласованными условиями, а организации необходимо будет подтвердить данные условия, которые являются выгодными для нее, так как ИС была обучена шаблону ее поведения и ценностям. Чтобы ИС соблюдала актуальность целей организации и при их изменении меняла свое поведение, необходимо иметь доверенное лицо (группу лиц), которые имеют самое точное и актуальное представление об интересах компании и регулярно «обслуживают» данную ИС посредством прохождения различных тестов. Например, ИС может спросить: «Какова на данный момент цена услуги добавления на существующий веб-сайт системы двухфакторной аутентификации?», на что сотрудник может ответить, например «Около 8000 рублей» или «Данная услуга нами не предоставляется». Таким образом, поведение системы будет регулярно обновляться. Главное преимущество данной ИС в том, что ее может обслуживать минимальное количество людей, например, всего два человека по 8 часов в день, а ИС в этот момент будет проводить консультации для десятков различных клиентов, фильтруя невыгодные предложения. Все, что остается сотрудникам организации – это подписать контракт.

В результате внедрения данной системы организация сможет повысить свой уровень продуктивности и степень удовлетворенности клиентов.

ЗАКЛЮЧЕНИЕ

При прохождении производственной практики (по профилю специальности) выполнено индивидуальное задание на тему: «Анализ бизнес-процесса «Оформление контрактов с новыми клиентами»».

В первом разделе дана подробная характеристика организации ООО «Твинс», включая контактные данные, организационную структуру с подробным ее описанием. Также в первом разделе описаны бизнес-процессы, которые наглядно отображают направленность деятельности данной организации и служат основной для дальнейшего изучения бизнес-процесса «Оформление контрактов с новыми клиентами».

Во втором разделе происходит детальный анализ изучаемого бизнес-процесса с помощью модели в нотации AS-IS и ее декомпозиции. Модель позволила выявить «узкие» места, сделала их более очевидными, что в дальнейшем может помочь при их устранении, которое производится с целью повышения эффективности деятельности самой организации.

В третьем разделе были приведены рекомендации, которые могут помочь организации избавиться от «узких» мест рассмотренного бизнес-процесса. В результате автоматизации бизнес-процесса по заключению контрактов с новыми клиентами организация ООО «Твинс» сможет сократить время на выполнение соответствующих операций и увеличить уровень удовлетворенности клиентов.

Таким образом, задачи производственной практики (по профилю специальности) выполнены, соответственно, цель достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Деятельность ООО «Твинс» в области информационных технологий // WebCanape : сайт. URL: <https://www.web-canape.ru/about/it-ooo-twins/>
2. Авторы // WebCanape : сайт. URL: <https://www.web-canape.ru/avtory/>
3. Назарова О.Б., Масленникова О.Е. Моделирование бизнес-процессов : учеб. 3-е изд. М. : ФЛИНТА, 2023. 261 с.
4. Кириллина Ю.В. Управление бизнес-процессами : учеб. М. : РТУ МИРЭА, 2022. 159 с.
5. Вершинина И.В. Применение методологии IDEF0 для функционального моделирования процессов швейного предприятия : монография М. : РГУ им. А.Н. Косыгина, 2023. 98 с.
6. Миндалёв И.В. Моделирование бизнес-процессов с помощью IDEF0, DFD, BPMN за 7 дней : учеб. Красноярск : КрасГАУ, 2016. 123 с.
7. Дронов В.Ю., Дронова Г.А. Бизнес-процесс «Обеспечение информационной безопасности организации» : учеб. Новосибирск : НГТУ, 2021. 76 с.
8. Либерман П.Ю., Яшкин А.Р. Маркетинг : методические указания Санкт-Петербург : СПбГУ ГА, 2020. 71 с.
9. Баланов А.Н. Биометрия. Разработка и внедрение систем идентификации : учеб. пособие для вузов : Санкт-Петербург : Лань, 2024. 228 с.
10. Сологаев В.И. Автоматизация инженерных систем : учеб. пособие Омск : СибАДИ, 2024. 50 с.
11. Баланов А.Н. DevOps: интеграция и автоматизация : учеб. пособие для вузов 2-е изд., стер. Санкт-Петербург : Лань, 2024. 240 с.
12. Кубрин С.С., Кучерин В.Н., Иванов И.М. Автоматическая информационная система : учеб. М. : РУТ (МИИТ), 2024. 92 с.
13. Гехт А.Б., Душкин Р.В., Неровный А.В. Гуманитарные проблемы искусственного интеллекта и его применения : Монография. Санкт-Петербург: СПбГУТ им. М.А. Бонч-Бруевича, 2024. 267 с.
14. Степанов Ю.А., Вылегжанина А.В., Бурмин Л.Н. Системы искусственного интеллекта : учеб. пособие Кемерово : КемГУ, 2024. 102 с.
15. Ложников П.С., Жумажанова С.С., Сулавко А.Е., Самотуга А.Е. Безопасность систем искусственного интеллекта : учеб. пособие Омск : ОмГТУ, 2023. 42 с.

**Приложение А   
Формулы для проведения количественного анализа**

Кур = N/L, (А.1)

где

Кур – коэффициент уровня;

N – количество блоков на уровне;

L – уровень декомпозиции.

, (А.2)

где

Kb – коэффициент сбалансированности;

i – номер блока;

Аi – количество стрелок, связанных с блоком i;

N – количество блоков на уровне.

Кэл.ф. = L ∙ (Nэл.ф./N), (А.3)

где

Кэл.ф. – коэффициент применения элементарных функций;

L – уровень декомпозиции;

Nэл.ф. – количество элементарных функций на уровне;

N – количество всех функций на уровне.

**ПРИЛОЖЕНИЕ Б   
Программный код**

|  |  |
| --- | --- |
| // 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;  }  }  // 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()  {  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();  } |