Государственное автономное профессиональное образовательное учреждение Мурманской области

«Мурманский колледж экономики и информационных технологий»

УТВЕРЖДАЮ

Начальник учебного отдела

ГАПОУ МО «МКЭиИТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

И.Г. Никитина

«\_\_» \_\_\_\_\_\_\_\_\_ 2018

Расчет стоимости использования вычислительных сервисов, а также оценка стоимости их простоя в ПАО «Мурманский траловый флот»

Наименование темы дипломного проекта

Д И П Л О М Н Ы Й П Р О Е К Т

пояснительная записка

по специальности 09.02.03 «Программирование в компьютерных системах»

Выпускник \_Зятына Евгений Эдуардович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество, подпись, дата)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество, подпись, дата)

Рецензент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество, подпись, дата)

Нормоконтролёр \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество, подпись, дата)

2018

Государственное автономное профессиональное образовательное учреждение Мурманской области

«Мурманский колледж экономики и информационных технологий»

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ  Начальник учебного отдела  ГАПОУ МО «МКЭиИТ»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  И.Г. Никитина  «\_\_» \_\_\_\_\_\_\_\_\_ 2018 |

**ЗАДАНИЕ НА ДИПЛОМНОЕ ПРОЕКТИРОВАНИЕ**

Студента(ки) \_Зятына Евгения Эдуардовича\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отделение: дневное

Курс 4 группа П1

Специальность 09.02.03 «Программирование в компьютерных системах»

ТемаРасчет стоимости использования вычислительных сервисов, а также

оценка стоимости их простоя в ПАО «Мурманский траловый флот»

Дипломный проект должен содержать:

* разработанный программный продукт;
* пояснительную записку.

Программный продукт должен реализовывать следующие основные функции:

* + ввод и редактирование данных;
  + поиск и фильтрация данных;
  + формирование выходных документов о заявках на предоставление вычислительных сервисов, а также стоимости их простоя;
  + расчёт стоимости использования вычислительных сервисов;
  + оценка стоимости планового/внепланового простоя сервисов.

Дата выдачи задания "\_\_" \_\_\_\_\_\_\_\_ 2018 года

Дата сдачи работы "\_\_" \_\_\_\_\_\_\_\_ 2018 года

Руководитель проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (расшифровка)

**Содержание**

[Введение 5](#_Toc516778864)

[1 Описание постановки задачи 7](#_Toc516778865)

[1.1 Описание предметной области 7](#_Toc516778866)

[1.1.1 Характеристика предприятия 7](#_Toc516778867)

[1.1.2 Описание технологии обработки информации выбранного для автоматизации вида деятельности 9](#_Toc516778868)

[1.2 Характеристика задачи 12](#_Toc516778869)

[1.3 Описание выходной информации 16](#_Toc516778870)

[1.4 Описание входной информации 16](#_Toc516778871)

[2 Организация и ведение информационной базы 19](#_Toc516778872)

[2.1 Описание систем классификации и кодирования 19](#_Toc516778873)

[2.2 Описание внутримашинной базы 23](#_Toc516778874)

[3 Описание программного обеспечения 41](#_Toc516778875)

[3.1 Описание ОС 41](#_Toc516778876)

[3.2 Описание СУБД 42](#_Toc516778877)

[3.3 Описание среды программирования и языка программирования 44](#_Toc516778878)

[3.4 Описание используемых компонент и проектных процедур 46](#_Toc516778879)

[4 Руководство пользователя 131](#_Toc516778880)

[4.1 Подготовка к работе 131](#_Toc516778881)

[4.2 Описание действий пользователя во время работы с программным продуктом 133](#_Toc516778882)

[4.2.1 Запуск программного продукта и авторизация 133](#_Toc516778883)

[4.2.2 Главный экран системного администратора 133](#_Toc516778884)

[4.2.3 Регистрация и редактирование пользователей ПП 134](#_Toc516778885)

[4.2.4 Редактирование вычислительной инфраструктуры 137](#_Toc516778886)

[4.2.4.1 Создание и редактирование конфигураций серверов 138](#_Toc516778887)

[4.2.4.2 Создание и редактирование конфигураций хранилищ данных 144](#_Toc516778888)

[4.2.5 Распределение вычислительных ресурсов между платформами 145](#_Toc516778889)

[4.2.6 Создание сервиса 146](#_Toc516778890)

[4.2.7 Обработка заявки на предоставление сервиса 148](#_Toc516778891)

[4.2.8 Подписка на предоставляемый сервис 149](#_Toc516778892)

[4.2.9 Планирование расписания обслуживания оборудования 150](#_Toc516778893)

[4.2.10 Регистрация простоя оборудования 151](#_Toc516778894)

[4.2.11 Расчет стоимости простоя сервиса 151](#_Toc516778895)

[4.3 Действия пользователя в аварийных ситуациях 162](#_Toc516778896)

[5 Выводы и предложения 163](#_Toc516778897)

[6 Перечень принятых сокращений 165](#_Toc516778898)

[7 Перечень терминов 166](#_Toc516778899)

[Список использованной литературы 167](#_Toc516778900)

[Приложение А. Выходные документы 169](#_Toc516778901)

[Приложение Б. Образцы входных документов 173](#_Toc516778902)

[Приложение В. Инфологическая модель БД 182](#_Toc516778903)

[Приложение Г. Распечатка содержимого файлов базы данных 184](#_Toc516778904)

[Приложение Д. Экранные формы 194](#_Toc516778905)

[Приложение Е. Листинг программы 216](#_Toc516778906)

[Приложение Ж. Инсталляционныйдиск 420](#_Toc516778907)

# **Введение**

Целью дипломного проектирования является разработка программного продута для автоматизации расчета стоимости использования вычислительных сервисов, а также оценки стоимости их простоя в ПАО «Мурманский траловый флот».

Актуальность решения данной задачи заключается в необходимости руководству предприятия знать стоимость предоставляемых вычислительных ресурсов, для собственных сотрудников, для грамотного распределения бюджета на компьютерную инфраструктуру, а также оценивать – во сколько обойдется ущерб от простоя оборудования. Разработанный программный продукт помогает решить поставленные задачи удобным, для конечного пользователя, способом.

Данная работа включает в себя следующие задачи:

* изучение предметной областипо предоставлению вычислительных ресурсов;
* составление финансовых формул для расчета стоимости использования вычислительных сервисов, а также для расчета ущерба от простоя оборудования;
* разработка базы данных;
* определениеосновного функционала приложения;
* изучение дополнительных материалов для разработки приложений;
* разработку программного продукта «ПАО МТФ - Облачные вычисления»;
* отладку и тестирование приложения.

Существующие подобные решения:

* AmazonWebServices;
* MicrosoftAzure;
* GoogleCloudPlatform.

Основные отличия разработанного программного продукта от существующих решений:

* продукт является проприетарным;
* продукт содержит конфигурирование вычислительной инфраструктуры;
* продукт предоставляет создание отчетов следующих видов:
  + 1. стоимость предоставления сервисов;
    2. заявка на предоставление сервиса;
    3. стоимость простоя предоставляемого сервиса.
* продукт предоставляет возможность распределения вычислительных ресурсов на предоставляемые платформы;
* в программном продукте существует возможность планирования расписания обслуживания;
* продукт поддерживает регистрацию простоев оборудования;
* продукт содержит функционал по учету используемого или обслуживаемого оборудования;
* в программном продукте имеется возможность автоматического или полуавтоматического расчета стоимости простоя сервиса.

# **1 Описание постановки задачи**

* 1. **Описание предметной области**
     1. **Характеристика предприятия**

Наименование предприятия: ПАО «Мурманский траловый флот». Предприятие расположено по адресу: г. Мурманск, ул. Шмидта, д. 43.

Сфера деятельности предприятиязаключается в добыче морепродуктов на траулерных судах, а также их переработки.

Основными видами деятельности предприятия являются:

* добыча и производство морепродуктов;
* переработка сырья;
* предоставление спутниковой и радиосвязи;
* транспортировка мороженной рыбопродукции, рыбной муки и консервов на собственных транспортных судах из районов промысла в российские и иностранные порты.

Предметом деятельности ОАО «Мурманский траловый флот» является добыча и производство (переработка) морепродуктов, транспортировка готовой продукции клиентам по всему миру, предоставление и обслуживание спутниковой и радиосвязи, участие в Российских и Международных выставках рыбопродукции.

Перечень и характеристика выпускаемой продукции представлена в таблице 1.

Таблица 1− Перечень выпускаемой продукции

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Характеристика** | |
| **№ банки** | **банок в ящике** |
| *Консервы и пресервы* | | |
| Скумбрия натуральная с добавлением масла | 3,6 | 48 |
| Сельдь натуральная с добавлением масла | 3 | 48 |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Характеристика** | |
| **№ банки** | **банок в ящике** |
| Сардина натуральная с добавлением масла | 6 | 48 |
| Печень трески натуральная | 3 | 48 |
| Треска натуральная с добавлением жира печени | 3 | 48 |
| Икра трески натуральная | 60 | 120 |
| Икра трески с морской капустой | 3 | 48 |
| Печень трески натуральная | 2 | 72 |
| Печень трески в томатном соусе | 3 | 48 |
| Печень трески по-мурмански | 60 | 120 |
| Паштет из печени и икры трески | 2 | 72 |
| Сельдь натуральная с добавлением масла | 6 | 48 |
| Сельдь с фасолью в томатном соусе | 6 | 48 |
| *Рыбная мука* | | |
| **Наименование** | **Виды упаковки** | |
| Рыбная кормовая стабилизированная антиокислителем | **-**полипропиленовый мешок;  -джутовый мешок;  -бумажный крафт-мешок; | |

Схема организационной структуры предприятия представлена на рисунке 1.

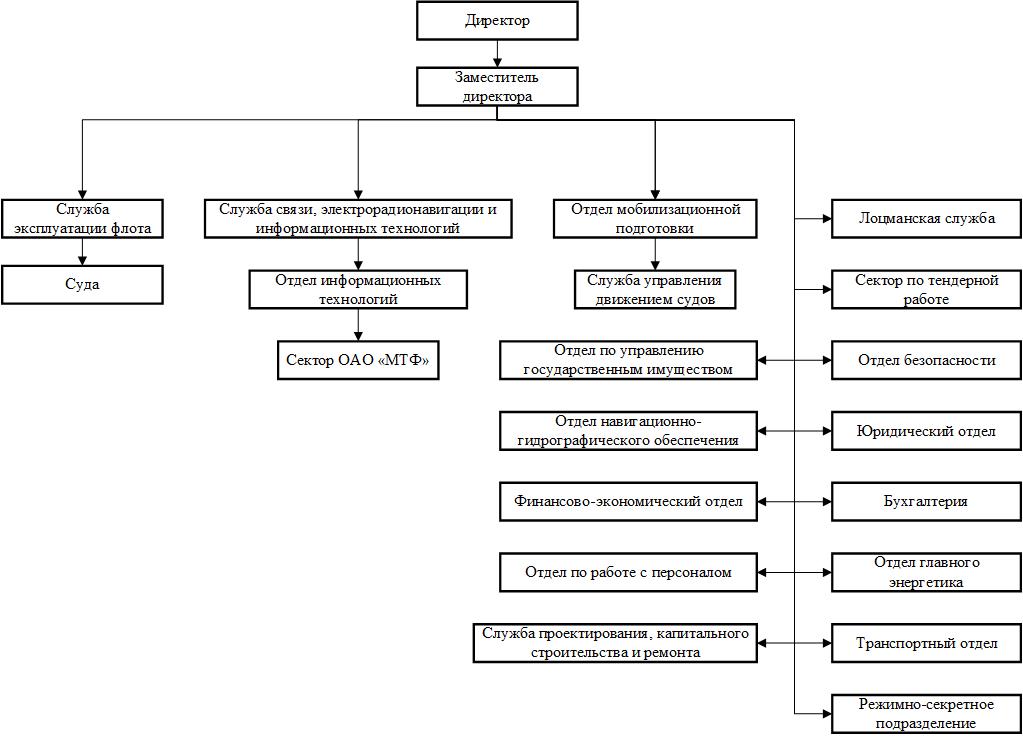


Рисунок 1 − Организационная структура предприятия

* + 1. **Описание технологии обработки информации выбранного для автоматизации вида деятельности**

Один из важнейших этапов деятельности IT отдела предприятия – рациональное конфигурирование вычислительных систем, так как оборудование для корпоративной деятельности очень дорогое. Специалисты предоставляют коммуникации и вычислительные ресурсы сотрудникам предприятия и анализируют загруженность инфраструктуры для ее своевременного расширения и/или обновления компонентов систем.

Основными недостатками являются:

* сложность расчета стоимости предоставляемых вычислительных ресурсов для более детального обоснования производимых расходов на оборудование;
* контроль загруженности инфраструктуры, из-за которого предприятие несет убытки, так как в случае отсутствия требуемых ресурсов сотруднику приходится ждать, пока предприятие согласует расширение вычислительных ресурсов;
* оценка стоимости простоя сервиса в случае планового обслуживания оборудования, а также внеплановых сбоев.

Разрабатываемый программный продукт позволяет решить вышеописанные проблемы следующим образом:

* системный администратор вносит подробное описание компонентов вычислительной инфраструктуры с описанием их стоимости и стоимости обслуживания каждого сервера;
* планирует расписание планового обслуживания оборудования;
* распределяет ресурсы между предоставляемыми платформами;
* к созданным платформам добавляет сервисы, выделяя доступное количество ресурсов, вследствие чего получает стоимость их предоставления в час. После создания запрос о их возможности предоставления отправляется директору предприятия;
* по существующим позициям планового расписания обслуживания оборудования приложение, в автоматическом режиме, производит попытку расчета стоимости простоя и переводит оборудование в состояние обслуживания, тем самым отключая предоставляемые сервисы, если не хватает вычислительных ресурсов для их обеспечения. При отрицательном результате, директор вводит финансовые характеристики предприятия и получает результат;
* директор принимает решение о предоставлении созданного сервиса, в случае положительного результата - сотрудники предприятия подписываются на предоставляемые сервисы;
* директор получает возможность получения отчетов об стоимости предоставления сервисов, а также оценку стоимости их простоя;
* системный администратор следит за загруженностью инфраструктуры, и в случае их нехваткизакупает новые компоненты и вносит их подробное описание.

Диаграмма IDEF0 моделирует функциональную структуру предметной области. Данные диаграммы представлены на рисунках 2 и 3.

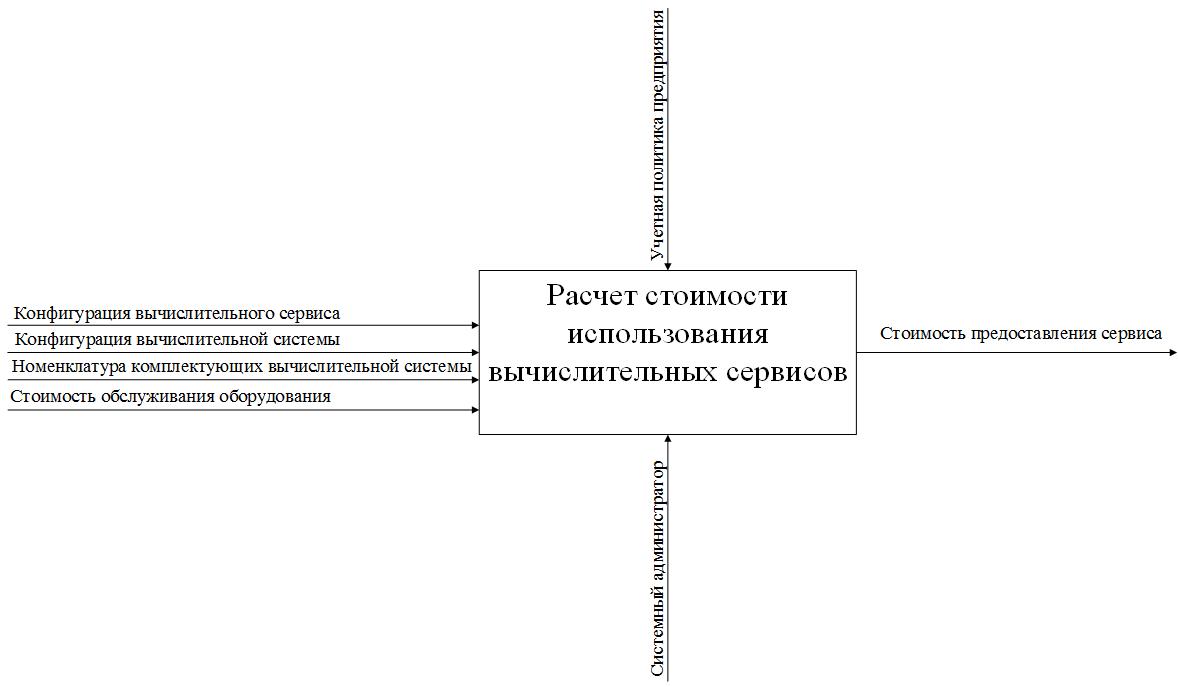


Рисунок 2 −IDEF0 диаграмма для задачи по рачету стоимости использования вычислительных сервисов

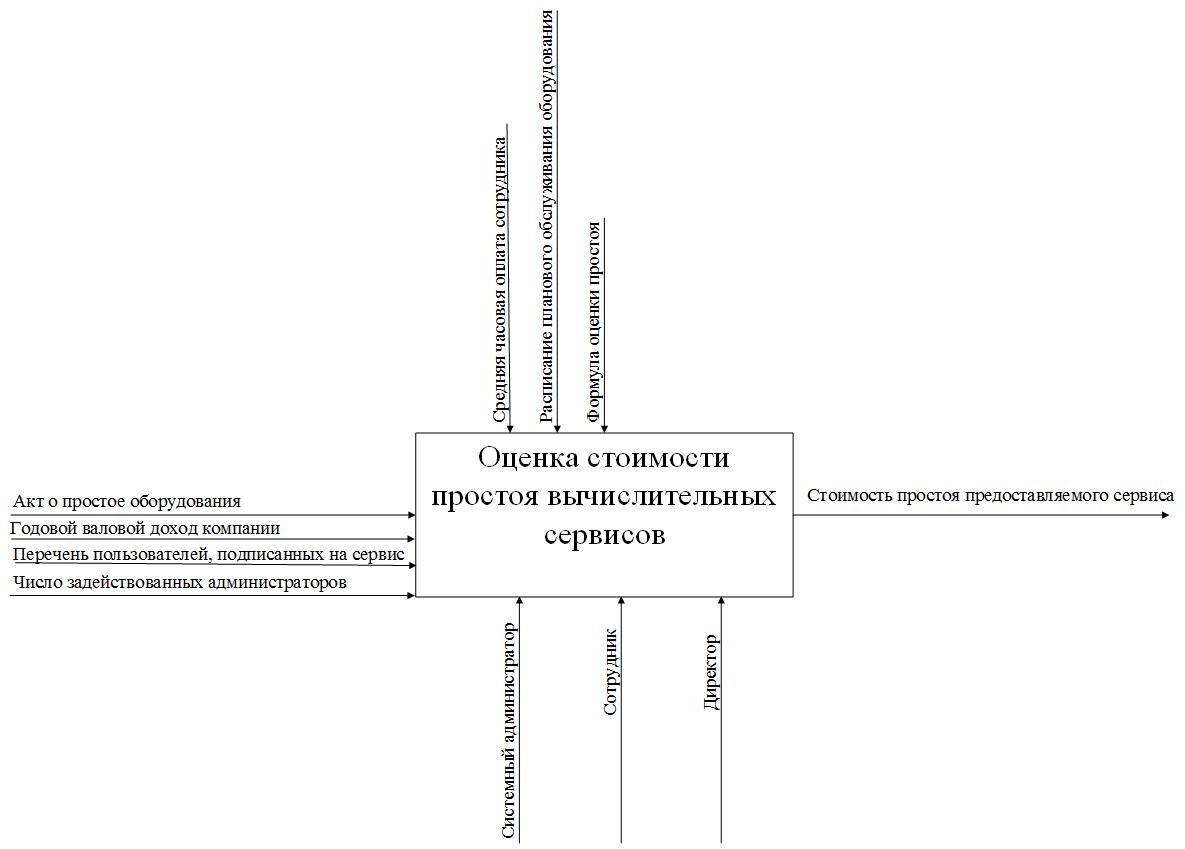


Рисунок 3 −IDEF0 диаграмма для задачи по оценке простоя вычислительных сервисов

## **1.2** **Характеристика задачи**

Наименование задачи: «Расчет стоимости использования вычислительных сервисови оценка стоимости их простоя в ПАО «Мурманский траловый флот».

Назначение задачи: учет и автоматизированный расчет стоимости использования вычислительных сервисов, предоставляемых предприятием, для оптимизации расходов на новое оборудование и своевременного расширения инфраструктуры; учет и автоматизация оценки стоимости их планового/внепланового простоя.

Перечень функций, реализуемых задачей:

* ввод и редактирование данных о:
  1. учетных записях пользователей;
  2. должностях сотрудников;
  3. конфигурациях серверов;
  4. платформах сервера;
  5. процессорах;
  6. разъёмах процессора;
  7. оперативной памяти;
  8. типов оперативной памяти;
  9. накопителях;
  10. интерфейсах накопителя;
  11. производителях;
  12. хранилищах данных;
  13. предоставляемых платформах;
  14. предоставляемых сервисах;
  15. типов предоставляемых сервисов;
  16. программном обеспечении;
  17. типах программного обеспечения;
  18. позиций планового обслуживания оборудования;
  19. характеристиках предприятия для расчета стоимости простоя сервиса;
  20. распределенном персонале предприятия для разрешения простоя оборудования;
  21. совершенном простое;
  22. подписанных сервисах для текущего сотрудника;
  23. учетных данных текущего пользователя;
* просмотр данных о:
  1. учетных записях пользователей;
  2. должностях сотрудников;
  3. конфигурациях серверов вычислительной инфраструктуры;
  4. хранилищах данных вычислительной инфраструктуры;
  5. платформах сервера;
  6. процессорах;
  7. разъёмах процессора;
  8. оперативной памяти;
  9. типов оперативной памяти;
  10. накопителях;
  11. интерфейсах накопителя;
  12. производителях;
  13. предоставляемых платформах;
  14. предоставляемых сервисах;
  15. типов предоставляемых сервисов;
  16. программном обеспечении;
  17. типах программного обеспечения;
  18. простое конфигураций серверов;
  19. простое хранилищ данных;
  20. простое сервисов;
  21. расписании обслуживания оборудования;
  22. подписанных сервисах для текущего сотрудника;
  23. доступных на подписку сервисов;
  24. заявках на предоставление сервисов;
  25. стоимости предоставления сервисов;
  26. простое сервисов;
  27. зарегистрированных пользователях системы;
* отбор данных по:
  1. платформе конфигурации сервера;
  2. процессору конфигурации сервера;
  3. производителю хранилища данных;
  4. модели хранилища данных;
  5. типу платформы сервиса;
  6. должности сотрудника;
  7. уровню привилегий сотрудника;
  8. актуальности позиций планового расписания обслуживания;
* поиск:
  1. производителя оборудования по его наименованию;
  2. типа оперативной памяти по ее наименованию;
  3. интерфейса накопителя по его наименованию;
  4. поиск разъёма процессора по его наименованию и/или производителю;
  5. типа сервиса по его наименованию;
  6. типа программного обеспечения по его наименованию;
  7. программного обеспечения по его наименованию и/или типу ПО;
* формирование отчетов о:
  1. стоимости предоставления сервисов;
  2. заявке на предоставление сервиса;
  3. стоимости простоя предоставляемого сервиса
* расчёт стоимости использования вычислительных сервисов;
* полуавтоматическаяили автоматическая оценка стоимости планового и внепланового простоя сервисов.

Объекты, участвующие в решении задачи:

* персонал предприятия;
* системный администратор;
* директор.

Данная задача связана тесно связана с учетом расходов и затрат предприятия на обеспечение производства производимой продукции.

## **1.3** **Описание выходной информации**

Описание выходной информации представлено в таблице в таблице 2.

Таблица 2 – Описание выходной информации

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Наименование** | **Форма представления** | **Перечень получателей** | **Количество экземпляров** | **Периодичность выдачи** |
| Отчет о стоимости предоставления сервисов | Печатный документ, экранная форма | Директор | 1 | По мере необходимости |
| Заявка на предоставление сервиса | Печатный документ, экранная форма | Директор, системный администратор | 1 | При обработке заявки на предоставление нового сервиса |
| Отчет о стоимости простоя предоставляемого сервиса | Печатный документ, экранная форма | Директор | 1 | При редактировании стоимости простоя предоставляемого сервиса |

## **1.4 Описание входной информации**

Описание входных документов представлено в таблице в таблице 3.

Таблица 3 – Описание входных документов

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Периодичность поступления** | **Перечень входной информации** |
| Технический паспорт на конфигурацию сервера | При распределении ресурсов между платформами | Порядковый номер конфигурации;  Наименование производителя и модели установленной платформы;  Наименование производителя и модели установленного процессора; |

Продолжение таблицы 3

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Периодичность поступления** | **Перечень входной информации** |
|  |  | Список установленной оперативной памяти;  Список установленных накопителей;  Стоимость обслуживания в месяц;  Общая стоимость конфигурации |
| Технический паспорт на конфигурацию хранилища данных | При распределении ресурсов между платформами | Порядковый номер хранилища данных;  Наименование производителя и модели;  Список установленных накопителей;  Стоимость обслуживания в месяц;  Общая стоимость хранилища данных |
| Акт простоя | При ликвидации нештатной ситуации вычислительной инфраструктуры | Дата и время начала простоя;  Дата и время окончания простоя;  Дата регистрации простоя;  Продолжительность;  Тип простоя;  Причина;  Задействованный персонал для разрешения возникшей ситуации |
| Справка о предоставлении средней заработной платы | Раз в месяц | Наименование должности;  Размер средней заработной платы;  Количество рабочих часов в неделю сотрудника |
| Личное дело работника | При регистрации учетных данных сотрудника | Логин;  Пароль;  Должность;  ФИО |

Продолжение таблицы 3

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Периодичность поступления** | **Перечень входной информации** |
| График планового обслуживания оборудования | Раз в месяц | Порядковый номер оборудования;  Краткая характеристика оборудования;  Дата и время начала планового обслуживания;  Дата и время окончания планового обслуживания;  Продолжительность |
| Штатное расписание | Раз в квартал | Число работников;  Средняя часовая оплата труда администратора;  Средняя часовая оплата труда работника |
| Отчет о финансовых результатах за год | Раз в год | Годовой валовой доход компании; |

# **2 Организация и ведение информационной базы**

## **2.1 Описание систем классификации и кодирования**

В задаче «Расчет стоимости использования виртуальных машин в ОАО «Мурманский траловый флот»» классификации и кодированию подлежат следующие объекты:

* уровень привилегий пользователя;
* тип программного обеспечения;
* тип платформы;
* тип сервиса;
* состояние сервиса;
* тип простоя;
* причина простоя;
* производитель;
* тип оперативной памяти;
* интерфейс обмена данных.

Комплектующие вычислительной системы отличаются большим разнообразием характеристик, на основании которых предприятие предоставляет вычислительные ресурсы сотрудникам. Для эффективной организации учета конфигураций вычислительной системы, а именно – подробного описания характеристик комплектующих, не обойтись без их кодирования.

Для всех объектов кодирования используется порядковая система кодирования.

Для объекта «Уровень привилегий пользователя» доступен диапазон значений кодаот 0 до 255, так как для хранения диапазона выбран тип данных tinyint. Фрагмент справочника приведен в таблице 4.

Таблица 4− Фрагмент объекта «Уровень привилегий пользователя»

|  |  |
| --- | --- |
| **Код** | **Наименование** |
| 1 | Директор |
| 2 | Системный администратор |
| 3 | Сотрудник |

Для объекта «Тип программного обеспечения» доступен диапазон значений кода от 0 до 255, так как для хранения диапазона выбран тип данных tinyint. Фрагмент справочника приведен в таблице 5.

Таблица 5− Фрагмент объекта «Тип программного обеспечения»

|  |  |
| --- | --- |
| **Код** | **Наименование** |
| 1 | Системное |
| 2 | Прикладное |
| 3 | Пакеты прикладных программ |

Дляобъекта «тип платформы» доступен диапазон значений кодаот -32768 до 32767, так как для хранения диапазона выбран тип данных smallint. Фрагмент справочника приведен в таблице 6.

Таблица 6− Фрагмент объекта «тип платформы»

|  |  |
| --- | --- |
| **Код** | **Наименование** |
| 1 | 1С |
| 2 | SQL |
| 3 | Cloud |

Для объекта «тип сервиса» доступен диапазон значений кодаот -32768 до 32767, так как для хранения диапазона выбран тип данных smallint. Фрагмент справочника приведен в таблице 7.

Таблица 7− Фрагмент объекта «тип сервиса»

|  |  |
| --- | --- |
| **Код** | **Наименование** |
| 1 | Тонкий клиент |
| 2 | VPN |
| 3 | Outlook |

Для объекта «состояние сервиса» доступен диапазон значений кода от 0 до 255, так как для хранения диапазона выбран тип данных tinyint. Фрагмент справочника приведен в таблице 8.

Таблица 8− Фрагмент объекта «состояние сервиса»

|  |  |
| --- | --- |
| **Код** | **Наименование** |
| 1 | Активен |
| 2 | Отправлена заявка |
| 3 | Отклонен |

Для объекта «тип простоя» доступен диапазон значений кода от 0 до 255, так как для хранения диапазона выбран тип данных tinyint. Фрагмент справочника приведен в таблице 9.

Таблица 9− Фрагмент объекта «тип простоя»

|  |  |
| --- | --- |
| **Код** | **Наименование** |
| 1 | Плановый |
| 2 | Внеплановый |

Для объекта «причина простоя» доступен диапазон значений кода от 0 до 255, так как для хранения диапазона выбран тип данных tinyint. Фрагмент справочника приведен в таблице 10.

Таблица 10− Фрагмент объекта «причина простоя»

|  |  |
| --- | --- |
| **Код** | **Наименование** |
| 1 | Резервное копирование |
| 2 | Обновление конфигурации |
| 3 | Выход оборудования из строя |

Для объекта «производитель» доступен диапазон значений кодаот -32768 до 32767, так как для хранения диапазона выбран тип данных smallint. Фрагмент справочника приведен в таблице 11.

Таблица 11− Фрагмент объекта «производитель»

|  |  |
| --- | --- |
| **Код** | **Наименование** |
| 1 | Dell |
| 2 | Intel |
| 3 | Supermicro |

Для объекта «тип оперативной памяти» доступендиапазон значений кода от 0 до 255, так как для хранения диапазона выбран тип данных tinyint. Фрагмент справочника приведен в таблице 12.

Таблица 12− Фрагмент объекта «тип оперативной памяти»

|  |  |
| --- | --- |
| **Код** | **Наименование** |
| 1 | DDR2 |
| 2 | DDR3 |
| 3 | DDR4 |

Для объекта «интерфейс обмена данных» доступен диапазон значений кода от 0 до 255, так как для хранения диапазона выбран тип данных tinyint. Фрагмент справочника приведен в таблице 13.

Таблица 13− Фрагмент объекта «интерфейс обмена данных»

|  |  |
| --- | --- |
| **Код** | **Наименование** |
| 1 | SATA3 |
| 2 | SAS 300 |
| 3 | SAS 600 |

## **2.2 Описание внутримашинной базы**

Перечень наименований файлов и их описание представлено в таблице 14.

Таблица 14 – Перечень наименований файлов и их описание

|  |  |
| --- | --- |
| **Наименование файла** | **Описание** |
| mtfservicesdb.mdf | Основной файл базы данных, хранится на удаленном сервере |
| mtfservicesdb\_log.mdf | Журнал транзакций базы данных, хранится на удаленном сервере |
| CostIdleParams.dat | Параметры для расчёта простоя сервисов, которые включают в себя:   * годовой валовой доход компании; * общее число работников предприятия; * среднюю часовую оплату труда администратора; * среднюю часовую оплату труда работника. |

Внутримашинная база включает в себя базу данных mtfservicesdb, которая хранит в себе информацию о:

* вычислительной инфраструктуре предприятия;
* распределении вычислительных ресурсов между платформами;
* предоставляемых сервисах;
* зарегистрированных сотрудниках предприятия;
* подписках сотрудников на предоставляемые вычислительные сервисы;
* расписании обслуживания оборудования;
* простоях вычислительных сервисов;
* должностях сотрудников и их среднюю заработную плату.

База данных хранится на сервере MSSQL, параметры подключения представлены в таблице 15.

Таблица 15 – Параметры подключения к базе данных

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| Адрес сервера СУБД | SSQL-12R2WEB01.client.parking.ru |
| Логин | zatinase\_zllbn |
| Пароль | Hjn7gN5qaC |

База данных включает в себя 25 таблиц и 10 справочников, перечень которых представлен в таблице 16.

Таблица 16 −Перечень таблиц базы данных и их описание

|  |  |  |
| --- | --- | --- |
| **Идентификатор таблицы** | **Наименование** | **Описание** |
| AdminIdle | Задействованный персонал простоя | Персонал, который был задействован для разрешения возникшей нештатной ситуации оборудования |
| CPU | Процессор | Номенклатура процессоров, которые используются на серверах и определяют характеристики, на основе которых происходит распределение вычислительных ресурсов, а также рассчитывается стоимость единицы ресурса |
| CpuSocket | Разъем процессора | Справочник наименований разъема процессора |
| IdleReason | Причина простоя | Справочник наименований причины простоя сервиса |
| IdleType | Тип простоя | Справочник наименований типа простоя сервиса |
| Manufacturer | Производитель | Справочник наименований производителя |

Продолжение таблицы 16

|  |  |  |
| --- | --- | --- |
| **Идентификатор таблицы** | **Наименование** | **Описание** |
| PaasType | Тип платформы сервисов | Справочник наименований вида платформы, к которым присоединяются сервисы |
| Platform | Платформа | Номенклатура платформ, на основе которых создается новый сервер и определяются основные характеристики конфигурации |
| Platform\_StorageInt | Интерфейс обмена данных платформы | Интерфейсыобмена данных и количество доступных портов на платформе сервера |
| Position | Должность | Описание наименования должности сотрудников, размера средней заработной платы, среднего количества рабочих часов в неделю |
| RAM | Оперативная память | Номенклатура оперативной памяти, которая используются на серверах и определяет характеристики, на основе которой происходит распределение вычислительных ресурсов, а также рассчитывается стоимость единицы ресурса |
| RamType | Тип оперативной памяти | Справочник наименований типа оперативной памяти |

Продолжение таблицы 16

|  |  |  |
| --- | --- | --- |
| **Идентификатор таблицы** | **Наименование** | **Описание** |
| RightsLevel | Уровень привилегий | Справочник наименований уровня привилегий пользователя приложения, на основе которого определяется предоставляемый функционал |
| SAN | Хранилище данных | Описание характеристик действующего хранилища, к которой можно будет присоединять предоставляемые сервисы |
| SAN\_MaintenanceShedule | Расписание планового обслуживания хранилища данных | Описание позиций расписания плановогообслуживания хранилища данных |
| SAN\_Storage | Накопитель хранилища данных | Накопители, которые используются на хранилище данных |
| SAN\_StorageInt | Интерфейс обмена данных накопителя | Интерфейсы обмена данных и количество доступных портов на хранилище данных |
| SANtoPaas | Хранилище данных платформы | Распределение хранилищ данных между типами платформ |
| Server | Конфигурация сервера | Описание характеристик действующейконфигурация сервера, к которой можно будет присоединять предоставляемые сервисы |
| Server\_MaintenanceShedule | Расписание планового обслуживания сервера | Описание позиций расписания плановогообслуживания сервера |

Продолжение таблицы 16

|  |  |  |
| --- | --- | --- |
| **Идентификатор таблицы** | **Наименование** | **Описание** |
| Server\_RAM | Оперативная память сервера | Оперативная память, которая используются на сервере |
| Server\_Storage | Накопитель сервера | Накопители, которые используются на сервере |
| ServerToPaas | Сервер платформы | Распределение серверов между типами платформ |
| Service | Сервис | Описание характеристик предоставляемого сервиса |
| ServiceIdle | Простой сервиса | Описание простоя оборудования, который затронул предоставляемые сервисы |
| ServiceState | Состояние сервиса | Справочник наименований состояния сервиса |
| ServiceType | Тип сервиса | Справочник наименований типа сервиса |
| SoftService | Программное обеспечение сервиса | Программное обеспечение, которое использует сервис |
| SoftType | Тип программного обеспечения | Справочник наименований типа сервиса программного обеспечения |
| Software | Программное обеспечение | Сведения о программном обеспечении, которое может быть использовано в сервисах |
| StrorageInterface | Интерфейс обмена данных | Справочник наименований вида интерфейсаобмена данных |

Продолжение таблицы 16

|  |  |  |
| --- | --- | --- |
| **Идентификатор таблицы** | **Наименование** | **Описание** |
| Strorage | Накопитель | Номенклатура накопителей данных, которые используются на серверах и хранилищах данных, и определяет характеристики, на основе которых происходит распределение вычислительных ресурсов, а также рассчитывается стоимость единицы ресурса |
| User | Пользователь | Сведения о зарегистрированном пользователе |
| UsingService | Использование сервисов | Подписки пользователей на предоставляемые сервисы |
| \_\_MigrationHistory | История миграций | История изменения модели данных |

Описание структуры таблиц приведено в таблицах 17 – 51.

Таблица 17 – Описание структуры записей таблицы «AdminIdle»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| ServiceId | Код сервиса | int | Первичный, внешний |
| UserTabNo | Табельный номер сотрудника | int | Первичный, внешний |

Таблица 18 – Описание структуры записей таблицы «CPU»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | int | Первичный |
| ManufacturerId | Код производителя | smallint | Внешний |
| Model | Наименование моделипроцессора | nvarchar(50) |  |

Продолжение таблицы 18

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| CpuSocketId | Код разъема процессора | tinyint | Внешний |
| CoreCount | Кол-во ядер (шт.) | smallint |  |
| Price | Цена (руб.) | money |  |
| Frequency | Тактовая частота (Ггц) | float |  |

Таблица 19 – Описание структуры записей таблицы «CpuSocket»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | tinyint | Первичный |
| Name | Наименование разъема процессора | nvarchar(50) |  |
| ManufacturerId | Код производителя | smallint | Внешний |

Таблица 20 – Описание структуры записей таблицы «IdleReason»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | tinyint | Первичный |
| Name | Наименование причины простоя | nvarchar(50) |  |

Таблица 21 – Описание структуры записей таблицы «IdleType»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | tinyint | Первичный |
| Name | Наименование типа простоя | nvarchar(50) |  |

Таблица 22 – Описание структуры записей таблицы «Manufacturer»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | smallint | Первичный |
| Name | Наименование производителя | nvarchar(100) |  |

Таблица 23 – Описание структуры записей таблицы «PaasType»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | smallint | Первичный |
| Name | Наименование типа платформы | nvarchar(100) |  |

Таблица 24 – Описание структуры записей таблицы «Platform»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | int | Первичный |
| ManufacturerId | Код производителя | smallint | Внешний |
| Model | Наименование модели платформы | nvarchar(100) |  |
| CpuSocketId | Код разъема процессора | tinyint | Внешний |
| CPUCount | Кол-во поддерживаемых процессоров (шт.) | tinyint |  |
| RamTypeId | Код типа оперативной памяти | tinyint | Внешний |
| RamVolumeMax | Максимальный поддерживаемый объем оперативной памяти (ГБ) | int |  |
| RamSocketCount | Кол-во разъемов оперативной памяти (шт.) | tinyint |  |
| Price | Цена (руб.) | money |  |

Таблица 25 – Описание структуры записей таблицы «Platform\_StorageInt»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| PlatformId | Код платформы | int | Первичный, внешний |
| InterfaceId | Код интерфейса обмена данных | tinyint | Первичный, внешний |
| SlotCount | Кол-во разъемов (шт.) | tinyint |  |

Таблица 26 – Описание структуры записей таблицы «Position»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | int | Первичный |
| Name | Наименование должности | nvarchar(100) |  |
| AvgSalary | Размер средней заработной платы (руб.) | money |  |

Продолжение таблицы 26

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** | |
| WorkHours | Среднее кол-во рабочих часов в неделю | tinyint | |  |

Таблица 27 – Описание структуры записей таблицы «RAM»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | int | Первичный |
| ManufacturerId | Код производителя | smallint | Внешний |
| Model | Наименование модели оперативной памяти | nvarchar(100) |  |
| RamTypeId | Код типа оперативной памяти | tinyint | Внешний |
| Volume | Объем модуля (ГБ) | int |  |
| Price | Цена (руб.) | money |  |

Таблица 28 – Описание структуры записей таблицы «RamType»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | tinyint | Первичный |
| Name | Наименование типа оперативной памяти | nvarchar(50) |  |

Таблица 29 – Описание структуры записей таблицы «RightsLevel»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | tinyint | Первичный |
| Name | Наименования уровня привилегий пользователя | nvarchar(50) |  |

Таблица 30 – Описание структуры записей таблицы «SAN»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | int | Первичный |
| ManufacturerId | Код производителя | smallint | Внешний |

Продолжение таблицы 30

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Model | Наименование модели хранилища данных | nvarchar(50) |  |
| AnnualMaintenance | Стоимость обслуживания (руб./мес.) | money |  |
| Price | Цена (руб.) | money |  |
| Active | Флаг, сообщающий – используется ли хранилище данных в текущий момент времени | bit |  |
| OnMaintenance | Флаг, сообщающий – находится ли хранилище данных на обслуживании в текущий момент времени | bit |  |

Таблица 31 – Описание структуры записей таблицы «SAN\_MaintenanceShedule»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | int | Первичный |
| SAN\_Id | Код хранилища данных | int | Внешний |
| BeginPeriod | Дата и время начала планового обслуживания | datetime |  |
| EndPeriod | Дата и время окончания планового обслуживания | datetime |  |

Таблица 32 – Описание структуры записей таблицы «SAN\_Storage»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| SAN\_Id | Код хранилища данных | int | Первичный, внешний |
| StorageId | Код накопителя | int | Первичный, внешний |
| Count | Кол-во установленных накопителей | smallint |  |

Таблица 33 – Описание структуры записей таблицы «SAN\_StorageInt»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| SAN\_Id | Код хранилища данных | int | Первичный, внешний |
| InterfaceId | Код интерфейса обмена данных | tinyint | Первичный, внешний |
| SlotCount | Кол-во доступных разъемов | tinyint |  |

Таблица 34 – Описание структуры записей таблицы «SANtoPaas»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| PaasTypeId | Код типа платформы | smallint | Первичный, внешний |
| SAN\_Id | Код хранилища данных | int | Первичный, внешний |

Таблица 35 – Описание структуры записей таблицы «Server»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | int | Первичный |
| PlatformId | Код платформы | int | Внешний |
| CPU\_Id | Код процессора | int | Внешний |
| Price | Стоимость (руб.) | money |  |
| AnnualMaintenance | Стоимость обслуживания (руб./мес.) | money |  |
| Active | Флаг, сообщающий – используется ли конфигурация сервера в текущий момент времени | bit |  |
| OnMaintenance | Флаг, сообщающий – находится ли конфигурация сервера на обслуживании в текущий момент времени | bit |  |

Таблица 36– Описание структуры записей таблицы «Server\_MaintenanceShedule»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | int | Первичный |
| ServerID | Код конфигурации сервера | int | Внешний |

Продолжение таблицы 36

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| BeginPeriod | Дата и время начала планового обслуживания | datetime |  |
| EndPeriod | Дата и время окончания планового обслуживания | datetime |  |

Таблица 37 – Описание структуры записей таблицы «Server\_RAM»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| ServerId | Кодконфигурации сервера | int | Первичный, внешний |
| RamId | Код оперативной памяти | int | Первичный, внешний |
| Count | Кол-во установленной оперативной памяти (шт.) | tinyint |  |

Таблица 38 – Описание структуры записей таблицы «Server\_Storage»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| ServerId | Код конфигурации сервера | int | Первичный, внешний |
| StorageId | Код накопителя | int | Первичный, внешний |
| Count | Кол-во установленных накопителей (шт.) | tinyint |  |

Таблица 39 – Описание структуры записей таблицы «ServerToPaas»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| PaasTypeId | Код типа платформы | smallint | Первичный, внешний |
| ServerId | Код конфигурации сервера | int | Первичный, внешний |

Таблица 40 – Описание структуры записей таблицы «Service»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | int | Первичный |
| PaasTypeId | Код типа платформы | smallint | Внешний |
| ServiceTypeId | Код типа сервиса | tinyint | Внешний |

Продолжение таблицы 40

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| CoreCount | Кол-во выделенных ядер процессора (шт.) | tinyint |  |
| RamCount | Объем выделенной оперативной памяти (ГБ) | float |  |
| HDDVolume | Объем выделенного пространства накопителей (ГБ) | float |  |
| CreateDate | Дата и время создания сервиса | datetime |  |
| DeleteDate | Дата и время удаления сервиса | datetime |  |
| ServiceStateId | Код состояния сервиса | tinyint | Внешний |
| CostPerHour | Стоимость предоставления (руб./час) | money |  |

Таблица 41 – Описание структуры записей таблицы «ServiceIdle»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | int | Первичный |
| Date | Дата фиксации простоя | datetime |  |
| ServiceId | Код сервиса | int | Внешний |
| BeginPeriod | Дата и время начала простоя | datetime |  |
| EndPeriod | Дата и время окончания простоя | datetime |  |
| IdleTypeId | Код типа простоя | tinyint | Внешний |
| IdleReasonId | Код причины простоя | tinyint | Внешний |
| Cost | Стоимость ущерба от простоя сервиса (руб.) | money |  |

Таблица 42 – Описание структуры записей таблицы «ServiceState»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | tinyint | Первичный |
| Name | Наименование состояния сервиса | nvarchar(50) |  |

Таблица 43 – Описание структуры записей таблицы «ServiceType»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | tinyint | Первичный |
| Name | Наименование типа сервиса | nvarchar(50) |  |

Таблица 44 – Описание структуры записей таблицы «SoftService»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| ServiceId | Код сервиса | int | Первичный, внешний |
| SoftwareId | Код программного обеспечения | int | Первичный, внешний |

Таблица 45 – Описание структуры записей таблицы «SoftType»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | tinyint | Первичный |
| Name | Код типа программного обеспечения | nvarchar(50) |  |

Таблица 46 – Описание структуры записей таблицы «Software»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | int | Первичный |
| Name | Наименование программного обеспечения | nvarchar(255) |  |
| SoftTypeId | Код типа программного обеспечения | tinyint | Внешний |
| Cost | Цена (руб.) | money |  |

Таблица 47 – Описание структуры записей таблицы «Strorage»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | int | Первичный |
| ManufacturerId | Код производителя | smallint | Внешний |
| Model | Наименование модели процессора | nvarchar(100) |  |
| StrorageInterfaceId | Код интерфейса обмена данных | tinyint | Внешний |

Продолжение таблицы 47

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Volume | Объем накопителя (ГБ) | int |  |
| Price | Цена (руб.) | money |  |

Таблица 48 – Описание структуры записей таблицы «StrorageInterface»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| Id | Код | tinyint | Первичный |
| Name | Наименование интерфейса обмена данных | nvarchar(50) |  |

Таблица 49 – Описание структуры записей таблицы «User»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| TabNo | Табельный номер | tinyint | Первичный |
| Login | Логин | nvarchar(50) |  |
| Password | Пароль | nvarchar(50) |  |
| Fio | ФИО | nvarchar(100) |  |
| RightLevelId | Код уровня привилегий | tinyint | Внешний |
| PositionId | Код должности | int | Внешний |

Таблица 50 – Описание структуры записей таблицы «UsingService»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| ServiceId | Код сервиса | int | Первичный, внешний |
| UserTabNo | Табельный номер сотрудника | int | Первичный, внешний |

Таблица 51 – Описание структуры записей таблицы «\_\_MigrationHistory»

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Наименование** | **Тип** | **Ключ** |
| MigrationId | Код | nvarchar(150) | Первичный, внешний |
| ContextKey | Ключевое наименование контекста данных | nvarchar(300) | Первичный, внешний |
| Model | Модель данных | varbinary(MAX) |  |
| ProductVersion | Версия фреймворка | nvarchar(32) |  |

Инфологическая модель БД представлена на рисунке В1.

Средствами заполнения и изменению содержимого базы данных являются:

* приложение «ПАО МТФ-Облачные вычисления»;
* утилита SQL Server Management Studio.

Таблицами, которые имеют предопределенные значения и намеренно не допускается возможность их изменения, являются:

* «IdleType»;
* «IdleReason»;
* «RightsLevel»;
* «ServiceState».

Так как программный продукт реализует 2 задачи, а именно: учет и автоматизированный расчет стоимости использования вычислительных сервисов; учет и автоматизация оценки стоимости их планового/внепланового простоя, то порядок заполнения таблиц несколько отличается.

Для выполнения первой задачи следует сначала зарегистрировать системного администратора, заполнив таблицу «User», указав его должность из таблицы «Position»[[1]](#footnote-1), а также уровень его привилегий доступа к приложению из таблицы «RightsLevel»1. Сразу же рекомендуется зарегистрировать директора и хотя-бы 1 сотрудника, выполнив действия, описанные ранее. Далее необходимо заполнить информацию о вычислительной инфраструктуре предприятия, а именно: таблицы «Server» - конфигурация сервера и «SAN» - конфигурацию хранилища данных. Однако, пред заполнением 2-х последних таблиц следует более детально описать характеристики комплектующих. Таблица «Server» имеет внешние ключи к таблицам «CPU» и «Platform». Для таблицы «СPU» нужно предварительно заполнить таблицы «Manufacturer» и «CpuSocket», которая также имеет внешний ключ к таблице «Manufacturer». Перед заполнением таблицы «Platform» следует заполнить таблицы что и для «CPU» (либо воспользоваться существующими записями, если таковые имеются), включая «RamType» и «Platform\_StorageInt». Но на этом заполнение характеристик для таблицы «Server» не окончено – также полагается добавить записи в таблицы «Server\_Storage» и «Server\_RAM». Таблица «Server\_Storage» имеет внешние ключи к таблицам «Server» и «Strorage». Так как таблица «Server» была предварительно заполнена, то остается только «Strorage», которую, прежде чем заполнить следует добавить запись в «StrorageInterface» и «Manufacturer» (либо воспользоваться существующими записями, если таковые имеются). Таблица «Server\_RAM» имеет внешние ключик таблицам «Server» и «RAM». Так как таблица «Server» была предварительно заполнена, то остается только «RAM», которую, прежде чем заполнить следует добавить запись в «RamType» и «Manufacturer» (либо воспользоваться существующими записями, если таковые имеются). После заполнения конфигурации сервера и его характеристик можно приступить к описанию хранилища данных, а именно – таблицы «SAN». Для заполнения данной таблицы требуется предварительно создать записи в таблице «Manufacturer». Также требуется добавить дополнительную информацию в таблицы «SAN\_StorageInt» и «SAN\_Storage». Таблица «SAN\_StorageInt» имеет внешние ключи к таблицам «SAN» и «StrorageInterface». Так как таблица «SAN» предварительно заполнена, то остается добавить записи в «StrorageInterface», либо воспользоваться существующими записями. Таблица «SAN\_Storage» имеет внешние ключи к таблицам «SAN» и «Strorage». Так как таблица «SAN» была предварительно заполнена, то остается добавить записи «Strorage», либо воспользоваться уже имеющимися. В случае добавления новой записи, следует добавить информацию в таблицы «StrorageInterface» и «Manufacturer», либо воспользоваться уже существующими записями. После заполнения информации о вычислительной инфраструктуре предприятия, можно приступить к добавлению записей об предоставляемых сервисах и распределять доступные вычислительные ресурсы. Для начала следует заполнить таблицу с типами платформ, а именно «PaasType». После этого можно связать существующие конфигурации серверов из таблицы «Server»cтипами платформ из таблицы «PaasType» в таблице «ServerToPaas», а также конфигурации хранилищ данных из таблицы «SAN» в таблице «SANtoPaas», тем самым определяя – какие ресурсы будут доступны на той или иной платформе. На основании доступных вычислительных ресурсов пользователь может их распределить по предоставляемым конфигурациям сервисов в таблице «Service». Она имеет внешние ключи к таблицам «PaasType», «ServiceType» и «ServiceState». Так как таблица «PaasType» была предварительно заполнена, то следует выбрать существующую запись для определения доступных ресурсов вычислительной инфраструктуры, распределенные между платформами ранее. Следует заполнить только таблицу «ServiceType», так как «ServiceState» имеет предопределенные значения. Также требуется распределить программное обеспечение на сервисы в таблице «SoftService» между существующими конфигурациями сервисов из таблицы «Service» и его описания содержащегося в таблице «Software», предварительно добавив туда записи, но перед этим следует заполнить таблицу «SoftType». После создания конфигураций сервисов сотрудник может подписаться на них через таблицу «UsingService».

Для выполнения 2 задачи, необходимо заполнить таблице в том порядке, который был описан для 1 задачи, при этом необходимо добавить записи в таблицу «ServiceIdle». Она имеет внешние ключи к таблицам «IdleType» и «IdleReason», которые не следует заполнять, так как они имеют предопределенные значения. Также рекомендуется заполнить таблицу расписания планового сервера «Server\_MaintenanceShedule» для существующих конфигураций сервисов из таблицы «Server»и таблицу планового обслуживания хранилищ данных «SAN\_MaintenanceShedule» для существующих конфигураций хранилищ данных из таблицы «SAN», тем самым можно добавлять корректные значения записей в таблицу «ServiceIdle», если существует связьмежду оборудованием из таблиц «Server» и «SAN»и конфигурациями сервисов из таблицы «Service», порядок создания которой был описан в 1 задаче.

Для обеспечения целостности базы данных на всех внешних ключах таблиц установлено каскадное обновление и удаление данных. А для защиты от несанкционированного доступа предоставляется учетная запись, описание которой представлено в таблице 15.

# **3 Описание программного обеспечения**

## **3.1 Описание ОС**

Программный продукт разрабатывался под управлением операционной системы Windows 10 редакции Pro 64-х разрядной версии. Основными ее достоинствами являются:

* универсальность. Может использоваться как на настольных ПК, так и на карманных, переносных устройствах. Все официальные приложения устанавливаются на любой гаджет с поддержкой Windows 10, а их интерфейс подстраивается под его особенности;
* большое количество редакций. Пользователь имеет возможность выбрать набор необходимого функционала, подходящий под его повседневные задачи, не переплачивая за ненужные функции;
* улучшенная производительность работы, в сравнении с предыдущими операционными системами компании Microsoft;
* минимальные системные требования не изменились с момента выхода Windows 7;
* любое обновление устанавливается в автоматическом режиме, не требующее перезагрузки ПК.

Одним из главных недостатков можно выделить проблемы с приватностью. Лицензионное соглашение Windows 10 дает Microsoft право на сбор вашей персональной информации и статистики – данных о местоположении, истории браузера и т.д. Как заверяет производитель, информация используется для подбора рекламных материалов и усовершенствования программных продуктов компании. Но многие пользователи слабо верят такому утверждению и не торопятся переходить на новую версию ОС.

## **3.2 Описание СУБД**

Для данного программного продукта была разработана база данных в СУБД «Microsoft SQL Server» версии 11.0.6248.0, используя утилиту «SQLServer Management Studio» версии 14.0.17199.0 для ее конфигурирования и администрирования.

Функциональные возможности СУБД:

* в качестве языка запросов используетсядиалект SQL, получивший название Transact-SQL (сокращённо T-SQL), являющуюся реализацией SQL-92 (стандарт ISO для SQL) с множественными расширениями;
* T-SQL позволяет использовать дополнительный синтаксис для хранимых процедур и обеспечивает поддержку транзакций (взаимодействие базы данных с управляющим приложением);
* поддержка Open Database Connectivity (ODBC) — интерфейс взаимодействия приложений с СУБД;
* начиная с версии SQL Server 2005 обеспечивается возможность подключения пользователей через веб-сервисы, использующие протокол SOAP. Это позволяет клиентским программам, не предназначенным для Windows, кроссплатформенно соединяться с SQL Server;
* поддержка зеркалирования и кластеризации баз данных. Кластер сервера SQL — это совокупность одинаково конфигурированных серверов; такая схема помогает распределить рабочую нагрузку между несколькими серверами;
* поддержка избыточного дублирования данных по трем сценариям:
  1. снимок: производится «снимок» базы данных, который сервер отправляет получателям;
  2. история изменений: всё изменения базы данных непрерывно передаются пользователям;
  3. синхронизация с другими серверами: Базы данных нескольких серверов синхронизируются между собой. Изменения всех баз данных происходят независимо друг от друга на каждом сервере, а при синхронизации происходит сверка данных. Данный тип дублирования предусматривает возможность разрешения противоречий между БД;
* встроена поддержка .NET Framework. Благодаря этому, хранимые процедуры БД могут быть написаны на любом языке платформы .NET, используя полный набор библиотек, доступных для .NET Framework, включая Common Type System (система обращения с типами данных в Microsoft .NET Framework).

Функциональные возможности утилиты по администрированию СУБД:

* подключение к любому компоненту SQL Server, таким как:
  1. Database Engine;
  2. службы Integration Services;
  3. службы Analysis Services;
  4. службы Reporting Services;
* встроен обозреватель объектов, который позволяет просматривать все объекты сервера, и предоставляет графический интерфейс для управления этими объектами;
* редактор запросов на языкеT-SQL;
* просмотр плана выполнения запросов;
* группировка сценариев в единое решение;
* использование встроенныхT-SQL шаблонов;
* монитор активности;
* создание резервных копий БД и восстановление из-под них;
* настройка свойств сервера, баз данных и других объектов;
* управление безопасностью сервера;
* настройка репликации баз данных.

## **3.3 Описание среды программирования и языка программирования**

Данный программный продукт был разработан с помощью интегрированной среды разработки MicrosoftVisualStudio 2017 редакции Communitycиспользованием языка C#.

Интерактивная интегрированная среда разработки Visual Studio — это оригинальная среда запуска, которая позволяет просматривать и изменять практически любой код, а также отлаживать, выполнять сборку и публиковать приложения для устройств с Android, iOS, Windows, а также в Интернете и облаке. Доступны версии как для компьютеров Mac, так и для компьютеров с Windows.

Функциональные возможности интегрированной среды разработки:

* разработка программ с использованием следующих языков программирования: C++, C#, Visual Basic, F#, JavaScript, TypeScript, Python.
* создание мобильных приложенийс помощью C#, Xamarin, Visual C++ или гибридные приложения на JavaScript с Apache Cordova;
* создание игр для мобильных устройств с использованием платформ Unity, Unreal, DirectX, Cocos и других;
* содержит эмулятор Android для запуска и отладки приложений Android;
* предоставляется набор инструментов, позволяющих с легкостью создавать облачные приложения на базе Microsoft Azure;
* создание веб-приложений с помощью технологий ASP.NET, Node.js, Python, JavaScript и TypeScript;
* использование большое числа встроенных шаблонов проектов;
* использование графического конструктора для создания пользовательского интерфейса;
* рефакторинг включает такие операции, как интеллектуальное переименование переменных, перемещение выделенных строк кода в отдельную функцию, перемещение кода в другие расположения, изменение порядка параметров функции и т. д;
* IntelliSense — это общий термин для набора очень популярных функций, отображающих сведения о типах в коде непосредственно в редакторе и в некоторых случаях, автоматически создающих небольшие отрывки кода;
* обозреватель объектов, позволяет исследовать сборки .NET или среды выполнения Windows в системе, просматривая содержащиеся в них типы;
* включает набор средств Microsoft SQL Server Data Tools (SSDT), которые используются для сборки, отладки, обслуживания и рефакторинга баз данных. Можно работать с проектом базы данных или напрямую с подключенным экземпляром базы данных (локально или удаленно).

Основные характеристики языка программирования:

* объектно-ориентированный язык со строгой типизацией, позволяющий разработчикам создавать различные безопасные и надежные приложения, работающие на платформе .NET Framework;
* инкапсулированные сигнатуры методов, именуемые делегатами, которые позволяют реализовать типобезопасные уведомления о событиях;
* свойства, выполняющие функцию акцессоров для закрытых переменных-членов;
* атрибуты, предоставляющие декларативные метаданные о типах во время выполнения;
* внутристрочные комментарии для XML-документации;
* LINQ для создания запросов к различным источникам данных.

## **3.4 Описание используемых компонент и проектных процедур**

Перечень используемых компонент представлен в таблице 52.

Таблица 52 – Перечень используемых компонент

|  |  |
| --- | --- |
| **Наименование компонента** | **Описание** |
| Button | Позволяет пользователю щелкнуть его для выполнения действия. На элементе управления могут отображаться текст и изображение. При щелчке кнопки мышью элемент управления выглядит так, как будто его нажимают и отпускают. |
| DataGridView | Предоставляет мощный и гибкий способ отображения данных в табличном формате. Элемент управления можно использовать для отображения представлений небольшого объема данных только для чтения, либо можно масштабировать его для отображения редактируемого представления очень больших наборов данных.  Для того чтобы реализовать пользовательское поведение в приложениях, элемент управления можно расширить несколькими способами. Например, можно программно задать собственные алгоритмы сортировки, создать собственные типы ячеек. В качестве источника данных могут использоваться хранилища данных различных типов, также может работать и без него. |
| ComboBox | Используется для отображения данных в раскрывающемся поле со списком. По умолчанию элемент управления состоит из двух частей: верхняя часть представляет собой текстовое поле, которое пользователь может ввести элемент списка. Вторая часть представляет собой список элементов, из которых пользователь может выбирать требуемый. |
| MenuStrip | Представляет собой главное меню формы. |
| ToolStrip | Панель инструментов, на которой можно разместить меню, элементы управления и пользовательские элементы управления в приложениях Windows Forms. |

Продолжение таблицы 52

|  |  |
| --- | --- |
| **Наименование компонента** | **Описание** |
| StatusStrip | Используется в формах в качестве области, обычно отображаемой в нижней части окна, в которой выводятся различные сведения о состоянии приложения. Элементы управления StatusStrip обычно имеют встроенные элементы управления ToolStripStatusLabel, отображающие текст или значки, которые показывают состояние, или ToolStripProgressBar, который отображает состояние завершения процесса графически. |
| ToolTip | Отображает текст при наведении указателя мыши на элементы управления. Компонент можно связать с любым элементом управления. |
| BindingSource | Инкапсулирует источник данных для привязки к элементам управления.  Компонент служит двум целям. Он обеспечивает уровень косвенного обращения при выполнении привязки элементов управления в форме к данным. Это достигается путем привязки компонента BindingSource к источнику данных и последующей привязки элементов управления в форме к компоненту BindingSource. Все последующие взаимодействия с данными, включая перемещение, сортировку, фильтрацию и обновление, осуществляются с помощью вызовов компонента BindingSource.  Добавление типа в компонент BindingSource с методом Add создает список этого типа. |
| Timer | Вызывает событие через определенные интервалы времени. |
| ContextMenuStrip | Предоставляет контекстное меню, связываемое с каким-либо элементом управления. |
| Label | Используется для отображения текста или изображения, которые не могут быть изменены пользователем. Отображаемые элементы обычно используются для идентификации объектов в форме. |

Продолжение таблицы 52

|  |  |
| --- | --- |
| **Наименование компонента** | **Описание** |
| TabControl | Используется для отображения нескольких вкладок, аналогичных разделителям в записной книжке или меткам в наборе папок в картотеке. Вкладки могут содержать различные элементы управления. |
| TableLayoutPanel | Упорядочивает содержимое в сетке. Поскольку макет выполняется как во время разработки, так и во время выполнения, его можно изменять динамически по мере изменения среды приложения. Это позволяет пропорционально изменять размер элементов управления в панели, так чтобы они учитывали изменения, такие как изменение размера родительского элемента управления или длины текста в случае локализации. |
| TextBox | Используются для получения входных данных от пользователя или для отображения текста. Текстовое полеможет иметь многострочный вид, осуществляя перенос текста по размеру элемента управления. |
| PictureBox | Используется для отображения графических объектов в формат растрового изображения, GIF, JPEG, метафайла или значок. |
| Panel | Используются для предоставления возможности группировки других элементов управления. Как правило панели используются для разделения формы по функциям. Аналогичен элементу GroupBox. Однако, тольков Panel можно использовать полосы прокрутки, но при этом в GroupBox имеется возможность отображения заголовка. |
| GroupBox | Используются для предоставления возможности группировки других элементов управления. Как правило группы используются для разделения формы по функциям. Аналогичен элементу Panel. Однако, только в GroupBox имеется возможность отображения заголовка, при этом в Panel можно использовать полосы прокрутки. |

Продолжение таблицы 52

|  |  |
| --- | --- |
| **Наименование компонента** | **Описание** |
| ListBox | Отображает список элементов, из которых пользователь может выбрать один или несколько необходимых. |
| DateTimePicker | Позволяет пользователю выбрать один элемент из списка дат или времени. Используемый для представления даты, он отображается в виде двух частей: раскрывающегося списка с даты, представленной в тексте и сетки, которая появляется, если щелкнуть стрелку вниз рядом со списком. |
| NumericUpDown | Выглядит как сочетание текстового поля и пары кнопок со стрелками для корректировки значения пользователем. Он выводит и задает отдельное числовое значение из списка вариантов. |
| ReportViewer | Включает возможности обработки и просмотра отчетов непосредственно в приложении. |

ПП состоит из решения, которое включает в себяследующие проекты:

* MTF\_Services.DataAccess (доступ к базе данных);
* MTF\_Services.Model (включает представления и сущности);
* MTF\_Services.WinForms (основноймодуль);
* MTF\_Servises.Installer (модульустановки).

Описание проектных процедур каждого проекта представлено в таблицах 53− 55.

Таблица 53 – Описание проектных процедур проекта «MTF\_Services.DataAccess»

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| MTF\_ServicesDbContext.cs | MTF\_ServicesDbContext | Конструктор класса контекста модели данных |
| OnModelCreating | Вызывается при инициализации модели для производного контекста, в котором определены дополнительные действия при развертывании модели данных |

Продолжение таблицы 53

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| Configuration.cs | Configuration | Конструктор класса конфигурации модели данных |
| Seed | Выполняется после развертывания последней миграции, обновляя записи в таблицах |
| 201806080127337\_Main.cs | Up | Перечень выполняемых операций с сущностями при обновлении базы данных |
| Down | Перечень выполняемых операций с сущностями при откате изменений базы данных |

Таблица 54 – Описание проектных процедур проекта «MTF\_Services.Model»

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| ConfigurationPartsInfocs | UpdateTotalPrice | Обновление итоговой стоимостиконфигурации сервера |
| PlatformServerItem.cs | PlatformServerItem | Конструктор класса описания сервера платформы |
| PlatfromSANItem.cs | PlatfromSANItem | Конструктор класса описания хранилища данных платформы |
| SANPartsInfo | UpdateTotalPrice | Обновление итоговой стоимостихранилища данных |
| ServicePartsInfo.cs | CalcSummaryVMCost | Расчет суммарной минимальной стоимости виртуальной машины |
| CalcCPUCostByHour | Расчет стоимости 1 ГГц процессора в час |
| CalcRAMCostByHour | Расчет стоимости 1 ГБ оперативной памяти в час |
| CalcServerStorageCostByHour | Расчет стоимости 1 ГБ накопителей сервера в час |
| CalcSanStorageCostByHour | Расчет стоимости 1 ГБ накопителей хранилища данных в час |

Продолжение таблицы 54

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| ServicePartsInfo.cs | CalcSoftwareCostByHour | Расчет стоимости программного обеспечения в час |
| CalcServerMaintenanceByHour | Расчет стоимости обслуживания сервера в час |
| CalcSanMaintenanceByHour | Расчет стоимости обслуживания хранилища данных в час |
| CalcTotalMaintenanceByHour | Расчет общей стоимости обслуживания оборудования в час |
| CalculateFinalPrice | Расчет итоговой стоимости сервиса |

Таблица 55 – Описание проектных процедур проекта «MTF\_Services.WinForms»

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| Program.cs | Main | Главная точка входа для приложения |
| CurrentDomain\_UnhandledException | Обработчик события перехвата необработанного исключения в текущем домене приложений |
| Application\_ThreadException | Обработчик события перехвата необработанного исключения в текущем потоке |
| ShowUserProfileForm.cs | ShowUserProfileForm | Конструктор формы просмотра данных текущей учетной записи |
| label2\_MouseHover | Обработчик события перемещения курсора мыши на метке "Пароль», который отображает пароль |
| label2\_MouseLeave | Обработчик события выхода курсора мыши на метке "Пароль», который скрывает пароль |
| ReportingForm.cs | ReportingForm(ReportType reportType) | Конструктор формы предварительного просмотра отчетов |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| ReportingForm.cs | ReportingForm(ServiceRequestItem serviceRequestItem, List<SoftwareInfo> softwaresInfo) | Конструктор формы для предварительного просмотра заявки на предоставление сервиса |
| ReportingForm(BindingList <ServiceDetailInfo> servicesDetailInfo, string costColumnHeaderName) | Конструктор формы для предварительного просмотра стоимости предоставления сервисов |
| ReportingForm(ServiceIdleParams serviceIdleParams) | Конструктор формы для предварительного просмотра информации о простое сервиса |
| MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния |
| MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |
| SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |
| SubScribeChildMenuItems(ToolStripItemCollection col) | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в строке состояния |
| ReportingForm\_Load | Обработчик события загрузки формы |
| LoginForm.cs | LoginForm | Конструктор формы авторизации пользователя |
| TextBox1\_KeyPress | Обработчик события нажатия клавиш на текстовом поле, который при нажатии "Enter" производит попытку авторизации. |
| LoginForm\_FormClosing | Обработчик события закрытия формы |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| LoginForm.cs | button1\_Click | Обработчик события нажатия на кнопку авторизации |
| label3\_MouseHover | Обработчик события перемещения курсора мыши на метке "Пароль", который отображает пароль |
| label3\_MouseLeave | Обработчик события выхода курсора мыши на метке "Пароль», который скрывает пароль |
| SysAdminForm.cs | UpdateBindings | Обновление привязок данных |
| HideServiceGrid | Скрытие таблицы с информацией о сервисах |
| HidePaasGrid | Скрытие таблицы с информацией о платформах |
| UpdateLabels | Обновление таблиц и меток, которые сообщают об отсутствии информации |
| SysAdminForm\_FormClosing | Обработчик события закрытия формы |
| MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния. |
| MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |
| SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |
| SubScribeChildMenuItems | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в строке состояния |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| SysAdminForm.cs | просмотрПрофиляToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы просмотра и редактирования текущей учетной записи |
| редактированиеИнфраструктурыToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который выполняет переход на форму редактирования инфраструктуры |
| простойToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который выполняет переход в меню простоя инфраструктуры |
| редактированиеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы редактирования учетных записей пользователей |
| регистрацияToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы регистрации новой учетной записи |
| сменитьПользователяToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который совершает выход из учетной записи |
| завершениеРаботыToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который завершает работу приложения |
| добавитьНовуюToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который выполняет переход на форму создания новой платформы |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| SysAdminForm.cs | редактироватьВыбраннуюToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, выполняет переход на форму редактирования выбранной платформы |
| добавитьНовыйToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который выполняет переход на форму создания нового сервиса |
| удалитьВыбранныйToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который выполняет удаление выбранной категории сервиса |
| SysAdminForm\_Activated | Обработчик события активации формы |
| удалитьВыбраннуюToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который выполняет удаление выбранной платформы |
| редакторПлатформИСервисовToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который выполняет переход на форму редактора платформ и сервисов |
| timer\_Tick | Обработчик события выполнения такта таймера |
| UserListForm.cs | UserListForm | Конструктор формы редактирования пользователей |
| InitBindings | Инициализация привязок данных |
| UpdateStatBar | Обновление строки состояния |
| MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния. |
| MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| UserListForm.cs | SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |
| SubScribeChildMenuItems | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в строке состояния |
| userInfoBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных пользователей |
| новыйПользовательToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы регистрации нового пользователя |
| редактироватьToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы редактирования выбранного пользователя |
| удалитьToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который выполняет удаление выбранного пользователя |
| всеЗаписиToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который выполняет просмотр всех зарегистрированных пользователей |
| отборToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно задания критериев для отбора зарегистрированных пользователей |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| UserListForm.cs | UserListForm\_KeyDown | Обработчик события нажатия клавиш клавиатуры на форме |
| UserConditionForm.cs | UserConditionForm | Конструктор формы отбора пользователей |
| InitBindings | Инициализация привязок |
| rightsLevelBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных уровней привилегий |
| positionBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных должностей |
| RunSelection | Выполнение отбора пользователей по условию |
| pictureBox1\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит отмену выбора в списке уровней привилегий |
| pictureBox2\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит отмену выбора в списке должностей |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену ввода условия отбора пользователей, закрывая форму |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит подтверждение совершенного отбора |
| comboBox1\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке уровней привилегии |
| comboBox2\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке должностей |
| EditUsrPositionForm.cs | EditUsrPositionForm() | Конструктор формы редактирования должностей сотрудников |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditUsrPositionForm.cs | EditUsrPositionForm(bool findMode) | Конструктор формы для выбора должности из списка |
| BindCollection | Привязка коллекции |
| EnDisFields | Активация/деактивация полей для редактирования |
| button2\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит добавление новой должности |
| button3\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит редактирование выбранной должности |
| btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |
| EditUsrPositionForm\_FormClosing | Обработчик события закрытия формы |
| positionBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных должностей |
| but\_Select\_Click | Обработчик события нажатия клавиши мыши на кнопку, который подтверждает выбранный элемент из списка должностей |
| textBox1\_KeyPress | Обработчик события нажатия клавиши в текстовом поле с наименованием должности, который ограничивает ввод только русских букв, дефиса, пробела и управляющих символов |
| textBox3\_KeyPress | Обработчик события нажатия клавиши в текстовом поле с кол-вом рабочих часов,который ограничивает ввод только цифр и управляющих символов |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditUsrPositionForm.cs | textBox2\_KeyPress | Обработчик события нажатия клавиши в текстовом поле с размером средней зарплаты, который ограничивает ввод только цифр, управляющих символов, и 1 запятой |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену последнего добавления/редактирования, закрывая форму |
| EditUsrPositionForm\_KeyDown | Обработчик события нажатий клавиш клавиатуры на форме |
| EditUsersForm.cs | EditUsersForm | Конструктор формы регистрации нового польователя |
| EditUsersForm(User selectedUser) | Конструктор формы редактирования учетных данных пользователя |
| InitBindings | Инициализация привязок |
| BindPositions | Привязка должностей сотрудника |
| pictureBox3\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно редактирования должностей сотрудников |
| pictureBox2\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно поиска должности сотрудника |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит отмену последнего добавления/редактирования, закрывая форму |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditUsersForm.cs | btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |
| EditUsersForm\_FormClosing | Обработчик события закрытия формы |
| rightsLevelBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных уровней привилегий |
| positionBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных должностей |
| pictureBox1\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит отмену выбранной должности |
| comboBox2\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке должностей |
| textBox1\_KeyPress | Обработчик события нажатия клавиши в текстовом поле с Логином, который ограничивает ввод только латиницы, цифр, символа нижнего подчеркивания и управляющих символов |
| textBox2\_KeyPress | Обработчик события нажатия клавиши в текстовом поле с ФИО, который ограничивает ввод только русских букв, пробела и управляющих символов |
| EditUsersForm\_KeyDown | Обработчик события нажатий клавиш клавиатуры на форме |
| ListOfServicesForm.cs | ListOfServicesForm | Конструктор формы списка сервисов |
| MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния. |
| MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| ListOfServicesForm.cs | SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |
| SubScribeChildMenuItems | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в |
| BindPlatformList | Инициализация привязки списка платформ с информацией о количестве пользователей и сервисов |
| BindServicesList | Инициализация привязки полного списка конфигураций сервисов с детальным описанием |
| UpdateStatBar | Обновление строки состояния |
| platformServiceUserBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных списка платформ |
| serviceDetailInfoBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных списка сервисов |
| picBtn\_AddNewPlatform\_Click | Обработчик события нажатия клавиши мыши на графический объект, который совершает переход на форму создания новой платформы |
| picBtn\_EditSelectedPlatform\_Click | Обработчик события нажатия клавиши мыши на графический объект, который совершает переход на форму редактирования выбранной платформы |
| picBtn\_DeleteSelectedPlatform\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление выбранной платформы |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| ListOfServicesForm.cs | picBtn\_AddNewService\_Click | Обработчик события нажатия клавиши мыши на графический объект, который совершает переход на форму создания нового сервиса |
| picBtn\_EditSelectedService\_Click | Обработчик события нажатия клавиши мыши на графический объект, который совершает переход на форму редактирования выбранного сервиса |
| picBtn\_DeleteSelectedService\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление выбранного сервиса |
| ttmi\_ShowAllServices\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который производит просмотр всех конфигураций сервисов |
| ttmi\_ShowServicesBySelectedPlatform\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который производит отбор сервисов по выбранной платформе |
| ListOfServicesForm\_Activated | Обработчик события активации формы |
| ListOfServicesForm\_FormClosing | Обработчик события закрытия формы |
| ListOfServicesForm\_KeyDown | Обработчик события нажатий клавиш клавиатуры на форме |
| EditServiceForm.cs | EditServiceForm | Конструктор формы создания нового сервиса |
| EditServiceForm(PaasType selectedPlatform) | Конструктор формы создания нового сервиса с выбранной платформой |
| EditServiceForm(Service selectedService) | Конструктор формы редактирования выбранного сервиса |
| PrepareBindingsToEditService | Инициализация привязок для редактирования конфигурации сервиса |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditServiceForm.cs | MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния. |
| MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |
| SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |
| SubScribeChildMenuItems | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в строке состояния |
| BindPlatforms | Инициализация привязки платформ |
| BindServiceTypes | Инициализация привязки типов сервиса |
| BindServerPartsInfo | Инициализация привязки информации о стоимости компонентов сервиса |
| UpdateServerPartsInfo | Обновление привязки информации о стоимости компонентов сервиса |
| BindSelectedSoftware | Привязка списка выбранного программного обеспечения |
| UpdateSelectedSoftware | Обновление привязки списка выбранного программного обеспечения |
| paasTypeBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных типов платформ |
| platformServerItemBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных серверов |
| platfromSANItemBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных хранилищ данных |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditServiceForm.cs | numericUpDown1\_ValueChanged | Обработчик события изменения значения в numericUpDown, который отвечает за кол-во используемых ядер |
| numericUpDown2\_ValueChanged | Обработчик события изменения значения в numericUpDown, который отвечает за кол-во используемой оперативной памяти |
| numericUpDown3\_ValueChanged | Обработчик события изменения значения в numericUpDown, который отвечает за кол-во используемый объем жесткого диска |
| pictureBox3\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно поиска типа сервиса |
| pictureBox5\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно редактирования типов сервисов |
| pictureBox2\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит отмену выбранного сервиса |
| serviceTypeBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных типов сервиса |
| picBtn\_AddSoftware\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно добавления программного обеспечения |
| picBtn\_DeleteSoftware\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление выбранного программного обеспечения |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditServiceForm.cs | btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену добавления/редактирования, закрывая форму |
| EditServiceForm\_FormClosing | Обработчик события закрытия формы |
| pictureBox1\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление выбранного типа платформы |
| serverPlarformInfoBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных платформ сервера |
| sANPlatformInfoBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных хранилищ данных |
| EditServiceForm\_KeyDown | Обработчик события нажатий клавиш клавиатуры на форме |
| EditPlatformForm.cs | EditPlatformForm | Конструктор формы создания новой платформы |
| EditPlatformForm(PaasType selectedPaas) | Конструктор формы редактирования выбранной платформы |
| InitEditPaas | Инициализация привязок выбранной платформы для редактирования |
| BindSelectedServers | Привязка списка выбранных конфигураций серверов |
| BindSelectedSANs | Привязка списка выбранных хранилищ данных |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditPlatformForm.cs | picBtn\_AddServer\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно добавления сервера к платформе |
| picBtn\_DeleteServer\_Click | Обработчик события нажатия клавиши мыши на графический объект,который осуществляет удаление выбранного сервера |
| picBtn\_AddSAN\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно добавления хранилища данных к платформе |
| picBtn\_DeleteSAN\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет удаление выбранного хранилища данных |
| btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену добавления/редактирования, закрывая форму |
| EditPlatformForm\_FormClosing | Обработчик события закрытия формы |
| EditPlatformForm\_KeyDown | Обработчик события нажатий клавиш клавиатуры на форме |
| SelectSoftwareForm.cs | SelectSoftwareForm | Конструктор формы выбора программного обеспечения |
| InitBindings | Инициализация привязок данных |
| MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния. |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| SelectSoftwareForm.cs | MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |
| SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |
| SubScribeChildMenuItems | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в строке состояния |
| созданиеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы создания нового программного обеспечения |
| редактированиеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы редактирования выбранного программного обеспечения |
| удалениеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который выполняет удаление выбранного программного обеспечения |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит подтверждение выбранного программного обеспечения |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену выбора программного обеспечения, закрывая форму |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| SelectSoftwareForm.cs | softwareInfoBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных программного обеспечения |
| SelectSoftwareForm\_FormClosing | Обработчик события закрытия формы |
| SelectSoftwareForm\_KeyDown | Обработчик события нажатий клавиш клавиатуры на форме |
| SelectServerForm.cs | SelectServerForm | Конструктор формы выбора конфигурации сервера |
| InitCollection | Инициализация привязки коллекции конфигураций серверов |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену выбора конфигурации сервера, закрывая форму |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит подтверждение выбранной конфигурации сервера |
| serverPaasInfoBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных конфигураций серверов |
| SelectServerForm\_KeyDown | Обработчик события нажатий клавиш клавиатуры на форме |
| SelectSANForm.cs | SelectSANForm | Конструктор формы выбора хранилища данных |
| InitCollection | Инициализация привязки коллекции хранилищ данных |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену выбора хранилища данных, закрывая форму |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| SelectSANForm.cs | btn\_OK\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит подтверждение выбранного хранилища данных |
| sANPaasInfoBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных конфигураций хранилищ данных |
| SelectSANForm\_KeyDown | Обработчик события нажатий клавиш клавиатуры на форме |
| EditSoftwareTypeForm.cs | EditSoftwareTypeForm | Конструктор формы редактирования типов программного обеспечения |
| EditSoftwareTypeForm(bool findMode) | Конструктор для выбора типа программного обеспечения из списка |
| BindCollection | Привязка коллекции |
| EnDisFields | Активация/деактивация полей для редактирования |
| softTypeBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных типов программного обеспечения |
| button2\_Click | Обработчик события нажатия клавиши мыши на кнопку, который совершает добавление нового типа программного обеспечения |
| button3\_Click | Обработчик события нажатия клавиши мыши на кнопку, который совершает редактирование выбранного типа программного обеспечения |
| btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditSoftwareTypeForm.cs | btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену последнего добавления/редактирования |
| but\_Select\_Click | Обработчик события нажатия клавиши мыши на кнопку, который совершает подтверждение выбранного типа программного обеспечения |
| EditSoftwareTypeForm\_FormClosing | Обработчик события закрытия формы |
| textBox2\_KeyPress | Ограничение ввода в наименовании добавляемого/редактируемого программного обеспечения |
| picBtn\_ClearCondition\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит очистку строки для поиска |
| picBtn\_FindByCondition\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит поиск типа программного обеспечения по наименованию |
| EditSoftwareTypeForm\_KeyDown | Обработчик события нажатий клавиш клавиатуры на форме |
| EditSoftwareForm.cs | EditSoftwareForm | Конструктор формы редактирования программного обеспечения |
| EditSoftwareForm(Software selectedSoftware) | Конструктор формы редактирования выбранного программного обеспечения |
| InitEditSoftware | Обновление привязки для редактирования выбранного программного обеспечения |
| BindCollection | Привязка коллекции |
| softwareInfoBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных программного обеспечения |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditSoftwareForm.cs | EnDisFields | Активация/деактивация полей для редактирования |
| button2\_Click | Обработчик события нажатия клавиши мыши на кнопку, который совершает добавление нового программного обеспечения |
| button3\_Click | Обработчик события нажатия клавиши мыши на кнопку, который совершает редактирование выбранного программного обеспечения |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену последнего добавления/редактирования |
| btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |
| EditSoftwareForm\_FormClosing | Обработчик события закрытия формы |
| but\_Select\_Click | Обработчик события нажатия клавиши мыши на кнопку, который совершает подтверждение выбранного программного обеспечения |
| pictureBox2\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит отмену выбранного типа программного обеспечения для поиска |
| softTypeBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных типов программного обеспечения |
| EditSoftwareForm\_KeyDown | Обработчик события нажатий клавиш клавиатуры на форме |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditServiceTypeForm.cs | EditServiceTypeForm | Конструктор формы редактирования типов сервиса |
| EditServiceTypeForm(bool findMode) | Конструктор для выбора типа сервиса из списка |
| BindCollection | Привязка коллекции |
| EnDisFields | Активация/деактивация полей для редактирования |
| serviceTypeBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных типов сервиса |
| button2\_Click | Обработчик события нажатия клавиши мыши на кнопку, который совершает добавление нового типа сервиса |
| button3\_Click | Обработчик события нажатия клавиши мыши на кнопку, который совершает редактирование выбранного типа сервиса |
| btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену последнего добавления/редактирования |
| but\_Select\_Click | Обработчик события нажатия клавиши мыши на кнопку, который совершает подтверждение выбранного типа сервиса |
| EditServiceTypeForm\_FormClosing | Обработчик события закрытия формы |
| textBox2\_KeyPress | Ограничение ввода в наименовании добавляемого/редактируемого типа сервиса |
| picBtn\_ClearCondition\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит очистку строки для поиска |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditServiceTypeForm.cs | picBtn\_FindByCondition\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит поиск типа сервиса по наименованию |
| EditServiceTypeForm\_KeyDown | Обработчик события нажатий клавиш клавиатуры на форме |
| EditInfrastructureFrom.cs | EditInfrastructureFrom | Конструктор формы редактирования вычислительной инфраструктуры |
| UpdateBindings | Обновление привязок данных |
| HideServerGrid | Скрытие таблицы с кратким описанием конфигураций серверов |
| HideSanGrid | Скрытие таблицы с кратким описанием конфигураций систем хранения данных |
| UpdateStatBar | Обновление строки состояния |
| UpdateLabels | Обновление таблиц и меток, которые сообщают об отсутствии информации |
| MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния. |
| MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |
| SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |
| SubScribeChildMenuItems | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в строке состояния |
| EditInfrastructureFrom\_FormClosing | Обработчик события закрытия формы |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditInfrastructureFrom.cs | созданиеНовойToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который выполняет переход на форму добавления новой конфигурации сервера |
| редактированиеВыбраннойToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который выполняет переход на форму редактирования выбранной конфигурации сервера |
| удалениеВыбраннойToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который осуществляет удаление выбранной конфигурации сервера |
| созданиеНовогоToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который выполняет переход на форму создания нового хранилища данных |
| редактированиеВыбранногоToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который выполняет переход на форму редактирования выбранного хранилища данных |
| удалениеВыбранногоToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который осуществляет удаление выбранного хранилища данных |
| EditInfrastructureFrom\_Activated | Обработчик события активации формы |
| timer\_Tick | Обработчик события выполнения такта таймера |
| EditInfrastructureFrom\_KeyDown | Обработчик нажатий клавиш на форме |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditInfrastructureFrom.cs | поискToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно для задания условия поиска конфигурации сервера |
| отборToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно для задания условия отбора конфигураций серверов |
| показатьВсеЗаписиToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который совершает показ всех конфигураций серверов |
| поискToolStripMenuItem1\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно для задания условия поиска конфигурации хранилища данных |
| отборToolStripMenuItem1\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно для задания условия отбора конфигураций хранилищ данных |
| показатьВсеЗаписиToolStripMenuItem1\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который совершает показ всех конфигураций хранилищ данных |
| tabControl1\_SelectedIndexChanged | Обработчик события изменения индекса текущей вкладки |
| EditServerConfigurationForm.cs | EditServerConfigurationForm | Конструктор формы создания новой конфигурации сервера |
| EditServerConfigurationForm(Server selectedServer) | Конструктор формы редактирования выбранной конфигурации сервера |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditServerConfigurationForm.cs | PrepareBindingsToEditServer | Инициализация привязок для редактирования конфигурации сервера |
| ComboBox\_Storage\_Format | Обработчик события форматирования отображаемого значения в списке накопителей |
| ComboBox\_RAM\_Format | Обработчик события форматирования отображаемого значения в списке оперативной памяти |
| ComboBox\_CPU\_Format | Обработчик события форматирования отображаемого значения в списке процессоров |
| ComboBox\_Platform\_Format | Обработчик события форматирования отображаемого значения в списке платформ |
| UpdateGridsFont | Обновление шрифтов таблиц |
| MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния. |
| MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |
| SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |
| SubScribeChildMenuItems | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в строке состояния |
| ManagePlatformControls | Управление элементами управления области платформы |
| ManageCpuControls | Управление элементами управления области процессоров |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditServerConfigurationForm.cs | ManageRamControls | Управление элементами управления области оперативной памяти |
| ManageStorageControls | Управление элементами управления области накопителей |
| BindPlatforms | Инициализация привязки платформ |
| BindInstalledRamOnServer | Инициализация привязки списка установленной памяти |
| BindInstalledStorageOnServer | Инициализация привязки списка установленных накопителей |
| BindCPUs | Получение списка доступных процессоров выбранной платформы |
| BindRAMs | Получение списка доступной оперативной памяти выбранной платформы |
| BindStorages | Получение списка доступных накопителей выбранной платформы |
| BindAvalibleInterfaces | Получение списка доступных интерфейсов накопителей выбранной платформы |
| BindConfigurationPartsInfo | Привязка объекта, который отображает цены компонентов |
| BindPlatfromInfo | Получение подробного описания платформы |
| CheckInstalledRam | Проверка установленной памяти в конфигурации с доступным списком ОЗУ |
| CheckInstalledStorage | Проверка установленных накопителей со списком доступных |
| platformBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных платформ |
| cPUBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных процессоров |
| rAMBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных оперативной памяти |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditServerConfigurationForm.cs | strorageBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных накопителей |
| picBtn\_DeletePlatform\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление текущей платформы из конфигурации |
| picBtn\_SearchPlatform\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно поиска платформы |
| pictureBox3\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление текущего процессора из конфигурации |
| pictureBox4\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление установленной оперативной памяти |
| pictureBox14\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление установленного накопителя |
| pictureBox1\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно поиска процессора |
| pictureBox5\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно поиска оперативной памяти |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditServerConfigurationForm.cs | pictureBox15\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно поиска накопителя |
| comboBox\_CPU\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке процессоров |
| pictureBox7\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет добавление выбранной памяти к установленной |
| pictureBox13\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет добавление выбранного накопителя к установленным |
| lbl\_NoStorages\_Click | Обработчик события нажатия клавиши мыши на текстовую метку, который открывает диалоговое окно добавления нового накопителя, если он не выбран |
| lbl\_NoRams\_Click | Обработчик события нажатия клавиши мыши на текстовую метку, который открывает диалоговое окно добавления новой оперативной памяти, если она не выбрана |
| lbl\_NoCPUs\_Click | Обработчик события нажатия клавиши мыши на текстовую метку, который открывает диалоговое окно добавления нового процессора, если он не выбран |
| pictureBox8\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет увеличение количества выбранной установленной оперативной памяти |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditServerConfigurationForm.cs | pictureBox9\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет уменьшение количества выбранной установленной оперативной памяти |
| pictureBox10\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет удаление выбранной установленной оперативной памяти |
| pictureBox21\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет увеличение количества выбранного установленного накопителя |
| pictureBox20\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет уменьшение количества выбранного установленного накопителя |
| pictureBox19\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет удаление выбранного установленного накопителя |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену последнего добавления/редактирования, закрывая форму |
| EditServerConfigurationForm\_FormClosing | Обработчик события закрытия формы |
| btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditSANForm.cs | EditSANForm | Конструктор формы создания нового хранилища данных |
| EditSANForm(SAN selectedSAN) | Конструктор формы редактирования выбранного хранилища данных |
| PrepareBindingsToEditSAN | Инициализация привязок для редактирования выбранного хранилища данных |
| UpdateGridsFont | Обновление шрифтов таблиц |
| BindManufacturer | Привязка списка производителей |
| BindAvalibleInterfacesNew | Привязка списка доступных интерфейсов накопителей для нового хранилища данных |
| BindAvalibleInterfacesForEdit | Привязка списка доступных интерфейсов накопителей для редактируемого хранилища данных |
| BindStorageInterfaces | Привязка списка существующих интерфейсов |
| BindInstalledStorageOnSAN | Привязка списка установленных накопителей на хранилище данных |
| BindSANPartsInfo | Привязка объекта, который отображает цены компонентов |
| BindStorages | Привязка списка доступных накопителей |
| CheckInstalledStorage | Проверка установленных накопителей со списком доступных |
| ManageStorageControls | Управление элементами управления области накопителей |
| pictureBox7\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет добавление выбранного интерфейса к доступным |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditSANForm.cs | lbl\_NoStorages\_Click | Обработчик события нажатия клавиши мыши на текстовую метку, который открывает диалоговое окно добавления нового накопителя, если он не выбран |
| strorageInterfaceBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных доступных интерфейсов накопителей |
| pictureBox14\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет удаление выбранного накопителя из установленных |
| pictureBox13\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет добавление выбранного накопителя к установленным |
| picBtn\_IncreaseStorage\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет увеличение количества выбранного установленного накопителя |
| picBtn\_DecreaseStorage\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет уменьшение количества выбранного установленного накопителя |
| picBtn\_DeleteStorage\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет удаление выбранного установленного накопителя |
| pictureBox21\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет увеличение количества выбранного установленного интерфейса накопителя |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditSANForm.cs | pictureBox20\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет уменьшение количества выбранного установленного интерфейса накопителя |
| pictureBox19\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет удаление выбранного установленного интерфейса накопителя |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену последнего редактирования, закрывая форму |
| EditSANForm\_FormClosing | Обработчик события закрытия формы |
| btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |
| pictureBox1\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет удаление выбранного производителя |
| manufacturerBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных производителей |
| pictureBox17\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно поиска производителя |
| pictureBox4\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно редактирования производителей |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditSANForm.cs | comboBox1\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке производителей |
| SelectStorageForm.cs | SelectStorageForm | Конструктор формы выбора накопителя |
| InitBindings | Инициализация привязок данных |
| MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния. |
| MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |
| SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |
| SubScribeChildMenuItems | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в строке состояния |
| созданиеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы создания нового накопителя |
| редактированиеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы редактирования выбранного накопителя |
| удалениеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который осуществляет удаление выбранного накопителя |
| storageInfoBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных накопителей |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| SelectStorageForm.cs | btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит отмену выбора накопителя, закрывая форму |
| SelectStorageForm\_FormClosing | Обработчик события закрытия формы |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит подтверждение выбранного накопителя |
| SelectRAMForm.cs | SelectRAMForm | Конструктор формы выбора оперативной памяти |
| InitBindings | Инициализация привязок данных |
| MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния. |
| MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |
| SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |
| SubScribeChildMenuItems | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в строке состояния |
| созданиеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы создания новой оперативной памяти |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| SelectRAMForm.cs | редактированиеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы редактирования выбранной оперативной памяти |
| удалениеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который осуществляет удаление выбранной оперативной памяти |
| rAMInfoBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных ОП |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит отмену выбора оперативной памяти, закрывая форму |
| SelectRAMForm\_FormClosing | Обработчик события закрытия формы |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит подтверждение выбранной оперативной памяти |
| SelectPlatformForm.cs | SelectPlatformForm | Конструктор формы выбора платформы |
| InitCollection | Инициализация привязки коллекции |
| MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния. |
| MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |
| SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| SelectPlatformForm.cs | SubScribeChildMenuItems | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в строке состояния |
| platformInfoBindingSource\_DataSourceChanged | Обработчик события изменения текущего источника данных платформ |
| platformInfoBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных платформ |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену выбора платформы, закрывая форму |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит подтверждение выбранной платформы |
| созданиеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы создания новой платформы |
| редактированиеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы редактирования выбранной платформы |
| удалениеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который осуществляет удаление выбранной платформы |
| btn\_ShowSupportedInterfacesList\_Click | Обработчик события нажатия клавиши мыши на кнопку, который открывает диалоговое окно формы просмотра списка доступных интерфейсов хранения выбранной платформы |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| SelectCPUForm.cs | SelectCPUForm | Конструктор формы выбора процессора |
| InitBindings | Инициализация привязок данных |
| MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния. |
| MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |
| SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |
| SubScribeChildMenuItems | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в строке состояния |
| созданиеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы создания нового процессора |
| редактированиеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы редактирования выбранного процессора |
| удалениеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который осуществляет удаление выбранного процессора |
| cPUInfoBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных процессоров |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| SelectCPUForm.cs | btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит отмену выбора процессора, закрывая форму |
| SelectCPUForm\_FormClosing | Обработчик события закрытия формы |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит подтверждение выбранного процессора |
| PlatformInterfacesForm.cs | PlatformInterfacesForm | Конструктор формы просмотра списка доступных интерфейсов хранения выбранной платформы |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит закрытие формы |
| EditStorageForm.cs | EditStorageForm | Конструктор формы создания нового накопителя |
| EditStorageForm(Strorage selectedStorage) | Конструктор формы редактирования выбранного накопителя |
| BindAll | Инициализация привязок всех коллекций |
| BindManufacturer | Инициализация привязки коллекции с производителями |
| BindStorageInterfaces | Инициализация привязки коллекции с интерфейсами обмена данных |
| InitEditStorage | Инициализация привязок для редактирования выбранного накопителя |
| manufacturerBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных производителей |
| strorageInterfaceBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных типа интерфейса накопителя |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditStorageForm.cs | pictureBox1\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление выбранного производителя |
| pictureBox2\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление выбранного интерфейса |
| EditStorageForm\_FormClosing | Обработчик события закрытия формы |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену создания / редактирования накопителя, закрывая форму |
| btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит сохранение добавляемой / редактируемой записи об накопителе |
| pictureBox4\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно формы редактирования производителей |
| pictureBox17\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно поиска производителя |
| pictureBox3\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно поиска интерфейса накопителя |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditStorageForm.cs | pictureBox5\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно редактирования |
| comboBox1\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке производителей |
| comboBox2\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке интерфейсов накопителя |
| textBox1\_KeyPress | Обработчик события нажатия клавиши в текстовом поле с наименованием, который ограничивает ввод только русских букв, пробела и управляющих символов |
| textBox2\_KeyPress | Обработчик события нажатия клавиши в текстовом поле с объемом, который ограничивает ввод только цифр и управляющих символов |
| textBox3\_KeyPress | Обработчик события нажатия клавиши в текстовом поле с ценой, который ограничивает ввод только цифр, управляющих символов, и 1 запятой |
| EditRAMForm.cs | EditRAMForm | Конструктор формы создания новой оперативной памяти |
| EditRAMForm(RAM selectedRAM) | Конструктор формы редактирования выбранной оперативной памяти |
| BindAll | Инициализация привязок всех коллекций |
| BindManufacturer | Инициализация привязки коллекции с производителями |
| BindRAMTypes | Инициализация привязки коллекции с типами оперативной памяти |
| InitEditRAM | Инициализация привязок для редактирования выбранной оперативной памяти |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditRAMForm.cs | manufacturerBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных производителей |
| ramTypeBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных типов оперативной памяти |
| pictureBox1\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление выбранного производителя |
| pictureBox2\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление выбранного типа оперативной памяти |
| EditRAMForm\_FormClosing | Обработчик события закрытия формы |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену создания / редактирования оперативной памяти, закрывая форму |
| btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит сохранение добавляемой / редактируемой записи об оперативной памяти |
| pictureBox4\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно формы редактирования производителей |
| pictureBox17\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно поиска производителя |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditRAMForm.cs | pictureBox5\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно редактирования типов оперативной памяти |
| pictureBox3\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно поиска типа оперативной памяти |
| comboBox1\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке производителей |
| comboBox2\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке типов оперативной памяти |
| textBox1\_KeyPress | Обработчик события нажатия клавиши в текстовом поле с наименованием, который ограничивает ввод только русских букв, пробела и управляющих символов |
| numericUpDown1\_KeyPress | Обработчик события нажатия клавиши в numericupdown с объемом модуля, который ограничивает ввод только цифр и управляющих символов |
| textBox3\_KeyPress | Обработчик события нажатия клавиши в текстовом поле с ценой, который ограничивает ввод только цифр, управляющих символов, и 1 запятой |
| EditPlatformForm.cs | EditPlatformForm | Конструктор формы создания новой платформы |
| EditPlatformForm(PlatformInfo selectedPlatform) | Конструктор формы редактирования выбраннойплатформы |
| InitEditPlatform | Инициализация привязок для редактирования выбранной платформы |
| BindAll | Инициализация привязок всех коллекций |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditPlatformForm.cs | BindManufacturer | Инициализация привязки коллекции с производителями |
| BindCpuSockets | Инициализация привязки коллекции с разъемами процессоров |
| BindRamTypes | Инициализация привязки коллекции с типами оперативной памяти |
| BindStorageInterfaces | Инициализация привязки коллекции с интерфейсами накопителей |
| BindAvalibleInterfacesNew | Привязка коллекции доступных интерфейсов накопителей для новой платформы |
| BindAvalibleInterfacesForEdit | Привязка коллекции доступных интерфейсов накопителей для редактируемой платформы |
| pictureBox17\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно поиска производителя |
| pictureBox4\_Click | Обработчик события нажатия клавиши мыши на графический объект, открывает диалоговое окно поиска разъема процессора |
| pictureBox5\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно поиска типа оперативной памяти |
| pictureBox1\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление выбранного производителя |
| pictureBox2\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление выбранного разъема |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditPlatformForm.cs | pictureBox3\_Click | Обработчик события нажатия клавиши мыши на графический объект, производит удаление выбранного типа оперативной памяти |
| manufacturerBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных производителей |
| cpuSocketBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных разъемов процессоров |
| ramTypeBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных типов оперативной памяти |
| comboBox2\_Format | Обработчик события форматирования отображаемого значения в списке разъемов процессоров |
| pictureBox7\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит добавление выбранного интерфейса к доступным |
| pictureBox21\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит увеличение количества выбранного интерфейса |
| pictureBox20\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит уменьшение количества выбранного интерфейса |
| pictureBox19\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление выбранного интерфейса |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditPlatformForm.cs | btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит сохранение добавляемой / редактируемой записи об платформе |
| EditPlatformForm\_FormClosing | Обработчик события закрытия формы |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену создания / редактирования платформы, закрывая форму |
| comboBox1\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке производителей |
| comboBox2\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке разъемов процессора |
| comboBox3\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке типов оперативной памяти |
| textBox1\_KeyPress | Обработчик события нажатия клавиши в текстовом поле с наименованием, который ограничивает ввод только русских букв, пробела и управляющих символов |
| textBox2\_KeyPress | Обработчик события нажатия клавиши в текстовом поле с макс. количеством ОЗУ, который ограничивает ввод только цифр и управляющих символов |
| numericUpDown1\_KeyPress | Обработчик события нажатия клавиши в numericupdown с количеством разъемов, который ограничивает ввод только цифр и управляющих символов |
| textBox3\_KeyPress | Обработчик события нажатия клавиши в текстовом поле с ценой, который ограничивает ввод только цифр, управляющих символов, и 1 запятой |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditCPUForm.cs | EditCPUForm | Конструктор формы создания нового процессора |
| EditCPUForm(CPU selectedCPU) | Конструктор формы редактирования выбранного процессора |
| BindAll | Инициализация привязок всех коллекций |
| BindManufacturer | Инициализация привязки коллекции с производителями |
| BindCPUSockets | Инициализация привязки коллекции с разъемами процессоров |
| InitEditCPU | Инициализация привязок для редактирования выбранного процессора |
| manufacturerBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных производителей |
| cpuSocketBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных разъемов процессоров |
| pictureBox1\_Click | Обработчик события нажатия клавиши мыши на графический объект, производит удаление выбранного производителя |
| pictureBox2\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит удаление выбранного разъема процессора |
| EditCPUForm\_FormClosing | Обработчик события закрытия формы |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит сохранение добавляемой / редактируемой записи об процессоре |
| btn\_Save\_Click | Обработчик события нажатия клавиши мыши на объект, производит добавление выбранного интерфейса к доступным |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditCPUForm.cs | pictureBox4\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно редактирования производителей |
| pictureBox17\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно поиска производителя |
| pictureBox5\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно редактирования разъемов процессора |
| pictureBox3\_Click | Обработчик события нажатия клавиши мыши на графический объект, который открывает диалоговое окно поиска разъема процессора |
| comboBox2\_Format | Обработчик события форматирования отображаемого значения в списке разъемов процессоров |
| EditCPUForm\_KeyDown | Обработчик нажатий клавиш на форме |
| textBox1\_KeyPress | Обработчик события нажатия клавиши в текстовом поле с наименованием, ограничивает ввод только русских букв, пробела и управляющих символов |
| numericUpDown1\_KeyPress | Обработчик события нажатия клавиши в numericupdown с количеством ядер, который ограничивает ввод только цифр и управляющих символов |
| textBox2\_KeyPress | Обработчик события нажатия клавиши в текстовом поле с тактовой частотой и ценой, который ограничивает ввод только цифр, управляющих символов, и 1 запятой |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditCPUForm.cs | comboBox1\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке производителей |
| comboBox2\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке разъемов процессора |
| EditStorageInterfaceForm.cs | EditStorageInterfaceForm | Конструктор формы редактирования интерфейсов накопителей |
| EditStorageInterfaceForm(bool findMode) | Конструктор для выбора интерфейса накопителя из списка |
| BindCollection | Привязка коллекции интерфейсов |
| button2\_Click | Обработчик события нажатия клавиши мыши на кнопку, который осуществляет добавление нового интерфейса накопителя |
| button3\_Click | Обработчик события нажатия клавиши мыши на кнопку, который осуществляет редактирование выбранного интерфейса накопителя |
| EnDisFields | Активация/деактивация полей для редактирования |
| strorageInterfaceBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных интерфейсов накопителя |
| btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену последнего добавления/редактирования |
| but\_Select\_Click | Обработчик события нажатия клавиши мыши на кнопку, который совершает подтверждение выбранного интерфейса накопителя |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditStorageInterfaceForm.cs | EditStorageInterfaceForm\_FormClosing | Обработчик события закрытия формы |
| textBox2\_KeyPress | Ограничение ввода в наименовании добавляемого/редактируемого интерфейса накопителя |
| picBtn\_ClearCondition\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит очистку строки для поиска |
| picBtn\_FindByCondition\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит поиск интерфейса накопителяпо наименованию |
| EditRAMTypeForm.cs | EditRAMTypeForm | Конструктор формы редактирования типов оперативной памяти |
| EditRAMTypeForm(bool findMode) | Конструктор для выбора типа оперативной памяти из списка |
| BindCollection | Привязка коллекции типов оперативной памяти |
| EnDisFields | Активация/деактивация полей для редактирования |
| button2\_Click | Обработчик события нажатия клавиши мыши на кнопку, который осуществляет добавление нового типа оперативной памяти |
| button3\_Click | Обработчик события нажатия клавиши мыши на кнопку, который осуществляет редактирование выбранного типа оперативной памяти |
| ramTypeBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных типов оперативной памяти |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditRAMTypeForm.cs | btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену последнего добавления/редактирования |
| but\_Select\_Click | Обработчик события нажатия клавиши мыши на кнопку, который совершает подтверждение выбранного типа оперативной памяти |
| EditRAMTypeForm\_FormClosing | Обработчик события закрытия формы |
| textBox2\_KeyPress | Ограничение ввода в наименовании добавляемого/редактируемого типа оперативной памяти |
| picBtn\_ClearCondition\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит очистку строки для поиска |
| picBtn\_FindByCondition\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит поиск типа оперативной памяти по наименованию |
| EditManufacturerForm.cs | EditManufacturerForm | Конструктор формы редактирования производителей |
| EditManufacturerForm(bool findMode) | Конструктор для выбора производителя из списка |
| BindCollection | Привязка коллекции производителей |
| button2\_Click | Обработчик события нажатия клавиши мыши на кнопку, который осуществляет добавление нового производителя |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditManufacturerForm.cs | button3\_Click | Обработчик события нажатия клавиши мыши на кнопку, который осуществляет редактирование выбранного производителя |
| EnDisFields | Активация/деактивация полей для редактирования |
| manufacturerBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных производителей |
| btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену последнего добавления/редактирования |
| but\_Select\_Click | Обработчик события нажатия клавиши мыши на кнопку, который совершает подтверждение выбранного производителя |
| EditManufacturerForm\_FormClosing | Обработчик события закрытия формы |
| textBox2\_KeyPress | Ограничение ввода в наименовании добавляемого/редактируемого производителя |
| picBtn\_ClearCondition\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит очистку строки для поиска |
| picBtn\_FindByCondition\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит поиск производителя по наименованию |
| EditCPUSocketForm.cs | EditCPUSocketForm | Конструктор формы редактирования разъемов процессоров |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditCPUSocketForm.cs | EditCPUSocketForm(bool findMode) | Конструктор для выбора разъема процессора из списка |
| BindCollection | Привязка коллекции разъемов процессора |
| BindManufacturersToFind | Привязка коллекции производителей для поиска записей |
| cPUSocketInfoBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных разъемов процессоров |
| EnDisFields | Активация/деактивация полей для редактирования |
| button2\_Click | Обработчик события нажатия клавиши мыши на кнопку, который осуществляет добавление нового разъема |
| button3\_Click | Обработчик события нажатия клавиши мыши на кнопку, который осуществляет редактирование выбранного разъема процессора |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену последнего добавления/редактирования |
| btn\_Save\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |
| EditCPUSocketForm\_FormClosing | Обработчик события закрытия формы |
| but\_Select\_Click | Обработчик события нажатия клавиши мыши на кнопку, который совершает подтверждение выбранного разъема |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditCPUSocketForm.cs | picBtn\_ClearCondition\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит очистку строки для поиска |
| picBtn\_FindByCondition\_Click | Обработчик события нажатия клавиши мыши на графический объект, который производит поиск разъема по наименованию и/или производителю |
| manufaturerToFindBS\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных разъемов для поиска |
| textBox2\_KeyPress | Ограничение ввода в наименовании добавляемого/редактируемого разъема процессора |
| ServersConditionConstructorForm.cs | ServersConditionConstructorForm | Конструктор формы отбора конфигураций серверов |
| InitBindings | Инициализация привязок |
| comboBox1\_Format | Обработчик события форматирования отображаемого значения в combobox для платформы |
| comboBox2\_Format | Обработчик события форматирования отображаемого значения в combobox для процессора |
| platformBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных платформ |
| cPUBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных процессоров |
| RunSelection | Выполнение отбора конфигураций серверов данных по условию |
| pictureBox1\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет отмену выбора платформы |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| ServersConditionConstructorForm.cs | pictureBox2\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет отмену выбора процессора |
| comboBox1\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке платформ |
| comboBox2\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке процессоров |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену совершенного отбора записей |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет подтверждение совершенного отбора конфигураций серверов |
| SAN\_ConditionConstructorForm.cs | SAN\_ConditionConstructorForm | Конструктор формы отбора конфигураций хранилищ данных |
| InitBindings | Инициализация привязок |
| manufacturerBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных уровней производителей |
| RunSelection | Выполнение отбора конфигураций хранилищ данных по условию |
| pictureBox1\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет отмену выбора производителя |
| pictureBox2\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет отмену выбора модели |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену совершенного отбора записей |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| SAN\_ConditionConstructorForm.cs | comboBox1\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке производителей |
| comboBox2\_SelectedIndexChanged | Обработчик события изменения текущего индекса в списке моделей |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на графический объект, который осуществляет подтверждение совершенного отбора конфигураций хранилищ данных |
| RegisterNewIdle.cs | RegisterNewIdle(Server server) | Конструктор формы регистрации нового простоя для сервера |
| RegisterNewIdle(SAN san) | Конструктор формы регистрации нового простоя для хранилища данных |
| BindCollections | Привязка коллекций |
| dateTimePicker\_Begin\_ValueChanged | Обработчик события изменения даты начала обслуживания, который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности |
| SetEndDate | Корректировка допустимой даты |
| CalculateDuration | Расчет разницы между окончанием и началом обслуживания |
| numericUpDown\_BeginHours\_ValueChanged | Обработчик события изменения часа начала обслуживания, который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности |
| numericUpDown\_BeginMinutes\_ValueChanged | Обработчик события изменения минуты начала обслуживания, который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| RegisterNewIdle.cs | dateTimePicker\_End\_ValueChanged | Обработчик события изменения даты окончания обслуживания, который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности |
| numericUpDown\_EndHours\_ValueChanged | Обработчик события изменения часа окончания обслуживания, который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности |
| numericUpDown\_EndMinutes\_ValueChanged | Обработчик события изменения минуты окончания обслуживания, который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности |
| comboBox\_DurationType\_SelectedIndexChanged | Обработчик события изменения индекса формата отображения длительности обслуживания |
| AvalibleUserBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных доступного персонала |
| UsedUserBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных задействованного персонала |
| idleTypeBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных типа простоя |
| idleReasonBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных простоя |
| btn\_RemoveFromUsed\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит удаление выбранного сотрудника из задействованных сотрудников |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| RegisterNewIdle.cs | btn\_AddToUsed\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит добавление выбранного сотрудника из задействованных сотрудников |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену регистрации простоя, закрывая форму |
| PlanningSchedule.cs | PlanningSchedule(ScheduleEditType scheduleEditType) | Конструктор формы планирования расписания обслуживания |
| PlanningSchedule(Server selectedServer) | Конструктор формы планирования расписания обслуживания для сервера |
| PlanningSchedule(SAN selectedSan) | Конструктор формы планирования расписания обслуживания для хранилища данных |
| BindCollection | Привязка коллекции |
| UpdateStatBar | Обновление строки состояния |
| MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния |
| MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |
| SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| PlanningSchedule.cs | SubScribeChildMenuItems | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в строке состояния |
| scheduleItemBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных позиций расписания обслуживания |
| новаяПозицияToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, выполняет переход на форму добавления новой позиции расписания обслуживания |
| редактироватьВыбраннуюПозициюToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который выполняет переход на форму редактирования выбранной позиции расписания обслуживания |
| всеЗаписиToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который совершает показ всех позиций расписания обслуживания |
| запланированныеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, совершает отбор запланированных позиций расписания обслуживания |
| завершенныеToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который совершает отбор завершенных позиций расписания обслуживания |
| минутыToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который меняет формат на минуты длительности позиции обслуживания |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| PlanningSchedule.cs | часыToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который меняет формат на часы длительности позиции обслуживания |
| удалитьВыбраннуюПозициюToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который осуществляет удаление выбранной позиции расписания обслуживания |
| EquipmentIdleMenu.cs | EquipmentIdleMenu | Конструктор формы меню простоя оборудования |
| MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния |
| MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |
| SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |
| SubScribeChildMenuItems | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в строке состояния |
| BindCollections | Привязка коллекций |
| UpdateServerIdleItems | Обновление коллекции простоя серверов |
| serverIdleItemBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных информации о простое сервера |
| sANIdleItemBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных информации о простое хранилища данных |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EquipmentIdleMenu.cs | tsmi\_ServerPlanningMaintenance\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы планирования расписания обслуживания выбранного сервера |
| tsmi\_ServerRegisterIdle\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы регистрации нового простоя сервера |
| tsmi\_SanPlanningMaintenance\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы планирования расписания обслуживания выбранного хранилища данных |
| tsmi\_SanRegisterIdle\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы регистрации нового простоя хранилища данных |
| EquipmentIdleMenu\_FormClosing | Обработчик события закрытия формы |
| timer\_Tick | Обработчик события выполнения такта таймера |
| EditScheduleItem.cs | EditScheduleItem(ScheduleEditType scheduleEditType) | Конструктор формы редактирования позиции расписания обслуживания |
| EditScheduleItem(Server selectedServer) | Конструктор добавления новой позиции расписания обслуживания севера |
| EditScheduleItem(SAN selectedSan) | Конструктор добавления новой позиции расписания обслуживания хранилища данных |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditScheduleItem.cs | EditScheduleItem(Server\_MaintenanceShedule serverMaintenanceShedule) | Конструктор редактирования позиции расписания обслуживания севера |
| EditScheduleItem(SAN\_MaintenanceShedule sanMaintenanceShedule) | Конструктор редактирования позиции расписания обслуживания хранилища данных |
| SetMinimumValuesToDefault | Установка минимальных значений для компонентов по умолчанию |
| UnSubscribeEvents | Отписка обработчиков событий изменений значений компонентов |
| SubscribeEvents | Подписка обработчиков событий изменений значений компонентов |
| dateTimePicker\_Begin\_ValueChanged | Обработчик события изменения даты начала обслуживания, который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности |
| SetEndDate | Корректировка допустимой даты |
| CalculateDuration | Расчет разницы между окончанием и началом обслуживания |
| numericUpDown\_BeginHours\_ValueChanged | Обработчик события изменения часа начала обслуживания, который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности |
| numericUpDown\_BeginMinutes\_ValueChanged | Обработчик события изменения минуты начала обслуживания, который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EditScheduleItem.cs | dateTimePicker\_End\_ValueChanged | Обработчик события изменения даты окончания обслуживания, который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности |
| numericUpDown\_EndHours\_ValueChanged | Обработчик события изменения часа окончания обслуживания, который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности |
| numericUpDown\_EndMinutes\_ValueChanged | Обработчик события изменения минуты окончания обслуживания, который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности |
| comboBox\_DurationType\_SelectedIndexChanged | Обработчик события изменения индекса формата отображения длительности обслуживания |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену создания / редактирования позиции расписания обслуживания, закрывая форму |
| EditScheduleItem\_FormClosin | Обработчик события закрытия формы |
| SubscribeServicesForm.cs | SubscribeServicesForm | Конструктор формы подписки сотрудника на новые сервисы |
| BindCollections | Привязка коллекций |
| AvalibleServiceTypeBS\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных доступных сервисов |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| SubscribeServicesForm.cs | UsedServiceTypeBS\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных сервисов |
| btn\_AddToUsed\_Click | Обработчик события нажатия клавиши мыши на кнопку, который совершает добавление к выбранным сервисам |
| btn\_RemoveFromUsed\_Click | Обработчик события нажатия клавиши мыши на кнопку, который совершает удаление из выбранных сервисов |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену изменений, закрывая форму |
| SubscribeServicesForm\_FormClosing | Обработчик события закрытия формы |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |
| EmployeeForm.cs | EmployeeForm | Конструктор главной формы сотрудника |
| EmployeeForm\_FormClosing | Обработчик события закрытия формы |
| MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния |
| MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |
| SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |
| SubScribeChildMenuItems | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в строке состояния |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| EmployeeForm.cs | просмотрПрофиляToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы просмотра и редактирования текущей учетной записи |
| сменитьПользователяToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который совершает выход из учетной записи |
| завершениеРаботыToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который завершает работу приложения |
| BindUsedServices | Привязка коллекции используемых сервисов сотрудника |
| подписатьсяНаНовыйСервисToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы подписки на новые сервисы |
| отписатьсяОтВыбранногоСервисаToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который совершает отписку от выбранного сервиса |
| userServiceBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных подписанных серверов |
| timer\_Tick | Обработчик события выполнения такта таймера |
| StaffDistributonForm.cs | StaffDistributonForm | Конструктор формы распределения персонала, задействованного в простое сервиса |
| BindCollections | Привязка коллекций |
| AvalibleUserBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных доступного персонала |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| StaffDistributonForm.cs | UsedUserBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных задействованного персонала |
| btn\_RemoveFromUsed\_Click | Обработчик события нажатия клавиши мыши на кнопку, который осуществляет удаление из выбранных сервисов |
| btn\_AddToUsed\_Click | Обработчик события нажатия клавиши мыши на кнопку, который осуществляет добавление к выбранным сервисам |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену подписки на новые сервисы, закрывая форму |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на кнопку, который выполняет сохранение изменений |
| StaffDistributonForm\_FormClosing | Обработчик события закрытия формы |
| ServiceRequestTreatment.cs | ServiceRequestTreatment | Конструктор формы обработки заявки на предоставление сервиса |
| UpdateTextFields | Обновление текстовых полей формы |
| comboBox\_PriceType\_SelectedIndexChanged | Обработчик события изменения индекса типа вывода стоимости |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит отмену заявки на создание сервиса, закрывая форму |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на кнопку, который производит подтверждение заявки на создание сервиса |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| ServiceRequestTreatment.cs | btn\_PrintReport\_Click | Обработчик события нажатия клавиши мыши на кнопку, который открывает диалоговое окно предварительного просмотра отчета заявки |
| EditIdleItemForm.cs | EditIdleItemForm | Конструктор формы расчета стоимости простоя сервиса |
| InitBindings | Инициализация привязок |
| CalcIdleCost | Расчет стоимости простоя |
| UpdateTextFields | Обновление текстовых полей |
| textBox9\_TextChanged | Изменение значения полей |
| costIdleParamsBindingSource\_CurrentItemChanged | Обработчик события изменения текущего элемента в списке параметров простоя |
| btn\_Cancel\_Click | Обработчик события нажатия клавиши мыши на кнопку, который осуществляет отмену изменений в параметрах простоя, закрывая форму |
| btn\_OK\_Click | Обработчик события нажатия клавиши мыши на кнопку, который осуществляет сохранение изменений в параметрах простоя |
| comboBox\_DurationType\_SelectedIndexChanged | Обработчик события изменения индекса формата отображения времени |
| распределениеПерсоналаToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы распределения задействованного персонала |
| btn\_PrintReport\_Click | Обработчик события нажатия клавиши мыши на кнопку, который открывает диалоговое окно предварительного просмотра отчета по текущему простою |
| DirectorForm.cs | DirectorForm | Конструктор главной формы директора |
| DirectorForm\_FormClosing | Обработчик события закрытия формы |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| DirectorForm.cs | BindAll | Привязка всех коллекций |
| BindServiceInfo | Привязка коллекции с обзором сервисов |
| BindServiceRequest | Привязка коллекции с заявками на создание сервиса |
| BindServiceCost | Получение коллекции со стоимостью сервисов |
| BindPlatformList | Инициализация привязки списка платформ с информацией о количестве пользователей и сервисов |
| BindUsersInformation | Привязка списка с информацией пользователей |
| BindServiceIdle | Привязка списка с информацией о простое сервисов |
| MenuItem\_MouseLeave | Обработчик события выхода курсора из элемента главного меню, который очищает область в строке состояния |
| MenuItem\_MouseEnter | Обработчик события наведения курсора на элемент главного меню, который выводит подсказку элемента в строку состояния |
| SubscribeMenuItems | Подписка элементов главного меню и панели элементов на события наведения и выхода курсора для отображения подсказок в строке состояния |
| SubScribeChildMenuItems | Подписка дочерних элементов главного меню на события наведения и выхода курсора для отображения подсказок в строке состояния |
| просмотрПрофиляToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы просмотра и редактирования текущей учетной записи |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| DirectorForm.cs | сменитьПользователяToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который совершает выход из текущей учетной записи |
| завершениеРаботыToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который завершает работу приложения |
| обработкаВыбраннойЗаявкиToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы обработки выбранной заявки |
| serviceRequestItemBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных заявок на создание сервиса |
| стоимостьПредоставленияСервисовToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно предварительного просмотра отчета по стоимости предоставления сервисов |
| idleItemBindingSource\_CurrentChanged | Обработчик события изменения текущего элемента в источнике данных заявок простоев сервисов |
| редактированиеВыбранногоПростояToolStripMenuItem\_Click | Обработчик события нажатия клавиши мыши на элемент главного меню, который открывает диалоговое окно формы редактирования выбранного простоя |
| timer\_Tick | Обработчик события выполнения такта таймера |
| CostIdleParams.cs | Save | Сохранение экземпляра параметров для расчёта простоя сервисов в файл |
| Load | Выгрузка экземпляра параметров для расчёта простоя сервисов из файла |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| Context.cs | TestConnection | Проверка наличия ссылки на контекст данных |
| SaveChangesAsync | Сохранение изменений в текущем контексте (асинхронный) |
| CancelChanges | Отмена изменений существующей сущности |
| SaveChanges | Сохранение изменений в текущем контексте |
| Login | Авторизация пользователя |
| Logout | Выход из учетной записи |
| GetPaasInfo | Получение списка с информацией о платформах |
| GetServiceInfo | Получение списка с информацией о сервисах |
| GetServersEquipmentState | Получение объекта, который описывает краткое состояние серверов |
| GetSanEquipmentState | Получение объекта, который описывает краткое состояние систем хранения данных |
| GetServerInfo | Получение списка краткого описания конфигураций серверов |
| GetSanInfo | Получение списка краткого описания конфигураций систем хранения данных |
| UsedPlatformsCount | Получение количества используемых платформ |
| UsedServiceCount | Получение количества используемых сервисов |
| ServiceConfigsCount | Получение количества используемых конфигураций сервисов |
| CheckActivePaas | Проверка наличия платформ, на которые распределены ресурсы |
| GetPaasTypeByName | Получение экземпляра типа сервиса по его наименованию |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| Context.cs | DeleteServiceType | Удаление выбранного типа сервиса |
| DeletePaasType | Удаление выбранного типа платформы |
| CheckInfrastructureToCreatePlatform | Проверка наличие экземпляров конфигураций серверов и хранилищ данных |
| DeleteServer | Удаление выбранной конфигурации сервера |
| GetPlatformsList | Получение списка платформ |
| GetCpusOfSocket | Получение списка процессоров по их разъему |
| GetRamsOfRamType | Получение списка памяти по ее типу |
| GetStoragesByAvaliblePlatformInterfaces | Получение списка доступных накопителей по указанным интерфейсам |
| GetAvalibleInterfacesOfPlarformBS | Получение преобразованного списка доступных интерфейсов накопителей для платформы |
| GetPlatformStorageIntFromAvalible | Получение обратно преобразованного списка доступных интерфейсов накопителей для платформы |
| GetInterfaceIdByName | Получение кода интерфейса по его наименованию |
| GetAvalibleInterfacesOfPlarform | Получение списка доступных интерфейсов выбранной платформы |
| AddNewServerConfiguration | Сохранение новой конфигурации сервера |
| EditServerConfiguration | Сохранение изменений в редактируемой сервера |
| GetServerByServerInfo | Получение ссылки на экземпляр сервера по его подробной информации |
| GetPlatformInclude | Получение экземпляра текущей платформы с коллекцией поддерживаемых интерфейсов |
| GetPlatformsInfo | Получение списка платформ с подробным описанием |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| Context.cs | GetPlatformByPlatformInfo | Получение экземпляра платформы из подробного описания платформы |
| DeletePlatform | Удаление выбранной платформы |
| GetPlatformInfoByPlatform | Получение подробного описания платформы по ее экземпляру |
| GetManufacturers | Получение списка производителей с привязкой |
| CheckManufacturerForDublicate | Проверка дублирования наименования производителя |
| GetCPUSocketsInfo | Получение списка разъемов процессора с наименованием производителя |
| GetCPUSocketByInfo | Получение разъема процессора по его информации |
| CheckCpuSocketForDublicate | Проверка дублирования разъема процессора |
| AddNewCPUSocket | Добавление нового разъема процессора |
| EditCPUSocket | Редактирование существующего разъема процессора |
| GetCPUSockets | Получение списка разъемов процессоров |
| GetRamsTypes | Получение списка типов ОЗУ |
| GetStorageInterfaces | Получение списка интерфейсов хранения |
| CheckPlatformForDublicate | Проверка дублирования платформы |
| AddNewPlatform | Добавление новой платформы |
| EditPlatform | Редактирование существующей платформы |
| GetRightsLevels | Получение списка уровней привилегий пользователя |
| GetPositions | Получение списка должностей |
| CheckUserLoginForDublicate | Проверка дублирования логина пользователя |
| AddNewUser | Добавление нового пользователя |
| EditUser | Редактирование пользователя |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| Context.cs | CheckPositionForDublicate | Проверка дублирования наименования должности |
| GetUsersInfo | Получение списка пользователей с подробным описанием |
| GetUserByUserInfo | Получение ссылки на пользователя по его подробному описанию |
| GetStoragesInfo | Получение списка накопителей с подробным описанием |
| GetStorageByStorageInfo | Получение ссылки на накопитель по его подробной информации |
| DeleteStrorage | Удаление выбранного накопителя |
| GetStorageByModelManufacturer | Получение экземпляра накопителя по ее модели и производителю |
| DeleteUser | Удаление выбранного пользователя |
| CheckStorageForDublicate | Проверка дублирования накопителя |
| AddNewStorage | Добавление нового накопителя |
| EditStorage | Редактирование существующего накопителя |
| GetStorageInterfacesBS | Получение списка интерфейсов накопителей с привязкой |
| CheckStorageInterfaceForDublicate | Проверка дублирования наименования интерфейса накопителя |
| GetCPUsInfo | Получение списка процессоров с подробным описанием |
| GetCPUByCPUInfo | Получение ссылки на процессор по его подробной информации |
| DeleteCPU | Удаление выбранного процессора |
| CheckCPUForDublicate | Проверка дублирования процессора |
| AddNewCPU | Добавление нового процессора |
| EditCPU | Редактирование существующего процессора |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| Context.cs | GetRAMsInfo | Получение списка оперативной памяти с подробным описанием |
| GetRAMByRAMInfo | Получение ссылки на оперативную память по его подробной информации |
| DeleteRAM | Удаление выбранной оперативной памяти |
| GetRAMByModelManufacturer | Получение экземпляра оперативной памяти по ее модели и производителю |
| CheckRAMForDublicate | Проверка дублирования оперативной памяти |
| AddNewRAM | Добавление новой оперативной памяти |
| EditRAM | Редактирование существующей оперативной памяти |
| GetRAMTypesBS | Получение списка типов оперативной памяти с привязкой |
| CheckRAMTypeForDublicate | Проверка дублирования наименования типа оперативной памяти |
| GetStoragesByAvalibleSANInterfaces | Получение списка доступных накопителей по указанным интерфейсам хранилища данных |
| GetStoragesByAvalibleInterfaces | Получение списка доступных накопителей по указанным интерфейсам |
| GetAvalibleInterfacesOfSANBS | Получение преобразованного списка доступных интерфейсов накопителей для хранилища данных с привязкой |
| GetAvalibleInterfacesOfSAN | Получение преобразованного списка доступных интерфейсов накопителей для хранилища данных |
| GetSANStorageIntFromAvalible | Получение обратно преобразованного списка доступных интерфейсов накопителей для хранилища данных |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| Context.cs | GetStorageInterfaceByName | Получение ссылки на экземпляр интерфейса накопителя по его наименованию |
| AddNewSAN | Сохранение нового хранилища данных |
| EditSAN | Сохранение изменений в редактируемом хранилище данных |
| DeleteSAN | Удаление выбранной конфигурации хранилища данных |
| GetSANBySANInfo | Получение ссылки на экземпляр файлового хранилища по его подробной информации |
| GetServicesByServer | Получение списка сервисов, которые используют выбранный сервер |
| GetServersByService | Получение списка используемых серверов выбранного сервиса |
| GetSANByService | Получение списка используемых хранилищ данных выбранного сервиса |
| GetServicesBySAN | Получение списка сервисов, которые используют выбранное хранилище данных |
| GetServerConfigs | Получение списка конфигураций серверов |
| GetServerConfigsByPaas | Получение списка используемых конфигураций серверов по выбранному типу платформы |
| GetSANsList | Получение списка хранилищ данных |
| GetSANsListByPaas | Получение списка используемых хранилищ данных по выбранному типу платформы |
| GetServerPaasInfoByServer | Получение информации о доступных/используемых ресурсов выбранного сервера |
| GetSANPaasInfoBySAN | Получение информации о доступных/используемых ресурсов выбранного хранилища данных |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| Context.cs | CheckPaasForDublicate | Проверка дублирования наименования платформы |
| GetServerByID | Получение ссылки на экземпляр сервера по его порядковому номеру |
| GetSANByID | Получение ссылки на экземпляр хранилища данных по его порядковому номеру |
| AddNewPaas | Сохранение новой платформы |
| EditPaas | Сохранение изменений в редактируемой платформе |
| GetPaasFromPaasInfo | Получение ссылки на экземпляр платформы по ее подробному описанию |
| GetPaasTypes | Получение списка типов платформ |
| GetServiceTypes | Получение списка типов сервиса |
| GetServersByPlatform | Получение списка серверов выбранной платформы |
| GetSANsByPlatform | Получение списка хранилищ данных выбранной платформы |
| GetServerDetailsByID | Получение детальной информации о сервере по его порядковому номеру |
| GetSANDetailsByID | Получение детальной информации о хранилище данных по его порядковому номеру |
| GetServerPlatformInfoByServer | Получение информации о доступных ресурсах сервера |
| GetSanPlatformInfoBySAN | Получение информации о доступных ресурсах хранилища данных |
| AddNewService | Создание нового сервиса |
| EditService | Редактирование выбранного сервиса |
| GetServiceStateByName | Получение экземпляра состояния сервиса по его наименованию |
| GetIdleTypeByName | Получение экземпляра типа простоя сервиса по его наименованию |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| Context.cs | GetIdleReasonByName | Получение экземпляра причины простоя сервиса по его наименованию |
| GetSoftwareById | Получение экземпляра программного обеспечения по его ID |
| GetSoftwaresInfo | Получение списка программного обеспечения с подробным описанием |
| GetSoftwareTypes | Получение списка типов программного обеспечения |
| GetSoftwareByInfo | Получение программного обеспечения по его информации |
| DeleteSoftware | Удаление выбранного программного обеспечения |
| CheckSoftwareForDublicate | Проверка дублирования программного обеспечения |
| AddNewSoftware | Добавление нового программного обеспечения |
| EditSoftware | Редактирование существующего программного обеспечения |
| GetSoftTypesBS | Получение списка типов программного обеспечения с привязкой |
| CheckSoftTypeForDublicate | Проверка дублирования наименования типа программного обеспечения |
| GetServiceTypesBS | Получение списка типов сервиса с привязкой |
| CheckServiceTypeForDublicate | Проверка дублирования наименования типа сервиса |
| GetPlatformServiceUsers | Получение списка платформ с информацией о количестве пользователей и сервисов |
| GetServicesDetailInfo | Получение полного списка конфигураций сервисов с детальным описанием |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| Context.cs | GetServicesCost | Получение полного списка конфигураций сервисов с детальным описанием |
| GetPlatformByPlatformServiceUser | Получение экземпляра платформы по ее подробной информации |
| GetServiceByServiceDetailInfo | Получение экземпляра сервиса по ее подробной информации |
| DeleteService | Удаление выбранного сервиса |
| GetServerIdleItems | Получение списка с информацией о простое серверов |
| GetSanIdleItems | Получение списка с информацией о простое хранилищ данных |
| GetServiceIdleItems | Получение списка с информацией о простое сервисов |
| AddNewServerSchedulePosition | Добавление новой позиции в расписании обслуживания сервера |
| AddNewSANSchedulePosition | Добавление новой позиции в расписании обслуживания хранилища данных |
| EditServerSchedulePosition | Изменение позиции в расписании обслуживания сервера |
| EditSANSchedulePosition | Изменений позиции в расписании обслуживания хранилища данных |
| CheckMaintenanceDateRange | Проверка уже запланированных позиций обслуживания в установленном диапазоне дат |
| GetScheduleItemsOfServer | Получение списка позиций обслуживания для выбранного сервера |
| GetScheduleItemsOfSAN | Получение списка позиций обслуживания для выбранного хранилища данных |
| GetServerMaintenanceSheduleByID | Получение ссылки на экземпляр позиции расписания обслуживания сервера |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| Context.cs | GetSANMaintenanceSheduleByID | Получение ссылки на экземпляр позиции расписания обслуживания хранилища данных |
| DeleteSelectedServiceScheduleMaintenance | Удаление выбранной позиции расписания обслуживания сервера |
| DeleteSelectedSANScheduleMaintenance | Удаление выбранной позиции расписания обслуживания хранилища данных |
| GetAdminsList | Получения списка пользователей, которые относятся к администраторам |
| GetIdleTypes | Получение списка типов простоя |
| GetIdleReasons | Получение списка причин простоя |
| RegisterNewIdle | Регистрация нового простоя сервисов |
| GetServiceReuqests | Получение списка заявок на создание сервисов |
| GetServiceByID | Получение экземпляра сервиса по его порядковому номеру |
| GetIdleItems | Получение списка с информацией о простое сервиса |
| CalculateServicesIdleBackground | Фоновый расчет стоимости простоя сервисов |
| GetServiceIdleByID | Получение экземпляра простоя сервиса по его номеру |
| CancelServiceRequest | Отмена заявки на создание сервиса |
| ConfirmServiceRequest | Принятие заявки на создание сервиса |
| GetUsedServices | Получение списка используемых сервисов текущим пользователем |
| UnSubscribeSelectedService | Отписка от выбранного сервиса |
| GetServiceTypeById | Получение ссылки на экземпляр типа сервиса по его порядковому номеру |
| GetAvalibleServiceTypes | Получение списка доступных типов сервисов |
| SubscribeToSelectedServices | Подписка на выбранные сервисы |

Продолжение таблицы 55

|  |  |  |
| --- | --- | --- |
| **Имя модуля** | **Имя процедуры (функции)** | **Назначение процедуры (функции)** |
| Context.cs | GetIdleParamsFormDatabase | Получение параметров для расчета стоимости простоя |
| UpdateServiceIdle | Обновление стоимости простоя сервиса |
| CheckServerMaintenance | Проверка обслуживания конфигураций серверов |
| CheckSANMaintenance | Проверка обслуживания хранилищ данных |
| GetCPUs | Получение полного списка процессоров |

# **4 Руководство пользователя**

## **4.1 Подготовка к работе**

Все пользователи программы, а именно: системный администратор, директор и прочий персонал, должны иметь базовые навыки работы на персональном компьютере.

В состав ПЭВМ должно входить:

* системный блок;
* монитор;
* клавиатура;
* мышь.

Также должен быть предусмотрен доступ к интернету, скорость которого не менее 64 кб/с.

Минимальные системные требования представлены в таблице 56.

Таблица56– Минимальные системные требования

|  |  |
| --- | --- |
| **Компонент** | **Минимальные требования** |
| Операционная система | Windows 10, Windows 8.1, Windows 8, Windows 7 спакетомобновления 1, Windows Server 2016, Windows Server 2012 R2, Windows Server 2012 или Windows Server 2008 R2 |
| Процессор | x86 или x64 с тактовой частотой от 1 ГГц |
| Оперативная память | 256 МБ |
| Свободное пространство на жестком диске | 80 МБ |
| Видеокарта | Видеоадаптер с поддержкой DirectX 9 |
| Дисплей | Разрешение экрана 1024 x 768 |
| Версия .NET | .NET 4.5.2 |

Также желательно наличие принтера для печати выходных документов, однако пользователь ПП может производить их экспорт в следующие форматы:

* документ Microsoft Office Word;
* книгу Microsoft Office Excel;
* PDF.

Первые два вида документа требуют установленного офисного пакета MicrosoftOfficeверсии не ниже 2007. Для экспорта отчета в PDFдолжно быть установлено приложение, которое позволит выполнять различные действия, после его сохранения. Например, «Adobe Acrobat Reader DC».

Состав дистрибутивного носителя программ и данных:

* «setup.exe» - установщик ПП;
* «MTF\_Services.Installer.msi» - пакетустановщикаWindows;
* «Readme.txt» - файл, содержащий информацию об учетных данных зарегистрированных пользователей приложения;
* «autorun.inf» - файл, используемый для автоматического запускаустановщикаПП;
* «icon.ico» - значок для загрузчика установщика.

Инсталлятор ПП должен запускаться автоматически при подключении диска, на котором он содержится. Если ничего не произошло, то следуетперейти в корневой каталог инсталляционного диска и запустить исполняемый файл «setup.exe». По ходу установки пользователю будет предложено выбрать путь для хранения файлов программы, по умолчанию «C:\Program Files (x86)\ПАО Мурманский траловый флот\MTF\_Services\»; а также для каких пользователей операционной системы следует провести установку. По окончании пользователь увидит сообщение об успешной установке, а также появится ярлык «ПАО МТФ - Облачные вычисления» на рабочем столе и создастся каталог «ПАО Мурманский траловый флот» в меню «Пуск».

# **4.2 Описание действий пользователя во время работы с программным продуктом**

4.2.1 Запуск программного продукта и авторизация

Перед запуском программы необходимо убедиться в том, что есть доступ к сети интернет. ПП можно запустить двумя способами:

* дважды щелкнув ЛКМ по ярлыку, который по умолчанию создается на рабочем столе после установки приложения;
* через меню «Пуск».

При запуске приложения происходит проверка подключения к базе данных, в случае ошибки программа сообщает пользователю об ошибке и завершает свою работу, иначе пользователю будет предоставлено окно авторизации, в соответствии с рисунком Д.1. В пунктах 1-2 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя. В случае прохождения авторизации пользователь попадает на форму главного меню, в зависимости от предоставленного ему уровня привилегий на использование ПП.Ссылки на примеры форм главного экрана по уровням привилегий представлено в таблице 57.

Таблица 57 – Ссылки на примеры форм главного экрана по уровням привилегий

|  |  |
| --- | --- |
| **Наименование уровня привилегий** | **Ссылка на пример** |
| Системный администратор | Рисунок Д.2 |
| Директор | Рисунок Д.3 |
| Персонал предприятия | Рисунок Д.4 |

4.2.2 Главный экран системного администратора

После авторизации, исходя из предоставленных уровня привилегий текущему пользователю, он попадает на форму, представленную на рисунке Д.2. На ней отображается состояние вычислительной инфраструктуры, а также предоставляемых платформ и сервисов в режиме реального времени. Доступный функционал перечислен в главном меню формы и включает в себя:

* добавление, редактирование и удаление платформ;
* добавление, редактирование и удаление сервисов;
* регистрация нового пользователя и редактирование существующих;
* редактирование вычислительной инфраструктуры;
* редактирования сведений о простоях оборудования.

Для реализации двух основных задач приложения, описание которых представлено в разделе 2.2, первым делом следует зарегистрировать учетные записи директора и сотрудников предприятия, которые будут использовать ПП. Для этого необходимо перейти в пункт главного меню «Пользователи», при этом существует 2 способа добавления нового сотрудника:

1. через подпункт «Регистрация», либо нажатием сочетания клавиш «Ctrl+U» открывается форма, представленная на рисунке Д.32;
2. через подпункт «Редактирование», либо нажатием сочетания клавиш «Ctrl+Shift+U», при этом открывается форма, представленная на рисунке Д.31.

4.2.3 Регистрация и редактирование пользователей ПП

Существует два способа регистрации пользователя, переход на формы, которых описан в предыдущем пункте.

В первом способе системный администратор осуществляет переход к непосредственному созданию новой учетной записи. Ему предстоит ввести данные входного документа «Личное дело сотрудника», перечень входной информации которого описан в таблице 3 пункта 1.4. В пунктах 3-6 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя. Каждой учетной записи необходимо назначить уровень привилегий, выбрав соответствующий из выпадающего списка. Также следует указать занимаемую должность, выбрав соответствующую из выпадающего списка, либо нажать на инструментальную кнопку поиска, которая в свою очередь откроет модальное диалоговое окно формы, представленной на рисунке Д.33. В появившимся окне пользователь имеет возможность выбрать соответствующую позицию и подтвердить выбор с помощью кнопки «Выбрать». На активной форме также имеется возможность редактирования информации о должностях, данные которого представлены во входном документе «Справка о предоставлении средней заработной платы», перечень входной информации которого описан в таблице 3 пункта 1.4. Для этого следует нажать на кнопку «Редактировать», выбрав соответствующую позицию. По окончании редактирования необходимо нажать на кнопку «Сохранить» для фиксации изменений. В пунктах 7-10 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя. В случае отсутствия должностей в выпадающем списке, следует нажать на инструментальную кнопку редактирования должностей, которая в свою очередь откроет модальное диалоговое окно формы, представленной на рисунке Д.33, и выполнить добавление информации о должностях, предварительно нажав на кнопку «Добавить». По окончании добавления необходимо нажать на кнопку «Сохранить» для фиксации изменений. Для отмены выбранной должности, пользователь имеет возможность нажать на инструментальную кнопку удаления текущей должности. Чтобы отменить регистрацию следует нажать на кнопку «Отмена», либо закрыть форму, подтвердив осуществление отмены.

Во втором способе системный администратор осуществляет переход на форму редактирования пользователей, которая представлена на рисунке Д.31. В пунктах 11-12 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя.

Доступный функционал перечислен в главном меню формы, который включает в себя:

* просмотр всех зарегистрированных пользователей в системе;
* отбор зарегистрированных пользователей по условию;
* добавление, редактирование и удаление учетных записей.

Чтобы отобразить всехзарегистрированных пользователейследует перейти в пункт главного меню «Вид» и выбрать подпункт «Все записи», либо нажать на кнопку просмотра всех записей на инструментальной панели. Также существует возможность просмотра всех записей через нажатие сочетания клавиш «Ctrl+A».

Для отбора зарегистрированных пользователей по условию следует перейти в пункт меню «Вид» и выбрать подпункт «Отбор», либо нажать на кнопку отбора записей на инструментальной панели или через нажатие сочетания клавиш «Ctrl+S». При этом откроется модальное диалоговое окно, представленное на рисунке Д.34. Для совершения отбора необходимо выбрать уровень привилегий и/или должность из выпадающих списков и нажать кнопку «ОК». В случае наличия отобранных записей, они будут отображены на форме редактирования пользователей. В пункте 13 перечня диагностических сообщений, который описан в таблице 58, представлена обработка результата некорректного действия пользователя.

Добавление нового пользователя можно выполнить несколькими способами:

* перейти в пункт меню «Правка» и выбрать подпункт «Новый пользователь»;
* нажать на кнопку добавления пользователя на инструментальной панели;
* нажатие сочетания клавиш «Ctrl+N»;
* выбрать пункт «Новый пользователь» в контекстном меню таблицы.

Редактирование выбранного пользователя можно выполнить несколькими способами:

* перейти в пункт меню «Правка» и выбрать подпункт «Редактировать»;
* нажать на кнопку редактирования выбранного пользователя на инструментальной панели;
* нажатие сочетания клавиш «Ctrl+E»;
* выбрать пункт «Редактировать» в контекстном меню таблицы.

При добавлении и редактировании открывается модальное диалоговое окно, представленное на рисунке Д.32. Описание действий, совершаемых в нем было описано в первом способе.

Редактирование выбранного пользователя можно выполнить несколькими способами:

* перейти в пункт меню «Правка» и выбрать подпункт «Удалить»;
* нажать на кнопку удаления выбранного пользователя на инструментальной панели;
* нажатие клавиши «Del»;
* выбрать пункт «Удалить» в контекстном меню таблицы.

При удалении появится диалоговое окно с подтверждением совершаемого действия.

4.2.4 Редактирование вычислительной инфраструктуры

Для редактирования вычислительной инфраструктуры предприятия существует форма, представленная на рисунке Д.5. В ней отображается в табличном виде существующие конфигурации серверов и хранилищ данных. Также на верхней панели располагается краткая информация их состояния, обновляемая в режиме реального времени. Доступный функционал перечислен в главном меню формы и включает в себя:

* создание, редактирование, удаление конфигурации сервера;
* поиск конфигурации сервера по условию;
* отбор конфигураций серверов по условию;
* просмотр всех конфигураций серверов;
* создание, редактирование, удаление конфигурации хранилища данных;
* поиск конфигурации хранилища данных по условию;
* отбор конфигураций хранилищ данных по условию;
* просмотр всех конфигураций хранилищ данных.

Большинство из перечисленных функций продублированы в инструментальной панели и контекстных меню на таблицах, а также имеют зарегистрированные комбинации клавиш для быстрого их выполнения.

В пунктах 14-23 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя.

4.2.4.1 Создание и редактирование конфигураций серверов

Для создания или редактирования выбранной конфигурации сервера следует перейти в пункт меню «Сервер» и выбрать соответствующие подпункты. При этом будет совершен переход на форму редактирования конфигурации сервера, представленной на рисунке Д.6.Пользователю предстоит ввести данные входного документа «Технический паспорт на конфигурацию сервера», перечень входной информации которого описан в таблице 3 пункта 1.4. В пунктах 24-43 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя.

Сначала требуется добавить платформу для определения доступных характеристик других комплектующих конфигурации. Это можно сделать, нажав на одну из меток, призывающих выбрать платформу, либо через пункт меню «Комплектующие» выбрать подпункт «Платформа». После этого появится модальное диалоговое окно выбора платформы, которое представлено на рисунке Д.10, предоставляющее следующий набор функций:

* создание, редактирование, удаление платформы;
* просмотр списка поддерживаемых интерфейсов;
* выбор требуемой платформы.

В пунктах 44-45 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя.

Если уже существуют созданные платформы, то следует нажать кнопку «ОК» для подтверждения выбранной платформы. Иначе нужно добавить новую платформу, это можно сделать несколькими способами:

* перейти в пункт меню «Правка» и выбрать подпункт «Создание»;
* нажать на кнопку создания новой платформы на инструментальной панели;
* выбрать пункт «Создание» в контекстном меню таблицы.

Таким образом открывается модальное диалоговое окно редактирования платформы, представленное на рисунке Д.14. Пользователю предстоит ввести более детальные данные входного документа «Технический паспорт на конфигурацию сервера», перечень входной информации которого описан в таблице 3 пункта 1.4. В пунктах 46-49 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя.

Пользователю ПП необходимо указать список поддерживаемых интерфейсов для платформы. Для этого нужно с помощью функциональной кнопки добавить выбранный тип интерфейса к существующим. После этого имеется возможность увеличения, уменьшения или удаления доступного интерфейса на выбранной платформе при помощи функциональных кнопок.

Также следует указать производителя, выбрав соответствующего из выпадающего списка, либо нажать на инструментальную кнопку поиска, которая в свою очередь откроет модальное диалоговое окно формы, представленной на рисунке Д.18. В появившимся окне пользователь имеет возможность выбрать соответствующее наименование и подтвердить выбор с помощью кнопки «Выбрать». На активной форме также имеется возможность их редактирования, для этого следует нажать на кнопку «Редактировать», выбрав соответствующегопроизводителя. По окончании редактирования необходимо нажать на кнопку «Сохранить» для фиксации изменений. В пунктах 50-53 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя. В случае отсутствия производителей в выпадающем списке, следует нажать на инструментальную кнопку их редактирования, которая в свою очередь откроет модальное диалоговое окно формы, представленной на рисунке Д.18, и выполнить добавление информации о производителях, предварительно нажав на кнопку «Добавить». По окончании добавления необходимо нажать на кнопку «Сохранить» для фиксации изменений.

Помимо этого, для платформы требуется указать разъем процессора, действия по добавлению и редактированию которого аналогичны что и для производителя, описанных в предыдущем абзаце. Отличие заключается в перечне диагностических сообщений, которых пользователь может получить в случае некорректных действий. Они описаны в пунктах 54-57 перечня диагностических сообщений, который описан в таблице 58.

В свою очередь, для платформы требуется указать тип поддерживаемой памяти, действия по добавлению и редактированию которого аналогичны что и для производителя, описанных двумя абзацами ранее. Отличие заключается в перечне диагностических сообщений, которых пользователь может получить в случае некорректных действий. Они описаны в пунктах 58-61 перечня диагностических сообщений, который описан в таблице 58.

После выбора платформы у пользователя появляется возможность добавления остальных комплектующих к конфигурации сервера, таких как:

* процессора;
* оперативной памяти;
* накопителях.

Процессор к конфигурации добавляется автоматически, если имеется список совместимых по разъему, который определен в выбранной платформе. Пользователь может выбирать его через выпадающий список, либо перейти к списку процессоров, через функциональную кнопку поиска, а также через пункт главного меню «Комплектующие» выбрать подпункт «Процессор». При этом появится модальное диалоговое окно выбора процессора, которое представлено на рисунке Д.11, в котором пользователь может осуществить создание нового, редактирование и удаление выбранного процессора, а также выбрать необходимый из списка по нажатию кнопки «ОК».

В пунктах 62-64 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя.

Если уже существуют созданные процессоры, то следует нажать кнопку «ОК» для подтверждения выбранного процессора. Иначе нужно добавить новый процессор, это можно сделать несколькими способами:

* перейти в пункт меню «Правка» и выбрать подпункт «Создание»;
* выбрать пункт «Создание» в контекстном меню таблицы.

Таким образом открывается модальное диалоговое окно редактирования процессора, представленное на рисунке Д.15. Пользователю предстоит ввести более детальные данные входного документа «Технический паспорт на конфигурацию сервера», перечень входной информации которого описан в таблице 3 пункта 1.4. В пунктах 65-67 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя. Среди вводимых характеристик пользователю требуется указать производителя процессора и его разъем, описаниередактирования которых было представлено ранее.

После выбора платформы у пользователя имеется список совместимой оперативной памяти по еетипу, который определен в выбранной платформе. Пользователь может выбирать его через выпадающий список, либо перейти к списку оперативной памяти, через функциональную кнопку поиска, а также через пункт главного меню «Комплектующие» выбрать подпункт «Оперативная память». При этом появится модальное диалоговое окно выбора оперативной памяти, которое представлено на рисунке Д.12, в котором пользователь может осуществить создание новой, редактирование и удаление выбраннойоперативной памяти, а также выбрать необходимую из списка по нажатию кнопки «ОК».

В пунктах 68-70 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя.

Если уже существует созданная оперативная память, то следует нажать кнопку «ОК» для подтверждения выбранной оперативной памяти. Иначе нужно добавить новуюоперативную память, это можно сделать несколькими способами:

* перейти в пункт меню «Правка» и выбрать подпункт «Создание»;
* выбрать пункт «Создание» в контекстном меню таблицы.

Таким образом открывается модальное диалоговое окно редактирования оперативной памяти, представленное на рисунке Д.16. Пользователю предстоит ввести более детальные данные входного документа «Технический паспорт на конфигурацию сервера», перечень входной информации которого описан в таблице 3 пункта 1.4. В пунктах 71-72 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя. Среди вводимых характеристик пользователю требуется указать производителяи тип оперативной памяти, описание редактирования которых было представлено ранее.

После выбора платформы у пользователя имеется список совместимыхнакопителейпо тем интерфейсам, которые определены в выбранной платформе. Пользователь может выбирать их через выпадающий список, либо перейти к списку накопителей, через функциональную кнопку поиска, а также через пункт главного меню «Комплектующие» выбрать подпункт «Накопители». При этом появится модальное диалоговое окно выбора накопителя, которое представлено на рисунке Д.13, в котором пользователь может осуществить создание нового, редактирование и удаление выбранногонакопителя, а также выбрать необходимый из списка по нажатию кнопки «ОК».

В пункте 73 перечня диагностических сообщений, который описан в таблице 58, представлена обработка результатов некорректных действий пользователя.

Если уже существует созданный накопитель, то следует нажать кнопку «ОК» для подтверждения выбранного накопителя. Иначе нужно добавить новый накопитель, это можно сделать несколькими способами:

* перейти в пункт меню «Правка» и выбрать подпункт «Создание»;
* выбрать пункт «Создание» в контекстном меню таблицы.

Таким образом открывается модальное диалоговое окно редактирования накопителя, представленное на рисунке Д.17. Пользователю предстоит ввести более детальные данные входного документа «Технический паспорт на конфигурацию сервера», перечень входной информации которого описан в таблице 3 пункта 1.4. В пунктах 74-75 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя. Среди вводимых характеристик пользователю требуется указать производителя, описание редактирования которого было представлено ранее.

Также следует указать интерфейс накопителя, выбрав соответствующего из выпадающего списка, либо нажать на инструментальную кнопку поиска, которая в свою очередь откроет модальное диалоговое окно формы, представленной на рисунке Д.21. В появившимся окне пользователь имеет возможность выбрать соответствующийинтерфейс и подтвердить выбор с помощью кнопки «Выбрать». На активной форме также имеется возможность их редактирования, для этого следует нажать на кнопку «Редактировать», выбрав соответствующий интерфейс. По окончании редактирования необходимо нажать на кнопку «Сохранить» для фиксации изменений. В пунктах 76-79 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя. В случае отсутствия интерфейсов в выпадающем списке, следует нажать на инструментальную кнопку их редактирования, которая в свою очередь откроет модальное диалоговое окно формы, представленной на рисунке Д.21, и выполнить добавление информации об интерфейсах, предварительно нажав на кнопку «Добавить». По окончании добавления необходимо нажать на кнопку «Сохранить» для фиксации изменений.

После проведения выше описанных действий пользователю следует ввести стоимость обслуживания конфигурации. Будет отображена итоговая стоимость, а также каждого компонента создаваемой или редактируемой конфигурации. Для сохранения изменений следует нажать кнопку «Сохранить». После чего будет совершен возврат на форму редактирования инфраструктуры, и будет обновлен список конфигураций серверов, а также краткое описание их состояния.

4.2.4.2 Создание и редактирование конфигураций хранилищ данных

Для создания или редактирования выбранной конфигурации хранилища данных следует перейти в пункт меню «Хранилище данных» и выбрать соответствующие подпункты. При этом будет совершен переход на форму редактирования конфигурации хранилища данных, представленной на рисунке Д.7.Пользователю предстоит ввести данные входного документа «Технический паспорт на хранилища данных», перечень входной информации которого описан в таблице 3 пункта 1.4. В пунктах 80-88 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя.

Перед тем как добавить накопители необходимо выбрать интерфейсы из списка доступных при помощи функциональной кнопки добавления выбранного интерфейса к доступным. По ходу добавления метка с просьбой добавления интерфейсов пропадет. Если есть совместимые накопители, то отобразится их список. В противном случае появится метка об их отсутствии, по нажатию на которую откроется модальное диалоговое выбора и редактирования накопителей, описание функционала которого описано в предыдущем пункте. Походу добавления накопителей в редактируемое хранилище данных будет рассчитываться его итоговая стоимость. Перед сохранением изменением изменений требуется ввести его стоимость обслуживания, затем можно нажать на кнопку «Сохранить» для фиксации изменений. После чего будет совершен возврат на форму редактирования инфраструктуры, и будет обновлен список конфигураций хранилищ данных, а также краткое описание их состояния.

4.2.5 Распределение вычислительных ресурсов между платформами

Перед тем как создавать сервисы необходимо определить платформы, для разграничения ресурсов. Для этого необходимо на главном меню системного администратора, которое представлено на рисунке Д.2, перейти в пункт главного меню «Платформы», при этом существует 2 способа добавления новой платформы:

1. через подпункт «Добавить новую», либо нажатием сочетания клавиш «Ctrl+N» открывается форма, представленная на рисунке Д.27;
2. через подпункт «Редактор платформ и сервисов», либо нажатием сочетания клавиш «Ctrl+Shift+E», при этом открывается форма, представленная на рисунке Д.26.

В первом способе, перед тем как перейти на форму добавления, система проверит наличия конфигураций серверов и хранилищ данных. В случае их отсутствия пользователь не сможет осуществить добавление.

В пунктах 89−94 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя.

Для создания новой платформы необходимо ввести ее наименование, а также добавить конфигурации серверов через функциональную кнопку «Добавить новый». При этом откроется модальное диалоговое окно выбора конфигурации сервера, представленного на рисунке Д.28. В пунктах 95−96 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя. По нажатию кнопки «ОК» произойдет добавление выбранной конфигурации сервера к платформе. А также необходимо добавить конфигурацию хранилища данных через функциональную кнопку «Добавить новый». При этом откроется модальное диалоговое окно выбора конфигурации хранилища данных, представленного на рисунке Д.29. В пунктах 97-98 перечня диагностических сообщений, который описан в таблице 58, представлены обработки результатов некорректных действий пользователя. По нажатию кнопки «ОК» произойдет добавление выбранной конфигурации хранилища данных к платформе. После совершенных действий, описанных ранее, можно нажать на кнопку «Сохранить» для фиксации изменений. После чего будет совершен возврат на главное меню администратора, и будет обновлен список платформ.

Во втором способепроисходит переход на форму, представленную на рисунке Д.26. С помощью функциональной кнопки «Добавление нового», либо через пункт меню «Правка» выбрать подпункт «Создание новой платформы» пользователю откроется форма добавления новой платформы, описание действий с которой представлено в первом способе.

4.2.6 Создание сервиса

После распределения вычислительных ресурсов между платформами, системный администратор может преступить к созданию сервисов. Для этого необходимо на главном меню системного администратора, которое представлено на рисунке Д.2, перейти в пункт главного меню «Сервисы», при этом существует 2 способа добавления новой платформы:

1. через подпункт «Добавить новый», либо нажатием сочетания клавиш «Alt+N» открывается форма, представленная на рисунке Д.30;
2. через подпункт «Редактор платформ и сервисов», либо нажатием сочетания клавиш «Ctrl+Shift+E», при этом открывается форма, представленная на рисунке Д.26.

В первом способе, перед тем, как перейти на форму добавления, система проверит наличия платформ, на которые распределены вычислительные ресурсы. В случае их отсутствия пользователь не сможет осуществить добавление.

Для создания нового сервиса, первым делом следует выбрать тип платформы, для определения доступных вычислительных ресурсов. Как только платформа будет выбрана, система отобразит общее и доступные значения ресурсов, а также их стоимость использования в час. При этом у пользователя установятся ограничения на выбранное количество требуемых ресурсов. При изменении каждого параметра сервиса, происходит вычисление стоимости использования. Также требуется добавить используемое программное обеспечение, нажав на функциональную кнопку его добавления. При этом откроется модальное диалоговое окно выбора программного обеспечения, представленного на рисунке Д.41, в котором пользователь может осуществить создание нового, редактирование и удаление выбранного программного обеспечения, а также выбрать необходимое из списка по нажатию кнопки «ОК».

Если уже существует созданное программное обеспечение, то следует нажать кнопку «ОК» для подтверждения его выбора. Иначе нужно добавить новое программное обеспечение, это можно сделать несколькими способами:

* перейти в пункт меню «Правка» и выбрать подпункт «Создание»;
* выбрать пункт «Создание» в контекстном меню таблицы.

Таким образом, открывается модальное диалоговое окно редактирования программного обеспечения, представленное на рисунке Д.42. Среди вводимых характеристик пользователю требуется наименование, его тип и стоимость лицензии. По окончании добавления необходимо нажать на кнопку «Сохранить» для фиксации изменений.

После выбора программного обеспечения, остается выбрать тип сервиса из выпадающего списка, либо нажать на инструментальную кнопку поиска, которая в свою очередь откроет модальное диалоговое окно формы, представленной на рисунке Д.43. В появившимся окне пользователь имеет возможность выбрать соответствующий тип сервиса и подтвердить выбор с помощью кнопки «Выбрать». На активной форме также имеется возможность их редактирования, для этого следует нажать на кнопку «Редактировать», выбрав соответствующий тип сервиса. По окончании редактирования необходимо нажать на кнопку «Сохранить» для фиксации изменений. В случае отсутствия типов сервиса в выпадающем списке, следует нажать на инструментальную кнопку их редактирования, которая в свою очередь откроет модальное диалоговое окно формы, представленной на рисунке Д.43, и выполнить добавление информации об типе сервиса, предварительно нажав на кнопку «Добавить». По окончании добавления необходимо нажать на кнопку «Сохранить» для фиксации изменений.

После проведения выше описанных действий пользователю следует нажать кнопку «Сохранить». После чего заявка на предоставление созданного сервиса будет отправлена директору, а также совершен возврат на форму редактирования главного меню системного администратора, где будет обновлен список предоставляемых сервисов и платформ.

4.2.7 Обработка заявки на предоставление сервиса

Для обработки заявки на предоставление нового сервиса, необходимо зайти под пользователем, который имеет права директора. После авторизации, он попадает на форму, представленную на рисунке Д.3. На ней отображается информация в табличном виде, такая как:

* общий обзор предоставляемых сервисов;
* заявки на предоставления созданных сервисов;
* стоимость предоставления сервисов;
* сведения о простое предоставляемых сервисов;
* обзор предоставляемых платформ;
* список зарегистрированных пользователей в системе.

Отображаемая информация обновляется в режиме реального времени.

Обработка заявки осуществляется следующим способом: перейти на вкладку «Обзор» и выбрать в ней вкладку «Заявки». Далее через пункт главного меню «Правка» выбрать подпункт «Обработка выбранной заявки», либо через функциональную кнопку на панели элементов. При этом откроется модальное диалоговое окно обработки заявки на предоставление сервиса, представленное на рисунке Д.35. Пользователю будет представлена детальная информация о сервисе, а также стоимость его предоставления, формат которой можно изменять через выпадающий список. Для принятия заявки следует нажать кнопку «Принять заявку», при этом появится диалоговое сообщение для подтверждения выбранного действия. В случае принятия, диалоговое окно будет закрыто, а обработанный сервис станет доступным для сотрудников предприятия, которые зарегистрировались в системе. Перед принятием решения у директора имеется возможность создания отчета по текущей заявки, для ее осуществления необходимо нажать на кнопку «Печать отчета», при этом откроется модальное диалоговое окно предварительного просмотра, представленное на рисунке Д.36.

Также на главном меню директора имеется возможность создания отчета по всем предоставляемым сервисам, для ее осуществления необходимо через пункт «Отчеты» главного меню выбрать подпункт «Стоимость предоставления сервисов», при этом откроется модальное диалоговое окно предварительного просмотра, представленное на рисунке Д.36.

4.2.8 Подписка на предоставляемый сервис

Для подписки на предоставляемый сервис, необходимо зайти под пользователем, который имеет права сотрудника. После авторизации, он попадает на форму, представленную на рисунке Д.4. На ней отображается информация о сервисах, на которые он подписан. Далее через пункт главного меню «Правка» выбрать подпункт «Подписаться на новый сервис», либо через функциональную кнопку на панели элементов. Если отсутствуют предоставляемые сервисы или нет тех сервисов, на которых пользователь не подписан, появится соответствующее диалоговое сообщение. Иначе откроется модальное диалоговое окно подписки на доступные сервисы, представленное на рисунке Д.39. В нем будет предоставлен список доступных сервисов для подписки слева, а также список сервисов, подписка на которых будет оформлена, справа. Пользователь должен нажать на кнопку «Подписаться на доступный сервис», а если он передумал, то он может отказаться от подписки, нажав на кнопку «Отписаться от выбранного сервиса». Для сохранения изменений следует нажать кнопку «ОК», после чего на главном меню сотрудника отобразятся сервисы, на которые он подписан.

4.2.9 Планирование расписания обслуживания оборудования

Для того, чтобы перейти на форму планирования расписания обслуживания оборудования, пользователь, находясь на главном меню системного администратора должен через пункт главного меню «Оборудование» выбрать подпункт «Простой». При этом будет совершен переход в меню простоя оборудования, представленного на рисунке Д.22. В нем отображается информация в табличном виде, такая как:

* состояние конфигураций серверов;
* состояние конфигураций хранилищ данных;
* состояние предоставляемых сервисов.

Отображаемая информация обновляется в режиме реального времени.

Для перехода на форму планирования расписания обслуживания для выбранной конфигурации сервера или хранилища данных, нужно в главном меню через пункт «Сервер» или «Хранилища данных» выбрать подпункт «Планирование расписания обслуживания», либо через контекстное меню требуемого типа оборудования. При этом откроется модальное диалоговое окно планирования расписания обслуживания, представленного на рисунке Д.23. В нем требуется через пункт главного меню «Правка» выбрать подпункт «Новая позиция», либо по нажатии кнопки на панели инструментов. В таком случае откроется модальное диалоговое окно планирования обслуживания, представленное на рисунке Д.24, в котором следует указать дату начала и окончания планируемого обслуживания. Для сохранения изменений нужно нажать на кнопку «ОК». После этого система автоматически переведет в режим облуживания требуемое оборудования, а также отключит сервисы, которые к ним привязаны в период, указанный при планировании.

4.2.10 Регистрация простоя оборудования

Для регистрации простоя оборудования системный администратор должен перейти в меню простоя оборудования, порядок перехода на которое описано в предыдущем пункте. Затем в главном меню через пункт «Сервер» или «Хранилища данных» выбрать подпункт «Регистрация нового простоя», либо через контекстное меню требуемого типа оборудования. При этом откроется модальное диалоговое окно регистрации нового простоя, представленного на рисунке Д.25. В котором следует указать:

* дату начала и окончания совершенного простоя;
* тип простоя;
* причину простоя;
* перечень пользователей, которые были задействованы в период простоя для решения нештатной ситуации.

Для сохранения изменений нужно нажать на кнопку «ОК». После этого система автоматически рассчитает стоимость простоя, которую увидит директор, если заполнена служебная информация, необходимая для расчета (ее заполняет директор 1 раз, а далее ее редактирует).

4.2.11 Расчет стоимости простоя сервиса

Для расчета простоя сервиса должна быть заполнена информация о совершенном простое, описание создания которой описано в пунктах 4.2.9 и 4.2.10. Расчет стоимости доступен директору. Следовательно, пользователь должен авторизоваться под соответствующей учетной записью. В меню директора нужно перейти на вкладку «Сервисы» и выбрать в ней вкладку «Простой». Не рассчитанное позиции помечены «Не рассчитан» в столбце «Стоимость». Далее через пункт меню «Правка» выбрать подпункт «Редактирование выбранного простоя». При этом откроется модальное диалоговое окно расчета стоимости простоя сервиса, представленного на рисунке Д.37. Для выполнения расчета следует заполнить следующие поля:

* годовой валовой доход компании;
* средняя часовая оплата труда администратора;
* средняя часовая оплата труда работника.

Также директор может править задействованный персонал во время простоя, для этого следует выбрать пункт «Распределение персонала». При этом откроется модальное диалоговое окно распределения персонала, представленное на рисунке Д.38. В котором директор определяет задействованный персонал. Для сохранения изменений нужно нажать на кнопку «ОК».

При этом такие поля как: средняя часовая оплата труда администратора и средняя часовая оплата труда администратора могут быть рассчитаны автоматически, если существуют требуемые данные в БД. В случае их правки будут использоваться данные, которые ввел директор.

Для сохранения полученного расчёта нажать на кнопку «ОК». Однако перед этим у директора имеется возможность создания отчета по текущему простою, для ее осуществления необходимо нажать на кнопку «Печать отчета», при этом откроется модальное диалоговое окно предварительного просмотра, представленное на рисунке Д.36.

Таблица 58 – Перечень диагностических сообщений

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Наименование** | **Ситуация действий пользователя** |
| 1 | Заполните все поля для прохождения авторизации! | Попытка авторизации без введенных данных |

Продолжение таблицы 58

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Наименование** | **Ситуация действий пользователя** |
| 2 | Неверный логин и/или пароль! | Попытка авторизации с неверными данными |
| 3 | Введите все поля для сохранения изменений! | Попытка создания новой учетной записи или редактирования существующей, не заполнив все поля |
| 4 | Пароли не совпадают! | Попытка создания новой учетной записи или редактирования существующей, введя неверное значения поля для подтверждения пароля |
| 5 | Пользователь с таким логином уже существует | Попытка создания новой учетной записи или редактирования существующей, с уже существующим логином в системе |
| 6 | Изменения не будут сохранены! Продолжить? | Закрытие формы при создании или редактировании учетной записи |
| 7 | Заполните все поля для сохранения изменений! | Попытка сохранения новой или отредактированной позиции должности сотрудников, не заполнив все поля |
| 8 | Должность с таким наименованием уже существует! | Попытка сохранения новой или отредактированной позиции должности сотрудников, с уже существующим наименованием в системе |
| 9 | Выберите должность из списка! | Подтверждение выбора должности, не выбрав при этом необходимую позицию |
| 10 | Изменения не будут сохранены! Продолжить? | Закрытие формы при создании или редактировании должности сотрудников, не совершив фиксацию изменений |
| 11 | Выберите пользователя из списка для его редактирования | Попытка редактирования не выбранного пользователя |
| 12 | Выберите пользователя из списка для его редактирования! | Попытка редактирования не выбранного пользователя |

Продолжение таблицы 58

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Наименование** | **Ситуация действий пользователя** |
| 13 | Не найдены пользователи по указанному условию | Попытка подтверждения вывода отобранных учетных записей пользователей, при отсутствии результата |
| 14 | Выберите конфигурацию сервера для редактирования! | Попытка перехода на форму редактирования конфигурации сервера, когда она не выбрана |
| 15 | Выберите конфигурацию сервера для ее удаления! | Удаление не выбранной конфигурации сервера |
| 16 | Выберите хранилище данных для редактирования! | Попытка перехода на форму редактирования конфигурации хранилища данных |
| 17 | Выберите хранилище данных для его удаления! | Удаление не выбранной конфигурации хранилища данных |
| 18 | Отсутствуют конфигурации серверов для совершения поиска! | Попытка открытия диалогового окна для задания условия поиска конфигурации сервера, при их отсутствии |
| 19 | Отсутствуют конфигурации серверов для совершения отбора! | Попытка открытия диалогового окна для задания условия отбора конфигураций серверов, при их отсутствии |
| 20 | Отсутствуют конфигурации серверов! | Попытка вывода всех конфигураций серверов, при их отсутствии |
| 21 | Отсутствуют конфигурации хранилищ данных для совершения поиска! | Попытка открытия диалогового окна для задания условия поиска конфигурации |
| 22 | Отсутствуют конфигурации хранилищ данных для совершения отбора! | Попытка открытия диалогового окна для задания условия отбора конфигураций хранилищ данных, при их отсутствии |
| 23 | Отсутствуют конфигурации хранилищ данных! | Попытка вывода всех конфигураций хранилищ данных, при их отсутствии |
| 24 | Платформа не выбрана! | Удаление не выбранной платформы |
| 25 | Процессор не выбран! | Удаление не выбранного процессора |
| 26 | Оперативная память не выбрана! | Удаление не выбранной оперативной памяти |
| 27 | Выбранная оперативная память не найдена в списке установленных! | Удаление выбранной оперативной памяти из списка, которая отсутствует в конфигурации |

Продолжение таблицы 58

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Наименование** | **Ситуация действий пользователя** |
| 28 | Накопитель не выбран! | Удаление не выбранного накопителя |
| 29 | Выбранный накопитель не найден в списке установленных! | Удаление выбранного накопителя из списка, который отсутствует в конфигурации сервера |
| 30 | Выберите оперативную память из списка! | Добавление не выбранной оперативной памяти в конфигурацию сервера |
| 31 | Все разъемы под оперативную память заняты! | Добавление выбранной оперативной памяти в конфигурацию сервера, когда все разъемы под нее уже заняты |
| 32 | Занят максимальный объем оперативной памяти! | Добавление выбранной оперативной памяти в конфигурацию сервера, когда достигнут максимальный объем, определенный платформой |
| 33 | Выберите накопитель из списка! | Добавление не выбранного накопителя в конфигурацию сервера |
| 34 | Все разъемы под интерфейс %название интерфейса% заняты! | Добавление выбранного накопителя в конфигурацию сервера или увеличение количества установленного, когда все разъемы под него уже заняты |
| 35 | Выберите установленную оперативную память из списка! | Попытка увеличения, уменьшения количества или удаления выбранной установленной оперативной памяти |
| 36 | Все разъемы под оперативную память заняты! | Попытка увеличения количества выбранной установленной оперативной памяти, когда все разъемы под нее уже заняты |
| 37 | Занят максимальный объем оперативной памяти! | Попытка увеличения количества выбранной установленной оперативной памяти, когда достигнут максимальный объем, определенный платформой |
| 38 | Выберите установленный накопитель из списка! | Попытка увеличения, уменьшения количества или удаления выбранного накопителя, когда он не выбран |

Продолжение таблицы 58

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Наименование** | **Ситуация действий пользователя** |
| 39 | Выберите платформу для сохранения конфигурации сервера! | Попытка сохранения конфигурации сервера, когда платформа не выбрана |
| 40 | Выберите процессор для сохранения конфигурации сервера! | Попытка сохранения конфигурации сервера, когда процессор не выбран |
| 41 | Установите оперативную память для сохранения конфигурации сервера! | Попытка сохранения конфигурации сервера, когда оперативная память не установлена |
| 42 | Установите накопители для сохранения конфигурации сервера! | Попытка сохранения конфигурации сервера, когда накопители не установлены |
| 43 | Введите стоимость обслуживания сервера! | Попытка сохранения конфигурации сервера, когда не введена стоимость планового обслуживания |
| 44 | Выберите платформу из списка! | Попытка подтверждения выбранной платформы или просмотра списка доступных интерфейсов, когда платформа не выбрана |
| 45 | Выберите платформу из списка или добавьте новую! | Попытка перехода на форму редактирования выбранной платформы или ее удаления, кода платформа не выбрана |
| 46 | Выберите интерфейс из списка! | Попытка добавление выбранного интерфейса к доступным, когда тот не выбран |
| 47 | Выберите интерфейс из списка или добавьте новый! | Попытка увеличения, уменьшения количества разъемов выбранного интерфейса, а также его удаления, когда тот не выбран |
| 48 | Заполните все поля для сохранения изменений! | Попытка сохранения новой или отредактированной платформы, когда были введены не все данные |
| 49 | Платформа с таким производителем и моделью уже существует! | Попытка сохранения новой или отредактированной платформы, с уже существующей комбинацией модели и производителя |

Продолжение таблицы 58

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Наименование** | **Ситуация действий пользователя** |
| 50 | Выберите производителя из списка или добавьте нового! | Подтверждение выбора производителя или его редактирования, при этом, не выбрав его |
| 51 | Введите наименование производителя! | Попытка сохранения нового или отредактированного производителя, не заполнив все поля |
| 52 | Производитель с таким наименованием уже существует! | Попытка сохранения нового или отредактированного производителя, с уже существующим наименованием в системе |
| 53 | Введите условие для поиска производителя! | Попытка поиска производителя по его наименованию, не введя условие |
| 54 | Выберите элемент из списка или добавьте новый! | Подтверждение выбора разъема процессора или его редактирования, при этом, не выбрав его |
| 55 | Заполните все поля перед сохранением! | Попытка сохранения нового или отредактированного разъема процессора, не заполнив все поля |
| 56 | Разъем с таким производителем и наименованием уже существует! | Попытка сохранения нового или отредактированного разъема процессора, с уже существующим наименованием |
| 57 | Введите условие для поиска разъема процессора! | Попытка поиска типа оперативной памяти по его наименованию, не введя условие |
| 58 | Выберите тип оперативной памяти из списка или добавьте новый! | Подтверждение выбора типа оперативной памяти или его редактирования, при этом, не выбрав его |
| 59 | Введите наименование типа оперативной памяти! | Попытка сохранения нового или отредактированного типа оперативной памяти, не заполнив все поля |

Продолжение таблицы 58

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Наименование** | **Ситуация действий пользователя** |
| 60 | Тип оперативной памяти с таким наименованием уже существует! | Попытка сохранения нового или отредактированного типа оперативной памяти, с уже существующим наименованием в системе |
| 61 | Введите условие для поиска типа оперативной памяти! | Попытка поиска типа оперативной памяти по его наименованию, не введя условие |
| 62 | Выберите процессор из списка для его редактирования! | Попытка перехода на форму редактирования выбранногопроцессора, когда он не выбран |
| 63 | Выберите процессор из списка для его удаления! | Попытка удаления выбранного процессора, когда он не выбран |
| 64 | Выбранный процессор не подходит к разъему платформы! | Попытка подтверждения выбранного процессора, при несовместимом разъеме с платформой |
| 65 | Заполните все поля для сохранения изменений! | Попытка сохранения нового или отредактированного процессора, когда были введены не все данные |
| 66 | Производитель процессора не совпадает с производителем разъема! | Попытка сохранения нового или отредактированного процессора, когда производитель процессора не совпадает с производителем разъема |
| 67 | Процессор с таким производителем и моделью уже существует! | Попытка сохранения нового или отредактированного процессора, с уже существующей комбинацией модели и производителя |
| 68 | Выберите оперативную память из списка для ее редактирования! | Попытка перехода на форму редактирования выбранной оперативной памяти, когда она не выбрана |
| 69 | Выберите оперативную память из списка для ее удаления! | Попытка удаления выбранной оперативной памяти, когда она не выбрана |
| 70 | Выберите оперативную память из списка! | Попытка подтверждения выбранной оперативной памяти, когда она не выбрана |

Продолжение таблицы 58

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Наименование** | **Ситуация действий пользователя** |
| 71 | Заполните все поля для сохранения изменений! | Попытка сохранения новой или отредактированной оперативной памяти, когда были введены не все данные |
| 72 | Оперативная память с таким производителем и моделью уже существует! | Попытка сохранения новой или отредактированной оперативной памяти, с уже существующей комбинацией модели и производителя |
| 73 | Выберите накопитель из списка для его редактирования! | Попытка перехода на форму редактирования выбранного накопителя, когда он не выбран |
| 74 | Заполните все поля для сохранения изменений! | Попытка сохранения нового или отредактированного накопителя, когда были введены не все данные |
| 75 | Накопитель с таким производителем и моделью уже существует! | Попытка сохранения нового или отредактированного накопителя, с уже существующей комбинацией модели и производителя |
| 76 | Выберите интерфейс накопителя из списка или добавьте новый! | Подтверждение выбора интерфейса накопителя, при этом, не выбрав его |
| 77 | Введите наименование интерфейса накопителя! | Попытка сохранения нового или отредактированного интерфейса накопителя, когда были введены не все данные |
| 78 | Интерфейс накопителя с таким наименованием уже существует! | Попытка сохранения нового или отредактированного интерфейса накопителя, с уже существующим наименованием в системе |
| 79 | Введите условие для поиска интерфейса накопителя! | Попытка поиска интерфейса накопителя по его наименованию, не введя условие |

Продолжение таблицы 58

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Наименование** | **Ситуация действий пользователя** |
| 80 | Выберите интерфейс из списка! | Попытка добавления выбранного интерфейса к доступным, при этом, не выбрав его |
| 81 | Накопитель не выбран! | Попытка удаления выбранного накопителя из установленных, при этом, не выбрав его |
| 82 | Выбранный накопитель не найден в списке установленных! | Удаление выбранного накопителя из списка, который отсутствует в конфигурации хранилища данных |
| 83 | Выберите установленный накопитель из списка! | Попытка увеличения, уменьшения количества или удаления выбранного установленного накопителя, когда он не выбран |
| 84 | Выберите интерфейс из списка или добавьте новый! | Попытка увеличения, уменьшения количества разъемов выбранного интерфейса, а также его удаления, когда тот не выбран |
| 85 | Выберите производителя для сохранения хранилища данных! | Попытка сохранения нового или отредактированного хранилища данных, не выбрав производителя |
| 86 | Выберите интерфейсы накопителей для сохранения хранилища данных! | Попытка сохранения нового или отредактированного хранилища данных, не выбрав производителя |
| 87 | Установите накопители для сохранения хранилища данных! | Попытка сохранения нового или отредактированного хранилища данных без установленных накопителей |
| 88 | Введите стоимость обслуживания хранилища данных! | Попытка сохранения нового или отредактированного хранилища данных, не введя стоимость обслуживания |
| 89 | Конфигурация сервера не выбрана! | Попытка удаления выбранной конфигурации сервера, когда она не выбрана |
| 90 | Хранилище данных сервера не выбрано! | Попытка удаления выбранного хранилища данных, когда он не выбран |
| 91 | Введите наименование платформы! | Попытка сохранения платформы, при не введенном ее наименовании |

Продолжение таблицы 58

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Наименование** | **Ситуация действий пользователя** |
| 92 | Выберите конфигурации серверов для сохранения платформы! | Попытка сохранения платформы, при не выбранных конфигурациях серверов |
| 93 | Выберите хранилища данных для сохранения платформы! | Попытка сохранения платформы, при не выбранных конфигурациях хранилищ данных |
| 94 | Платформа с таким наименованием уже существует! | Попытка сохранения платформы, с уже существующим наименованием в системе |
| 95 | Выбранная конфигурация сервера уже присутствует в списке выбранных! | Попытка добавления уже распределенной конфигурации сервера к текущей платформе |
| 96 | Выберите сервер из списка! | Попытка подтверждения выбранной конфигурации сервера, когда она не выбрана |
| 97 | Выбранное файловое хранилище уже присутствует в списке выбранных! | Попытка добавления уже распределенной конфигурации хранилища данных к текущей платформе |
| 98 | Выберите хранилище данных из списка! | Попытка подтверждения выбранной конфигурации хранилища данных, когда она не выбрана |

## **4.3 Действия пользователя в аварийных ситуациях**

При эксплуатации программного продукта возможны аварийные ситуации, описание и действия для решения, которых представлены в таблице 59.

Таблица 59 – Описание аварий ных ситуаций и действия, необходимые для их решения

|  |  |
| --- | --- |
| **Аварийная ситуация** | **Действия пользователя** |
| Программный продукт перестал отвечать на действия пользователя | 1. Открыть диспетчер задач; 2. найти и завершить процесс приложения «MTF\_Services.exe»; 3. перезапустить приложениt. |
| Полное или частичное отсутствие, искажение ранее сохраненной информации | 1. перезапустить приложение; 2. если ситуация не разрешена, связаться с администратором. |
| Получение сообщения об необработанном исключении | Нажать кнопку «Повторить», либо «Пропустить» для игнорирования ошибки. Либо «Закрыть» для перезапуска приложения |

# **5 Выводы и предложения**

Технологии машинной обработки информации постепенно вытесняют ручные методы. Каждая современная организация задумывается о внедрении автоматизированных систем или уже успешно использует их преимущества.

Создав программный продукт по расчету стоимости вычислительных сервисов, а также стоимости их простоя; руководство предприятия будет иметь возможность более детально алализировать расходы бюджета и в дальнейшем не допускать перерасходов на закупку компонентов для вычислительной инфраструктуры. Создав функционал по оценке простоя предоставляемых вычислительных сервисов, руководство иожет планировать своевременно расширять вычислительную инфраструктуру.

Так как на данный момент расчет стоимости предоставления вычислительных сервисов не является совершенным, в дальнейшем планируется расширить предметную область с учетом затрат на электроэнергию, а таже ввода ожидаемого дохода на конкретный вид деятельности предприятия, для которой был выделен сервис.

Также планируется создание полноценного сервиса, а именно:

* разработка APIдля выполненой задачи;
* создание интерфейса;
* создание мобильного приложения;
* расширение возможностей по импорту данных в приложение;
* расширение возможностей по создаю отчетов в различные форматы, а также их автоматическую рассылку на электронную почту директора предприятия при регистрации новых сбоев оборудования;
* создание диаграмм по стоимости предоставления сервисов и их простоя;
* оценка загруженности компонетов вычислительной инфраструктуры, чтобы системный администратор мог более детально мог оценивать – сколько реусурсов затрачено на действующие сервисы и доступно для создания новых.

# **6 Перечень принятых сокращений**

Аббревиатуры:

* ОАО – открытое акционерное общество;
* ОС – операционная система;
* ПАО – публичное акционерное общество;
* ПК – персональный компьютер;
* ПО – программное обеспечение;
* ПЭВМ – персональная электронно-вычислительная машина;
* СУБД – система управления базами данных;
* API – application programming interface (интерфейс прикладного программирования);
* IT – Information Technology (Информационные технологии).

Принятые сокращения:

* ГБ – гигабайт;
* Ггц – гигагерц.

# **7 Перечень терминов**

Сервис – одна из форм облачных вычислений, модель обслуживания, при которой подписчикам предоставляется готовое прикладное программное обеспечение, полностью обслуживаемое провайдером;

Платформа – модель предоставления облачных вычислений, при которой потребитель получает доступ к использованию информационно-технологических платформ: операционных систем, систем управления базами данных, связующему программному обеспечению, средствам разработки и тестирования, размещённым у облачного провайдера;

Интерфейс – совокупность средств, методов и правил взаимодействия между элементами системы;

Виртуальная машина – программная и/или аппаратная система, эмулирующая аппаратное обеспечение некоторой платформы;

API – набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах.

# **Список использованной литературы**

1. Албахари Д., Албахари Б. C# 6.0. Справочник. Полное описание языка – Вильямс, 2017. – 1040 с;
2. Алексей Гультяев – «Виртуальные машины: несколько компьютеров в одном» // Санкт-Петербург, издательский центр «Питер», 2006, 250c;
3. Бен-ГанИцик – «Microsoft SQL Server 2012. Основы T-SQL» // Москва, издательство ООО "Эксмо", 2015 – 212c;
4. Ботт Э. Windows 10 для IT-профессионалов – Эком, 2016. – 173 с;
5. В.М. Илюшечкин – «Основы использования и проектирования баз данных» // Москва, издательство «Юрайт», 2016 – 214c;
6. Карвальо Л. – «Windows Server 2012 Hyper-V. Книга рецептов» // Москва, издательство «ДМК пресс», 2013 – 304c;
7. Джесси Рассел – «Виртуальная машина» // Print-on-Demand (POD), 2018 -140c;
8. Методические рекомендации по выполнению и оформлению выпускной квалификационной работы (дипломного проекта) для студентов специальности 09.02.03 Программирование в компьютерных системах – ГАПОУ МО «МКЭиИТ», 2017 – 36 с;
9. Обзор Microsoft SQL Server [Электронный ресурс] URL: http://www.tadviser.ru/index.php/%D0%9F%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82:Microsoft\_SQL\_Server(дата обращения: 10.06.18);
10. Основные возможности среды SQL Server Management Studio (SSMS) [Электронный ресурс] URL: http://info-comp.ru/softprodobes/526-environment-sql-server-management-studio.html (дата обращения: 07.06.18);
11. Adam Beguelin, Jack Dongarra, Weicheng Jiang – «PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Network Parallel Computing (Scientific and Engineering Computation) » // The MIT Press, 1994 - 299c;
12. Dave Mishchenko – «VMware ESXi: Planning, Implementation, and Security» // Cengage Learning PTR, 2010 – 480c;
13. Ken St. Cyr - «Microsoft Exchange Server 2010 Administration Instant Reference» // Sybex, 2016 – 566c;
14. Rebecca Fitzhugh – «vSphere Virtual Machine Management» // Packt Publishing - ebooks Account, 2014 - 367c;
15. Windows 10: преимущества и особенности [Электронный ресурс] URL: http://blog-pc.ru/windows-10-preimushhestva-i-osobennosti.html (дата обращения: 08.06.18).

# **Приложение А.**

**Выходные документы**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | |  | | --- | | ПАО «Мурмансикй тралловый флот»  Россия, 183038, г. Мурманск, ул. Шмидта, д. 43  E-Mail: mtf@mtf.ru  Телефон: (815-2) 288-111  Факс: (815-2) 288-070 | | | |  |
|  |  |  |  |  |  |
| |  | | --- | | **Стоимость предоставления сервисов** | | | | | | |
|  |  |  |  |  |  |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **Платформа** | **Сервис** | **№ п/п** | **Дата создания** | **Состояние** | **Кол-во пользователей** | **Стоимость руб./час** | | 1С | Толстый клиент | 2 | 14.02.2017 | Активен | 1 | 46,0468 | | **Всего** | **2** |  |  | **1** | **46,0468** | | SQL | Oracle DB | 4 | 14.03.2018 | Активен | 2 | 37,6455 | | **Всего** | **4** |  |  | **2** | **37,6455** | | Тех. поддержка | Помощь сотрудникам | 3 | 14.11.2017 | Активен | 3 | 24,2045 | | **Всего** | **3** |  |  | **3** | **24,2045** | | **Всего** |  | **9** |  |  | **6** | **107,8968** | | | | | |  |
|  |  |  |  |  |  |
| |  | | --- | | Генеральный директор:  Старков Денис Алексеевич | | | |  | |  | | --- | | Подпись\_\_\_\_\_\_\_\_\_: | |  |
|  |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | |  | | --- | | ПАО «Мурмансикй тралловый флот»  Россия, 183038, г. Мурманск, ул. Шмидта, д. 43  E-Mail: mtf@mtf.ru  Телефон: (815-2) 288-111  Факс: (815-2) 288-070 | | | |  |
|  |  |  |  |  |  |
| |  | | --- | | **Стоимость простоя предоставляемого сервиса** | | | | | |  |
|  |  |  |  |  |  |
| |  | | --- | | **№ п/п** 1 **Сервис** Тонкий клиент **Платформа** 1С | | | | | |  |
|  |  |  |  |  |  |
| |  | | --- | | **Начало** 28.05.2018 23:09**Окончание** 30.05.2018 4:08  **Продолжительность Минуты:** 1739 | | | | | |  |
|  |  |  |  |  |  |
| |  | | --- | | **Тип простоя** Внеплановый **Причина** Выход оборудования из строя | | | | | |  |
|  |  |  |  |  |  |
| |  | | --- | | **Кол-во отключенных сотрудников** 3 | | | | | |  |
|  |  |  |  |  |  |
| |  | | --- | | **Задействовано персонала** 3 | | | | | |  |
|  |  |  |  |  |  |
| |  | | --- | | **Расходы на отключение сервера** 15749,73 **Потерянный доход** 188352,00 | | | | | |  |
|  |  |  |  |  |  |
| |  | | --- | | **Итого:** 204101,73 | | | | | |  |
|  |  |  |  |  |  |
| |  | | --- | | Генеральный директор:  Старков Денис Алексеевич | | | |  | |  | | --- | | Подпись:\_\_\_\_\_\_\_\_ | |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  | |  | | --- | | ПАО «Мурмансикй тралловый флот»  Россия, 183038, г. Мурманск, ул. Шмидта, д. 43  E-Mail: mtf@mtf.ru  Телефон: (815-2) 288-111  Факс: (815-2) 288-070 | | | |  |
|  |  |  |  |  |  |  |  |  |
|  | |  | | --- | | **Заявка на предоставление сервиса** | | | | | | | |  |
|  |  |  |  |  |  |  |  |  |
|  | |  | | --- | | Наименование | | |  | |  | | --- | | Тонкий клиент | | | | |  |
|  |  |  |  |  |  |  |  |  |
|  | |  | | --- | | Платформа | | |  | |  | | --- | | 1С | | | | |  |
|  |  |  |  |  |  |  |  |  |
|  | |  | | --- | | Дата создания | | |  | |  | | --- | | 14.02.2018 | | | | |  |
|  |  |  |  |  |  |  |  |  |
|  | |  | | --- | | Стоимость (руб./час) | | |  | |  | | --- | | 5,18 | | | | |  |
|  |  |  |  |  |  |  |  |  |
|  | |  | | --- | | Установленное программное обсеспечение | | | | | | | |  |
|  |  |  |  |  |  |  |  |  |
|  | |  |  |  | | --- | --- | --- | | **Тип ПО** | **Наименование** | **Стоимость (руб.)** | | Системное | VMware vSphere 6.7 (ESXi) | 60289,00 | | Windows 10 Pro | 12950,00 | | ***Всего:*** |  | **73239,00** | | | | | | | |  |
|  |  |  |  |  |  |  |  |  |
|  | |  | | --- | | Генеральный директор:  Старков Денис Алексеевич | | | | | |  | |  | | --- | | Подпись:\_\_\_\_\_\_ | |  |

# **Приложение Б.**

**Образцы входных документов**

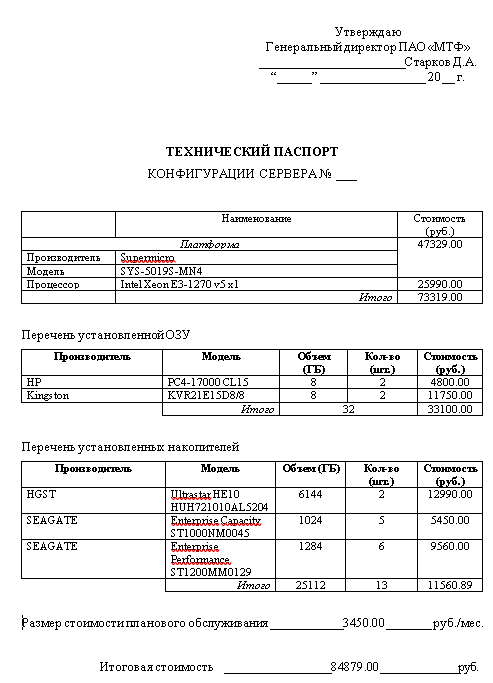


Рисунок Б.1 –Технический паспорт на конфигурацию сервера

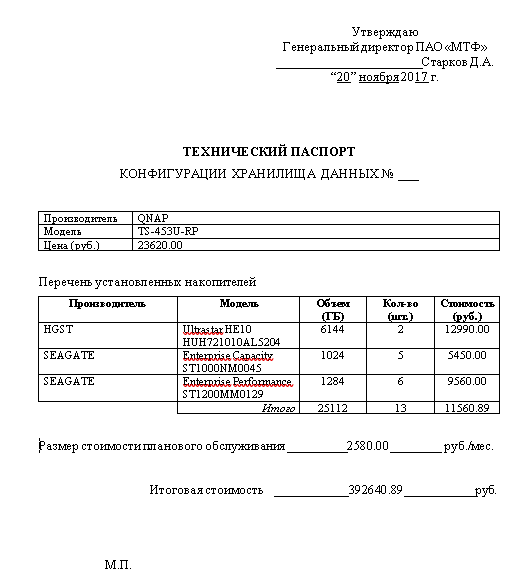


Рисунок Б.2 – Технический паспорт на конфигурацию хранилища данных

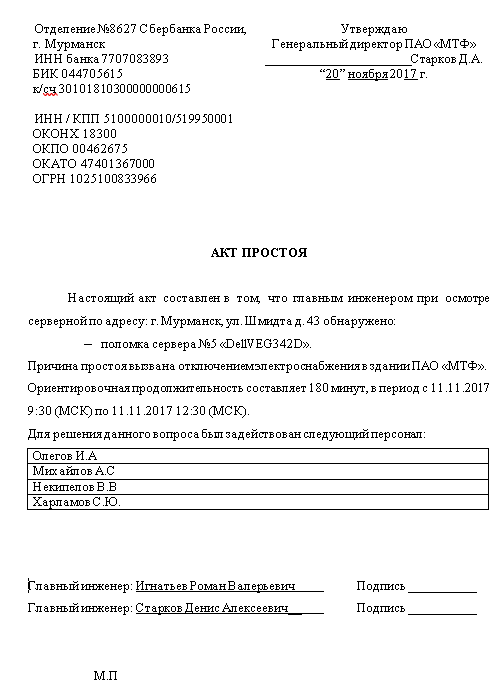


Рисунок Б.3 – Акт простоя



Рисунок Б.4 – Справка о предоставлении средней заработной платы

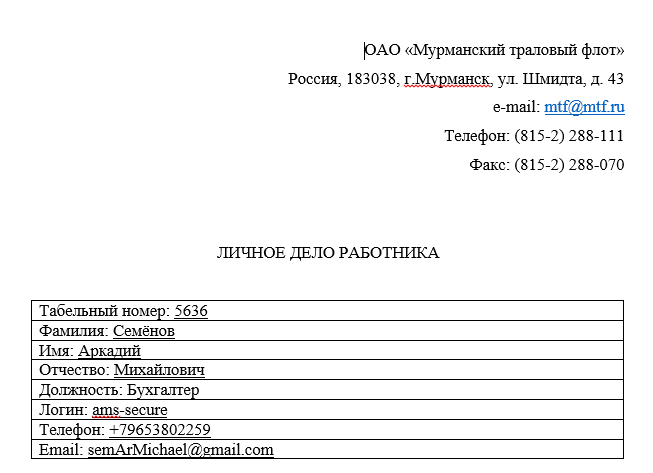


Рисунок Б.5 – Личное дело работника

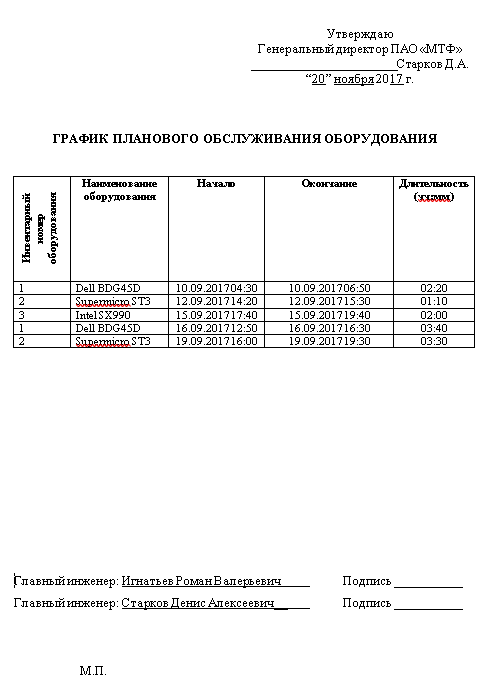


Рисунок Б.6 – График планового обслуживания оборудования

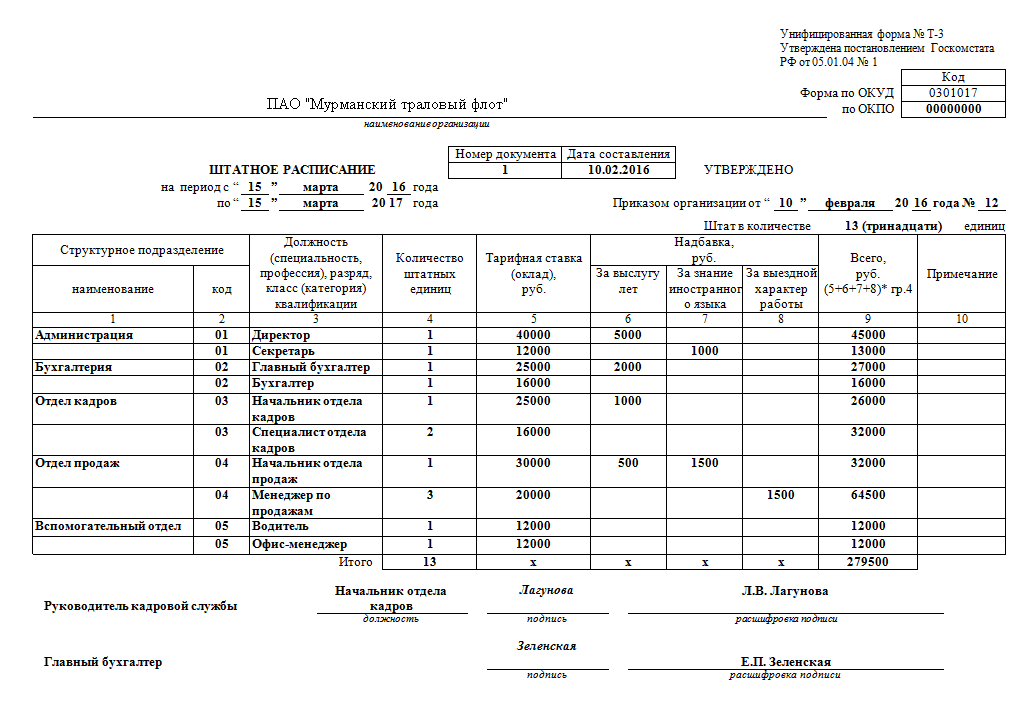


Рисунок Б.7 – Штатное расписание

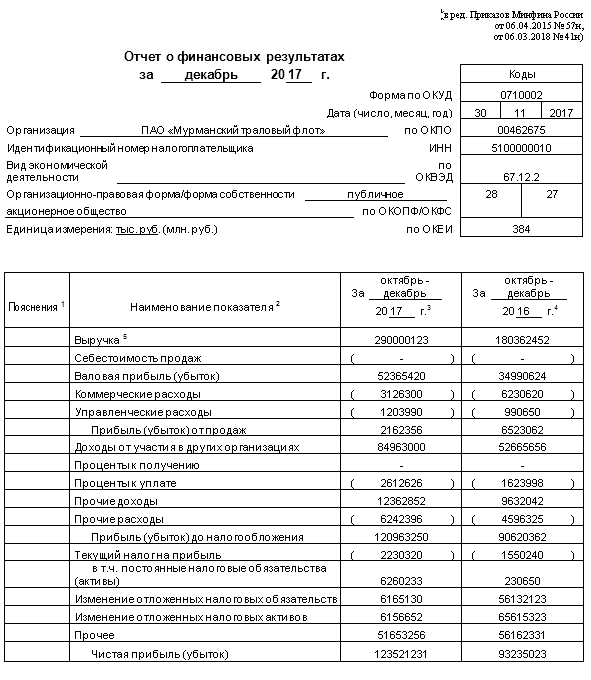


Рисунок Б.8 – Отчет о финансовых результатах за год

# **Приложение В.**

**Инфологическая модель БД**

# **Приложение Г.**

**Распечатка содержимого файлов базы данных**

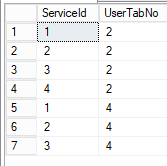


Рисунок Г.1 – Таблица «AdminIdle»

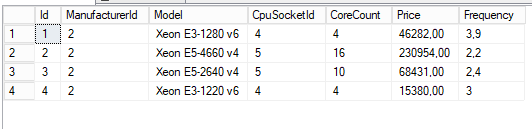


Рисунок Г.2 – Таблица «CPU»

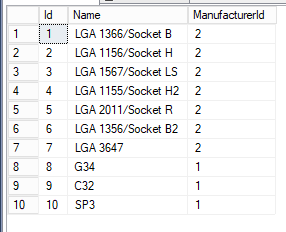


Рисунок Г.3 – Таблица «CpuSocket»

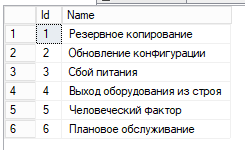


Рисунок Г.4 – Таблица «IdleReason»

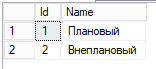


Рисунок Г.5 – Таблица «IdleType»



Рисунок Г.6 – Таблица «Manufacturer»

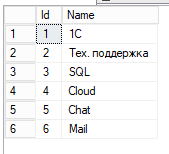


Рисунок Г.7 – Таблица «PaasType»

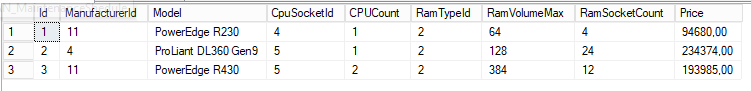
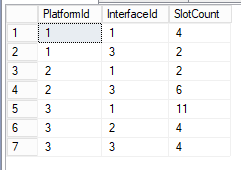
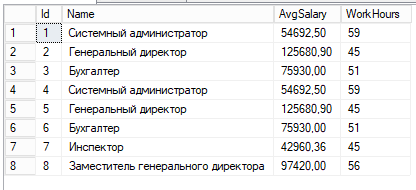


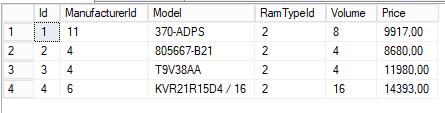
Рисунок Г.8 – Таблица «Platform»



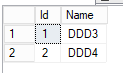
РисунокГ.9 – Таблица «Platform\_StorageInt»



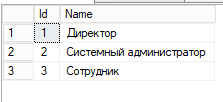
РисунокГ.10 – Таблица «Position»



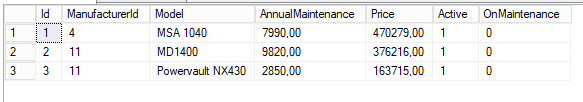
РисунокГ.11 – Таблица «RAM»



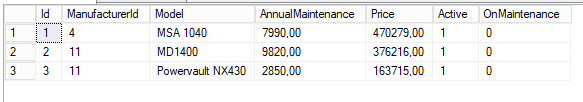
РисунокГ.12 – Таблица «RamType»



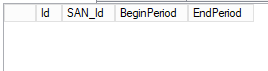
РисунокГ.13 – Таблица «RightsLevel»



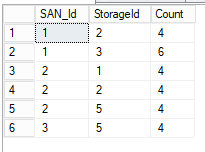
РисунокГ.14 – Таблица «SAN»



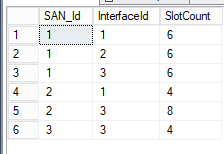
РисунокГ.15 – Таблица «SAN»



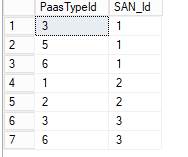
РисунокГ.16 – Таблица «SAN\_MaintenanceShedule»



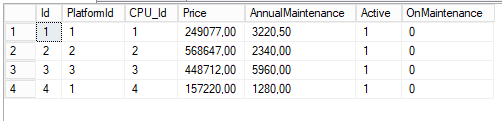
РисунокГ.17 – Таблица «SAN\_Storage»



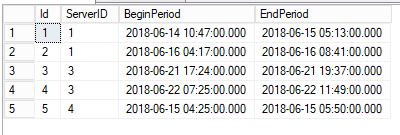
РисунокГ.18 – Таблица «SAN\_StorageInt»



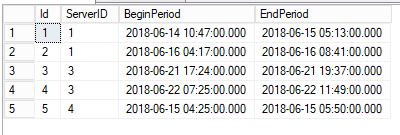
РисунокГ.19 – Таблица «SANtoPaas»



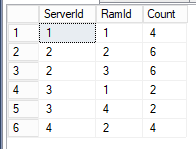
РисунокГ.20 – Таблица «Server»



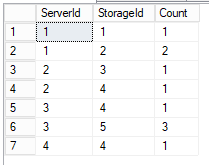
РисунокГ.21 – Таблица «Server\_MaintenanceShedule»



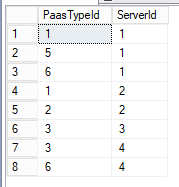
РисунокГ.22 – Таблица «Server\_MaintenanceShedule»



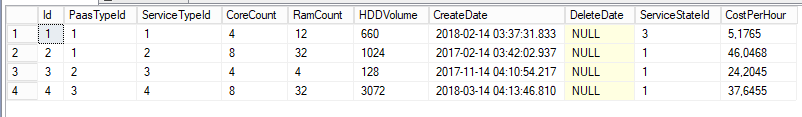
РисунокГ.23 – Таблица «Server\_RAM »



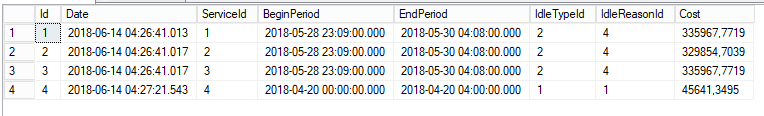
РисунокГ.24 – Таблица «Server\_Storage»



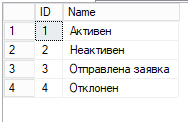
РисунокГ.25 – Таблица «ServerToPaas»



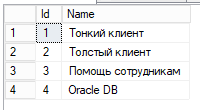
РисунокГ.26 – Таблица «Service»



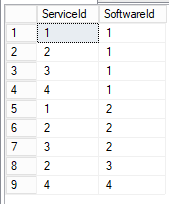
РисунокГ.27 – Таблица «ServiceIdle»



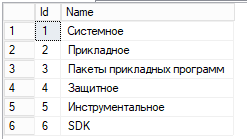
РисунокГ.28 – Таблица «ServiceState»



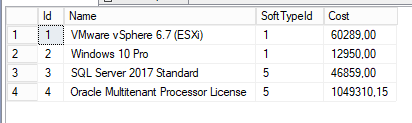
РисунокГ.29 – Таблица «ServiceType»



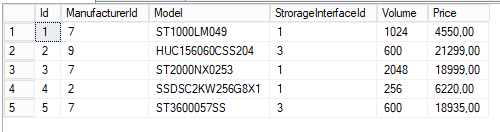
РисунокГ.30 – Таблица «SoftService»



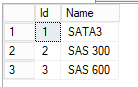
РисунокГ.31 – Таблица «SoftType»



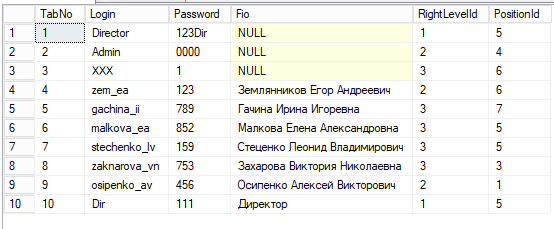
РисунокГ.32 – Таблица «Software»



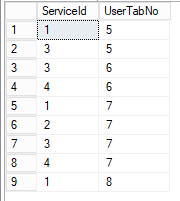
РисунокГ.33 – Таблица «Strorage»



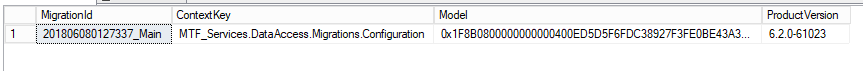
РисунокГ.34 – Таблица «StrorageInterface»



РисунокГ.35 – Таблица «User»



РисунокГ.36 – Таблица «UsingService»



РисунокГ.37 – Таблица «\_\_MigrationHistory»

# **Приложение Д.**

**Экранные формы**

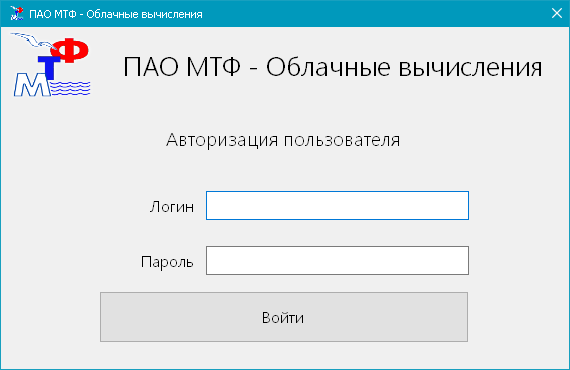


Рисунок Д.1 – Форма авторизации пользователей

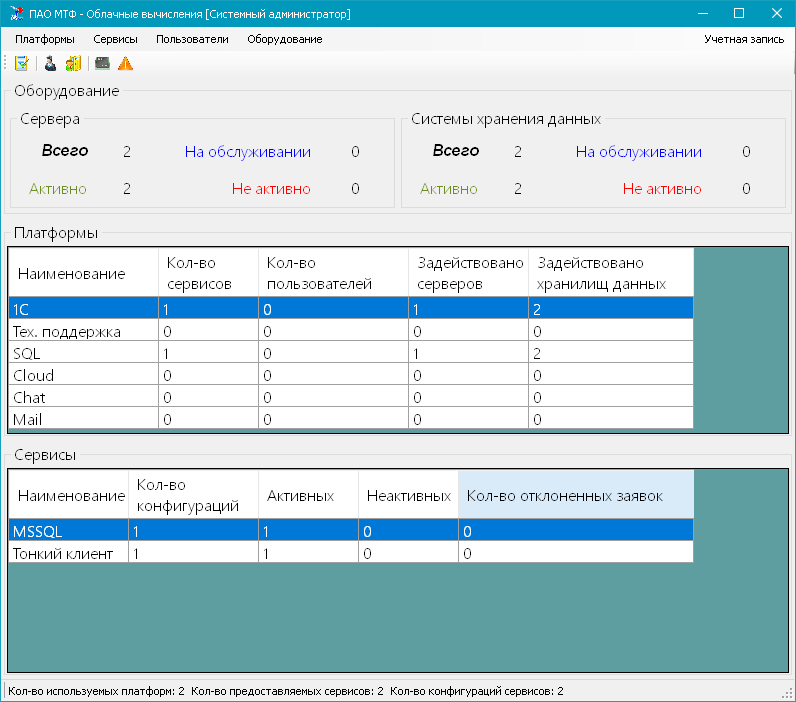


Рисунок Д.2 –Форма главного экрана для системного администратора

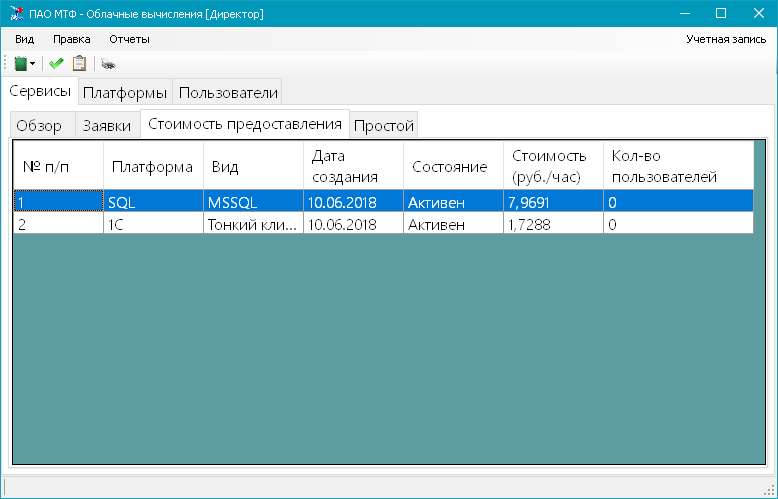


Рисунок Д.3 –Форма главного экрана для директора

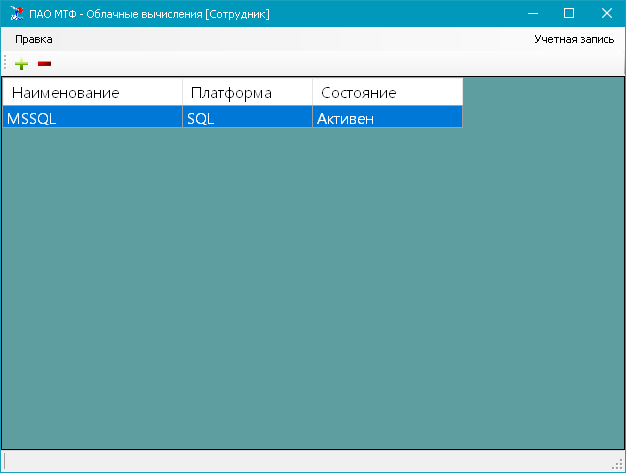


Рисунок Д.4 –Форма главного экрана для сотрудника

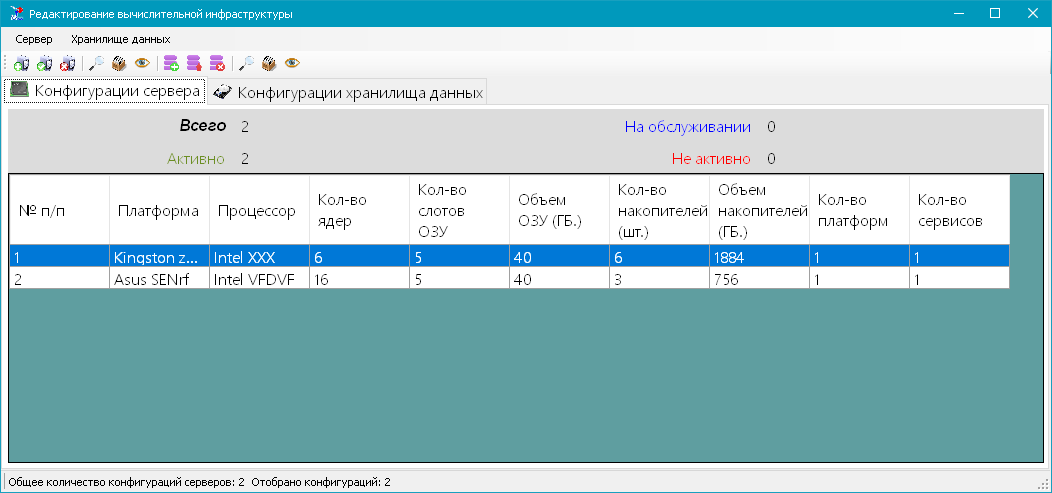


Рисунок Д.5 – Форма редактирования вычислительной инфраструктуры

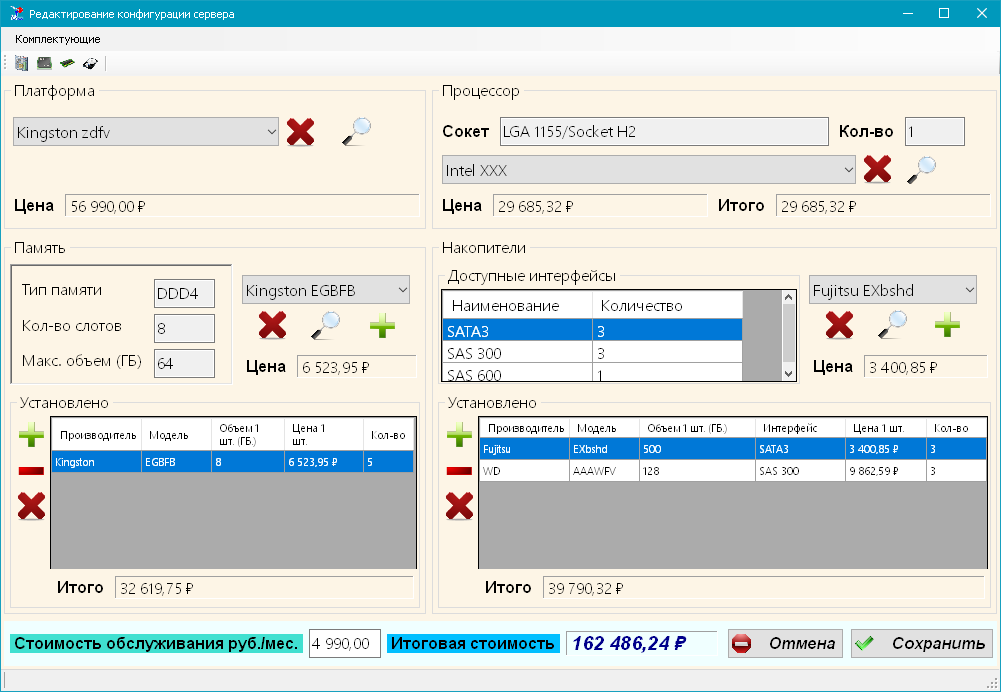


Рисунок Д.6 – Форма редактирования конфигурации сервера

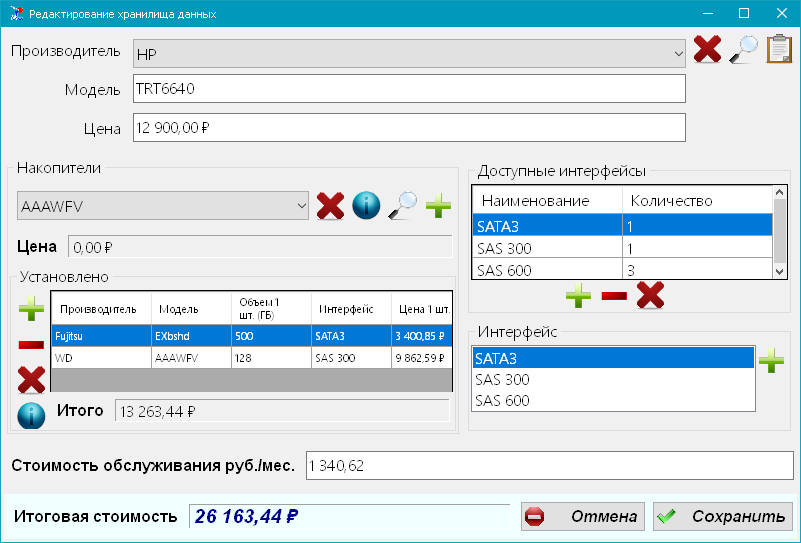


Рисунок Д.7 – Форма редактирования конфигурации хранилища данных

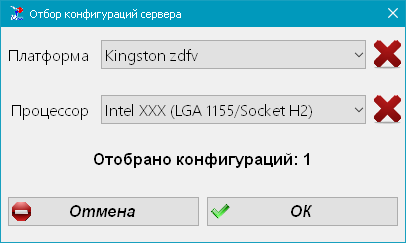


Рисунок Д.8 – Форма отбора конфигураций сервера

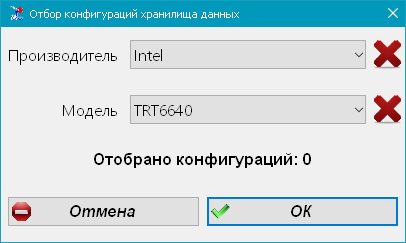


Рисунок Д.9 – Форма отбора конфигураций хранилища данных

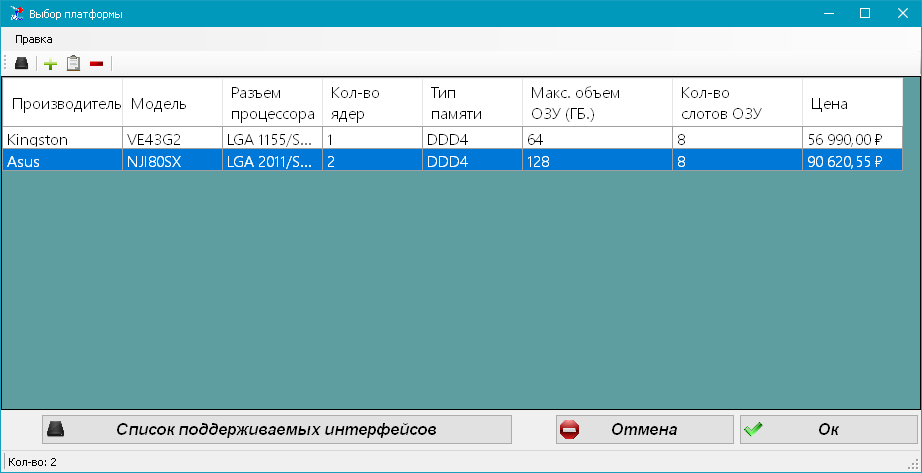


Рисунок Д.10 – Форма выбора платформы

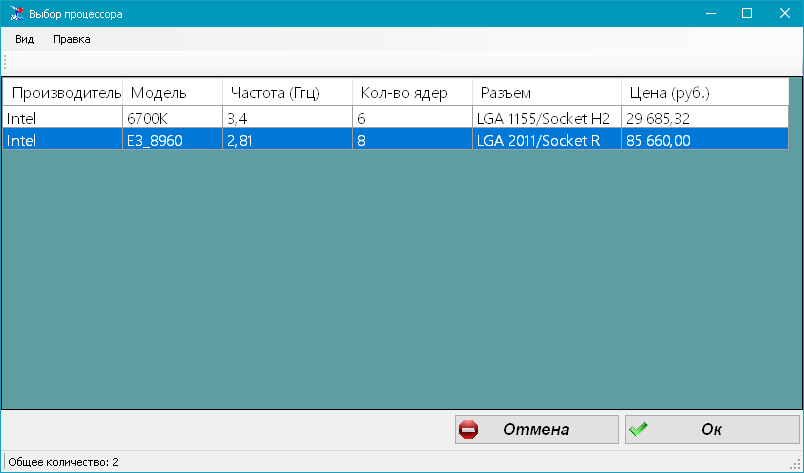


Рисунок Д.11 – Форма выбора процессора

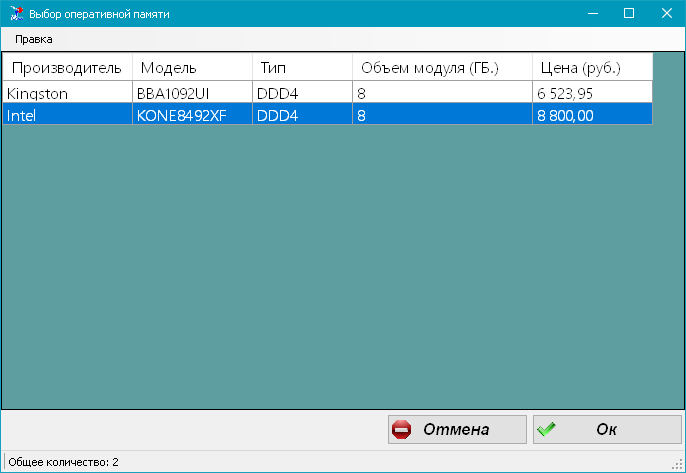


Рисунок Д.12 – Форма выбора оперативной памяти

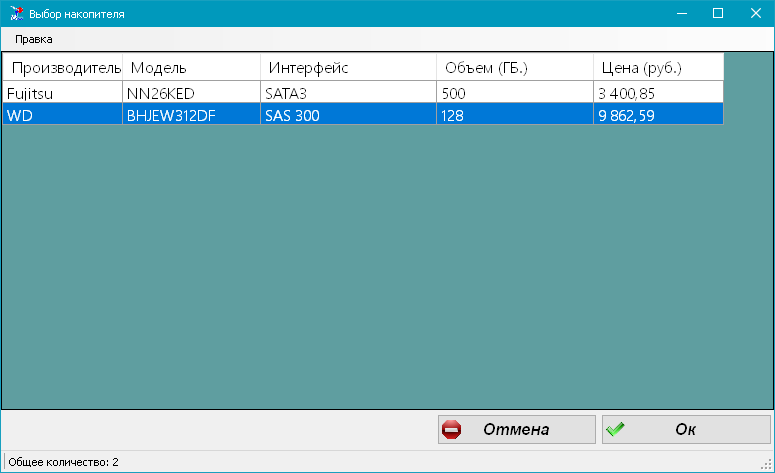


Рисунок Д.13 – Форма выбора накопителя

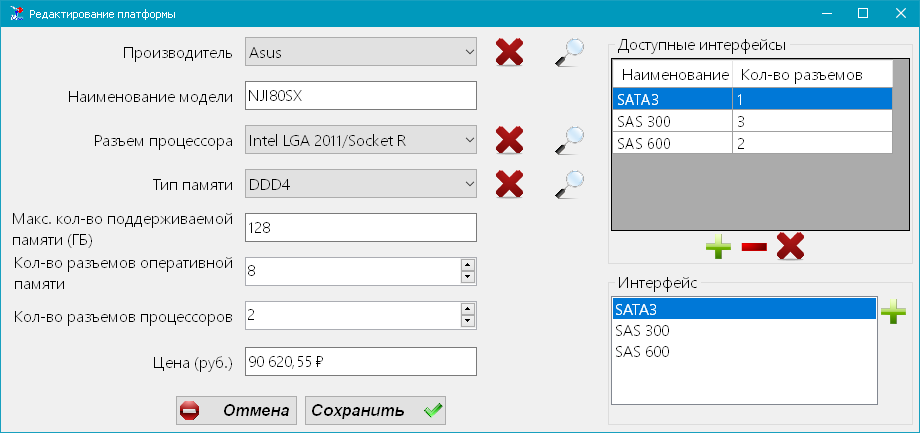


Рисунок Д.14 – Форма редактирования платформы

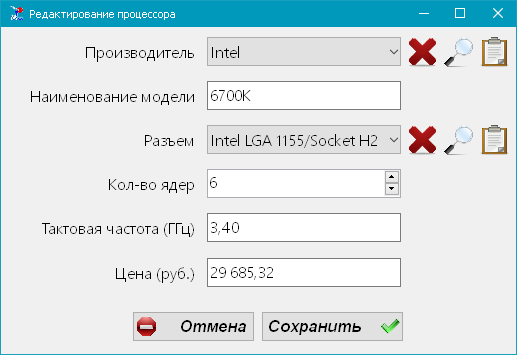


Рисунок Д.15 – Форма редактирования процессора

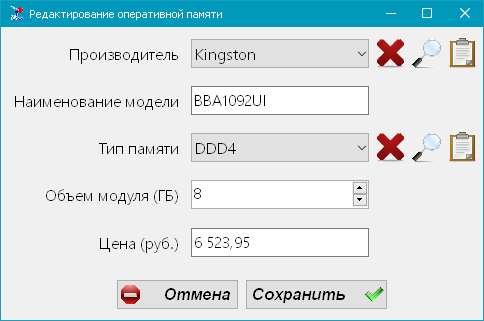


Рисунок Д.16 – Форма редактирования оперативной памяти

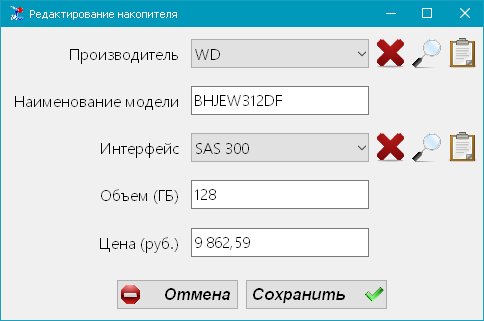


Рисунок Д.17 – Форма редактирования накопителя

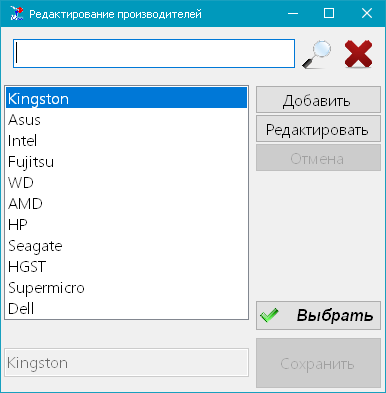


Рисунок Д.18 – Форма редактирования производителей

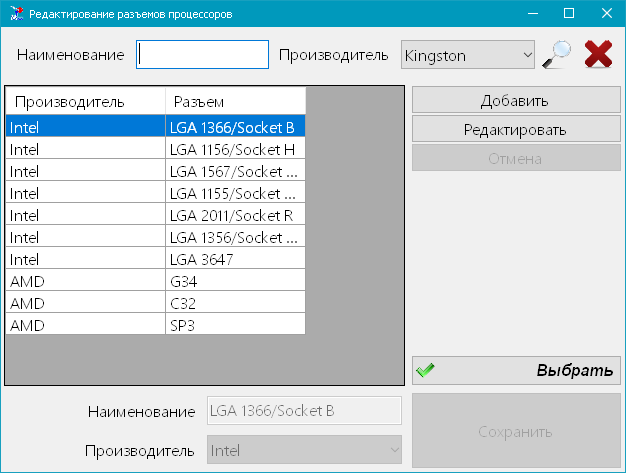


Рисунок Д.19 – Форма редактирования разъемов процессоров

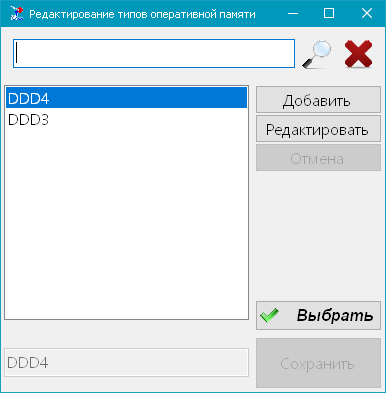


Рисунок Д.20 – Форма редактирования типов оперативной памяти

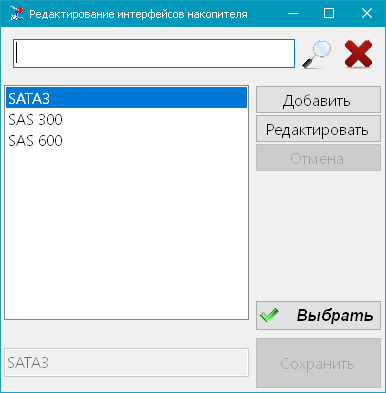


Рисунок Д.21 – Форма редактирования интерфейсов накопителя

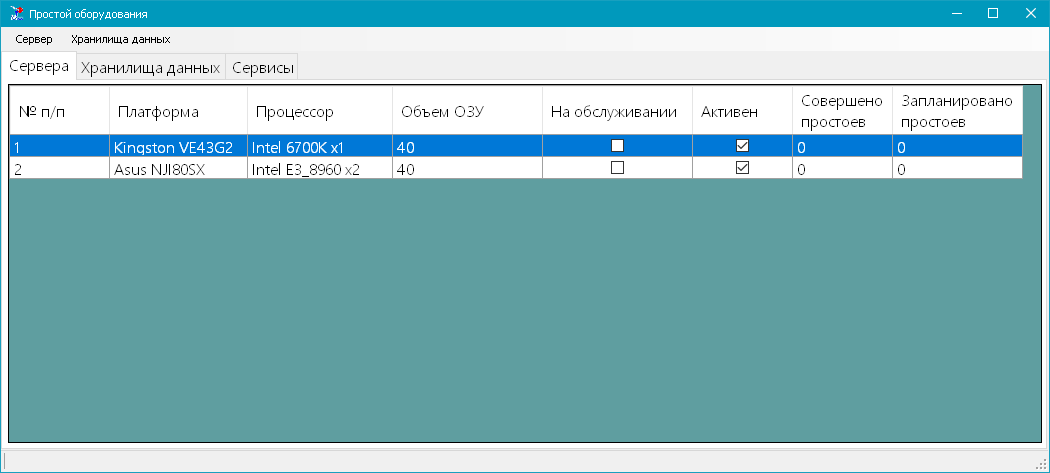


Рисунок Д.22 – Форма просмотра состояния оборудования

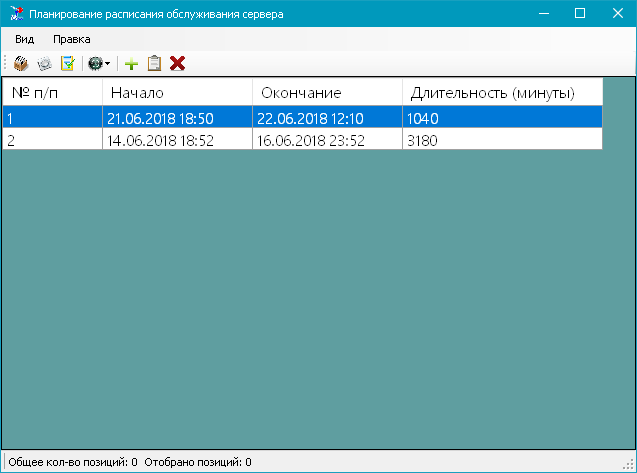


Рисунок Д.23 – Форма просмотра позиций планового обслуживания оборудования

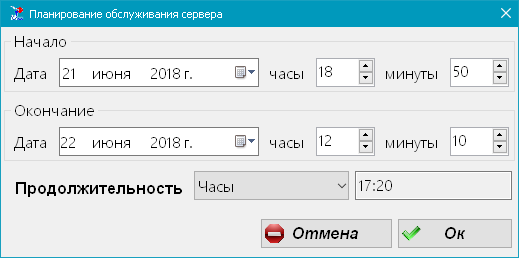


Рисунок Д.24 – Форма планированияпозиции расписания планового обслуживания оборудования

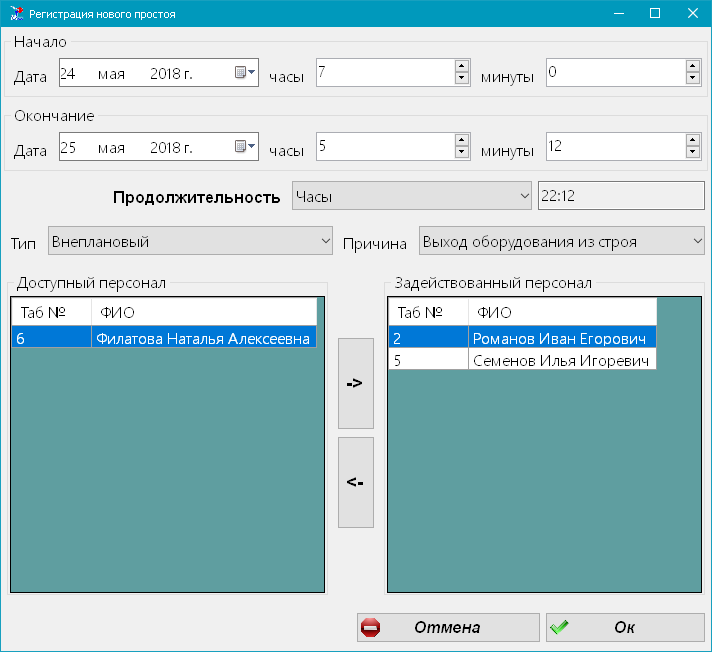


Рисунок Д.25 – Форма регистрации совершенного простоя оборудования

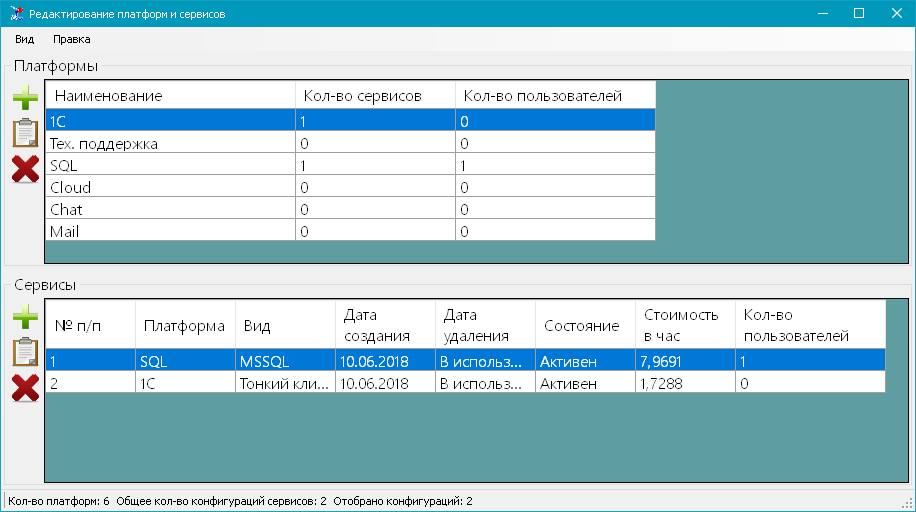


Рисунок Д.26 – Форма редактирования платформ и сервисов

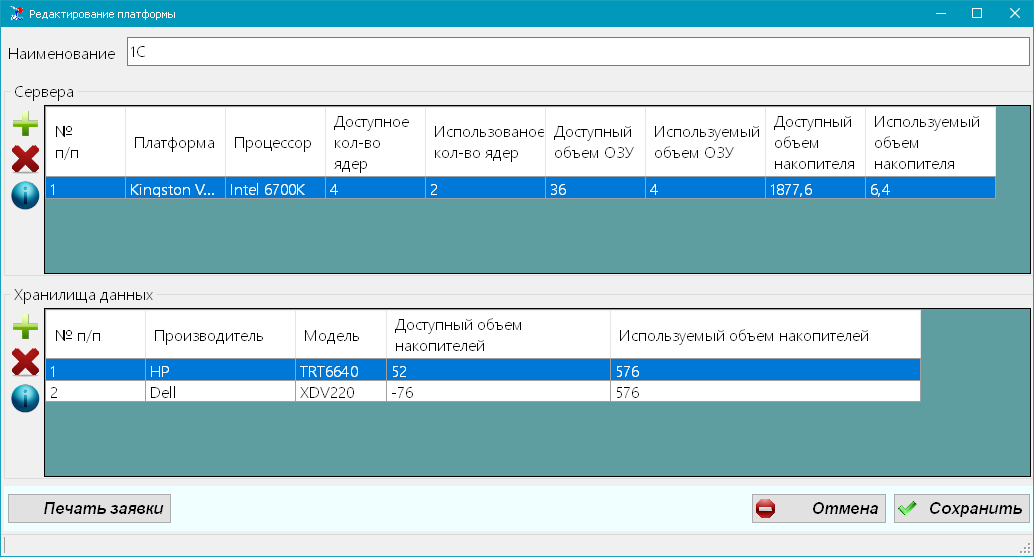


Рисунок Д.27 – Форма редактирования платформы

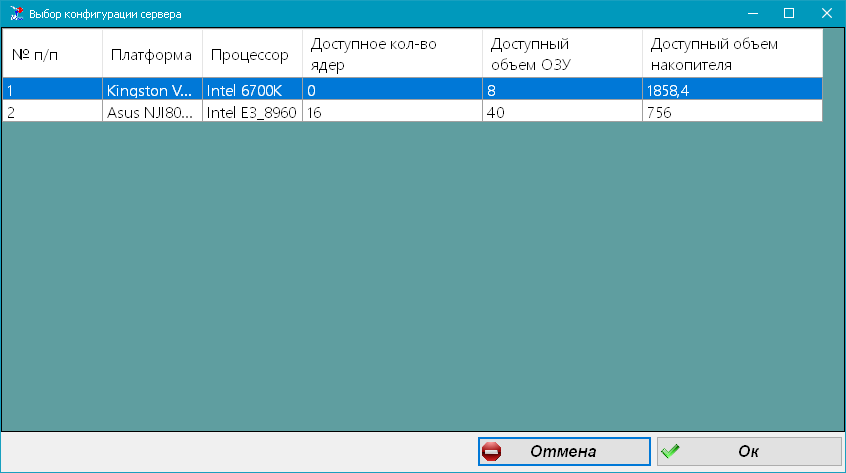


Рисунок Д.28 – Форма выбора конфигурации сервера для редактируемой платформы

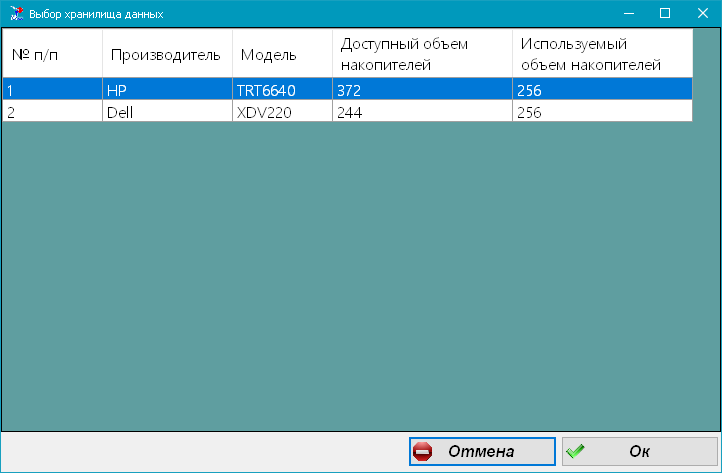


Рисунок Д.29 – Форма выбора хранилища данных для редактируемой платформы

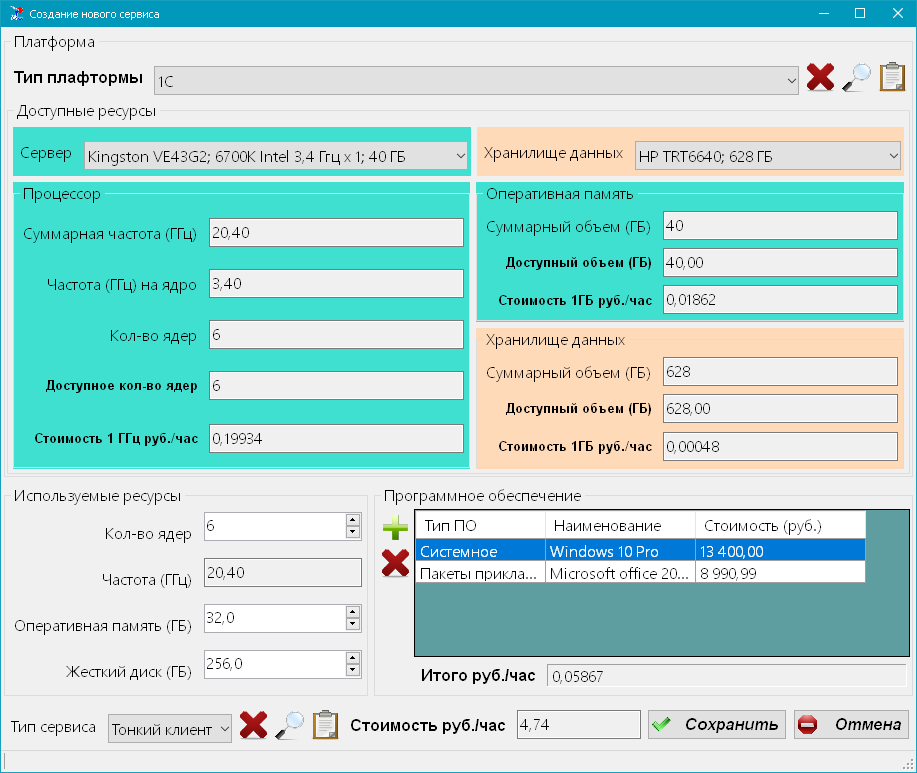


Рисунок Д.30 – Форма редактирования конфигурации сервиса

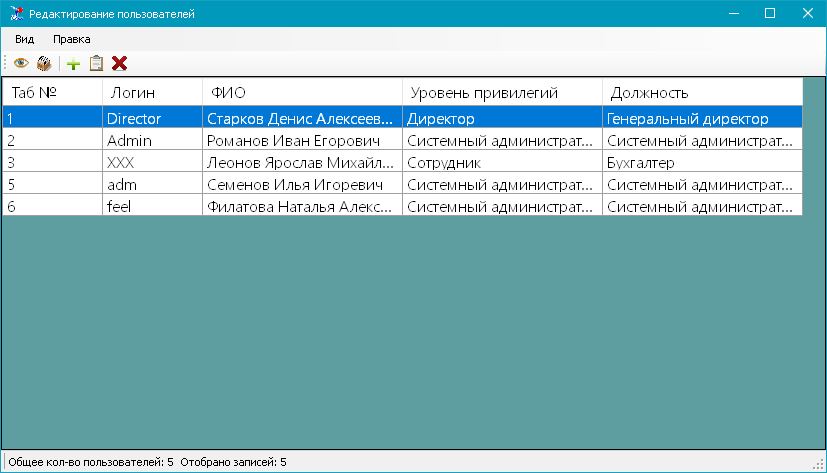


Рисунок Д.31 – Форма редактирования зарегистрированных пользователей

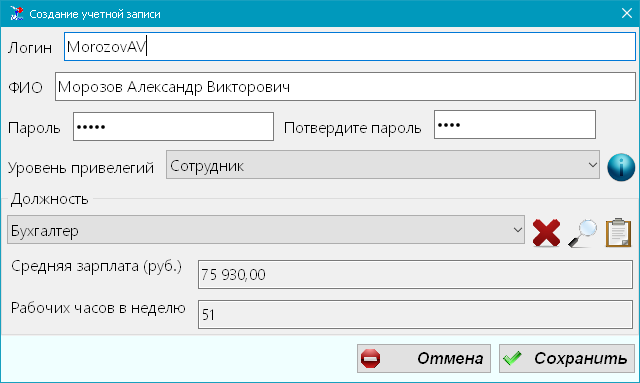


Рисунок Д.32 – Форма регистрации новой учетной записи

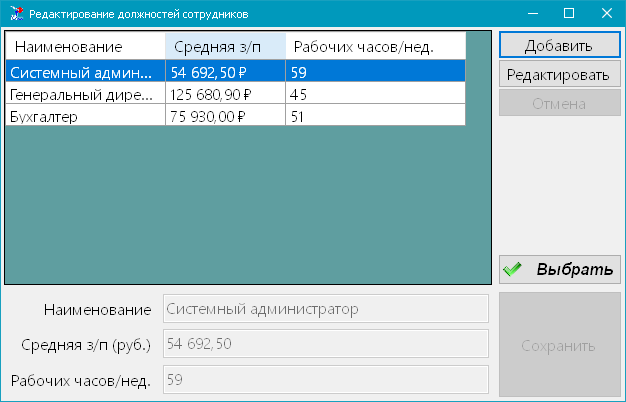


Рисунок Д.33 – Форма редактирования должностей сотрудников

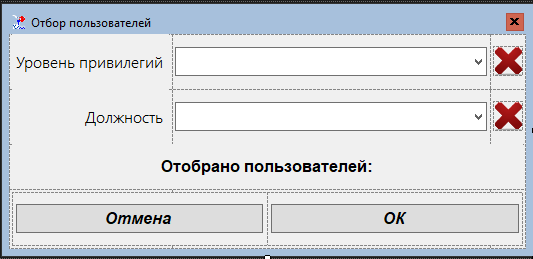


Рисунок Д.34 – Форма отбора пользователей

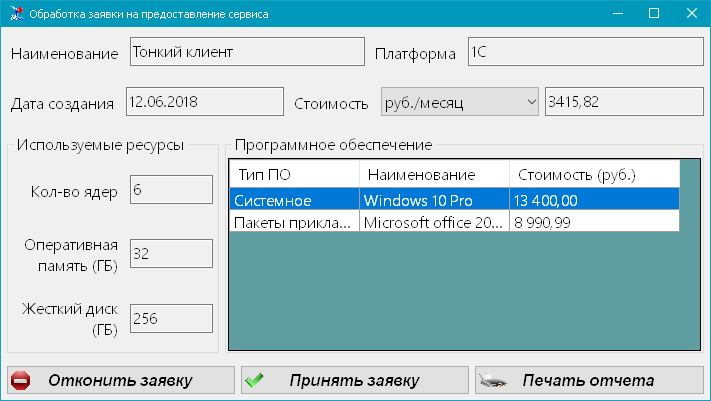


Рисунок Д.35 – Форма обработки заявки на предоставление сервиса

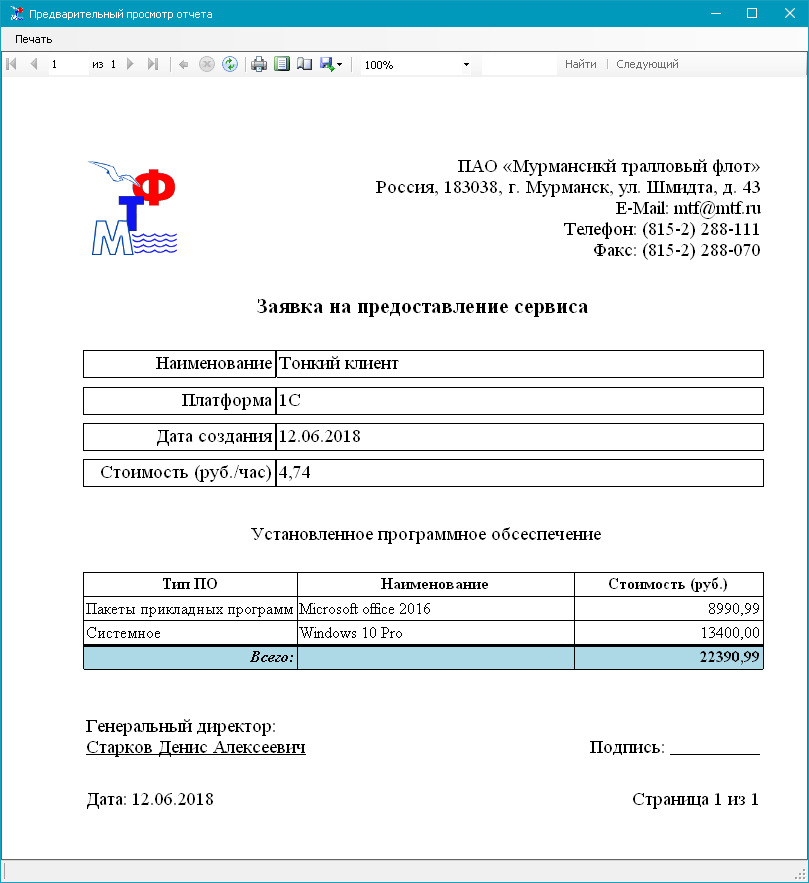


Рисунок Д.36 – Форма предварительного просмотра отчета

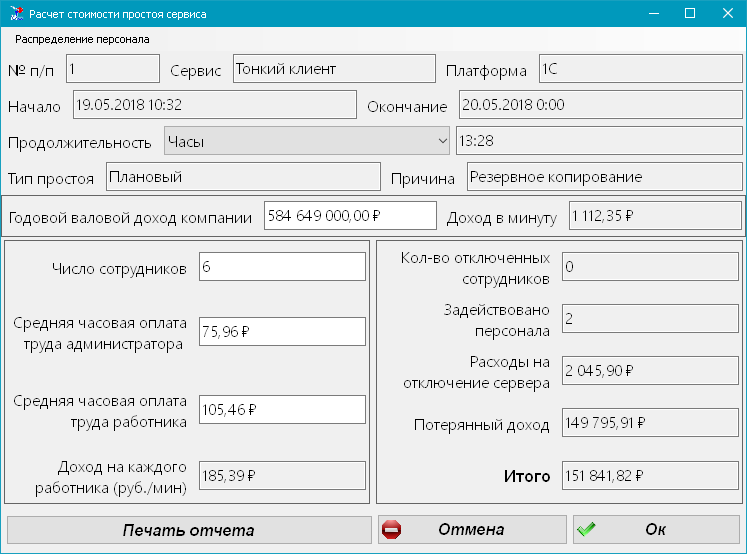


Рисунок Д.37 – Форма расчета стоимости простоя сервиса

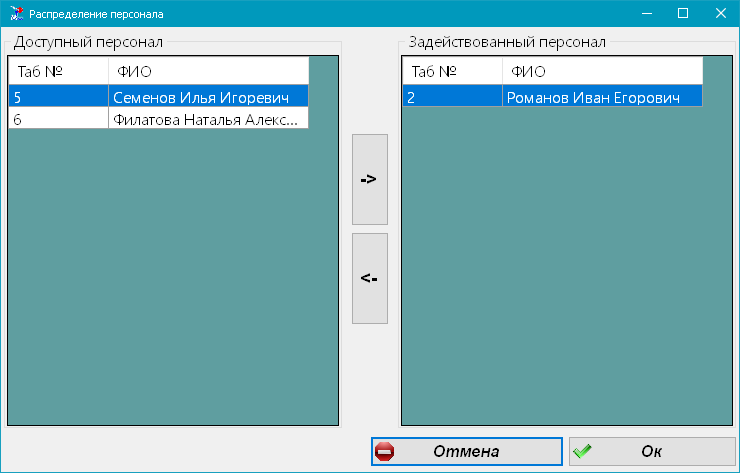


Рисунок Д.38 – Форма распределения персонала, который задействован для решения нештатной ситуации

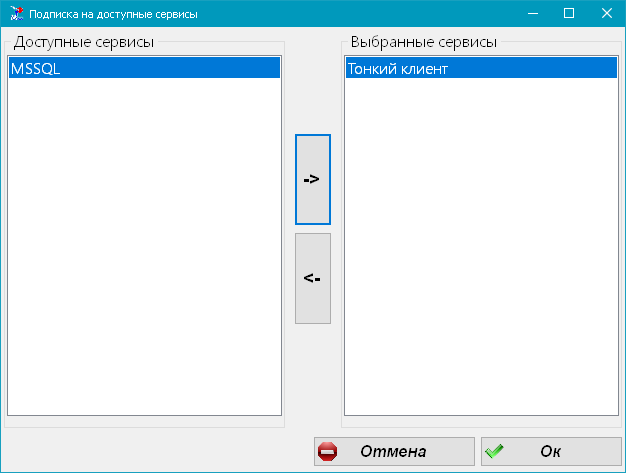


Рисунок Д.39 – Форма оформления подписки на предоставляемые сервисы

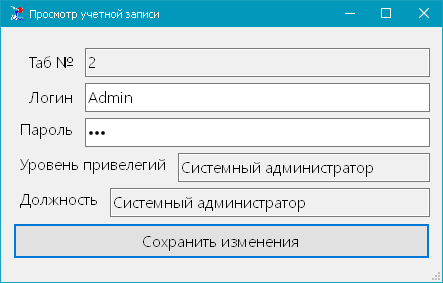


Рисунок Д.40 – Форма просмотра текущей учетной записи

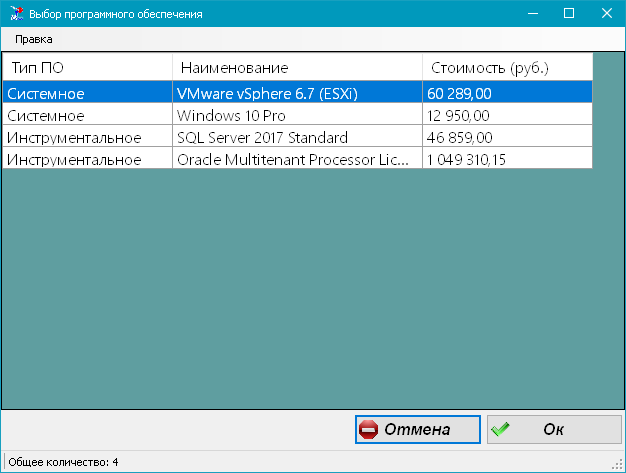


Рисунок Д.41 – Форма выбора программного обеспечения

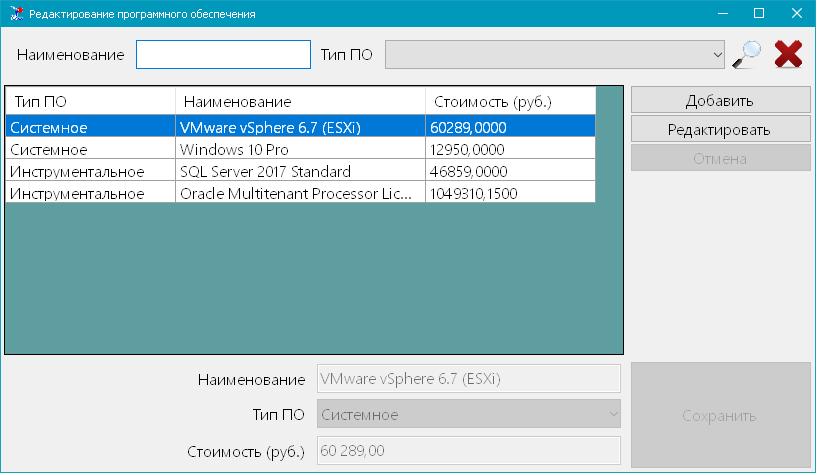


Рисунок Д.42 – Форма редактирования программного обеспечения

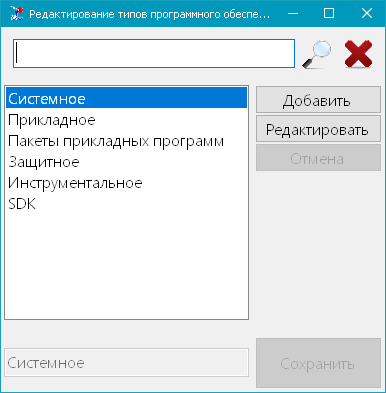


Рисунок Д.43 – Форма редактирования типов сервиса

# **Приложение Е.**

**Листинг программы**

Модуль «201806080127337\_Main.cs»

namespace MTF\_Services.DataAccess.Migrations

{

using System;

using System.Data.Entity.Migrations;

public partial class Main : DbMigration

{

/// <summary>

/// Перечень выполняемых операций с сущностями при обновлении базы данных

/// </summary>

public override void Up()

{

CreateTable(

"dbo.CPU",

c => new

{

Id = c.Int(nullable: false, identity: true),

ManufacturerId = c.Short(nullable: false),

Model = c.String(nullable: false, maxLength: 50),

CpuSocketId = c.Byte(nullable: false),

CoreCount = c.Short(nullable: false),

Price = c.Decimal(nullable: false, storeType: "money"),

Frequency = c.Double(nullable: false),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.CpuSocket", t => t.CpuSocketId, cascadeDelete: true)

.ForeignKey("dbo.Manufacturer", t => t.ManufacturerId, cascadeDelete: true)

.Index(t => t.ManufacturerId)

.Index(t => t.CpuSocketId);

CreateTable(

"dbo.CpuSocket",

c => new

{

Id = c.Byte(nullable: false, identity: true),

Name = c.String(nullable: false, maxLength: 50),

ManufacturerId = c.Short(nullable: false),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.Manufacturer", t => t.ManufacturerId)

.Index(t => t.ManufacturerId);

CreateTable(

"dbo.Manufacturer",

c => new

{

Id = c.Short(nullable: false, identity: true),

Name = c.String(nullable: false, maxLength: 100),

})

.PrimaryKey(t => t.Id);

CreateTable(

"dbo.Platform",

c => new

{

Id = c.Int(nullable: false, identity: true),

ManufacturerId = c.Short(nullable: false),

Model = c.String(nullable: false, maxLength: 100),

CpuSocketId = c.Byte(nullable: false),

CPUCount = c.Byte(nullable: false),

RamTypeId = c.Byte(nullable: false),

RamVolumeMax = c.Int(nullable: false),

RamSocketCount = c.Byte(nullable: false),

Price = c.Decimal(nullable: false, storeType: "money"),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.CpuSocket", t => t.CpuSocketId, cascadeDelete: true)

.ForeignKey("dbo.Manufacturer", t => t.ManufacturerId, cascadeDelete: true)

.ForeignKey("dbo.RamType", t => t.RamTypeId, cascadeDelete: true)

.Index(t => t.ManufacturerId)

.Index(t => t.CpuSocketId)

.Index(t => t.RamTypeId);

CreateTable(

"dbo.Platform\_StorageInt",

c => new

{

PlatformId = c.Int(nullable: false),

InterfaceId = c.Byte(nullable: false),

SlotCount = c.Byte(nullable: false),

})

.PrimaryKey(t => new { t.PlatformId, t.InterfaceId })

.ForeignKey("dbo.Platform", t => t.PlatformId, cascadeDelete: true)

.ForeignKey("dbo.StrorageInterface", t => t.InterfaceId, cascadeDelete: true)

.Index(t => t.PlatformId)

.Index(t => t.InterfaceId);

CreateTable(

"dbo.StrorageInterface",

c => new

{

Id = c.Byte(nullable: false, identity: true),

Name = c.String(maxLength: 50),

})

.PrimaryKey(t => t.Id);

CreateTable(

"dbo.SAN\_StorageInt",

c => new

{

SAN\_Id = c.Int(nullable: false),

InterfaceId = c.Byte(nullable: false),

SlotCount = c.Byte(nullable: false),

})

.PrimaryKey(t => new { t.SAN\_Id, t.InterfaceId })

.ForeignKey("dbo.SAN", t => t.SAN\_Id, cascadeDelete: true)

.ForeignKey("dbo.StrorageInterface", t => t.InterfaceId, cascadeDelete: true)

.Index(t => t.SAN\_Id)

.Index(t => t.InterfaceId);

CreateTable(

"dbo.SAN",

c => new

{

Id = c.Int(nullable: false, identity: true),

ManufacturerId = c.Short(nullable: false),

Model = c.String(nullable: false, maxLength: 50),

AnnualMaintenance = c.Decimal(storeType: "money"),

Price = c.Decimal(nullable: false, storeType: "money"),

Active = c.Boolean(nullable: false),

OnMaintenance = c.Boolean(nullable: false),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.Manufacturer", t => t.ManufacturerId, cascadeDelete: true)

.Index(t => t.ManufacturerId);

CreateTable(

"dbo.PaasType",

c => new

{

Id = c.Short(nullable: false, identity: true),

Name = c.String(nullable: false, maxLength: 100),

})

.PrimaryKey(t => t.Id);

CreateTable(

"dbo.Server",

c => new

{

Id = c.Int(nullable: false, identity: true),

PlatformId = c.Int(nullable: false),

CPU\_Id = c.Int(nullable: false),

Price = c.Decimal(storeType: "money"),

AnnualMaintenance = c.Decimal(storeType: "money"),

Active = c.Boolean(nullable: false),

OnMaintenance = c.Boolean(nullable: false),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.Platform", t => t.PlatformId, cascadeDelete: true)

.ForeignKey("dbo.CPU", t => t.CPU\_Id, cascadeDelete: false)

.Index(t => t.PlatformId)

.Index(t => t.CPU\_Id);

CreateTable(

"dbo.Server\_MaintenanceShedule",

c => new

{

Id = c.Int(nullable: false, identity: true),

ServerID = c.Int(nullable: false),

BeginPeriod = c.DateTime(nullable: false),

EndPeriod = c.DateTime(nullable: false),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.Server", t => t.ServerID, cascadeDelete: true)

.Index(t => t.ServerID);

CreateTable(

"dbo.Server\_RAM",

c => new

{

ServerId = c.Int(nullable: false),

RamId = c.Int(nullable: false),

Count = c.Byte(nullable: false),

})

.PrimaryKey(t => new { t.ServerId, t.RamId })

.ForeignKey("dbo.RAM", t => t.RamId)

.ForeignKey("dbo.Server", t => t.ServerId, cascadeDelete: true)

.Index(t => t.ServerId)

.Index(t => t.RamId);

CreateTable(

"dbo.RAM",

c => new

{

Id = c.Int(nullable: false, identity: true),

ManufacturerId = c.Short(nullable: false),

Model = c.String(nullable: false, maxLength: 100),

RamTypeId = c.Byte(nullable: false),

Volume = c.Int(nullable: false),

Price = c.Decimal(nullable: false, storeType: "money"),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.Manufacturer", t => t.ManufacturerId, cascadeDelete: true)

.ForeignKey("dbo.RamType", t => t.RamTypeId, cascadeDelete: true)

.Index(t => t.ManufacturerId)

.Index(t => t.RamTypeId);

CreateTable(

"dbo.RamType",

c => new

{

Id = c.Byte(nullable: false, identity: true),

Name = c.String(nullable: false, maxLength: 50),

})

.PrimaryKey(t => t.Id);

CreateTable(

"dbo.Server\_Storage",

c => new

{

ServerId = c.Int(nullable: false),

StorageId = c.Int(nullable: false),

Count = c.Byte(nullable: false),

})

.PrimaryKey(t => new { t.ServerId, t.StorageId })

.ForeignKey("dbo.Server", t => t.ServerId, cascadeDelete: true)

.ForeignKey("dbo.Strorage", t => t.StorageId)

.Index(t => t.ServerId)

.Index(t => t.StorageId);

CreateTable(

"dbo.Strorage",

c => new

{

Id = c.Int(nullable: false, identity: true),

ManufacturerId = c.Short(nullable: false),

Model = c.String(nullable: false, maxLength: 100),

StrorageInterfaceId = c.Byte(nullable: false),

Volume = c.Int(nullable: false),

Price = c.Decimal(nullable: false, storeType: "money"),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.Manufacturer", t => t.ManufacturerId, cascadeDelete: true)

.ForeignKey("dbo.StrorageInterface", t => t.StrorageInterfaceId, cascadeDelete: true)

.Index(t => t.ManufacturerId)

.Index(t => t.StrorageInterfaceId);

CreateTable(

"dbo.SAN\_Storage",

c => new

{

SAN\_Id = c.Int(nullable: false),

StorageId = c.Int(nullable: false),

Count = c.Short(),

})

.PrimaryKey(t => new { t.SAN\_Id, t.StorageId })

.ForeignKey("dbo.Strorage", t => t.StorageId)

.ForeignKey("dbo.SAN", t => t.SAN\_Id, cascadeDelete: true)

.Index(t => t.SAN\_Id)

.Index(t => t.StorageId);

CreateTable(

"dbo.Service",

c => new

{

Id = c.Int(nullable: false, identity: true),

PaasTypeId = c.Short(nullable: false),

ServiceTypeId = c.Byte(nullable: false),

CoreCount = c.Byte(),

RamCount = c.Double(),

HDDVolume = c.Double(),

CreateDate = c.DateTime(),

DeleteDate = c.DateTime(),

ServiceStateId = c.Byte(nullable: false),

CostPerHour = c.Decimal(storeType: "money"),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.PaasType", t => t.PaasTypeId, cascadeDelete: true)

.ForeignKey("dbo.ServiceState", t => t.ServiceStateId, cascadeDelete: true)

.ForeignKey("dbo.ServiceType", t => t.ServiceTypeId, cascadeDelete: true)

.Index(t => t.PaasTypeId)

.Index(t => t.ServiceTypeId)

.Index(t => t.ServiceStateId);

CreateTable(

"dbo.ServiceIdle",

c => new

{

Id = c.Int(nullable: false, identity: true),

Date = c.DateTime(nullable: false),

ServiceId = c.Int(nullable: false),

BeginPeriod = c.DateTime(nullable: false),

EndPeriod = c.DateTime(nullable: false),

IdleTypeId = c.Byte(nullable: false),

IdleReasonId = c.Byte(nullable: false),

Cost = c.Decimal(storeType: "money"),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.IdleReason", t => t.IdleReasonId, cascadeDelete: true)

.ForeignKey("dbo.IdleType", t => t.IdleTypeId, cascadeDelete: true)

.ForeignKey("dbo.Service", t => t.ServiceId, cascadeDelete: true)

.Index(t => t.ServiceId)

.Index(t => t.IdleTypeId)

.Index(t => t.IdleReasonId);

CreateTable(

"dbo.IdleReason",

c => new

{

Id = c.Byte(nullable: false, identity: true),

Name = c.String(nullable: false, maxLength: 50),

})

.PrimaryKey(t => t.Id);

CreateTable(

"dbo.IdleType",

c => new

{

Id = c.Byte(nullable: false, identity: true),

Name = c.String(nullable: false, maxLength: 50),

})

.PrimaryKey(t => t.Id);

CreateTable(

"dbo.User",

c => new

{

TabNo = c.Int(nullable: false, identity: true),

Login = c.String(nullable: false, maxLength: 50),

Password = c.String(nullable: false, maxLength: 50),

Fio = c.String(maxLength: 100),

RightLevelId = c.Byte(nullable: false),

PositionId = c.Int(nullable: false),

})

.PrimaryKey(t => t.TabNo)

.ForeignKey("dbo.Position", t => t.PositionId, cascadeDelete: true)

.ForeignKey("dbo.RightsLevel", t => t.RightLevelId, cascadeDelete: true)

.Index(t => t.RightLevelId)

.Index(t => t.PositionId);

CreateTable(

"dbo.Position",

c => new

{

Id = c.Int(nullable: false, identity: true),

Name = c.String(nullable: false, maxLength: 100),

AvgSalary = c.Decimal(nullable: false, storeType: "money"),

WorkHours = c.Byte(nullable: false),

})

.PrimaryKey(t => t.Id);

CreateTable(

"dbo.RightsLevel",

c => new

{

Id = c.Byte(nullable: false),

Name = c.String(nullable: false, maxLength: 50),

})

.PrimaryKey(t => t.Id);

CreateTable(

"dbo.ServiceState",

c => new

{

ID = c.Byte(nullable: false, identity: true),

Name = c.String(nullable: false, maxLength: 50),

})

.PrimaryKey(t => t.ID);

CreateTable(

"dbo.ServiceType",

c => new

{

Id = c.Byte(nullable: false, identity: true),

Name = c.String(nullable: false, maxLength: 50),

})

.PrimaryKey(t => t.Id);

CreateTable(

"dbo.Software",

c => new

{

Id = c.Int(nullable: false, identity: true),

Name = c.String(nullable: false, maxLength: 255),

SoftTypeId = c.Byte(nullable: false),

Cost = c.Decimal(nullable: false, storeType: "money"),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.SoftType", t => t.SoftTypeId, cascadeDelete: true)

.Index(t => t.SoftTypeId);

CreateTable(

"dbo.SoftType",

c => new

{

Id = c.Byte(nullable: false, identity: true),

Name = c.String(nullable: false, maxLength: 50),

})

.PrimaryKey(t => t.Id);

CreateTable(

"dbo.SAN\_MaintenanceShedule",

c => new

{

Id = c.Int(nullable: false, identity: true),

SAN\_Id = c.Int(nullable: false),

BeginPeriod = c.DateTime(nullable: false),

EndPeriod = c.DateTime(nullable: false),

})

.PrimaryKey(t => t.Id)

.ForeignKey("dbo.SAN", t => t.SAN\_Id, cascadeDelete: true)

.Index(t => t.SAN\_Id);

CreateTable(

"dbo.SANtoPaas",

c => new

{

PaasTypeId = c.Short(nullable: false),

SAN\_Id = c.Int(nullable: false),

})

.PrimaryKey(t => new { t.PaasTypeId, t.SAN\_Id })

.ForeignKey("dbo.PaasType", t => t.PaasTypeId, cascadeDelete: true)

.ForeignKey("dbo.SAN", t => t.SAN\_Id, cascadeDelete: true)

.Index(t => t.PaasTypeId)

.Index(t => t.SAN\_Id);

CreateTable(

"dbo.ServerToPaas",

c => new

{

PaasTypeId = c.Short(nullable: false),

ServerId = c.Int(nullable: false),

})

.PrimaryKey(t => new { t.PaasTypeId, t.ServerId })

.ForeignKey("dbo.PaasType", t => t.PaasTypeId, cascadeDelete: true)

.ForeignKey("dbo.Server", t => t.ServerId, cascadeDelete: true)

.Index(t => t.PaasTypeId)

.Index(t => t.ServerId);

CreateTable(

"dbo.AdminIdle",

c => new

{

ServiceId = c.Int(nullable: false),

UserTabNo = c.Int(nullable: false),

})

.PrimaryKey(t => new { t.ServiceId, t.UserTabNo })

.ForeignKey("dbo.ServiceIdle", t => t.ServiceId, cascadeDelete: true)

.ForeignKey("dbo.User", t => t.UserTabNo, cascadeDelete: true)

.Index(t => t.ServiceId)

.Index(t => t.UserTabNo);

CreateTable(

"dbo.SoftService",

c => new

{

ServiceId = c.Int(nullable: false),

SoftwareId = c.Int(nullable: false),

})

.PrimaryKey(t => new { t.ServiceId, t.SoftwareId })

.ForeignKey("dbo.Service", t => t.ServiceId, cascadeDelete: true)

.ForeignKey("dbo.Software", t => t.SoftwareId, cascadeDelete: true)

.Index(t => t.ServiceId)

.Index(t => t.SoftwareId);

CreateTable(

"dbo.UsingService",

c => new

{

ServiceId = c.Int(nullable: false),

UserTabNo = c.Int(nullable: false),

})

.PrimaryKey(t => new { t.ServiceId, t.UserTabNo })

.ForeignKey("dbo.Service", t => t.ServiceId, cascadeDelete: true)

.ForeignKey("dbo.User", t => t.UserTabNo, cascadeDelete: true)

.Index(t => t.ServiceId)

.Index(t => t.UserTabNo);

}

/// <summary>

/// Перечень выполняемых операций с сущностями при откате изменений базы данных

/// </summary>

public override void Down()

{

DropForeignKey("dbo.Server", "CPU\_Id", "dbo.CPU");

DropForeignKey("dbo.SAN\_StorageInt", "InterfaceId", "dbo.StrorageInterface");

DropForeignKey("dbo.SAN\_StorageInt", "SAN\_Id", "dbo.SAN");

DropForeignKey("dbo.SAN\_Storage", "SAN\_Id", "dbo.SAN");

DropForeignKey("dbo.SAN\_MaintenanceShedule", "SAN\_Id", "dbo.SAN");

DropForeignKey("dbo.UsingService", "UserTabNo", "dbo.User");

DropForeignKey("dbo.UsingService", "ServiceId", "dbo.Service");

DropForeignKey("dbo.SoftService", "SoftwareId", "dbo.Software");

DropForeignKey("dbo.SoftService", "ServiceId", "dbo.Service");

DropForeignKey("dbo.Software", "SoftTypeId", "dbo.SoftType");

DropForeignKey("dbo.Service", "ServiceTypeId", "dbo.ServiceType");

DropForeignKey("dbo.Service", "ServiceStateId", "dbo.ServiceState");

DropForeignKey("dbo.AdminIdle", "UserTabNo", "dbo.User");

DropForeignKey("dbo.AdminIdle", "ServiceId", "dbo.ServiceIdle");

DropForeignKey("dbo.User", "RightLevelId", "dbo.RightsLevel");

DropForeignKey("dbo.User", "PositionId", "dbo.Position");

DropForeignKey("dbo.ServiceIdle", "ServiceId", "dbo.Service");

DropForeignKey("dbo.ServiceIdle", "IdleTypeId", "dbo.IdleType");

DropForeignKey("dbo.ServiceIdle", "IdleReasonId", "dbo.IdleReason");

DropForeignKey("dbo.Service", "PaasTypeId", "dbo.PaasType");

DropForeignKey("dbo.ServerToPaas", "ServerId", "dbo.Server");

DropForeignKey("dbo.ServerToPaas", "PaasTypeId", "dbo.PaasType");

DropForeignKey("dbo.Strorage", "StrorageInterfaceId", "dbo.StrorageInterface");

DropForeignKey("dbo.Server\_Storage", "StorageId", "dbo.Strorage");

DropForeignKey("dbo.SAN\_Storage", "StorageId", "dbo.Strorage");

DropForeignKey("dbo.Strorage", "ManufacturerId", "dbo.Manufacturer");

DropForeignKey("dbo.Server\_Storage", "ServerId", "dbo.Server");

DropForeignKey("dbo.Server\_RAM", "ServerId", "dbo.Server");

DropForeignKey("dbo.Server\_RAM", "RamId", "dbo.RAM");

DropForeignKey("dbo.RAM", "RamTypeId", "dbo.RamType");

DropForeignKey("dbo.Platform", "RamTypeId", "dbo.RamType");

DropForeignKey("dbo.RAM", "ManufacturerId", "dbo.Manufacturer");

DropForeignKey("dbo.Server\_MaintenanceShedule", "ServerID", "dbo.Server");

DropForeignKey("dbo.Server", "PlatformId", "dbo.Platform");

DropForeignKey("dbo.SANtoPaas", "SAN\_Id", "dbo.SAN");

DropForeignKey("dbo.SANtoPaas", "PaasTypeId", "dbo.PaasType");

DropForeignKey("dbo.SAN", "ManufacturerId", "dbo.Manufacturer");

DropForeignKey("dbo.Platform\_StorageInt", "InterfaceId", "dbo.StrorageInterface");

DropForeignKey("dbo.Platform\_StorageInt", "PlatformId", "dbo.Platform");

DropForeignKey("dbo.Platform", "ManufacturerId", "dbo.Manufacturer");

DropForeignKey("dbo.Platform", "CpuSocketId", "dbo.CpuSocket");

DropForeignKey("dbo.CpuSocket", "ManufacturerId", "dbo.Manufacturer");

DropForeignKey("dbo.CPU", "ManufacturerId", "dbo.Manufacturer");

DropForeignKey("dbo.CPU", "CpuSocketId", "dbo.CpuSocket");

DropIndex("dbo.UsingService", new[] { "UserTabNo" });

DropIndex("dbo.UsingService", new[] { "ServiceId" });

DropIndex("dbo.SoftService", new[] { "SoftwareId" });

DropIndex("dbo.SoftService", new[] { "ServiceId" });

DropIndex("dbo.AdminIdle", new[] { "UserTabNo" });

DropIndex("dbo.AdminIdle", new[] { "ServiceId" });

DropIndex("dbo.ServerToPaas", new[] { "ServerId" });

DropIndex("dbo.ServerToPaas", new[] { "PaasTypeId" });

DropIndex("dbo.SANtoPaas", new[] { "SAN\_Id" });

DropIndex("dbo.SANtoPaas", new[] { "PaasTypeId" });

DropIndex("dbo.SAN\_MaintenanceShedule", new[] { "SAN\_Id" });

DropIndex("dbo.Software", new[] { "SoftTypeId" });

DropIndex("dbo.User", new[] { "PositionId" });

DropIndex("dbo.User", new[] { "RightLevelId" });

DropIndex("dbo.ServiceIdle", new[] { "IdleReasonId" });

DropIndex("dbo.ServiceIdle", new[] { "IdleTypeId" });

DropIndex("dbo.ServiceIdle", new[] { "ServiceId" });

DropIndex("dbo.Service", new[] { "ServiceStateId" });

DropIndex("dbo.Service", new[] { "ServiceTypeId" });

DropIndex("dbo.Service", new[] { "PaasTypeId" });

DropIndex("dbo.SAN\_Storage", new[] { "StorageId" });

DropIndex("dbo.SAN\_Storage", new[] { "SAN\_Id" });

DropIndex("dbo.Strorage", new[] { "StrorageInterfaceId" });

DropIndex("dbo.Strorage", new[] { "ManufacturerId" });

DropIndex("dbo.Server\_Storage", new[] { "StorageId" });

DropIndex("dbo.Server\_Storage", new[] { "ServerId" });

DropIndex("dbo.RAM", new[] { "RamTypeId" });

DropIndex("dbo.RAM", new[] { "ManufacturerId" });

DropIndex("dbo.Server\_RAM", new[] { "RamId" });

DropIndex("dbo.Server\_RAM", new[] { "ServerId" });

DropIndex("dbo.Server\_MaintenanceShedule", new[] { "ServerID" });

DropIndex("dbo.Server", new[] { "CPU\_Id" });

DropIndex("dbo.Server", new[] { "PlatformId" });

DropIndex("dbo.SAN", new[] { "ManufacturerId" });

DropIndex("dbo.SAN\_StorageInt", new[] { "InterfaceId" });

DropIndex("dbo.SAN\_StorageInt", new[] { "SAN\_Id" });

DropIndex("dbo.Platform\_StorageInt", new[] { "InterfaceId" });

DropIndex("dbo.Platform\_StorageInt", new[] { "PlatformId" });

DropIndex("dbo.Platform", new[] { "RamTypeId" });

DropIndex("dbo.Platform", new[] { "CpuSocketId" });

DropIndex("dbo.Platform", new[] { "ManufacturerId" });

DropIndex("dbo.CpuSocket", new[] { "ManufacturerId" });

DropIndex("dbo.CPU", new[] { "CpuSocketId" });

DropIndex("dbo.CPU", new[] { "ManufacturerId" });

DropTable("dbo.UsingService");

DropTable("dbo.SoftService");

DropTable("dbo.AdminIdle");

DropTable("dbo.ServerToPaas");

DropTable("dbo.SANtoPaas");

DropTable("dbo.SAN\_MaintenanceShedule");

DropTable("dbo.SoftType");

DropTable("dbo.Software");

DropTable("dbo.ServiceType");

DropTable("dbo.ServiceState");

DropTable("dbo.RightsLevel");

DropTable("dbo.Position");

DropTable("dbo.User");

DropTable("dbo.IdleType");

DropTable("dbo.IdleReason");

DropTable("dbo.ServiceIdle");

DropTable("dbo.Service");

DropTable("dbo.SAN\_Storage");

DropTable("dbo.Strorage");

DropTable("dbo.Server\_Storage");

DropTable("dbo.RamType");

DropTable("dbo.RAM");

DropTable("dbo.Server\_RAM");

DropTable("dbo.Server\_MaintenanceShedule");

DropTable("dbo.Server");

DropTable("dbo.PaasType");

DropTable("dbo.SAN");

DropTable("dbo.SAN\_StorageInt");

DropTable("dbo.StrorageInterface");

DropTable("dbo.Platform\_StorageInt");

DropTable("dbo.Platform");

DropTable("dbo.Manufacturer");

DropTable("dbo.CpuSocket");

DropTable("dbo.CPU");

}

}

}

Модуль «Configuration.cs»

using MTF\_Services.Model;

namespace MTF\_Services.DataAccess.Migrations

{

using System;

using System.Data.Entity;

using System.Data.Entity.Migrations;

using System.Linq;

/// <summary>

/// Класс конфигурации модели данных

/// </summary>

internal sealed class Configuration : DbMigrationsConfiguration<MTF\_Services.DataAccess.MTF\_ServicesDbContext>

{

/// <summary>

/// Конструктор класса конфигурации модели данных

/// </summary>

public Configuration()

{

AutomaticMigrationsEnabled = true;

}

/// <summary>

/// Выполняется после развертывания последней миграции,

/// обновляя записи в таблицах

/// </summary>

/// <param name="context"></param>

protected override void Seed(MTF\_Services.DataAccess.MTF\_ServicesDbContext context)

{

context.RightsLevel.AddOrUpdate(

rl => rl.Name,

new RightsLevel { Id = 1, Name = "Директор" },

new RightsLevel { Id = 2, Name = "Системныйадминистратор" },

new RightsLevel { Id = 3, Name = "Сотрудник" });

context.Position.AddOrUpdate(

p => p.Name,

new Position

{

Name = "Системный администратор",

AvgSalary = new decimal(54692.50),

WorkHours = 59

},

new Position

{

Name = "Генеральныйдиректор",

AvgSalary = new decimal(125680.90),

WorkHours = 45

},

new Position

{

Name = "Бухгалтер",

AvgSalary = new decimal(75930.00),

WorkHours = 51

});

context.SaveChanges();

context.User.AddOrUpdate(

u => u.Login,

new User

{

Login = "Director",

Password = "123Dir",

RightsLevel = context.RightsLevel.Single(rl => rl.Name.Equals("Директор")),

Position = context.Position.Single(p => p.Name.Equals("Генеральныйдиректор")),

Fio = "Старков Денис Алексеевич"

},

new User

{

Login = "Admin",

Password = "0000",

RightsLevel = context.RightsLevel.Single(rl => rl.Name.Equals("Системныйадминистратор")),

Position = context.Position.Single(p => p.Name.Equals("Системныйадминистратор")),

Fio = "РомановСтаниславВалерьевич"

});

context.Manufacturer.AddOrUpdate(

m => m.Name,

new Manufacturer { Name = "AMD" },

new Manufacturer { Name = "Intel" },

new Manufacturer { Name = "Asus" },

new Manufacturer { Name = "HP" },

new Manufacturer { Name = "WD" },

new Manufacturer { Name = "Kingston" },

new Manufacturer { Name = "Seagate" },

new Manufacturer { Name = "Fujitsu" },

new Manufacturer { Name = "HGST" },

new Manufacturer { Name = "Supermicro" },

new Manufacturer { Name = "Dell" }

);

context.SaveChanges();

var amd = context.Manufacturer.Single(m => m.Name.Equals("AMD"));

var intel = context.Manufacturer.Single(m => m.Name.Equals("Intel"));

context.CpuSocket.AddOrUpdate(

new CpuSocket

{

Manufacturer = intel,

Name = "LGA 1366/Socket B"

},

new CpuSocket

{

Manufacturer = intel,

Name = "LGA 1156/Socket H"

},

new CpuSocket

{

Manufacturer = intel,

Name = "LGA 1567/Socket LS"

},

new CpuSocket

{

Manufacturer = intel,

Name = "LGA 1155/Socket H2"

},

new CpuSocket

{

Manufacturer = intel,

Name = "LGA 2011/Socket R"

},

new CpuSocket

{

Manufacturer = intel,

Name = "LGA 1356/Socket B2"

},

new CpuSocket

{

Manufacturer = intel,

Name = "LGA 3647"

},

new CpuSocket

{

Manufacturer = amd,

Name = "G34"

},

new CpuSocket

{

Manufacturer = amd,

Name = "C32"

},

new CpuSocket

{

Manufacturer = amd,

Name = "SP3"

}

);

context.RamType.AddOrUpdate(

rt => rt.Name,

new RamType { Name = "DDD3" },

new RamType { Name = "DDD4" }

);

context.StrorageInterface.AddOrUpdate(

si => si.Name,

new StrorageInterface { Name = "SATA3" },

new StrorageInterface { Name = "SAS 300" },

new StrorageInterface { Name = "SAS 600" }

);

context.PaasType.AddOrUpdate(

pt => pt.Name,

new PaasType { Name = "1С" },

new PaasType { Name = "Тех. поддержка" },

new PaasType { Name = "SQL" },

new PaasType { Name = "Cloud" },

new PaasType { Name = "Chat" },

new PaasType { Name = "Mail" }

);

context.SoftType.AddOrUpdate(

st => st.Name,

new SoftType { Name = "Системное" },

new SoftType { Name = "Прикладное" },

new SoftType { Name = "Пакетыприкладныхпрограмм" },

new SoftType { Name = "Защитное" },

new SoftType { Name = "Инструментальное" },

new SoftType { Name = "SDK" }

);

context.ServiceState.AddOrUpdate(

ss => ss.Name,

new ServiceState { Name = "Активен" },

new ServiceState { Name = "Неактивен" },

new ServiceState { Name = "Отправленазаявка" },

new ServiceState { Name = "Отклонен" }

);

context.IdleType.AddOrUpdate(

it => it.Name,

new IdleType { Name = "Плановый" },

new IdleType { Name = "Внеплановый" }

);

context.IdleReason.AddOrUpdate(

ir => ir.Name,

new IdleReason { Name = "Резервноекопирование" },

new IdleReason { Name = "Обновление конфигурации" },

new IdleReason { Name = "Сбойпитания" },

new IdleReason { Name = "Выход оборудования из строя" },

new IdleReason { Name = "Человеческий фактор" },

new IdleReason { Name = "Плановое обслуживание" }

);

}

}

}

Модуль «MTF\_ServicesDbContext.cs»

using System.Data.Entity;

using MTF\_Services.Model;

namespace MTF\_Services.DataAccess

{

/// <summary>

/// Класс контекста модели данных

/// </summary>

public class MTF\_ServicesDbContext : DbContext

{

/// <summary>

/// Конструктор класса контекста модели данных

/// </summary>

public MTF\_ServicesDbContext() :base("MTF\_ServicesBase")

{

}

public virtual DbSet<CPU> CPU { get; set; }

public virtual DbSet<CpuSocket> CpuSocket { get; set; }

public virtual DbSet<IdleReason> IdleReason { get; set; }

public virtual DbSet<IdleType> IdleType { get; set; }

public virtual DbSet<Manufacturer> Manufacturer { get; set; }

public virtual DbSet<PaasType> PaasType { get; set; }

public virtual DbSet<Platform> Platform { get; set; }

public virtual DbSet<Platform\_StorageInt> Platform\_StorageInt { get; set; }

public virtual DbSet<Position> Position { get; set; }

public virtual DbSet<RAM> RAM { get; set; }

public virtual DbSet<RamType> RamType { get; set; }

public virtual DbSet<RightsLevel> RightsLevel { get; set; }

public virtual DbSet<SAN> SAN { get; set; }

public virtual DbSet<SAN\_MaintenanceShedule> SAN\_MaintenanceShedule { get; set; }

public virtual DbSet<SAN\_Storage> SAN\_Storage { get; set; }

public virtual DbSet<SAN\_StorageInt> SAN\_StorageInt { get; set; }

public virtual DbSet<Server> Server { get; set; }

public virtual DbSet<Server\_MaintenanceShedule> Server\_MaintenanceShedule { get; set; }

public virtual DbSet<Server\_RAM> Server\_RAM { get; set; }

public virtual DbSet<Server\_Storage> Server\_Storage { get; set; }

public virtual DbSet<Service> Service { get; set; }

public virtual DbSet<ServiceIdle> ServiceIdle { get; set; }

public virtual DbSet<ServiceState> ServiceState { get; set; }

public virtual DbSet<ServiceType> ServiceType { get; set; }

public virtual DbSet<SoftType> SoftType { get; set; }

public virtual DbSet<Software> Software { get; set; }

public virtual DbSet<Strorage> Strorage { get; set; }

public virtual DbSet<StrorageInterface> StrorageInterface { get; set; }

public virtual DbSet<User> User { get; set; }

/// <summary>

/// Вызывается при инициализации модели для производного контекста,

/// в котором определены дополнительные действия при развертывании модели данных

/// </summary>

/// <param name="modelBuilder"></param>

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<CPU>()

.Property(e => e.Price)

.HasPrecision(19, 4);

modelBuilder.Entity<CPU>()

.HasMany(e => e.Server)

.WithRequired(e => e.CPU)

.HasForeignKey(e => e.CPU\_Id);

modelBuilder.Entity<Manufacturer>()

.HasMany(e => e.CpuSocket)

.WithRequired(e => e.Manufacturer)

.WillCascadeOnDelete(false);

modelBuilder.Entity<PaasType>()

.HasMany(e => e.SAN)

.WithMany(e => e.PaasType)

.Map(m => m.ToTable("SANtoPaas").MapLeftKey("PaasTypeId"));

modelBuilder.Entity<PaasType>()

.HasMany(e => e.Server)

.WithMany(e => e.PaasType)

.Map(m => m.ToTable("ServerToPaas").MapLeftKey("PaasTypeId").MapRightKey("ServerId"));

modelBuilder.Entity<Platform>()

.Property(e => e.Price)

.HasPrecision(19, 4);

modelBuilder.Entity<Position>()

.Property(e => e.AvgSalary)

.HasPrecision(19, 4);

modelBuilder.Entity<RAM>()

.Property(e => e.Price)

.HasPrecision(19, 4);

modelBuilder.Entity<RAM>()

.HasMany(e => e.Server\_RAM)

.WithRequired(e => e.RAM)

.WillCascadeOnDelete(false);

modelBuilder.Entity<RightsLevel>()

.HasMany(e => e.User)

.WithRequired(e => e.RightsLevel)

.HasForeignKey(e => e.RightLevelId);

modelBuilder.Entity<SAN>()

.Property(e => e.AnnualMaintenance)

.HasPrecision(19, 4);

modelBuilder.Entity<SAN>()

.Property(e => e.Price)

.HasPrecision(19, 4);

modelBuilder.Entity<SAN>()

.HasMany(e => e.SAN\_MaintenanceShedule)

.WithRequired(e => e.SAN)

.HasForeignKey(e => e.SAN\_Id);

modelBuilder.Entity<SAN>()

.HasMany(e => e.SAN\_Storage)

.WithRequired(e => e.SAN)

.HasForeignKey(e => e.SAN\_Id);

modelBuilder.Entity<SAN>()

.HasMany(e => e.SAN\_StorageInt)

.WithRequired(e => e.SAN)

.HasForeignKey(e => e.SAN\_Id);

modelBuilder.Entity<Server>()

.Property(e => e.Price)

.HasPrecision(19, 4);

modelBuilder.Entity<Server>()

.Property(e => e.AnnualMaintenance)

.HasPrecision(19, 4);

modelBuilder.Entity<Service>()

.Property(e => e.CostPerHour)

.HasPrecision(19, 4);

modelBuilder.Entity<Service>()

.HasMany(e => e.Software)

.WithMany(e => e.Service)

.Map(m => m.ToTable("SoftService").MapLeftKey("ServiceId").MapRightKey("SoftwareId"));

modelBuilder.Entity<Service>()

.HasMany(e => e.User)

.WithMany(e => e.Service)

.Map(m => m.ToTable("UsingService").MapLeftKey("ServiceId").MapRightKey("UserTabNo"));

modelBuilder.Entity<ServiceIdle>()

.Property(e => e.Cost)

.HasPrecision(19, 4);

modelBuilder.Entity<ServiceIdle>()

.HasMany(e => e.User)

.WithMany(e => e.ServiceIdle)

.Map(m => m.ToTable("AdminIdle").MapLeftKey("ServiceId").MapRightKey("UserTabNo"));

modelBuilder.Entity<Software>()

.Property(e => e.Cost)

.HasPrecision(19, 4);

modelBuilder.Entity<Strorage>()

.Property(e => e.Price)

.HasPrecision(19, 4);

modelBuilder.Entity<Strorage>()

.HasMany(e => e.SAN\_Storage)

.WithRequired(e => e.Strorage)

.HasForeignKey(e => e.StorageId)

.WillCascadeOnDelete(false);

modelBuilder.Entity<Strorage>()

.HasMany(e => e.Server\_Storage)

.WithRequired(e => e.Strorage)

.HasForeignKey(e => e.StorageId)

.WillCascadeOnDelete(false);

modelBuilder.Entity<StrorageInterface>()

.HasMany(e => e.Platform\_StorageInt)

.WithRequired(e => e.StrorageInterface)

.HasForeignKey(e => e.InterfaceId);

modelBuilder.Entity<StrorageInterface>()

.HasMany(e => e.SAN\_StorageInt)

.WithRequired(e => e.StrorageInterface)

.HasForeignKey(e => e.InterfaceId);

}

}

}

Модуль «AvalibleInterface.cs»

namespace MTF\_Services.Model.Views

{

public class AvalibleInterface

{

public string Name { get; set; }

public byte Slot\_Count { get; set; }

}

}

Модуль «ConfigurationPartsInfo.cs»

namespace MTF\_Services.Model.Views

{

public class ConfigurationPartsInfo

{

public decimal PlatfromPrice { get; set; }

public decimal OneCPUPrice { get; set; }

public decimal CPUSumPrice { get; set; }

public decimal SelectedRamPrice { get; set; }

public decimal RAMSumPrice { get; set; }

public decimal SelectedStoragePrice { get; set; }

public decimal StorageSumPrice { get; set; }

public decimal TotalPrice { get; set; }

public decimal Maintenance { get; set; }

/// <summary>

/// Обновление итоговой стоимости конфигурации сервера

/// </summary>

public void UpdateTotalPrice()

{

TotalPrice = PlatfromPrice + CPUSumPrice + RAMSumPrice + StorageSumPrice + SelectedStoragePrice;

}

}

}

Модуль «CPUInfo.cs»

namespace MTF\_Services.Model.Views

{

public class CPUInfo

{

public string Manufacturer { get; set; }

public string Model { get; set; }

public double Frequency { get; set; }

public short Core\_Count { get; set; }

public string CpuSocket { get; set; }

public decimal Price { get; set; }

}

}

Модуль «CPUSocketInfo.cs»

namespace MTF\_Services.Model.Views

{

public class CPUSocketInfo

{

public string Manufacturer { get; set; }

public string Socket { get; set; }

}

}

Модуль «EquipmentState.cs»

namespace MTF\_Services.Model.Views

{

public class EquipmentState

{

public int Total { get; set; }

public int Active { get; set; }

public int UnActive { get; set; }

public int OnMaintenace { get; set; }

}

}

Модуль «IdleItem.cs»

using System;

namespace MTF\_Services.Model.Views

{

public class IdleItem

{

public int Id { get; set; }

public string ServiceName { get; set; }

public string PaasName { get; set; }

public DateTime BeginDate { get; set; }

public DateTime EndDate { get; set; }

public string Duration { get; set; }

public string Cost { get; set; }

}

}

Модуль «InstalledRamOnServer.cs»

namespace MTF\_Services.Model.Views

{

public class InstalledRamOnServer

{

public string Manufacturer { get; set; }

public string Model { get; set; }

public int SingleVolume { get; set; }

public decimal Price { get; set; }

public byte Count { get; set; }

}

}

Модуль «InstalledStorageOnSAN.cs»

namespace MTF\_Services.Model.Views

{

public class InstalledStorageOnSAN : InstalledStorageOnServer

{

}

}

Модуль «InstalledStorageOnServer.cs»

namespace MTF\_Services.Model.Views

{

public class InstalledStorageOnServer

{

public string Manufacturer { get; set; }

public string Model { get; set; }

public int Volume { get; set; }

public string Interface { get; set; }

public decimal Price { get; set; }

public byte Count { get; set; }

}

}

Модуль «PaasInfo.cs»

namespace MTF\_Services.Model.Views

{

public class PaasInfo

{

public string Name { get; set; }

public int ServiceCount { get; set; }

public int UserCount { get; set; }

public int ServerCount { get; set; }

public int SANCount { get; set; }

}

}

Модуль «PlatformInfo.cs»

namespace MTF\_Services.Model.Views

{

public class PlatformInfo

{

public string Manufacturer { get; set; }

public string Model { get; set; }

public byte CPU\_Count { get; set; }

public int RamVolume\_Max { get; set; }

public byte RamSocketCount { get; set; }

public decimal Price { get; set; }

public string CPUSocket { get; set; }

public string RAMType { get; set; }

}

}

Модель «PlatformServerItem.cs»

using System.Linq;

namespace MTF\_Services.Model.Views

{

public class PlatformServerItem

{

public int ID { get; set; }

public string Name { get; set; }

/// <summary>

/// Конструктор класса описания сервера платформы

/// </summary>

public PlatformServerItem(Server server)

{

ID = server.Id;

Name = $"{server.Platform.Manufacturer.Name} {server.Platform.Model}; " +

$"{server.CPU.Model} {server.CPU.Manufacturer.Name} {server.CPU.Frequency} Ггцх {server.Platform.CPUCount}; " +

$"{server.Server\_RAM.Sum(sr => sr.RAM.Volume \* sr.Count)} ГБ";

}

}

}

Модуль «PlatformServiceUser.cs»

namespace MTF\_Services.Model.Views

{

public class PlatformServiceUser

{

public string Name { get; set; }

public int ServiceCount { get; set; }

public int UserCount { get; set; }

}

}

Модуль «PlatfromSANItem.cs»

using System.Linq;

namespace MTF\_Services.Model.Views

{

public class PlatfromSANItem

{

public int ID { get; set; }

public string Name { get; set; }

/// <summary>

/// Конструктор класса описания хранилища данных платформы

/// </summary>

public PlatfromSANItem(SAN san)

{

ID = san.Id;

Name = $"{san.Manufacturer.Name} {san.Model}; {san.SAN\_Storage.Sum(ss => ss.Strorage.Volume)} ГБ";

}

}

}

Модуль «RAMInfo.cs»

namespace MTF\_Services.Model.Views

{

public class RAMInfo

{

public string Manufacturer { get; set; }

public string Model { get; set; }

public string RamType { get; set; }

public int Volume { get; set; }

public decimal Price { get; set; }

}

}

Модуль «SAN\_Info.cs»

namespace MTF\_Services.Model.Views

{

public class SAN\_Info

{

public int ID { get; set; }

public string Manufacturer { get; set; }

public string Model { get; set; }

public int StorageCount { get; set; }

public int Volume { get; set; }

public int PaasCount { get; set; }

public int ServiceCount { get; set; }

}

}

Модуль «SANIdleItem.cs»

namespace MTF\_Services.Model.Views

{

public class SANIdleItem

{

public int ID { get; set; }

public string Manufacturer { get; set; }

public string Model { get; set; }

public int StorageCount { get; set; }

public int StorageVolume { get; set; }

public bool OnMaintenance { get; set; }

public bool Active { get; set; }

public int CompleteIdleCount { get; set; }

public int SheduledIdleCount { get; set; }

}

}

Модуль «SANPaasInfo.cs»

namespace MTF\_Services.Model.Views

{

public class SANPaasInfo

{

public int Id { get; set; }

public string Manufacturer { get; set; }

public string Model { get; set; }

public double AvalibleVolume { get; set; }

public double UsedVolume { get; set; }

}

}

Модуль «SANPartsInfo.cs»

namespace MTF\_Services.Model.Views

{

public class SANPartsInfo

{

public decimal InitialPrice { get; set; }

public decimal SelectedStoragePrice { get; set; }

public decimal StorageSumPrice { get; set; }

public decimal TotalPrice { get; set; }

public decimal Maintenance { get; set; }

/// <summary>

/// Обновление итоговой стоимости хранилища данных

/// </summary>

public void UpdateTotalPrice()

{

TotalPrice = InitialPrice + StorageSumPrice;

}

}

}

Модуль «SANPlatformInfo.cs»

namespace MTF\_Services.Model.Views

{

public class SANPlatformInfo

{

public int SummaryVolume { get; set; }

public double AvalibleVolume { get; set; }

}

}

Модуль «ScheduleItem.cs»

using System;

namespace MTF\_Services.Model.Views

{

public class ScheduleItem

{

public int Id { get; set; }

public DateTime BeginDate { get; set; }

public DateTime EndDate { get; set; }

public string Duration { get; set; }

}

}

Модуль «ServerIdleItem.cs»

namespace MTF\_Services.Model.Views

{

public class ServerIdleItem

{

public int ID { get; set; }

public string Plarform { get; set; }

public string CPU { get; set; }

public int RAM { get; set; }

public bool OnMaintenance { get; set; }

public bool Active { get; set; }

public int CompleteIdleCount { get; set; }

public int SheduledIdleCount { get; set; }

}

}

Модуль «ServerInfo.cs»

namespace MTF\_Services.Model.Views

{

public class ServerInfo

{

public int Id { get; set; }

public string Platform { get; set; }

public int CpuCount { get; set; }

public string CPU { get; set; }

public int RamSlotCount { get; set; }

public int RamVolume { get; set; }

public int StorageCount { get; set; }

public int StorageVolume { get; set; }

public int PaasCount { get; set; }

public int ServiceCount { get; set; }

}

}

Модуль «ServerPaasInfo.cs»

namespace MTF\_Services.Model.Views

{

public class ServerPlarformInfo

{

public double SumFrequency { get; set; }

public double FrequencyPerCore { get; set; }

public int CoreCount { get; set; }

public int AvalibleCoreCount { get; set; }

public int SumRAMVolume { get; set; }

public double AvalibleRAMVolume { get; set; }

}

}

Модуль «ServiceDetailInfo.cs»

using System;

namespace MTF\_Services.Model.Views

{

public class ServiceDetailInfo

{

public int ID { get; set; }

public string PaasType { get; set; }

public string ServiceType { get; set; }

public DateTime? Create\_Date { get; set; }

public DateTime? Delete\_Date { get; set; }

public string ServiceState { get; set; }

public decimal CostPerHour { get; set; }

public int UserCount { get; set; }

}

}

Модуль «ServiceIdleItem.cs»

namespace MTF\_Services.Model.Views

{

public class ServiceIdleItem

{

public int ID { get; set; }

public string Name { get; set; }

public string PaasTypeName { get; set; }

public string CurrentState { get; set; }

public int UsersCount { get; set; }

public int CompleteIdleCount { get; set; }

public int SheduledIdleCount { get; set; }

}

}

Модуль «ServiceInfo.cs»

namespace MTF\_Services.Model.Views

{

public class ServiceInfo

{

public string Name { get; set; }

public int Total { get; set; }

public int Active { get; set; }

public int UnActive { get; set; }

public int Rejected { get; set; }

}

}

Модуль «ServicePartsInfo.cs»

using System.Collections.Generic;

using System.Linq;

namespace MTF\_Services.Model.Views

{

public class ServicePartsInfo

{

/// <summary>

/// Cуммарная минимальная стоимость виртуальной машины

/// </summary>

public decimal SummaryVMCost => CalcSummaryVMCost();

/// <summary>

/// Стоимость 1 ГГц процессора в час

/// </summary>

public decimal CPUCostByHour => CalcCPUCostByHour();

/// <summary>

/// Стоимость 1 ГБ оперативной памяти в час

/// </summary>

public decimal RAMCostByHour => CalcRAMCostByHour();

/// <summary>

/// Стоимость 1 ГБ накопителей сервера в час

/// </summary>

public decimal ServerStorageCostByHour => CalcServerStorageCostByHour();

/// <summary>

/// Стоимость 1 ГБ накопителей хранилища данных в час

/// </summary>

public decimal SanStorageCostByHour => CalcSanStorageCostByHour();

/// <summary>

/// Стоимость программного обеспечения в час

/// </summary>

public decimal SoftwareCostByHour => CalcSoftwareCostByHour();

/// <summary>

/// Стомость обслуживания сервера в час

/// </summary>

public decimal ServerMaintenanceByHour => CalcServerMaintenanceByHour();

/// <summary>

/// Стомость обслуживания хранилища данных в час

/// </summary>

public decimal SanMaintenanceByHour => CalcSanMaintenanceByHour();

/// <summary>

/// Стоимость обслуживания оборудования в час

/// </summary>

public decimal TotalMaintenanceByHour => CalcTotalMaintenanceByHour();

#region Рассчет стоимости единицы ресурсов в час

/// <summary>

/// Расчет суммарной минимальной стоимости виртуальной машины

/// </summary>

private decimal CalcSummaryVMCost()

{

return CPUCostByHour + RAMCostByHour + ServerStorageCostByHour + SanStorageCostByHour +

TotalMaintenanceByHour;

}

/// <summary>

/// Расчет стоимости 1 ГГц процессора в час

/// </summary>

public decimal CalcCPUCostByHour()

{

try

{

decimal percentByTotalPrice = 1 / (ServerTotalPrice / CPUPrice);

decimal fiveYearsCost = (ServerTotalPrice / (decimal)CPUFrequency) \* percentByTotalPrice;

return fiveYearsCost / HoursInFiveYears;

}

catch

{

return 0;

}

}

/// <summary>

/// Расчет стоимости 1 ГБ оперативной памяти в час

/// </summary>

public decimal CalcRAMCostByHour()

{

try

{

decimal percentByTotalPrice = 1 / (ServerTotalPrice / RAMPrice);

decimal fiveYearsCost = (ServerTotalPrice / (decimal)ServerRAMVolume) \* percentByTotalPrice;

return fiveYearsCost / HoursInFiveYears;

}

catch

{

return 0;

}

}

/// <summary>

/// Расчет стоимости 1 ГБ накопителей сервера в час

/// </summary>

public decimal CalcServerStorageCostByHour()

{

try

{

decimal percentByTotalPrice = 1 / (ServerTotalPrice / ServerStoragePrice);

decimal fiveYearsCost = (ServerTotalPrice / (decimal)ServerStorageVolume) \* percentByTotalPrice;

return fiveYearsCost / HoursInFiveYears;

}

catch

{

return 0;

}

}

/// <summary>

/// Расчет стоимости 1 ГБ накопителей хранилища данных в час

/// </summary>

public decimal CalcSanStorageCostByHour()

{

try

{

decimal percentByTotalPrice = 1 / (SanTotalPrice / SanStoragePrice);

decimal fiveYearsCost = (SanTotalPrice / (decimal)SanStorageVolume) \* percentByTotalPrice;

return fiveYearsCost / HoursInFiveYears;

}

catch

{

return 0;

}

}

/// <summary>

/// Расчет стоимости программного обеспечения в час

/// </summary>

public decimal CalcSoftwareCostByHour()

{

try

{

decimal percentByTotalPrice = 1 / ((ServerTotalPrice + SanTotalPrice) / SoftwarePrice);

decimal fiveYearsCost = SoftwarePrice \* percentByTotalPrice;

return fiveYearsCost / HoursInFiveYears;

}

catch

{

return 0;

}

}

/// <summary>

/// Расчет стоимости обслуживания сервера в час

/// </summary>

private decimal CalcServerMaintenanceByHour()

{

try

{

decimal percentByTotalPrice = 1 / (ServerTotalPrice / ServerMaintenance);

decimal fiveYearsCost = ServerMaintenance \* percentByTotalPrice;

return fiveYearsCost / HoursInFiveYears;

}

catch

{

return 0;

}

}

/// <summary>

/// Расчет стоимости обслуживания хранилища данных в час

/// </summary>

private decimal CalcSanMaintenanceByHour()

{

try

{

decimal percentByTotalPrice = 1 / (SanTotalPrice / SanMaintenance);

decimal fiveYearsCost = SanMaintenance \* percentByTotalPrice;

return fiveYearsCost / HoursInFiveYears;

}

catch

{

return 0;

}

}

/// <summary>

/// Расчет общей стоимости обслуживания оборудования в час

/// </summary>

private decimal CalcTotalMaintenanceByHour()

{

return ServerMaintenanceByHour + SanMaintenanceByHour + SoftwareCostByHour;

}

#endregion

#region Предварительные рассчеты

/// <summary>

/// Часов в пяти годах

/// </summary>

private const ushort HoursInFiveYears = 43800;

/// <summary>

/// Выбранныйсервер

/// </summary>

public Server Server { get; set; }

/// <summary>

/// Выбранное хранилище данных

/// </summary>

public SAN San { get; set; }

/// <summary>

/// Редактируемыйсервис

/// </summary>

public Service Service { get; set; }

/// <summary>

/// Выбранное программное обеспечение

/// </summary>

public List<SoftwareInfo> SoftwareInfo { get; set; }

#region Сервер

/// <summary>

/// Частота процессора сервера

/// </summary>

public double CPUFrequency => Server?.CPU.Frequency \* CPUCount ?? 0;

/// <summary>

/// Объемоперативнойпамятисервера

/// </summary>

private double ServerRAMVolume => Server?.Server\_RAM.Sum(sr => sr.RAM.Volume \* sr.Count) ?? 0;

/// <summary>

/// Объемнакопителейсервера

/// </summary>

private double ServerStorageVolume => Server?.Server\_Storage.Sum(ss => ss.Strorage.Volume \* ss.Count) ?? 0;

/// <summary>

/// Кол-вопроцессороввплатформе

/// </summary>

public byte CPUCount => Server?.Platform.CPUCount ?? 0;

/// <summary>

/// Исходная стоимость процессора

/// </summary>

private decimal CPUPrice => Server?.CPU.Price ?? 0;

/// <summary>

/// Исходная стоимость оперативной памяти

/// </summary>

private decimal RAMPrice => Server?.Server\_RAM.Sum(r => r.RAM.Price \* r.Count) ?? 0;

/// <summary>

/// Исходнаястоимостьнакопителейсевера

/// </summary>

private decimal ServerStoragePrice => Server?.Server\_Storage.Sum(ss => ss.Strorage.Price \* ss.Count) ?? 0;

/// <summary>

/// Исходнаястоимостьобслуживаниясервера

/// </summary>

private decimal ServerMaintenance => Server.AnnualMaintenance ?? 0;

/// <summary>

/// Стоимостьсервера

/// </summary>

private decimal ServerTotalPrice => Server.Price ?? 0;

#endregion

#region Хранилищеданных

/// <summary>

/// Объем накопителей хранилища данных

/// </summary>

private decimal SanStorageVolume => San?.SAN\_Storage.Sum(ss => ss.Strorage.Volume \* ss.Count.Value) ?? 0;

/// <summary>

/// Исходнаястоимостьхранилищаданных

/// </summary>

private decimal SanTotalPrice => San.Price;

/// <summary>

/// Исходная стоимость накопиетелей хранилища данных

/// </summary>

private decimal SanStoragePrice => San?.SAN\_Storage.Sum(ss => ss.Strorage.Price \* ss.Count.Value) ?? 0;

/// <summary>

/// Исходнаястоимостьобслуживанияхранилищаданных

/// </summary>

private decimal SanMaintenance => San?.AnnualMaintenance ?? 0;

#endregion

#region Программное обеспечение

/// <summary>

/// Исходная стоимость программного обеспечения

/// </summary>

public decimal SoftwarePrice => SoftwareInfo?.Sum(ss => ss.Cost) ?? 0;

#endregion

#endregion

#region Рассчетстоимостиресурсов

/// <summary>

/// Указаннаячастотапроцессора

/// </summary>

public double Frequency { get; set; }

/// <summary>

/// Указанное количество оперативной памяти

/// </summary>

public double RAMVolume { get; set; }

/// <summary>

/// Указанный объем жесткого диска

/// </summary>

public double StorageVolume { get; set; }

/// <summary>

/// Расчет итоговой стоимости сервиса

/// </summary>

private decimal CalculateFinalPrice()

{

try

{

return ((decimal)Frequency \* CPUCostByHour) + ((decimal)Frequency \* RAMCostByHour) + ((decimal)RAMVolume \* SanStorageCostByHour) +

SummaryVMCost;

}

catch

{

return 0;

}

}

/// <summary>

/// Итоговаястоимостьсервиса

/// </summary>

public decimal FinalPrice => CalculateFinalPrice();

#endregion

}

}

Модуль «ServiceRequestItem.cs»

using System;

namespace MTF\_Services.Model.Views

{

public class ServiceRequestItem

{

public int Id { get; set; }

public string PaasTypeName { get; set; }

public string Name { get; set; }

public DateTime CreateDate { get; set; }

public decimal CostPerHour { get; set; }

}

}

Модуль «ServiceSoftInfo.cs»

namespace MTF\_Services.Model.Views

{

public class ServiceSoftInfo

{

public string Type { get; set; }

public string Name { get; set; }

}

}

Модуль «SoftwareInfo.cs»

namespace MTF\_Services.Model.Views

{

public class SoftwareInfo

{

public int Id { get; set; }

public string SoftType { get; set; }

public string Software { get; set; }

public decimal Cost { get; set; }

}

}

Модуль «StorageInfo.cs»

namespace MTF\_Services.Model.Views

{

public class StorageInfo

{

public string Manufacturer { get; set; }

public string Model { get; set; }

public string StrorageInterface { get; set; }

public int Volume { get; set; }

public decimal Price { get; set; }

}

}

Модуль «UserInfo.cs»

namespace MTF\_Services.Model.Views

{

public class UserInfo

{

public int TabNo { get; set; }

public string Login { get; set; }

public string RightsLevel { get; set; }

public string Position { get; set; }

public string Fio { get; set; }

}

}

Модуль «UserService.cs»

namespace MTF\_Services.Model.Views

{

public class UserService

{

public int ServiceTypeId { get; set; }

public string Name { get; set; }

public string PaasName { get; set; }

public string State { get; set; }

}

}

Модуль «CPU.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("CPU")]

public partial class CPU

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public CPU()

{

Server = new HashSet<Server>();

}

public int Id { get; set; }

public short ManufacturerId { get; set; }

[Required]

[StringLength(50)]

public string Model { get; set; }

public byte CpuSocketId { get; set; }

public short CoreCount { get; set; }

[Column(TypeName = "money")]

public decimal Price { get; set; }

public double Frequency { get; set; }

public virtual CpuSocket CpuSocket { get; set; }

public virtual Manufacturer Manufacturer { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Server> Server { get; set; }

}

}

Модуль «CpuSocket.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("CpuSocket")]

public partial class CpuSocket

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public CpuSocket()

{

CPU = new HashSet<CPU>();

Platform = new HashSet<Platform>();

}

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public byte Id { get; set; }

[Required]

[StringLength(50)]

public string Name { get; set; }

public short ManufacturerId { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<CPU> CPU { get; set; }

public virtual Manufacturer Manufacturer { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Platform> Platform { get; set; }

}

}

Модуль «IdleReason.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("IdleReason")]

public partial class IdleReason

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public IdleReason()

{

ServiceIdle = new HashSet<ServiceIdle>();

}

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public byte Id { get; set; }

[Required]

[StringLength(50)]

public string Name { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<ServiceIdle> ServiceIdle { get; set; }

}

}

Модуль «IdleType.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("IdleType")]

public partial class IdleType

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public IdleType()

{

ServiceIdle = new HashSet<ServiceIdle>();

}

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public byte Id { get; set; }

[Required]

[StringLength(50)]

public string Name { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<ServiceIdle> ServiceIdle { get; set; }

}

}

Модуль «Manufacturer.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("Manufacturer")]

public partial class Manufacturer

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Manufacturer()

{

CPU = new HashSet<CPU>();

CpuSocket = new HashSet<CpuSocket>();

Platform = new HashSet<Platform>();

RAM = new HashSet<RAM>();

SAN = new HashSet<SAN>();

Strorage = new HashSet<Strorage>();

}

public short Id { get; set; }

[Required]

[StringLength(100)]

public string Name { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<CPU> CPU { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<CpuSocket> CpuSocket { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Platform> Platform { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<RAM> RAM { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<SAN> SAN { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Strorage> Strorage { get; set; }

}

}

Модуль «PaasType.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("PaasType")]

public partial class PaasType

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public PaasType()

{

Service = new HashSet<Service>();

SAN = new HashSet<SAN>();

Server = new HashSet<Server>();

}

public short Id { get; set; }

[Required]

[StringLength(100)]

public string Name { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Service> Service { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<SAN> SAN { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Server> Server { get; set; }

}

}

Модуль «Platform.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("Platform")]

public partial class Platform

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Platform()

{

Platform\_StorageInt = new HashSet<Platform\_StorageInt>();

Server = new HashSet<Server>();

}

public int Id { get; set; }

public short ManufacturerId { get; set; }

[Required]

[StringLength(100)]

public string Model { get; set; }

public byte CpuSocketId { get; set; }

public byte CPUCount { get; set; }

public byte RamTypeId { get; set; }

public int RamVolumeMax { get; set; }

public byte RamSocketCount { get; set; }

[Column(TypeName = "money")]

public decimal Price { get; set; }

public virtual CpuSocket CpuSocket { get; set; }

public virtual Manufacturer Manufacturer { get; set; }

public virtual RamType RamType { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Platform\_StorageInt> Platform\_StorageInt { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Server> Server { get; set; }

}

}

Модуль «Platform\_StorageInt.cs»

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

public partial class Platform\_StorageInt

{

[Key]

[Column(Order = 0)]

[DatabaseGenerated(DatabaseGeneratedOption.None)]

public int PlatformId { get; set; }

[Key]

[Column(Order = 1)]

public byte InterfaceId { get; set; }

public byte SlotCount { get; set; }

public virtual Platform Platform { get; set; }

public virtual StrorageInterface StrorageInterface { get; set; }

}

}

Модуль «Position.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("Position")]

public partial class Position

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Position()

{

User = new HashSet<User>();

}

public int Id { get; set; }

[Required]

[StringLength(100)]

public string Name { get; set; }

[Column(TypeName = "money")]

public decimal AvgSalary { get; set; }

public byte WorkHours { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<User> User { get; set; }

}

}

Модуль «RAM.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("RAM")]

public partial class RAM

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public RAM()

{

Server\_RAM = new HashSet<Server\_RAM>();

}

public int Id { get; set; }

public short ManufacturerId { get; set; }

[Required]

[StringLength(100)]

public string Model { get; set; }

public byte RamTypeId { get; set; }

public int Volume { get; set; }

[Column(TypeName = "money")]

public decimal Price { get; set; }

public virtual Manufacturer Manufacturer { get; set; }

public virtual RamType RamType { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Server\_RAM> Server\_RAM { get; set; }

}

}

Модуль «RamType.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("RamType")]

public partial class RamType

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public RamType()

{

Platform = new HashSet<Platform>();

RAM = new HashSet<RAM>();

}

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public byte Id { get; set; }

[Required]

[StringLength(50)]

public string Name { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Platform> Platform { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<RAM> RAM { get; set; }

}

}

Модуль «RightsLevel.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("RightsLevel")]

public partial class RightsLevel

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public RightsLevel()

{

User = new HashSet<User>();

}

[Key]

public byte Id { get; set; }

[Required]

[StringLength(50)]

public string Name { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<User> User { get; set; }

}

}

Модуль «SAN.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("SAN")]

public partial class SAN

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public SAN()

{

SAN\_MaintenanceShedule = new HashSet<SAN\_MaintenanceShedule>();

SAN\_Storage = new HashSet<SAN\_Storage>();

SAN\_StorageInt = new HashSet<SAN\_StorageInt>();

PaasType = new HashSet<PaasType>();

}

public int Id { get; set; }

public short ManufacturerId { get; set; }

[Required]

[StringLength(50)]

public string Model { get; set; }

[Column(TypeName = "money")]

public decimal? AnnualMaintenance { get; set; }

[Column(TypeName = "money")]

public decimal Price { get; set; }

public bool Active { get; set; }

public bool OnMaintenance { get; set; }

public virtual Manufacturer Manufacturer { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<SAN\_MaintenanceShedule> SAN\_MaintenanceShedule { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<SAN\_Storage> SAN\_Storage { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<SAN\_StorageInt> SAN\_StorageInt { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<PaasType> PaasType { get; set; }

}

}

Модуль «SAN\_MaintenanceShedule.cs»

using System;

namespace MTF\_Services.Model

{

public partial class SAN\_MaintenanceShedule

{

public int Id { get; set; }

public int SAN\_Id { get; set; }

public DateTime BeginPeriod { get; set; }

public DateTime EndPeriod { get; set; }

public virtual SAN SAN { get; set; }

}

}

Модуль «SAN\_Storage.cs»

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

public partial class SAN\_Storage

{

[Key]

[Column(Order = 0)]

[DatabaseGenerated(DatabaseGeneratedOption.None)]

public int SAN\_Id { get; set; }

[Key]

[Column(Order = 1)]

[DatabaseGenerated(DatabaseGeneratedOption.None)]

public int StorageId { get; set; }

public short? Count { get; set; }

public virtual SAN SAN { get; set; }

public virtual Strorage Strorage { get; set; }

}

}

Модуль «SAN\_StorageInt.cs»

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

public partial class SAN\_StorageInt

{

[Key]

[Column(Order = 0)]

[DatabaseGenerated(DatabaseGeneratedOption.None)]

public int SAN\_Id { get; set; }

[Key]

[Column(Order = 1)]

public byte InterfaceId { get; set; }

public byte SlotCount { get; set; }

public virtual SAN SAN { get; set; }

public virtual StrorageInterface StrorageInterface { get; set; }

}

}

Модуль «Server.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("Server")]

public partial class Server

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Server()

{

Server\_MaintenanceShedule = new HashSet<Server\_MaintenanceShedule>();

Server\_RAM = new HashSet<Server\_RAM>();

Server\_Storage = new HashSet<Server\_Storage>();

PaasType = new HashSet<PaasType>();

}

public int Id { get; set; }

public int PlatformId { get; set; }

public int CPU\_Id { get; set; }

[Column(TypeName = "money")]

public decimal? Price { get; set; }

[Column(TypeName = "money")]

public decimal? AnnualMaintenance { get; set; }

public bool Active { get; set; }

public bool OnMaintenance { get; set; }

public virtual CPU CPU { get; set; }

public virtual Platform Platform { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Server\_MaintenanceShedule> Server\_MaintenanceShedule { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Server\_RAM> Server\_RAM { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Server\_Storage> Server\_Storage { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<PaasType> PaasType { get; set; }

}

}

Модуль «Server\_MaintenanceShedule.cs»

using System;

namespace MTF\_Services.Model

{

public partial class Server\_MaintenanceShedule

{

public int Id { get; set; }

public int ServerID { get; set; }

public DateTime BeginPeriod { get; set; }

public DateTime EndPeriod { get; set; }

public virtual Server Server { get; set; }

}

}

Модуль «Server\_RAM.cs»

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

public partial class Server\_RAM

{

[Key]

[Column(Order = 0)]

[DatabaseGenerated(DatabaseGeneratedOption.None)]

public int ServerId { get; set; }

[Key]

[Column(Order = 1)]

[DatabaseGenerated(DatabaseGeneratedOption.None)]

public int RamId { get; set; }

public byte Count { get; set; }

public virtual RAM RAM { get; set; }

public virtual Server Server { get; set; }

}

}

Модуль «Server\_Storage.cs»

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

public partial class Server\_Storage

{

[Key]

[Column(Order = 0)]

[DatabaseGenerated(DatabaseGeneratedOption.None)]

public int ServerId { get; set; }

[Key]

[Column(Order = 1)]

[DatabaseGenerated(DatabaseGeneratedOption.None)]

public int StorageId { get; set; }

public byte Count { get; set; }

public virtual Server Server { get; set; }

public virtual Strorage Strorage { get; set; }

}

}

Модуль «Service.cs»

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("Service")]

public partial class Service

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Service()

{

ServiceIdle = new HashSet<ServiceIdle>();

Software = new HashSet<Software>();

User = new HashSet<User>();

}

public int Id { get; set; }

public short PaasTypeId { get; set; }

public byte ServiceTypeId { get; set; }

public byte? CoreCount { get; set; }

public double? RamCount { get; set; }

public double? HDDVolume { get; set; }

public DateTime? CreateDate { get; set; }

public DateTime? DeleteDate { get; set; }

public byte ServiceStateId { get; set; }

[Column(TypeName = "money")]

public decimal? CostPerHour { get; set; }

public virtual PaasType PaasType { get; set; }

public virtual ServiceState ServiceState { get; set; }

public virtual ServiceType ServiceType { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<ServiceIdle> ServiceIdle { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Software> Software { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<User> User { get; set; }

}

}

Модуль «ServiceIdle.cs»

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("ServiceIdle")]

public partial class ServiceIdle

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public ServiceIdle()

{

User = new HashSet<User>();

}

public int Id { get; set; }

public DateTime Date { get; set; }

public int ServiceId { get; set; }

public DateTime BeginPeriod { get; set; }

public DateTime EndPeriod { get; set; }

public byte IdleTypeId { get; set; }

public byte IdleReasonId { get; set; }

[Column(TypeName = "money")]

public decimal? Cost { get; set; }

public virtual IdleReason IdleReason { get; set; }

public virtual IdleType IdleType { get; set; }

public virtual Service Service { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<User> User { get; set; }

}

}

Модуль «ServiceState.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("ServiceState")]

public partial class ServiceState

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public ServiceState()

{

Service = new HashSet<Service>();

}

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public byte ID { get; set; }

[Required]

[StringLength(50)]

public string Name { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Service> Service { get; set; }

}

}

Модуль «ServiceType.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("ServiceType")]

public partial class ServiceType

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public ServiceType()

{

Service = new HashSet<Service>();

}

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public byte Id { get; set; }

[Required]

[StringLength(50)]

public string Name { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Service> Service { get; set; }

}

}

Модуль «SoftType.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("SoftType")]

public partial class SoftType

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public SoftType()

{

Software = new HashSet<Software>();

}

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public byte Id { get; set; }

[Required]

[StringLength(50)]

public string Name { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Software> Software { get; set; }

}

}

Модуль «Software.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("Software")]

public partial class Software

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Software()

{

Service = new HashSet<Service>();

}

public int Id { get; set; }

[Required]

[StringLength(255)]

public string Name { get; set; }

public byte SoftTypeId { get; set; }

[Column(TypeName = "money")]

public decimal Cost { get; set; }

public virtual SoftType SoftType { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Service> Service { get; set; }

}

}

Модуль «Strorage.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("Strorage")]

public partial class Strorage

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Strorage()

{

SAN\_Storage = new HashSet<SAN\_Storage>();

Server\_Storage = new HashSet<Server\_Storage>();

}

public int Id { get; set; }

public short ManufacturerId { get; set; }

[Required]

[StringLength(100)]

public string Model { get; set; }

public byte StrorageInterfaceId { get; set; }

public int Volume { get; set; }

[Column(TypeName = "money")]

public decimal Price { get; set; }

public virtual Manufacturer Manufacturer { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<SAN\_Storage> SAN\_Storage { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Server\_Storage> Server\_Storage { get; set; }

public virtual StrorageInterface StrorageInterface { get; set; }

}

}

Модуль «StrorageInterface.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("StrorageInterface")]

public partial class StrorageInterface

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public StrorageInterface()

{

Platform\_StorageInt = new HashSet<Platform\_StorageInt>();

SAN\_StorageInt = new HashSet<SAN\_StorageInt>();

Strorage = new HashSet<Strorage>();

}

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public byte Id { get; set; }

[StringLength(50)]

public string Name { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Platform\_StorageInt> Platform\_StorageInt { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<SAN\_StorageInt> SAN\_StorageInt { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Strorage> Strorage { get; set; }

}

}

Модуль «User.cs»

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MTF\_Services.Model

{

[Table("User")]

public partial class User

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public User()

{

ServiceIdle = new HashSet<ServiceIdle>();

Service = new HashSet<Service>();

}

[Key]

public int TabNo { get; set; }

[Required]

[StringLength(50)]

public string Login { get; set; }

[Required]

[StringLength(50)]

public string Password { get; set; }

[StringLength(100)]

public string Fio { get; set; }

public byte RightLevelId { get; set; }

public int PositionId { get; set; }

public virtual Position Position { get; set; }

public virtual RightsLevel RightsLevel { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<ServiceIdle> ServiceIdle { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Service> Service { get; set; }

}

}

Модуль «Context.cs»

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data.Entity;

using System.Linq;

using System.Threading.Tasks;

using MTF\_Services.DataAccess;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

namespace MTF\_Services.WinForms.Data

{

/// <summary>

/// Контекст для работы с базой данных.

/// </summary>

public class Context

{

/// <summary>

/// Модельданных.

/// </summary>

private static MTF\_ServicesDbContext \_ctx;

/// <summary>

/// Текущийпользователь.

/// </summary>

public static User CurrentUser { get; private set; }

/// <summary>

/// Статический конструктор, в котором происходит инициализация модели данных.

/// </summary>

static Context()

{

\_ctx = new MTF\_ServicesDbContext();

}

/// <summary>

/// Проверка наличия ссылки на контекст данных.

/// </summary>

/// <returns>Наличие ссылки на контекст данных</returns>

public bool TestConnection()

{

return \_ctx != null;

}

/// <summary>

/// Сохранение изменений в текущем контексте (ассинхронный)

/// </summary>

/// <returns>Число затронутых записей</returns>

public async Task<int> SaveChangesAsync()

{

return await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Отмена изменений существующей сущности

/// </summary>

/// <typeparam name="T">Тип измененной сущности</typeparam>

/// <param name="entity">Измененная сущность</param>

/// <returns>Исходный экземпляр сущности</returns>

public T CancelChanges<T>(T entity) where T : class

{

var changedEntity = \_ctx.ChangeTracker.Entries().SingleOrDefault(ent => ent.Entity.Equals(entity));

if (changedEntity != null)

{

changedEntity.CurrentValues.SetValues(changedEntity.OriginalValues);

changedEntity.State = EntityState.Unchanged;

}

return changedEntity?.Entity as T;

}

/// <summary>

/// Сохранение изменений в текущем контексте

/// </summary>

/// <returns>Число затронутых записей</returns>

public int SaveChanges()

{

return \_ctx.SaveChanges();

}

/// <summary>

/// Авторизацияпользователя.

/// </summary>

/// <param name="login">Логин</param>

/// <param name="password">Пароль</param>

/// <returns>Наличие ссылки на объект текущего пользователя</returns>

public async Task<bool> Login(string login, string password)

{

CurrentUser = await \_ctx.User.SingleOrDefaultAsync(u => u.Login.Equals(login) && u.Password.Equals(password));

return CurrentUser != null;

}

/// <summary>

/// Выходизучетнойзаписи.

/// </summary>

public void Logout()

{

CurrentUser = null;

}

#region Views

/// <summary>

/// Получение списка с информацией о платформах.

/// </summary>

/// <returns>Коллекция с информацией о платформах</returns>

public async Task<BindingList<PaasInfo>> GetPaasInfo()

{

var qry = from paasType in \_ctx.PaasType

join service in \_ctx.Service on paasType.Id equals service.PaasTypeId into se

from subServ in se.DefaultIfEmpty()

select new PaasInfo

{

Name = paasType.Name,

SANCount = paasType.SAN.Count,

ServerCount = paasType.Server.Count,

ServiceCount = paasType.Service.Count,

UserCount = subServ.User.Count

};

var paasInfos = await qry.ToListAsync();

var xx = paasInfos.GroupBy(q => q.Name)

.Select(q => q.First())

.ToList();

return new BindingList<PaasInfo>(xx);

}

/// <summary>

/// Получение списка с информацией о сервисах.

/// </summary>

/// <returns>Коллекция с информацией о сервисах</returns>

public BindingList<ServiceInfo> GetServiceInfo()

{

var qry = from serviceType in \_ctx.ServiceType

join service in \_ctx.Service on serviceType.Id equals service.ServiceTypeId into se

from subServ in se.DefaultIfEmpty()

select new ServiceInfo

{

Name = serviceType.Name,

Total = serviceType.Service.Count,

Active = serviceType.Service.Count(x => x.ServiceState.Name == "Активен"),

UnActive = serviceType.Service.Count(x =>

x.ServiceState.Name == "Неактивен" || x.ServiceState.Name == "Отклонен"),

Rejected = serviceType.Service.Count(x => x.ServiceState.Name == "Отклонен")

};

var xx = qry.ToList().GroupBy(q => q.Name)

.Select(q => q.First())

.ToList();

return new BindingList<ServiceInfo>(xx);

}

/// <summary>

/// Получение объекта, который описывает краткое состояние серверов.

/// </summary>

/// <returns>Краткое описание состояния серверов</returns>

public async Task<EquipmentState> GetServersEquipmentState()

{

var es = new EquipmentState();

var servers = await \_ctx.Server.ToListAsync();

es.Total = servers.Count;

foreach (var serv in servers)

{

if (serv.Active)

es.Active++;

else

es.UnActive++;

if (serv.OnMaintenance)

es.OnMaintenace++;

}

return es;

}

/// <summary>

/// Получение объекта, который описывает краткое состояние систем хранения данных.

/// </summary>

/// <returns>Краткое описание состояния систем хранения данных</returns>

public async Task<EquipmentState> GetSanEquipmentState()

{

var es = new EquipmentState();

var sans = await \_ctx.SAN.ToListAsync();

es.Total = sans.Count;

foreach (var s in sans)

{

if (s.Active)

es.Active++;

else

es.UnActive++;

if (s.OnMaintenance)

es.OnMaintenace++;

}

return es;

}

/// <summary>

/// Получение списка краткого описания конфигураций серверов

/// </summary>

/// <returns>Коллекция с кратким описнием конфигураций серверов</returns>

public BindingList<ServerInfo> GetServerInfo()

{

var qry = from serv in \_ctx.Server

join cpu in \_ctx.CPU on serv.CPU\_Id equals cpu.Id into subCpu

from cp in subCpu.DefaultIfEmpty()

select new ServerInfo

{

Id = serv.Id,

CPU = cp.Manufacturer.Name + " " + cp.Model,

CpuCount = cp.CoreCount \* serv.Platform.CPUCount,

RamSlotCount = serv.Server\_RAM.Sum(sr => sr.Count),

RamVolume = serv.Server\_RAM.Sum(sr => sr.Count \* sr.RAM.Volume),

StorageCount = serv.Server\_Storage.Sum(ss => ss.Count),

StorageVolume = serv.Server\_Storage.Sum(ss => ss.Count \* ss.Strorage.Volume),

Platform = serv.Platform.Manufacturer.Name + " " + serv.Platform.Model,

ServiceCount = serv.PaasType.Count > 0 ? serv.PaasType.Select(x => x.Service.Count).Sum() : 0,

PaasCount = serv.PaasType.Count

};

return new BindingList<ServerInfo>(qry.ToList());

}

/// <summary>

/// Получение списка краткого описания конфигураций систем хранения данных

/// </summary>

/// <returns>Коллекция с кратким описнием конфигураций систем хранения данных</returns>

public BindingList<SAN\_Info> GetSanInfo()

{

var qry = from san in \_ctx.SAN

select new SAN\_Info

{

ID = san.Id,

Manufacturer = san.Manufacturer.Name,

Model = san.Model,

StorageCount = san.SAN\_Storage.Sum(ss => ss.Count.Value),

Volume = san.SAN\_Storage.Sum(ss => ss.Strorage.Volume \* ss.Count.Value),

PaasCount = san.PaasType.Count,

ServiceCount = san.PaasType.Count > 0 ? san.PaasType.Select(x => x.Service.Count).Sum() : 0

};

return new BindingList<SAN\_Info>(qry.ToList());

}

#endregion

#region SysAdmin\_statusInfo

/// <summary>

/// Получение количества используемых платформ.

/// </summary>

/// <param name="paasInfos">Список с информацией о платформах</param>

/// <returns>Количество используемых платформ</returns>

public int UsedPlatformsCount(BindingList<PaasInfo> paasInfos)

{

return paasInfos.Count(pi => pi.ServiceCount > 0);

}

/// <summary>

/// Получение количества используемых сервисов.

/// </summary>

/// <param name="serviceInfos">Список с информацией о предоставляемых сервисах</param>

/// <returns>Количество используемых сервисов</returns>

public int UsedServiceCount(BindingList<ServiceInfo> serviceInfos)

{

return serviceInfos.Count;

}

/// <summary>

/// Получение количества используемых конфигураций сервисов.

/// </summary>

/// <param name="serviceInfos">Список с информацией о предоставляемых сервисах</param>

/// <returns>Количество используемых конфигураций сервисов</returns>

public int ServiceConfigsCount(BindingList<ServiceInfo> serviceInfos)

{

return serviceInfos.Sum(si => si.Active);

}

/// <summary>

/// Проверка наличия платформ, на которые распределены ресурсы

/// </summary>

/// <returns>Количествоплатформ</returns>

public int CheckActivePaas()

{

return \_ctx.PaasType.Count(p => p.Server.Count > 0 && p.SAN.Count > 0);

}

/// <summary>

/// Получение экземпляра типа сервиса по его наименованию

/// </summary>

/// <param name="name">Наименованиетипасервиса</param>

/// <returns>Экземпляртипасервиса</returns>

public async Task<ServiceType> GetPaasTypeByName(string name)

{

return await \_ctx.ServiceType.SingleAsync(st => st.Name.Equals(name));

}

/// <summary>

/// Удаление выбранного типа сервиса

/// </summary>

/// <param name="serviceTypeToDel">Тип сервиса, который следует удалить</param>

public async Task DeleteServiceType(ServiceType serviceTypeToDel)

{

\_ctx.ServiceType.Remove(serviceTypeToDel);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Удаление выбранного типа платформы

/// </summary>

/// <param name="paasTypeToDel">Тип платформы, который следует удалить</param>

public async Task DeletePaasType(PaasType paasTypeToDel)

{

\_ctx.PaasType.Remove(paasTypeToDel);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Проверка наличие экземпляров конфигураций серверов и хранилищ данных

/// </summary>

public bool CheckInfrastructureToCreatePlatform()

{

return \_ctx.Server.Any() && \_ctx.SAN.Any();

}

#endregion

#region ServerConfiguration

/// <summary>

/// Удалениевыбраннойконфигурациисервера

/// </summary>

/// <param name="serverToDel">Конфигурация сервера, которую следует удалить</param>

public async Task DeleteServer(Server serverToDel)

{

\_ctx.Server.Remove(serverToDel);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Получение списка платформ

/// </summary>

/// <returns>Списокплатформ</returns>

public BindingList<Platform> GetPlatformsList()

{

return new BindingList<Platform>(\_ctx.Platform.ToList());

}

/// <summary>

/// Получение списка процессоров по их разъему

/// </summary>

/// <param name="socketName">Разъемпроцессора</param>

/// <returns>Список доступных процессоров по указанному разъему</returns>

public BindingList<CPU> GetCpusOfSocket(string socketName)

{

var qry = from cpu in \_ctx.CPU

where cpu.CpuSocket.Name.Equals(socketName)

select cpu;

return new BindingList<CPU>(qry.ToList());

}

/// <summary>

/// Получение списка памяти по ее типу.

/// </summary>

/// <param name="ramType">Типпамяти</param>

/// <returns>Спискок доступной памяти по указанному типу</returns>

public BindingList<RAM> GetRamsOfRamType(string ramType)

{

var qry = from ram in \_ctx.RAM

where ram.RamType.Name.Equals(ramType)

select ram;

return new BindingList<RAM>(qry.ToList());

}

/// <summary>

/// Получение списка доступных накопителей по указанным интерфейсам

/// </summary>

/// <param name="avalibleInterfaces">Списокдоступныхинтерфейсов</param>

/// <returns>Список доступных накопителей по укзанным интерфейсам</returns>

public BindingList<Strorage> GetStoragesByAvaliblePlatformInterfaces(ICollection<Platform\_StorageInt> avalibleInterfaces)

{

var avInt = avalibleInterfaces.ToList();

var listOfAvalibleStorages =

\_ctx.Strorage.ToList().Where(s => avInt.Exists(nterf => nterf.StrorageInterface.Equals(s.StrorageInterface)));

return new BindingList<Strorage>(listOfAvalibleStorages.ToList());

}

/// <summary>

/// Получение преобразованного списка доступных интерфейсов накопителей для платформы

/// </summary>

/// <param name="storageInterfaces">Интерфейсынакопителейплатформы</param>

/// <returns>Преобразованный список доступных интерфейсов накопителей</returns>

public BindingList<AvalibleInterface> GetAvalibleInterfacesOfPlarformBS(ICollection<Platform\_StorageInt> storageInterfaces)

{

return new BindingList<AvalibleInterface>(GetAvalibleInterfacesOfPlarform(storageInterfaces));

}

/// <summary>

/// Получение обратно преобразованного списка доступных интерфейсов накопителей для платформы

/// </summary>

/// <param name="platform">Платформа</param>

/// <param name="avalibleInterfaces">Преобразованный список доступных интерфейсов накопителей</param>

/// <returns>Обработно преобразованный список доступных интерфейсов накопителей</returns>

public List<Platform\_StorageInt> GetPlatformStorageIntFromAvalible(Platform platform,

List<AvalibleInterface> avalibleInterfaces)

{

var plStInt = new List<Platform\_StorageInt>();

avalibleInterfaces.ForEach(ai => plStInt.Add(new Platform\_StorageInt

{

StrorageInterface = GetInterfaceIdByName(ai.Name),

Platform = platform,

SlotCount = ai.Slot\_Count

}));

return plStInt;

}

/// <summary>

/// Получение кода интерфейса по его наименованию

/// </summary>

/// <param name="intName">Наименованиеинтерфейса</param>

/// <returns>Кодинтерфейса</returns>

private StrorageInterface GetInterfaceIdByName(string intName)

{

return \_ctx.StrorageInterface.Single(si => si.Name.Equals(intName));

}

/// <summary>

/// Получение списка доступных интерфейсов выбранной платформы

/// </summary>

private List<AvalibleInterface> GetAvalibleInterfacesOfPlarform(ICollection<Platform\_StorageInt> storageInterfaces)

{

var storInt = storageInterfaces.ToList();

var avInt = new List<AvalibleInterface>();

storInt.ForEach(si => avInt.Add(new AvalibleInterface

{

Name = si.StrorageInterface.Name,

Slot\_Count = si.SlotCount

}));

return avInt;

}

/// <summary>

/// Сохранение новой конфигурации сервера

/// </summary>

/// <param name="newServer">Новаяконфигурациясервера</param>

public async Task AddNewServerConfiguration(Server newServer)

{

\_ctx.Server.Add(newServer);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Сохранение изменений в редактируемой сервера

/// </summary>

/// <param name="editedServer">Редактируемуясервера</param>

public async Task EditServerConfiguration(Server editedServer)

{

\_ctx.Server.Attach(editedServer);

\_ctx.Entry(editedServer).State = EntityState.Modified;

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Получение ссылки на экземпляр сервера по его подробной информации

/// </summary>

/// <param name="selectedServer">Подробная информация о сервере</param>

/// <returns>Ссылка на экземпляр сервера</returns>

public async Task<Server> GetServerByServerInfo(ServerInfo selectedServer)

{

return await \_ctx.Server

.Include(s => s.Server\_RAM)

.Include(s => s.Server\_Storage)

.Include(s => s.CPU)

.Include(s => s.Platform)

.SingleAsync(s => s.Id == selectedServer.Id);

}

/// <summary>

/// Получение экземпляра текущей платформы с коллекцией поддерживаемых интерфейсов

/// </summary>

/// <param name="currentPlatform">Текущаяплатформа</param>

/// <returns>Экземпляр текущей платформы с коллекцией поддерживаемых интерфейсов</returns>

public Platform GetPlatformInclude(Platform currentPlatform)

{

return \_ctx.Platform

.Include(p => p.Platform\_StorageInt)

.Single(p => p.Id == currentPlatform.Id);

}

#endregion

#region SelectPlatform

/// <summary>

/// Получениеспискаплатформсподробнымописанием.

/// </summary>

/// <returns>Список платформ с подробным описанием</returns>

public BindingList<PlatformInfo> GetPlatformsInfo()

{

var qry = from platform in \_ctx.Platform

select new PlatformInfo

{

Model = platform.Model,

Manufacturer = platform.Manufacturer.Name,

CPUSocket = platform.CpuSocket.Name,

CPU\_Count = platform.CPUCount,

RAMType = platform.RamType.Name,

RamSocketCount = platform.RamSocketCount,

RamVolume\_Max = platform.RamVolumeMax,

Price = platform.Price

};

return new BindingList<PlatformInfo>(qry.ToList());

}

/// <summary>

/// Получение экземпляра платформы из подробного описания платформы

/// </summary>

/// <param name="selectedPlatformInfo">Подробноеописаниеплатформы</param>

/// <returns>Экземплярплатформы</returns>

public Platform GetPlatformByPlatformInfo(PlatformInfo selectedPlatformInfo)

{

return \_ctx.Platform

.Include(pl => pl.Platform\_StorageInt)

.Include(pl => pl.Manufacturer)

.Include(pl => pl.CpuSocket)

.Include(pl => pl.RamType)

.SingleOrDefault(p =>

p.Model.Equals(selectedPlatformInfo.Model) &&

p.Manufacturer.Name.Equals(selectedPlatformInfo.Manufacturer));

}

/// <summary>

/// Удаление выбранной платформы

/// </summary>

/// <param name="platformToDel">Платформа, которую следует удалить</param>

public async Task DeletePlatform(Platform platformToDel)

{

\_ctx.Platform.Remove(platformToDel);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Получение подробного описания платформы по ее экземпляру

/// </summary>

/// <param name="platform">Экземплярплатформы</param>

/// <returns>Подробное описание платформы</returns>

public PlatformInfo GetPlatformInfoByPlatform(Platform platform)

{

return new PlatformInfo

{

Model = platform.Model,

Manufacturer = platform.Manufacturer.Name,

CPUSocket = platform.CpuSocket.Name,

CPU\_Count = platform.CPUCount,

RAMType = platform.RamType.Name,

RamSocketCount = platform.RamSocketCount,

RamVolume\_Max = platform.RamVolumeMax,

Price = platform.Price

};

}

#endregion

#region Manufacturers

/// <summary>

/// Получение списка производителей с привязкой

/// </summary>

/// <returns>Списокпроизводителей</returns>

public BindingList<Manufacturer> GetManufacturers()

{

\_ctx.Manufacturer.Load();

return \_ctx.Manufacturer.Local.ToBindingList();

}

/// <summary>

/// Проверка дублирования наименования производителя

/// </summary>

/// <param name="name">Наименование производителя</param>

/// <returns>Наличие производителя с таким наименованием</returns>

public async Task<bool> CheckManufacturerForDublicate(string name)

{

return await \_ctx.Manufacturer.SingleOrDefaultAsync(m => m.Name.ToUpper().Equals(name.ToUpper())) != null;

}

#endregion

#region CPU\_Sockets

/// <summary>

/// Получение списка разъемов процессора с наименованием производителя

/// </summary>

/// <returns>Списиокразъемовпроцессора</returns>

public BindingList<CPUSocketInfo> GetCPUSocketsInfo()

{

var qry = from cpuSocket in \_ctx.CpuSocket

select new CPUSocketInfo

{

Manufacturer = cpuSocket.Manufacturer.Name,

Socket = cpuSocket.Name

};

return new BindingList<CPUSocketInfo>(qry.ToList());

}

/// <summary>

/// Получение разъема процессора по его информации

/// </summary>

/// <param name="selectedItem">Информацияоразъеме</param>

/// <returns>Разъемпроцессора</returns>

public async Task<CpuSocket> GetCPUSocketByInfo(CPUSocketInfo selectedItem)

{

return await \_ctx.CpuSocket.SingleAsync(cs =>

cs.Name.Equals(selectedItem.Socket) && cs.Manufacturer.Name.Equals(selectedItem.Manufacturer));

}

/// <summary>

/// Проверка дублирования разъема процессора

/// </summary>

/// <param name="name">Наименованиеразъемв</param>

/// <param name="manufacturerName">Наименованиепроизводителя</param>

/// <returns>Наличие разъема у производителя с таким наименованием</returns>

public async Task<bool> CheckCpuSocketForDublicate(string name, string manufacturerName)

{

return await \_ctx.CpuSocket.SingleOrDefaultAsync(cs =>

cs.Name.ToUpper().Equals(name.ToUpper()) && cs.Manufacturer.Name.Equals(manufacturerName)) != null;

}

/// <summary>

/// Добавление нового разъема процессора

/// </summary>

/// <param name="newSocket">Новыйразъемпроцессора</param>

public async Task AddNewCPUSocket(CpuSocket newSocket)

{

\_ctx.CpuSocket.Add(newSocket);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Редактирование существующего разъема процессора

/// </summary>

/// <param name="newSocket">Существующий разъем процессора</param>

public async Task EditCPUSocket(CpuSocket editedSocket)

{

\_ctx.CpuSocket.Attach(editedSocket);

\_ctx.Entry(editedSocket).State = EntityState.Modified;

await \_ctx.SaveChangesAsync();

}

#endregion

#region EditPlatform

/// <summary>

/// Получение списка разъемов процессоров

/// </summary>

/// <returns>Список разъемов процессоров</returns>

public BindingList<CpuSocket> GetCPUSockets()

{

return new BindingList<CpuSocket>(\_ctx.CpuSocket.ToList());

}

/// <summary>

/// Получение списка типов ОЗУ

/// </summary>

/// <returns>СписоктиповОЗУ</returns>

public BindingList<RamType> GetRamsTypes()

{

return new BindingList<RamType>(\_ctx.RamType.ToList());

}

/// <summary>

/// Получение списка интерфейсов храненения

/// </summary>

/// <returns>Список интерфейсов храненения</returns>

public BindingList<StrorageInterface> GetStorageInterfaces()

{

return new BindingList<StrorageInterface>(\_ctx.StrorageInterface.ToList());

}

/// <summary>

/// Проверка дублирования платформы

/// </summary>

/// <param name="model">Модель</param>

/// <param name="manufacturer">Производитель</param>

/// <returns>Наличие такой модели платформы у производителя</returns>

public async Task<bool> CheckPlatformForDublicate(string model, string manufacturer)

{

return await \_ctx.Platform.SingleOrDefaultAsync(pl =>

pl.Model.ToUpper().Equals(model.ToUpper()) && pl.Manufacturer.Name.Equals(manufacturer)) != null;

}

/// <summary>

/// Добавление новой платформы

/// </summary>

/// <param name="newPlatform">Новаяплатформа</param>

public async Task AddNewPlatform(Platform newPlatform)

{

\_ctx.Platform.Add(newPlatform);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Редактирование существующей платформы

/// </summary>

/// <param name="editedPlatform">Существующаяплатформа</param>

public async Task EditPlatform(Platform editedPlatform)

{

\_ctx.Platform.Attach(editedPlatform);

\_ctx.Entry(editedPlatform).State = EntityState.Modified;

await \_ctx.SaveChangesAsync();

}

#endregion

#region EditUsers

/// <summary>

/// Получение списка уровней привелегий пользователя

/// </summary>

/// <returns>Список уровней привелегий пользователя</returns>

public BindingList<RightsLevel> GetRightsLevels()

{

return new BindingList<RightsLevel>(\_ctx.RightsLevel.ToList());

}

/// <summary>

/// Получение списка должностей

/// </summary>

/// <returns>Списокдолжностей</returns>

public BindingList<Position> GetPositions()

{

\_ctx.Position.Load();

return \_ctx.Position.Local.ToBindingList();

}

/// <summary>

/// Проверка дублирования логина пользователя

/// </summary>

/// <param name="login">Новыйлогин</param>

/// <returns>Наличие уже существующего логина</returns>

public async Task<bool> CheckUserLoginForDublicate(string login)

{

return await \_ctx.User.SingleOrDefaultAsync(u => u.Login.ToUpper().Equals(login.ToUpper())) != null;

}

/// <summary>

/// Добавление нового пользователя

/// </summary>

/// <param name="user">Новыйпользователь</param>

public async Task AddNewUser(User user)

{

\_ctx.User.Add(user);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Редактирование пользователя

/// </summary>

/// <param name="user">Новый пользователь</param>

public async Task EditUser(User user)

{

\_ctx.User.Attach(user);

\_ctx.Entry(user).State = EntityState.Modified;

await \_ctx.SaveChangesAsync();

}

#endregion

#region EditPosition

/// <summary>

/// Проверка дублирования наименования должности

/// </summary>

/// <param name="currentPositionName">Наименованиедолжности</param>

/// <returns>Наличие должности с таким наименованием</returns>

public async Task<bool> CheckPositionForDublicate(string positionName)

{

return await \_ctx.Position.SingleOrDefaultAsync(p => p.Name.ToUpper().Equals(positionName.ToUpper())) != null;

}

#endregion

#region UserList

/// <summary>

/// Получение списка пользователей с подробным описанием

/// </summary>

/// <returns>Список пользователей с подробным описанием</returns>

public BindingList<UserInfo> GetUsersInfo()

{

var usr = \_ctx.User

.Include(u => u.Position)

.Include(u => u.RightsLevel)

.ToList();

var usrInfo = new List<UserInfo>();

usr.ForEach(u => usrInfo.Add(new UserInfo

{

TabNo = u.TabNo,

Login = u.Login,

Position = u.Position.Name,

RightsLevel = u.RightsLevel.Name,

Fio = u.Fio

}));

return new BindingList<UserInfo>(usrInfo);

}

/// <summary>

/// Получение ссылки на пользователя по его подробному описанию

/// </summary>

/// <param name="currentUser">Текущая информация о пользователе</param>

/// <returns>Ссылка на пользователя</returns>

public async Task<User> GetUserByUserInfo(UserInfo currentUser)

{

if (currentUser == null)

return null;

return await \_ctx.User.SingleOrDefaultAsync(u => u.TabNo == currentUser.TabNo);

}

#endregion

#region SelectStorage

/// <summary>

/// Получение списка накопителей с подробным описанием

/// </summary>

/// <returns>Список накопителей с подробным описанием</returns>

public BindingList<StorageInfo> GetStoragesInfo()

{

var stor = from strorage in \_ctx.Strorage

select new StorageInfo

{

Manufacturer = strorage.Manufacturer.Name,

Model = strorage.Model,

Price = strorage.Price,

StrorageInterface = strorage.StrorageInterface.Name,

Volume = strorage.Volume

};

return new BindingList<StorageInfo>(stor.ToList());

}

/// <summary>

/// Получение ссылки на накопитель по его подробной информации

/// </summary>

/// <param name="currentStorage">Информацияобнакопителе</param>

/// <returns>Ссылка на накопитель</returns>

public async Task<Strorage> GetStorageByStorageInfo(StorageInfo currentStorage)

{

if (currentStorage == null)

return null;

return await \_ctx.Strorage.SingleOrDefaultAsync(s =>

s.Model.Equals(currentStorage.Model) && s.Manufacturer.Name.Equals(currentStorage.Manufacturer));

}

/// <summary>

/// Удаление выбранного накопителя

/// </summary>

/// <param name="strorageToDel">Накопитель, который следует удалить</param>

public async Task DeleteStrorage(Strorage strorageToDel)

{

\_ctx.Strorage.Remove(strorageToDel);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Получение экземпляра накопителя по ее модели и производителю

/// </summary>

/// <param name="model">Наименованиемодели</param>

/// <param name="manufacturer">Наименованиепроизводителя</param>

/// <returns>Экземплярнакопителя</returns>

public Strorage GetStorageByModelManufacturer(string model, string manufacturer)

{

return \_ctx.Strorage.Single(r => r.Model.Equals(model) && r.Manufacturer.Name.Equals(manufacturer));

}

/// <summary>

/// Удаление выбранного пользователя

/// </summary>

/// <param name="userToDel">Пользователь, которого следует удалить</param>

public async Task DeleteUser(User userToDel)

{

\_ctx.User.Remove(userToDel);

await \_ctx.SaveChangesAsync();

}

#endregion

#region EditStorage

/// <summary>

/// Проверкадублированиянакопителя

/// </summary>

/// <param name="model">Модель</param>

/// <param name="manufacturer">Производитель</param>

/// <returns>Наличие существующего накопителя с такой моделью и производителем</returns>

public async Task<bool> CheckStorageForDublicate(string model, string manufacturer)

{

return await \_ctx.Strorage.SingleOrDefaultAsync(s =>

s.Model.ToUpper().Equals(model.ToUpper()) && s.Manufacturer.Name.Equals(manufacturer)) != null;

}

/// <summary>

/// Добавление нового накопителя

/// </summary>

/// <param name="newStorage">Новыйнакопитель</param>

public async Task AddNewStorage(Strorage newStorage)

{

\_ctx.Strorage.Add(newStorage);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Редактирование существующего накопителя

/// </summary>

/// <param name="editedStrorage">Существующийнакопитель</param>

public async Task EditStorage(Strorage editedStrorage)

{

\_ctx.Strorage.Attach(editedStrorage);

\_ctx.Entry(editedStrorage).State = EntityState.Modified;

await \_ctx.SaveChangesAsync();

}

#endregion

#region StorageInterfaces

/// <summary>

/// Получение списка интерфейсов накопителей с привязкой

/// </summary>

/// <returns>Список интерфейсов накопителей</returns>

public BindingList<StrorageInterface> GetStorageInterfacesBS()

{

\_ctx.StrorageInterface.Load();

return \_ctx.StrorageInterface.Local.ToBindingList();

}

/// <summary>

/// Проверка дублирования наименования интерфейса накопителя

/// </summary>

/// <param name="name">Наименование интерфейса накопителя</param>

/// <returns>Наличие интерфейса накопителя с таким наименованием</returns>

public async Task<bool> CheckStorageInterfaceForDublicate(string name)

{

return await \_ctx.StrorageInterface.SingleOrDefaultAsync(m => m.Name.ToUpper().Equals(name.ToUpper())) != null;

}

#endregion

#region SelectCPU

/// <summary>

/// Получение списка процессоров с подробным описанием

/// </summary>

/// <returns>Список процессоров с подробным описанием</returns>

public BindingList<CPUInfo> GetCPUsInfo()

{

var cp = from cpu in \_ctx.CPU

select new CPUInfo

{

Model = cpu.Model,

Manufacturer = cpu.Manufacturer.Name,

CpuSocket = cpu.CpuSocket.Name,

Core\_Count = cpu.CoreCount,

Price = cpu.Price,

Frequency = cpu.Frequency

};

return new BindingList<CPUInfo>(cp.ToList());

}

/// <summary>

/// Получение ссылки на процессор по его подробной информации

/// </summary>

/// <param name="currentCPU">Информацияобпроцессоре</param>

/// <returns>Ссылка на процессор</returns>

public async Task<CPU> GetCPUByCPUInfo(CPUInfo currentCPU)

{

if (currentCPU == null)

return null;

return await \_ctx.CPU.SingleOrDefaultAsync(s =>

s.Model.Equals(currentCPU.Model) && s.Manufacturer.Name.Equals(currentCPU.Manufacturer));

}

/// <summary>

/// Удаление выбранного процессора

/// </summary>

/// <param name="cpuToDel">Процессор, который следует удалить</param>

public async Task DeleteCPU(CPU cpuToDel)

{

\_ctx.CPU.Remove(cpuToDel);

await \_ctx.SaveChangesAsync();

}

#endregion

#region EditCPU

/// <summary>

/// Проверкадублированияпроцессора

/// </summary>

/// <param name="model">Модель</param>

/// <param name="manufacturer">Производитель</param>

/// <returns>Наличие существующего процессора с такой моделью и производителем</returns>

public async Task<bool> CheckCPUForDublicate(string model, string manufacturer)

{

return await \_ctx.CPU.SingleOrDefaultAsync(s =>

s.Model.ToUpper().Equals(model.ToUpper()) && s.Manufacturer.Name.Equals(manufacturer)) != null;

}

/// <summary>

/// Добавление нового процессора

/// </summary>

/// <param name="newCPU">Новыйпроцессор</param>

public async Task AddNewCPU(CPU newCPU)

{

\_ctx.CPU.Add(newCPU);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Редактирование существующего процессора

/// </summary>

/// <param name="editedCPU">Существующийпроцессор</param>

public async Task EditCPU(CPU editedCPU)

{

\_ctx.CPU.Attach(editedCPU);

\_ctx.Entry(editedCPU).State = EntityState.Modified;

await \_ctx.SaveChangesAsync();

}

#endregion

#region SelectRAM

/// <summary>

/// Получение списка оперативной памяти с подробным описанием

/// </summary>

/// <returns>Список оперативной памяти с подробным описанием</returns>

public BindingList<RAMInfo> GetRAMsInfo()

{

var cp = from ram in \_ctx.RAM

select new RAMInfo

{

Model = ram.Model,

Manufacturer = ram.Manufacturer.Name,

RamType = ram.RamType.Name,

Volume = ram.Volume,

Price = ram.Price

};

return new BindingList<RAMInfo>(cp.ToList());

}

/// <summary>

/// Получение ссылки на оперативную память по его подробной информации

/// </summary>

/// <param name="currentRAM">Информация об оперативной памяти</param>

/// <returns>Ссылка на оперативную память</returns>

public async Task<RAM> GetRAMByRAMInfo(RAMInfo currentRAM)

{

if (currentRAM == null)

return null;

return await \_ctx.RAM.SingleOrDefaultAsync(s =>

s.Model.Equals(currentRAM.Model) && s.Manufacturer.Name.Equals(currentRAM.Manufacturer));

}

/// <summary>

/// Удаление выбранной оперативной памяти

/// </summary>

/// <param name="ramToDel">Оперативная память, которую следует удалить</param>

public async Task DeleteRAM(RAM ramToDel)

{

\_ctx.RAM.Remove(ramToDel);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Получение экземпляра оперативной памяти по ее модели и производителю

/// </summary>

/// <param name="model">Наименованиемодели</param>

/// <param name="manufacturer">Наименованиепроизводителя</param>

/// <returns>Экземпляроперативнойпамяти</returns>

public async Task<RAM> GetRAMByModelManufacturer(string model, string manufacturer)

{

return await \_ctx.RAM.SingleAsync(r => r.Model.Equals(model) && r.Manufacturer.Name.Equals(manufacturer));

}

#endregion

#region EditRAM

/// <summary>

/// Проверка дублирования оперативной памяти

/// </summary>

/// <param name="model">Модель</param>

/// <param name="manufacturer">Производитель</param>

/// <returns>Наличие существующей оперативной памяти с такой моделью и производителем</returns>

public async Task<bool> CheckRAMForDublicate(string model, string manufacturer)

{

return await \_ctx.RAM.SingleOrDefaultAsync(s =>

s.Model.ToUpper().Equals(model.ToUpper()) && s.Manufacturer.Name.Equals(manufacturer)) != null;

}

/// <summary>

/// Добавление новой оперативной памяти

/// </summary>

/// <param name="newRAM">Новая оперативная память</param>

public async Task AddNewRAM(RAM newRAM)

{

\_ctx.RAM.Add(newRAM);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Редактирование существующей оперативной памяти

/// </summary>

/// <param name="editedRAM">Существующая оперативная память</param>

public async Task EditRAM(RAM editedRAM)

{

\_ctx.RAM.Attach(editedRAM);

\_ctx.Entry(editedRAM).State = EntityState.Modified;

await \_ctx.SaveChangesAsync();

}

#endregion

#region RAM\_Types

/// <summary>

/// Получение списка типов оперативной памяти с привязкой

/// </summary>

/// <returns>Список типов оперативной памяти</returns>

public BindingList<RamType> GetRAMTypesBS()

{

\_ctx.RamType.Load();

return \_ctx.RamType.Local.ToBindingList();

}

/// <summary>

/// Проверка дублирования наименования типа оперативной памяти

/// </summary>

/// <param name="name">Наименование типа оперативной памяти</param>

/// <returns>Наличие типа оперативной памяти с таким наименованием</returns>

public async Task<bool> CheckRAMTypeForDublicate(string name)

{

return await \_ctx.RamType.SingleOrDefaultAsync(m => m.Name.ToUpper().Equals(name.ToUpper())) != null;

}

#endregion

#region EditSAN

/// <summary>

/// Получение списка доступных накопителей по укзанным интерфейсам хранилища данных

/// </summary>

/// <param name="avalibleInterfaces">Списокдоступныхинтерфейсов</param>

/// <returns>Список доступных накопителей по укзанным интерфейсам хранилища данных</returns>

public BindingList<Strorage> GetStoragesByAvalibleSANInterfaces(ICollection<SAN\_StorageInt> avalibleInterfaces)

{

var avInt = avalibleInterfaces.ToList();

var listOfAvalibleStorages =

\_ctx.Strorage.ToList().Where(s => avInt.Exists(nterf => nterf.StrorageInterface.Equals(s.StrorageInterface)));

return new BindingList<Strorage>(listOfAvalibleStorages.ToList());

}

/// <summary>

/// Получение списка доступных накопителей по укзанным интерфейсам

/// </summary>

/// <param name="avalibleInterfaces">Списокдоступныхинтерфейсов</param>

/// <returns>Список доступных накопителей по укзанным интерфейсам</returns>

public BindingList<Strorage> GetStoragesByAvalibleInterfaces(ICollection<AvalibleInterface> avalibleInterfaces)

{

var avInt = avalibleInterfaces.ToList();

var listOfAvalibleStorages =

\_ctx.Strorage.ToList().Where(s => avInt.Exists(x => x.Name.Equals(s.StrorageInterface.Name)));

return new BindingList<Strorage>(listOfAvalibleStorages.ToList());

}

/// <summary>

/// Получение преобразованного списка доступных интерфейсов накопителей для хранилища данных с привязкой

/// </summary>

/// <param name="storageInterfaces">Интерфейсы накопителей хранилища данных</param>

/// <returns>Преобразованный список доступных интерфейсов накопителей</returns>

public BindingList<AvalibleInterface> GetAvalibleInterfacesOfSANBS(ICollection<SAN\_StorageInt> storageInterfaces)

{

return new BindingList<AvalibleInterface>(GetAvalibleInterfacesOfSAN(storageInterfaces));

}

/// <summary>

/// Получение преобразованного списка доступных интерфейсов накопителей для хранилища данных

/// </summary>

/// <param name="storageInterfaces">Интерфейсы накопителей хранилища данных</param>

/// <returns>Преобразованный список доступных интерфейсов накопителей</returns>

private List<AvalibleInterface> GetAvalibleInterfacesOfSAN(ICollection<SAN\_StorageInt> storageInterfaces)

{

var storInt = storageInterfaces.ToList();

var avInt = new List<AvalibleInterface>();

storInt.ForEach(si => avInt.Add(new AvalibleInterface

{

Name = si.StrorageInterface.Name,

Slot\_Count = si.SlotCount

}));

return avInt;

}

/// <summary>

/// Получение обратно преобразованного списка доступных интерфейсов накопителей для хранилища данных

/// </summary>

/// <param name="san">Хранилищеданных</param>

/// <param name="avalibleInterfaces">Преобразованный список доступных интерфейсов накопителей</param>

/// <returns>Обработно преобразованный список доступных интерфейсов накопителей</returns>

public List<SAN\_StorageInt> GetSANStorageIntFromAvalible(SAN san,

List<AvalibleInterface> avalibleInterfaces)

{

var sanStInt = new List<SAN\_StorageInt>();

avalibleInterfaces.ForEach(ai => sanStInt.Add(new SAN\_StorageInt

{

StrorageInterface = GetInterfaceIdByName(ai.Name),

SAN = san,

SlotCount = ai.Slot\_Count

}));

return sanStInt;

}

/// <summary>

/// Получение ссылки на экземпляр интерфейса накопителя по его наименованию

/// </summary>

/// <param name="aiName">Наименование интерфейса накопителя</param>

/// <returns>Ссылка на экземпляр интерфейса</returns>

public StrorageInterface GetStorageInterfaceByName(string aiName)

{

return \_ctx.StrorageInterface.Single(si => si.Name.Equals(aiName));

}

/// <summary>

/// Сохранение нового хранилища данных

/// </summary>

/// <param name="newSAN">Новоехранилищеданных</param>

public async Task AddNewSAN(SAN newSAN)

{

\_ctx.SAN.Add(newSAN);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Сохранение изменений в редактируемом хранилище данных

/// </summary>

/// <param name="editedSAN">Редактируемоехранилищеданных</param>

public async Task EditSAN(SAN editedSAN)

{

\_ctx.SAN.Attach(editedSAN);

\_ctx.Entry(editedSAN).State = EntityState.Modified;

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Удаление выбранной конфигурации хранилища данных

/// </summary>

/// <param name="sanToDel">Конфигурация хранилища данных, которую следует удалить</param>

public async Task DeleteSAN(SAN sanToDel)

{

\_ctx.SAN.Remove(sanToDel);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Получение ссылки на экземпляр файлового хранилища по его подробной информации

/// </summary>

/// <param name="selectedSAN">Подробная информация о файловом хралилище</param>

/// <returns>Ссылка на экземпляр файлового хранилища</returns>

public async Task<SAN> GetSANBySANInfo(SAN\_Info selectedSAN)

{

return await \_ctx.SAN

.Include(s => s.SAN\_Storage)

.Include(s => s.SAN\_StorageInt)

.SingleAsync(s => s.Id == selectedSAN.ID);

}

#endregion

#region EditPaas

/// <summary>

/// Получение списка сервисов, которые используют выбранный сервер

/// </summary>

/// <param name="s">Выбранныйсервер</param>

/// <returns>Список сервисов, которые используют выбранный сервер</returns>

public List<Service> GetServicesByServer(Server s)

{

var paasTypes = s.PaasType.ToList();

return \_ctx.Service

.Include(serv => serv.ServiceIdle)

.ToList().Where(x => paasTypes.Contains(x.PaasType)).ToList();

}

/// <summary>

/// Получение списка используемых серверов выбранного сервиса

/// </summary>

/// <param name="s">Выбранныйсервис</param>

/// <returns>Список используемых серверов выбранного сервиса</returns>

public List<Server> GetServersByService(Service s)

{

return \_ctx.Server.ToList().Where(x => x.PaasType.ToList().Contains(s.PaasType)).ToList();

}

/// <summary>

/// Получение списка используемых хранилищ данных выбранного сервиса

/// </summary>

/// <param name="s">Выбранныйсервис</param>

/// <returns>Список используемых хранилищ данных выбранного сервиса </returns>

public List<SAN> GetSANByService(Service s)

{

return \_ctx.SAN.ToList().Where(x => x.PaasType.ToList().Contains(s.PaasType)).ToList();

}

/// <summary>

/// Получение списка сервисов, которые используют выбранное хранилище данных

/// </summary>

/// <param name="s">Выбранное хранилище данных</param>

/// <returns>Список сервисов, которые используют выбранное хранилище данных</returns>

public List<Service> GetServicesBySAN(SAN s)

{

var paasTypes = s.PaasType.ToList();

return \_ctx.Service.ToList().Where(x => paasTypes.Contains(x.PaasType)).ToList();

}

/// <summary>

/// Получение списка конфигураций серверов

/// </summary>

/// <returns>Список конфигураций серверов</returns>

public async Task<List<Server>> GetServerConfigs()

{

return await \_ctx.Server.ToListAsync();

}

/// <summary>

/// Получение списка используемых конфигураций серверов по выбранному типу плфатформы

/// </summary>

/// <returns>Список используемых конфигураций серверов выбранной платформы</returns>

public List<Server> GetServerConfigsByPaas(PaasType paasType)

{

var servers = \_ctx.Server

.Include(pt => pt.PaasType)

.ToList();

return servers.Where(pt => pt.PaasType.Contains(paasType)).ToList();

}

/// <summary>

/// Получение списка хранилищ данных

/// </summary>

/// <returns>Список хранилищ данных</returns>

public async Task<List<SAN>> GetSANsList()

{

return await \_ctx.SAN.ToListAsync();

}

/// <summary>

/// Получение списка используемых хранилищ данных по выбранному типу платформы

/// </summary>

/// <returns>Список используемых хранилищ данных по выбранному типу платформы</returns>

public List<SAN> GetSANsListByPaas(PaasType paasType)

{

var sans = \_ctx.SAN

.Include(pt => pt.PaasType)

.ToList();

return sans.Where(pt => pt.PaasType.Contains(paasType)).ToList();

}

/// <summary>

/// Получение информации о доступных/используемых ресурсов выбранного сервера

/// </summary>

/// <param name="selectedServer">Выбранныйсервер</param>

/// <returns>Информация о доступных/используемых ресурсов выбранного сервера</returns>

public ServerPaasInfo GetServerPaasInfoByServer(Server selectedServer)

{

var sInfo = GetServerInfo();

var servInfo = sInfo.Single(si => si.Id == selectedServer.Id);

var servicesByServer = GetServicesByServer(selectedServer);

var usedCoreCount = servicesByServer.Sum(cp => cp.CoreCount.Value);

var usedRAMVolume = servicesByServer.Sum(rv => rv.RamCount.Value);

var usedStorageVolume = (servicesByServer.Sum(sv => sv.HDDVolume.Value) / 10);

return new ServerPaasInfo

{

Id = selectedServer.Id,

CPU = servInfo.CPU,

Platform = servInfo.Platform,

AvalibleCoreCount = servInfo.CpuCount - usedCoreCount,

AvalibleRAMVolume = servInfo.RamVolume - usedRAMVolume,

AvalibleStorageVolume = servInfo.StorageVolume - usedStorageVolume,

UsedCoreCount = usedCoreCount,

UsedRAMVolume = usedRAMVolume,

UsedStorageVolume = usedStorageVolume

};

}

/// <summary>

/// Получение информации о доступных/используемых ресурсов выбранного хранилища данных

/// </summary>

/// <param name="selectedSAN">Выбранноехранилищеданных</param>

/// <returns>Информация о доступных/используемых ресурсов выбранного хранилища данных</returns>

public SANPaasInfo GetSANPaasInfoBySAN(SAN selectedSAN)

{

var sInfo = GetSanInfo();

var sanInfo = sInfo.Single(si => si.ID == selectedSAN.Id);

var servicesBySAN = GetServicesBySAN(selectedSAN);

var usedStorageVolume = (servicesBySAN.Sum(sv => sv.HDDVolume.Value));

return new SANPaasInfo

{

Id = selectedSAN.Id,

Manufacturer = sanInfo.Manufacturer,

Model = sanInfo.Model,

AvalibleVolume = sanInfo.Volume - usedStorageVolume,

UsedVolume = usedStorageVolume

};

}

/// <summary>

/// Проверка дублирования наименования платформы

/// </summary>

/// <param name="name">Наименование платформы</param>

/// <returns>Наличие платформы с таким наименованием</returns>

public async Task<bool> CheckPaasForDublicate(string name)

{

return await \_ctx.PaasType.SingleOrDefaultAsync(s => s.Name.ToUpper().Equals(name.ToUpper())) != null;

}

/// <summary>

/// Получение ссылки на экземпляр сервера по его порядковому номеру

/// </summary>

/// <param name="id">Порядковый номер сервера</param>

/// <returns>Ссылка на экземпляр сервера</returns>

public async Task<Server> GetServerByID(int id)

{

return await \_ctx.Server.SingleAsync(s => s.Id == id);

}

/// <summary>

/// Получение ссылки на экземпляр хранилища данных по его порядковому номеру

/// </summary>

/// <param name="id">Порядковый номер сервера</param>

/// <returns>Ссылка на экземпляр хранилища данных</returns>

public async Task<SAN> GetSANByID(int id)

{

return await \_ctx.SAN.SingleAsync(s => s.Id == id);

}

/// <summary>

/// Сохранение новой платформы

/// </summary>

/// <param name="newPaas">Новаяплатформа</param>

public async Task AddNewPaas(PaasType newPaas)

{

\_ctx.PaasType.Add(newPaas);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Сохранение изменений в редактируемой платформе

/// </summary>

/// <param name="editedPaas">Редактируемаяплатформа</param>

public async Task EditPaas(PaasType editedPaas)

{

\_ctx.PaasType.Attach(editedPaas);

\_ctx.Entry(editedPaas).State = EntityState.Modified;

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Получение ссылки на экземпляр платформы по ее подробному описанию

/// </summary>

/// <param name="selectedPlatform">Выбранное подробное описание платформы</param>

/// <returns>Ссылка на экземпляр платформы </returns>

public async Task<PaasType> GetPaasFromPaasInfo(PaasInfo selectedPlatform)

{

return await \_ctx.PaasType

.Include(pt => pt.Service)

.SingleAsync(pt => pt.Name.Equals(selectedPlatform.Name));

}

#endregion

#region EditService

/// <summary>

/// Получениеспискатиповплатформ

/// </summary>

/// <returns>Cписоктиповплатформ</returns>

public BindingList<PaasType> GetPaasTypes()

{

var lst = \_ctx.PaasType.Where(pt => pt.Server.Count > 0 && pt.SAN.Count > 0).ToList();

return new BindingList<PaasType>(lst);

}

/// <summary>

/// Получение списка типов сервиса

/// </summary>

/// <returns>Списоктиповсервиса</returns>

public BindingList<ServiceType> GetServiceTypes()

{

var lst = \_ctx.ServiceType.ToList();

return new BindingList<ServiceType>(lst);

}

/// <summary>

/// Получение списка серверов выбранной платформы

/// </summary>

/// <param name="currentPaasType">Выбраннаяплатформа</param>

/// <returns>Список серверов выбранной платформы</returns>

public BindingList<PlatformServerItem> GetServersByPlatform(PaasType currentPaasType)

{

var psi = new List<PlatformServerItem>();

\_ctx.Server.ToList().Where(s => s.PaasType.ToList().Exists(p => p.Id == currentPaasType.Id)).ToList()

.ForEach(x => psi.Add(new PlatformServerItem(x)));

return new BindingList<PlatformServerItem>(psi);

}

/// <summary>

/// Получение списка хранилищ данных выбранной платформы

/// </summary>

/// <param name="currentPaasType">Выбраннаяплатформа</param>

/// <returns>Список хранилищ данных выбранной платформы</returns>

public BindingList<PlatfromSANItem> GetSANsByPlatform(PaasType currentPaasType)

{

var psi = new List<PlatfromSANItem>();

\_ctx.SAN.ToList().Where(s => s.PaasType.ToList().Exists(p => p.Id == currentPaasType.Id)).ToList().ForEach(x => psi.Add(new PlatfromSANItem(x)));

return new BindingList<PlatfromSANItem>(psi);

}

/// <summary>

/// Получение детальной информации о сервере по его порядковому номеру

/// </summary>

/// <param name="id">Порядковый номер сервера</param>

/// <returns>Детальная информация о сервере</returns>

public Server GetServerDetailsByID(int id)

{

return \_ctx.Server

.Include(cp => cp.CPU)

.Include(pl => pl.Platform)

.Include(ss => ss.Server\_Storage)

.Include(sr => sr.Server\_RAM)

.Include(srv => srv.PaasType)

.Single(s => s.Id == id);

}

/// <summary>

/// Получение детальной информации о хранилище данных по его порядковому номеру

/// </summary>

/// <param name="id">Порядковый номер хранилища данных</param>

/// <returns>Детальная информация о хранилище данных</returns>

public SAN GetSANDetailsByID(int id)

{

return \_ctx.SAN

.Include(s => s.SAN\_Storage)

.Single(s => s.Id == id);

}

/// <summary>

/// Получение информации о доступных ресурсах сервера

/// </summary>

/// <param name="selectedServer">Исходныйэкземплярсервера</param>

/// <returns>Информация о доступных ресурсах сервера</returns>

public ServerPlarformInfo GetServerPlatformInfoByServer(Server selectedServer)

{

var servicesByServer = GetServicesByServer(selectedServer);

var usedCoreCount = servicesByServer.Sum(cp => cp.CoreCount.Value);

var usedRAMVolume = servicesByServer.Sum(rv => rv.RamCount.Value);

return new ServerPlarformInfo

{

SumFrequency = (selectedServer.CPU.Frequency \* selectedServer.Platform.CPUCount) \* selectedServer.CPU.CoreCount,

AvalibleCoreCount = (selectedServer.CPU.CoreCount \* selectedServer.Platform.CPUCount) - usedCoreCount,

CoreCount = selectedServer.CPU.CoreCount \* selectedServer.Platform.CPUCount,

AvalibleRAMVolume = selectedServer.Server\_RAM.Sum(sr => sr.RAM.Volume \* sr.Count) - usedRAMVolume,

FrequencyPerCore = selectedServer.CPU.Frequency,

SumRAMVolume = selectedServer.Server\_RAM.Sum(sr => sr.RAM.Volume \* sr.Count)

};

}

/// <summary>

/// Получение информации о доступных ресурсах хранилища данных

/// </summary>

/// <param name="selectedSAN">Исходный экземпляр хранилища данных</param>

/// <returns>Информация о доступных ресурсах хранилища данных</returns>

public SANPlatformInfo GetSanPlatformInfoBySAN(SAN selectedSAN)

{

var servicesBySAN = GetServicesBySAN(selectedSAN);

var usedStorageVolume = (servicesBySAN.Sum(sv => sv.HDDVolume.Value));

return new SANPlatformInfo

{

SummaryVolume = selectedSAN.SAN\_Storage.Sum(ss => ss.Strorage.Volume \* ss.Count.Value),

AvalibleVolume = selectedSAN.SAN\_Storage.Sum(ss => ss.Strorage.Volume \* ss.Count.Value) - usedStorageVolume

};

}

/// <summary>

/// Создание нового сервиса

/// </summary>

/// <param name="newService">Экземплярновогосервиса</param>

public async Task AddNewService(Service newService)

{

newService.CreateDate = DateTime.Now;

newService.ServiceState = await GetServiceStateByName("Отправленазаявка");

\_ctx.Service.Add(newService);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Редактирование выбранного сервиса

/// </summary>

/// <param name="editedService">Экземплярвыбранногосервиса</param>

public async Task EditService(Service editedService)

{

editedService.ServiceState = await GetServiceStateByName("Неактивен");

\_ctx.ServiceIdle.Add(new ServiceIdle

{

Date = DateTime.Now,

BeginPeriod = DateTime.Now,

EndPeriod = DateTime.Now.AddDays(1),

Service = editedService,

IdleType = await GetIdleTypeByName("Внеплановый"),

IdleReason = await GetIdleReasonByName("Обновлениеконфигурации")

});

\_ctx.Service.Attach(editedService);

\_ctx.Entry(editedService).State = EntityState.Modified;

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Получение экземпляра состояния сервиса по его наименованию

/// </summary>

/// <param name="name">Наименование состояния сервиса</param>

/// <returns>Экземпляр состояния сервиса</returns>

private async Task<ServiceState> GetServiceStateByName(string name)

{

return await \_ctx.ServiceState.SingleAsync(ss => ss.Name.Equals(name));

}

/// <summary>

/// Получение экземпляра типа простоя сервиса по его наименованию

/// </summary>

/// <param name="name">Наименование типа простоя сервиса</param>

/// <returns>Экземпляр типа простоя сервиса</returns>

private async Task<IdleType> GetIdleTypeByName(string name)

{

return await \_ctx.IdleType.SingleAsync(it => it.Name.Equals(name));

}

/// <summary>

/// Получение экземпляра причины простоя сервиса по его наименованию

/// </summary>

/// <param name="name">Наименование причины простоя сервиса</param>

/// <returns>Экземпляр причины простоя сервиса</returns>

private async Task<IdleReason> GetIdleReasonByName(string name)

{

return await \_ctx.IdleReason.SingleAsync(ir => ir.Name.Equals(name));

}

/// <summary>

/// Получение экземляра программного обеспечения по его ID

/// </summary>

/// <param name="id">ID программного обеспечения</param>

/// <returns>Экземляр программного обеспечения</returns>

public async Task<Software> GetSoftwareById(int id)

{

return await \_ctx.Software.SingleAsync(s => s.Id == id);

}

#endregion

#region SelectSoftware

/// <summary>

/// Получение списка программного обеспечения с подробным описанием

/// </summary>

/// <returns>Список программного обеспечения с подробным описанием</returns>

public BindingList<SoftwareInfo> GetSoftwaresInfo()

{

var soft = from software in \_ctx.Software

select new SoftwareInfo

{

Id = software.Id,

Software = software.Name,

SoftType = software.SoftType.Name,

Cost = software.Cost

};

return new BindingList<SoftwareInfo>(soft.ToList());

}

/// <summary>

/// Получение списка типов программного обеспечения

/// </summary>

/// <returns>Список типов программного обеспечения</returns>

public BindingList<SoftType> GetSoftwareTypes()

{

return new BindingList<SoftType>(\_ctx.SoftType.ToList());

}

/// <summary>

/// Получение программного обеспечения по его информации

/// </summary>

/// <param name="selectedItem">Информация о программном обеспечении</param>

/// <returns>Экземпляр программного обеспечения</returns>

public Software GetSoftwareByInfo(SoftwareInfo selectedItem)

{

return \_ctx.Software.Single(cs =>

cs.Name.Equals(selectedItem.Software) && cs.SoftType.Name.Equals(selectedItem.SoftType));

}

/// <summary>

/// Удаление выбранного программного обеспечения

/// </summary>

/// <param name="softwareToDel">Программное обеспечение, которую следует удалить</param>

public async Task DeleteSoftware(Software softwareToDel)

{

\_ctx.Software.Remove(softwareToDel);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Проверка дублирования программного обеспечения

/// </summary>

/// <param name="name">Наименование программного обеспечения</param>

/// <param name="softTypeName">Наименование типа программного обеспечения</param>

/// <returns>Наличие программного обеспечения с таким наименованием и типом</returns>

public async Task<bool> CheckSoftwareForDublicate(string name, string softTypeName)

{

return await \_ctx.Software.SingleOrDefaultAsync(cs =>

cs.Name.ToUpper().Equals(name.ToUpper()) && cs.SoftType.Name.Equals(softTypeName)) != null;

}

/// <summary>

/// Добавление нового программного обеспечения

/// </summary>

/// <param name="newSoftware">Новое программное обеспечение</param>

public async Task AddNewSoftware(Software newSoftware)

{

\_ctx.Software.Add(newSoftware);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Редактирование существующего программного обеспечения

/// </summary>

/// <param name="editedSoftware">Существующее программное обеспечение</param>

public async Task EditSoftware(Software editedSoftware)

{

\_ctx.Software.Attach(editedSoftware);

\_ctx.Entry(editedSoftware).State = EntityState.Modified;

await \_ctx.SaveChangesAsync();

}

#endregion

#region SoftTypes

/// <summary>

/// Получение списка типов программного обеспечения с привязкой

/// </summary>

/// <returns>Список типов программного обеспечения</returns>

public BindingList<SoftType> GetSoftTypesBS()

{

\_ctx.SoftType.Load();

return \_ctx.SoftType.Local.ToBindingList();

}

/// <summary>

/// Проверка дублирования наименования типа программного обеспечения

/// </summary>

/// <param name="name">Наименование типа программного обеспечения</param>

/// <returns>Наличие типа программного обеспечения с таким наименованием</returns>

public async Task<bool> CheckSoftTypeForDublicate(string name)

{

return await \_ctx.SoftType.SingleOrDefaultAsync(m => m.Name.ToUpper().Equals(name.ToUpper())) != null;

}

#endregion

#region ServiceTypes

/// <summary>

/// Получениеспискатиповсервисаспривязкой

/// </summary>

/// <returns>Списоктиповсервиса</returns>

public BindingList<ServiceType> GetServiceTypesBS()

{

\_ctx.ServiceType.Load();

return \_ctx.ServiceType.Local.ToBindingList();

}

/// <summary>

/// Проверка дублирования наименования типа сервиса

/// </summary>

/// <param name="name">Наименованиетипасервиса</param>

/// <returns>Наличие типа сервиса с таким наименованием</returns>

public async Task<bool> CheckServiceTypeForDublicate(string name)

{

return await \_ctx.ServiceType.SingleOrDefaultAsync(m => m.Name.ToUpper().Equals(name.ToUpper())) != null;

}

#endregion

#region ListOfServices

/// <summary>

/// Получение списка платформ с информацией о количестве пользователей и сервисов

/// </summary>

/// <returns>Список платформ с информацией о количестве пользователей и сервисов</returns>

public BindingList<PlatformServiceUser> GetPlatformServiceUsers()

{

var qry = from paasType in \_ctx.PaasType

select new PlatformServiceUser

{

Name = paasType.Name,

ServiceCount = paasType.Service.Count,

UserCount = paasType.Service.Count > 0 ? paasType.Service.Sum(uc => uc.User.Count) : 0

};

return new BindingList<PlatformServiceUser>(qry.ToList());

}

/// <summary>

/// Получение полного списка конфигураций сервисов с детальным описанием

/// </summary>

/// <returns>Полный список конфигураций сервисов с детальным описанием</returns>

public BindingList<ServiceDetailInfo> GetServicesDetailInfo()

{

var qry = from service in \_ctx.Service

select new ServiceDetailInfo

{

ID = service.Id,

PaasType = service.PaasType.Name,

ServiceType = service.ServiceType.Name,

ServiceState = service.ServiceState.Name,

UserCount = service.User.Count,

CostPerHour = service.CostPerHour ?? 0,

Create\_Date = service.CreateDate,

Delete\_Date = service.DeleteDate

};

return new BindingList<ServiceDetailInfo>(qry.ToList());

}

/// <summary>

/// Получение полного списка конфигураций сервисов с детальным описанием

/// </summary>

/// <returns>Полный список конфигураций сервисов с детальным описанием</returns>

public BindingList<ServiceDetailInfo> GetServicesCost()

{

var qry = from service in \_ctx.Service

where !service.ServiceState.Name.Equals("Отклонен") && !service.ServiceState.Name.Equals("Отправленазаявка")

select new ServiceDetailInfo

{

ID = service.Id,

PaasType = service.PaasType.Name,

ServiceType = service.ServiceType.Name,

ServiceState = service.ServiceState.Name,

UserCount = service.User.Count,

CostPerHour = service.CostPerHour ?? 0,

Create\_Date = service.CreateDate

};

return new BindingList<ServiceDetailInfo>(qry.ToList());

}

/// <summary>

/// Получение экземпляра платформы по ее подробной информации

/// </summary>

/// <param name="platformServiceUser">Подробнаяинформацияоплатформе</param>

/// <returns>Экземплярплатформы</returns>

public async Task<PaasType> GetPlatformByPlatformServiceUser(PlatformServiceUser platformServiceUser)

{

return await \_ctx.PaasType.SingleAsync(p => p.Name.Equals(platformServiceUser.Name));

}

/// <summary>

/// Получение экземпляра сервиса по ее подробной информации

/// </summary>

/// <param name="serviceDetailInfo">Подробнаяинформацияосервисе</param>

/// <returns>Экземплярсервиса</returns>

public async Task<Service> GetServiceByServiceDetailInfo(ServiceDetailInfo serviceDetailInfo)

{

return await \_ctx.Service.SingleAsync(s => s.Id == serviceDetailInfo.ID);

}

/// <summary>

/// Удаление выбранного сервиса

/// </summary>

/// <param name="serviceToDel">Сервис, которую следует удалить</param>

public async Task DeleteService(Service serviceToDel)

{

\_ctx.Service.Remove(serviceToDel);

await \_ctx.SaveChangesAsync();

}

#endregion

#region EquipmentIdleMenu

/// <summary>

/// Получение списка с информацией о простое серверов

/// </summary>

/// <returns>Список с информацией о простое серверов</returns>

public BindingList<ServerIdleItem> GetServerIdleItems()

{

var qry = from server in \_ctx.Server

from service in \_ctx.Service

join serviceIdle in \_ctx.ServiceIdle on service.Id equals serviceIdle.ServiceId into si

from subServ in si.DefaultIfEmpty()

join platform in \_ctx.Platform on server.PlatformId equals platform.Id

join cpu in \_ctx.CPU on server.CPU\_Id equals cpu.Id

where server.PaasType.Contains(service.PaasType) || subServ == null

select new ServerIdleItem

{

ID = server.Id,

Plarform = platform.Manufacturer.Name + " " + platform.Model,

CPU = cpu.Manufacturer.Name + " " + cpu.Model + " x" + platform.CPUCount,

RAM = server.Server\_RAM.Sum(sr => sr.RAM.Volume \* sr.Count),

OnMaintenance = server.OnMaintenance,

Active = server.Active,

CompleteIdleCount = si.Count(s => s.EndPeriod <= DateTime.Now),

SheduledIdleCount = server.Server\_MaintenanceShedule.Count(sms => sms.BeginPeriod >= DateTime.Now)

};

var xx = qry.ToList().GroupBy(q => q.ID)

.Select(q => q.First())

.ToList();

return new BindingList<ServerIdleItem>(xx);

}

/// <summary>

/// Получение списка с информацией о простое хранилищ данных

/// </summary>

/// <returns>Список с информацией о простое хранилищ данных</returns>

public BindingList<SANIdleItem> GetSanIdleItems()

{

var qry = from san in \_ctx.SAN

from service in \_ctx.Service

join serviceIdle in \_ctx.ServiceIdle on service.Id equals serviceIdle.ServiceId into si

from subServ in si.DefaultIfEmpty()

where san.PaasType.Contains(service.PaasType) || subServ == null

select new SANIdleItem

{

ID = san.Id,

Manufacturer = san.Manufacturer.Name,

Model = san.Model,

StorageCount = san.SAN\_Storage.Sum(ss => ss.Count.Value),

StorageVolume = san.SAN\_Storage.Sum(ss => ss.Count.Value \* ss.Strorage.Volume),

OnMaintenance = san.OnMaintenance,

Active = san.Active,

CompleteIdleCount = si.Count(s => s.EndPeriod <= DateTime.Now),

SheduledIdleCount = san.SAN\_MaintenanceShedule.Count(sms => sms.BeginPeriod >= DateTime.Now)

};

var xx = qry.ToList().GroupBy(q => q.ID)

.Select(q => q.First())

.ToList();

return new BindingList<SANIdleItem>(xx);

}

/// <summary>

/// Получение списка с информацией о простое сервисов

/// </summary>

/// <returns>Список с информацией о простое сервисов</returns>

public BindingList<ServiceIdleItem> GetServiceIdleItems()

{

var qry = from service in \_ctx.Service

from server in \_ctx.Server

from san in \_ctx.SAN

where san.PaasType.Contains(service.PaasType) && server.PaasType.Contains(service.PaasType)

select new ServiceIdleItem

{

ID = service.Id,

Name = service.ServiceType.Name,

PaasTypeName = service.PaasType.Name,

CurrentState = service.ServiceState.Name,

UsersCount = service.User.Count,

CompleteIdleCount = san.SAN\_MaintenanceShedule.Count(sms => sms.EndPeriod <= DateTime.Now) + server.Server\_MaintenanceShedule.Count(sms => sms.EndPeriod <= DateTime.Now),

SheduledIdleCount = san.SAN\_MaintenanceShedule.Count(sms => sms.BeginPeriod >= DateTime.Now) + server.Server\_MaintenanceShedule.Count(sms => sms.BeginPeriod >= DateTime.Now)

};

return new BindingList<ServiceIdleItem>(qry.ToList());

}

#endregion

#region EditScheduleItem

/// <summary>

/// Добавление новой позиции в расписании обслуживания сервера

/// </summary>

/// <param name="serverMaintenanceShedule">Новая позиции в расписании обслуживания сервера</param>

public async Task AddNewServerSchedulePosition(Server\_MaintenanceShedule serverMaintenanceShedule)

{

\_ctx.Server\_MaintenanceShedule.Add(serverMaintenanceShedule);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Добавление новой позиции в расписании обслуживания хранилища данных

/// </summary>

/// <param name="sanMaintenanceShedule">Новая позиции в расписании обслуживания хранилища данных</param>

public async Task AddNewSANSchedulePosition(SAN\_MaintenanceShedule sanMaintenanceShedule)

{

\_ctx.SAN\_MaintenanceShedule.Add(sanMaintenanceShedule);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Изменение позиции в расписании обслуживания сервера

/// </summary>

/// <param name="serverMaintenanceShedule">Измененная позиция в расписании обслуживания сервера</param>

public async Task EditServerSchedulePosition(Server\_MaintenanceShedule serverMaintenanceShedule)

{

\_ctx.Server\_MaintenanceShedule.Attach(serverMaintenanceShedule);

\_ctx.Entry(serverMaintenanceShedule).State = EntityState.Modified;

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Изменений позиции в расписании обслуживания хранилища данных

/// </summary>

/// <param name="sanMaintenanceShedule">Измененная позиция в расписании обслуживания хранилища данных</param>

public async Task EditSANSchedulePosition(SAN\_MaintenanceShedule sanMaintenanceShedule)

{

\_ctx.SAN\_MaintenanceShedule.Attach(sanMaintenanceShedule);

\_ctx.Entry(sanMaintenanceShedule).State = EntityState.Modified;

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Проверка уже запланированных позиций обсулживания в установленном диапазоне дат

/// </summary>

/// <param name="positionId">Номерредактируемойзаписи</param>

/// <param name="id">Код выбранного сервера или хранилища данных</param>

/// <param name="isServer">Флаг, который укзаывает какую таблицу проверять</param>

/// <param name="beginDateTime">Датаначалаобсулживания</param>

/// <param name="endDateTime">Датаокончанияобсулживания</param>

public async Task<bool> CheckMaintenanceDateRange(int positionId, int id, bool isServer, DateTime beginDateTime, DateTime endDateTime)

{

if (isServer)

{

if (positionId < 0)

return await \_ctx.Server\_MaintenanceShedule

.AnyAsync(sms => sms.ServerID == id && sms.BeginPeriod >= beginDateTime && sms.EndPeriod <= endDateTime);

else

return await \_ctx.Server\_MaintenanceShedule

.AnyAsync(sms => sms.ServerID == id && sms.BeginPeriod >= beginDateTime && sms.EndPeriod <= endDateTime && sms.Id != positionId);

}

else

{

if (positionId < 0)

return await \_ctx.SAN\_MaintenanceShedule

.AnyAsync(sms => sms.SAN\_Id == id && sms.BeginPeriod >= beginDateTime && sms.EndPeriod <= endDateTime);

else

return await \_ctx.SAN\_MaintenanceShedule

.AnyAsync(sms => sms.SAN\_Id == id && sms.BeginPeriod >= beginDateTime && sms.EndPeriod <= endDateTime && sms.Id != positionId);

}

}

#endregion

#region PlanningSchedule

/// <summary>

/// Получение списка позиций обслуживания для выбранного сервера

/// </summary>

/// <param name="selectedServer">Выбранныйсервер</param>

/// <returns>Список позиций обсулуживания для выбранного сервера</returns>

public BindingList<ScheduleItem> GetScheduleItemsOfServer(Server selectedServer)

{

var serverMaintenanceShedules = \_ctx.Server\_MaintenanceShedule.Where(sms => sms.ServerID == selectedServer.Id).ToList();

var scheduleItems = new List<ScheduleItem>();

serverMaintenanceShedules.ForEach(sms => scheduleItems.Add(new ScheduleItem

{

Id = sms.Id,

BeginDate = sms.BeginPeriod,

EndDate = sms.EndPeriod,

Duration = (sms.EndPeriod - sms.BeginPeriod).TotalMinutes.ToString()

}));

return new BindingList<ScheduleItem>(scheduleItems);

}

/// <summary>

/// Получение списка позиций обслуживания для выбранного хранилища данных

/// </summary>

/// <param name="selectedSAN">Выбранноехранилищеданных</param>

/// <returns>Список позиций обсулуживания для выбранного хранилища данных</returns>

public BindingList<ScheduleItem> GetScheduleItemsOfSAN(SAN selectedSAN)

{

var serverMaintenanceShedules = \_ctx.SAN\_MaintenanceShedule.Where(sms => sms.SAN\_Id == selectedSAN.Id).ToList();

var scheduleItems = new List<ScheduleItem>();

serverMaintenanceShedules.ForEach(sms => scheduleItems.Add(new ScheduleItem

{

Id = sms.Id,

BeginDate = sms.BeginPeriod,

EndDate = sms.EndPeriod,

Duration = (sms.EndPeriod - sms.BeginPeriod).TotalMinutes.ToString()

}));

return new BindingList<ScheduleItem>(scheduleItems);

}

/// <summary>

/// Полулчение ссылки на экземпляр позиции расписания обслуживания сервера

/// </summary>

/// <param name="id">Кодпозиции</param>

/// <returns>Ссылка на экземпляр позиции расписания обслуживания сервера</returns>

public async Task<Server\_MaintenanceShedule> GetServerMaintenanceSheduleByID(int id)

{

return await \_ctx.Server\_MaintenanceShedule.SingleAsync(sms => sms.Id == id);

}

/// <summary>

/// Полулчение ссылки на экземпляр позиции расписания обслуживания хранилища данных

/// </summary>

/// <param name="id">Кодпозиции</param>

/// <returns>Ссылка на экземпляр позиции расписания обслуживания хранилища данных</returns>

public async Task<SAN\_MaintenanceShedule> GetSANMaintenanceSheduleByID(int id)

{

return await \_ctx.SAN\_MaintenanceShedule.SingleAsync(sms => sms.Id == id);

}

/// <summary>

/// Удаление выбранной позиции расписания обслуживания сервера

/// </summary>

/// <param name="selectedServerMaintenanceShedule">Удаляемаяпозиция</param>

public async Task DeleteSelectedServiceScheduleMaintenance(Server\_MaintenanceShedule selectedServerMaintenanceShedule)

{

\_ctx.Server\_MaintenanceShedule.Remove(selectedServerMaintenanceShedule);

await \_ctx.SaveChangesAsync();

}

/// <summary>

/// Удаление выбранной позиции расписания обслуживания хранилища данных

/// </summary>

/// <param name="selectedServerMaintenanceShedule">Удаляемаяпозиция</param>

public async Task DeleteSelectedSANScheduleMaintenance(SAN\_MaintenanceShedule selectedSanMaintenanceShedule)

{

\_ctx.SAN\_MaintenanceShedule.Remove(selectedSanMaintenanceShedule);

await \_ctx.SaveChangesAsync();

}

#endregion

#region RegisterNewIdle

/// <summary>

/// Получения списка пользователей, которые относятся к администраторам

/// </summary>

/// <returns>Список пользователей, которые относятся к администраторам</returns>

public List<User> GetAdminsList()

{

return \_ctx.User.Where(usr => usr.RightsLevel.Name.Equals("Системный администратор")).ToList();

}

/// <summary>

/// Получение списка типов простоя

/// </summary>

/// <returns>Список типов простоя</returns>

public BindingList<IdleType> GetIdleTypes()

{

return new BindingList<IdleType>(\_ctx.IdleType.ToList());

}

/// <summary>

/// Получение списка причин простоя

/// </summary>

/// <returns>Список причин простоя</returns>

public BindingList<IdleReason> GetIdleReasons()

{

return new BindingList<IdleReason>(\_ctx.IdleReason.ToList());

}

/// <summary>

/// Регистрация нового простоя сервисов

/// </summary>

/// <param name="services">Список сервисов на простой</param>

/// <param name="usedPersonal">Задействованный персонал</param>

/// <param name="beginDateTime">Дата начала простоя</param>

/// <param name="endDateTime">Датаокончанияпростоя</param>

/// <param name="selectedIdleType">Типпростоя</param>

/// <param name="selectedIdleReason">Причинапростоя</param>

public void RegisterNewIdle(List<Service> services, List<User> usedPersonal, DateTime beginDateTime, DateTime endDateTime, IdleType selectedIdleType, IdleReason selectedIdleReason)

{

var qry = from sb in \_ctx.Service.AsEnumerable()

where services.Exists(srv => srv.Id == sb.Id) && sb.CreateDate.Value < beginDateTime

&& !(sb.DeleteDate != null && sb.DeleteDate <= endDateTime)

select new ServiceIdle

{

ServiceId = sb.Id,

Date = DateTime.Now,

BeginPeriod = beginDateTime,

EndPeriod = endDateTime,

IdleType = selectedIdleType,

IdleReason = selectedIdleReason,

User = usedPersonal

};

\_ctx.ServiceIdle.AddRange(qry.AsEnumerable());

\_ctx.SaveChanges();

}

#endregion

#region DirectorForm

/// <summary>

/// Получение списка заявок на создание сервисов

/// </summary>

/// <returns>Список заявок на создание сервисов</returns>

public BindingList<ServiceRequestItem> GetServiceReuqests()

{

var qry = from service in \_ctx.Service

where service.ServiceState.Name.Equals("Отправленазаявка")

select new ServiceRequestItem

{

Id = service.Id,

Name = service.ServiceType.Name,

PaasTypeName = service.PaasType.Name,

CreateDate = service.CreateDate.Value,

CostPerHour = service.CostPerHour.Value

};

return new BindingList<ServiceRequestItem>(qry.ToList());

}

/// <summary>

/// Получение экземпляра сервиса по его порядковому номеру

/// </summary>

/// <param name="id">Порядковый номер сервиса</param>

/// <returns>Экземпляр сервиса по его порязковому номеру</returns>

public Service GetServiceByID(int id)

{

return \_ctx.Service.Single(s => s.Id == id);

}

/// <summary>

/// Получение списка с информацией о простое сервиса

/// </summary>

/// <returns>Список с информацией о простое сервиса</returns>

public BindingList<IdleItem> GetIdleItems()

{

var idleItems = \_ctx.ServiceIdle

.Include(si => si.Service)

.Include(si => si.Service.ServiceType)

.Include(si => si.Service.PaasType)

.AsEnumerable();

var qry = from serviceIdle in idleItems

select new IdleItem

{

Id = serviceIdle.Id,

ServiceName = serviceIdle.Service.ServiceType.Name,

PaasName = serviceIdle.Service.PaasType.Name,

BeginDate = serviceIdle.BeginPeriod,

EndDate = serviceIdle.EndPeriod,

Duration = (serviceIdle.EndPeriod - serviceIdle.BeginPeriod).TotalMinutes.ToString(),

Cost = serviceIdle.Cost != null ? serviceIdle.Cost.Value.ToString("F") : "Нерасчитана"

};

return new BindingList<IdleItem>(qry.ToList());

}

#endregion

#region Расчет стоимости простоя

/// <summary>

/// Фоновый расчет стоимости простоя сервисов

/// </summary>

public void CalculateServicesIdleBackground()

{

// Нерасчитанные экземпляры простоя сервисов

var idlesToCalc = \_ctx.ServiceIdle.Where(si => si.Cost == null).ToList();

if (idlesToCalc.Count == 0)

return;

// Годовойдоходкомпанни

CostIdleParams idleParams = null;

// Минутвгоду

int minutesInYear = 525600;

try

{

// Попытказагрузкиизфайла

idleParams = CostIdleParams.Load();

// При отсутсвтсвии файла расчет обрывается

if (idleParams == null)

return;

}

catch

{

// в случае ошибки расчет обрывается

return;

}

// Доходвминуту

decimal incomeInMinute = idleParams.YearIncome / minutesInYear;

// Среднийдоходадминистраторавминуту

decimal avgAdminSalaryPerMin = idleParams.AvgAdminSalary / 60;

// Среднийдоходработникавминуту

decimal avgEmployeeSalaryPerMin = idleParams.AvgEmployeeSalary / 60;

// Доход на каждого работника (руб./мин)

decimal incomePerEmployee = incomeInMinute / idleParams.TotalEmployeeCount;

foreach (ServiceIdle serviceIdle in idlesToCalc)

{

// Продолжительность (мин.)

double duration = (serviceIdle.EndPeriod - serviceIdle.BeginPeriod).TotalMinutes;

// количествопользователей, отключенныхприэтом

int disEmployesCount = serviceIdle.Service.User.Count;

// количество администраторов, задействованных для этого

int usedAdminsCount = serviceIdle.User.Count;

// Плановые расходы на администраторов (руб./мин.)

decimal adminPlannedCost = (decimal)duration \* usedAdminsCount \* avgAdminSalaryPerMin;

// Плановые расходы на конечных пользователей (руб./мин.)

decimal employeePlannedCost = (decimal)duration \* disEmployesCount \* avgEmployeeSalaryPerMin;

// Плановыерасходынаотключениесервера

decimal servicePlannedCost = adminPlannedCost + employeePlannedCost;

// Потерянныйдоход (руб./мин.)

decimal brokenIncome = incomePerEmployee \* (decimal)duration;

// Итоговаястоимостьпростоясервиса

serviceIdle.Cost = (servicePlannedCost + brokenIncome);

}

idlesToCalc.ForEach(itc => \_ctx.ServiceIdle.Single(si => si.Id == itc.Id).Cost = itc.Cost);

\_ctx.SaveChanges();

}

/// <summary>

/// Получение экземпляра простоя сервиса по его номеру

/// </summary>

/// <param name="id">Номерзаписи</param>

/// <returns>Экземпляр простоя сервиса</returns>

public ServiceIdle GetServiceIdleByID(int id)

{

return \_ctx.ServiceIdle

.Include(si => si.Service)

.Include(si => si.Service.ServiceState)

.Include(si => si.Service.PaasType)

.Include(si => si.User)

.Single(si => si.Id == id);

}

#endregion

#region ServiceRequestTreatment

/// <summary>

/// Отменазаявкинасозданиесервиса

/// </summary>

/// <param name="selectedService">Отклоняемыйсервис</param>

public void CancelServiceRequest(Service selectedService)

{

var service = \_ctx.Service.Single(s => s.Id == selectedService.Id);

service.ServiceState = \_ctx.ServiceState.Single(ss => ss.Name.Equals("Отклонен"));

\_ctx.SaveChanges();

}

/// <summary>

/// Принятие заявки на создание сервиса

/// </summary>

/// <param name="selectedService">Принемаемыйсервис</param>

public void ConfirmServiceRequest(Service selectedService)

{

var service = \_ctx.Service.Single(s => s.Id == selectedService.Id);

service.ServiceState = \_ctx.ServiceState.Single(ss => ss.Name.Equals("Активен"));

//todo Исправить случай с плановым обсуживанием

var serversByService = GetServersByService(selectedService);

var sanByService = GetSANByService(selectedService);

serversByService.ForEach(s => \_ctx.Server.Single(serv => serv.Id == s.Id).Active = true);

sanByService.ForEach(s => \_ctx.SAN.Single(san => san.Id == s.Id).Active = true);

\_ctx.SaveChanges();

}

#endregion

#region EmployeeForm

/// <summary>

/// Получение списка используемых сервисов текущим пользователем

/// </summary>

/// <returns>Список используемых сервисов текущим пользователем</returns>

public BindingList<UserService> GetUsedServices()

{

var xx = \_ctx.Service

.Include(s => s.User)

.ToList()

.Where(s => s.User.Contains(CurrentUser))

.GroupBy(s => s.ServiceType);

var x = xx.Select(g => new UserService

{

ServiceTypeId = g.Key.Id,

Name = g.Key.Name,

PaasName = g.Select(m => m.PaasType.Name).First(),

State = g.All(s => s.ServiceState.Name.Equals("Неактивен")) ? "Неактивен" : "Активен"

});

return new BindingList<UserService>(x.ToList());

}

/// <summary>

/// Отпискаотвыбранногосервиса

/// </summary>

/// <param name="selectedServiceType">Выбранныйтипсервиса</param>

public void UnSubscribeSelectedService(ServiceType selectedServiceType)

{

var qry = from service in \_ctx.Service.AsEnumerable()

where service.User.Contains(CurrentUser) && service.ServiceType.Id == selectedServiceType.Id

select service;

var services = qry.ToList();

User currentUser = \_ctx.User.Single(usr => usr.TabNo == CurrentUser.TabNo);

services.ForEach(s => currentUser.Service.Remove(s));

\_ctx.SaveChanges();

}

/// <summary>

/// Получение ссылки на экземпляр типа сервиса по его порядковому номеру

/// </summary>

/// <param name="serviceTypeId">Порядковыйномерсервиса</param>

/// <returns>Ссылка на экземпляр типа сервиса</returns>

public ServiceType GetServiceTypeById(int serviceTypeId)

{

return \_ctx.ServiceType.Single(st => st.Id == serviceTypeId);

}

#endregion

#region SubscribeServicesForm

/// <summary>

/// Получениеспискадоступныхтиповсервисов

/// </summary>

/// <returns></returns>

public BindingList<ServiceType> GetAvalibleServiceTypes()

{

var qry = \_ctx.Service

.Include(s => s.User)

.ToList()

.Where(s => !s.User.Contains(CurrentUser))

.GroupBy(s => s.ServiceType);

var serviceTypes = new List<ServiceType>();

qry.ToList().ForEach(st =>

{

if (st.All(s => s.ServiceState.Name.Equals("Активен") || s.ServiceState.Name.Equals("Неактивен")))

serviceTypes.Add(st.Key);

});

return new BindingList<ServiceType>(serviceTypes);

}

/// <summary>

/// Подписка на выбранные сервисы

/// </summary>

/// <param name="selectedServiceTypes">Выбранныекатегориисервисов</param>

public void SubscribeToSelectedServices(List<ServiceType> selectedServiceTypes)

{

var qry = from service in \_ctx.Service.AsEnumerable()

where selectedServiceTypes.Exists(z => z.Id == service.ServiceType.Id) &&

(service.ServiceState.Name.Equals("Активен") || service.ServiceState.Name.Equals("Неактивен"))

select service;

var services = qry.ToList();

User currentUser = \_ctx.User.Single(usr => usr.TabNo == CurrentUser.TabNo);

services.ForEach(s => currentUser.Service.Add(s));

\_ctx.SaveChanges();

}

#endregion

#region EditIdleItemForm

/// <summary>

/// Получение параметров для расчета стоимости простоя

/// </summary>

/// <returns>Параметры для расчета стоимости простоя</returns>

public CostIdleParams GetIdleParamsFormDatabase()

{

var costIdleParams = new CostIdleParams();

var qryAdmin = from position in \_ctx.Position

join user in \_ctx.User on position.Id equals user.PositionId

join rightsLevel in \_ctx.RightsLevel on user.RightLevelId equals rightsLevel.Id

where rightsLevel.Name.Equals("Системныйадминистратор")

select position.AvgSalary;

var qryEmployee = from position in \_ctx.Position

join user in \_ctx.User on position.Id equals user.PositionId

join rightsLevel in \_ctx.RightsLevel on user.RightLevelId equals rightsLevel.Id

where rightsLevel.Name.Equals("Сотрудник")

select position.AvgSalary;

var adm = qryAdmin.ToList();

var empl = qryEmployee.ToList();

costIdleParams.AvgAdminSalary = adm.Count > 0 ? adm.Average(x => x / 720) : 0;

costIdleParams.AvgEmployeeSalary = empl.Count > 0 ? empl.Average(x => x / 720) : 0;

costIdleParams.TotalEmployeeCount = \_ctx.User.Count();

return costIdleParams;

}

/// <summary>

/// Обновление стоимости простоя сервиса

/// </summary>

/// <param name="serviceIdle">Обновленный экземпляр простоя сервиса</param>

public void UpdateServiceIdle(ServiceIdle serviceIdle)

{

\_ctx.ServiceIdle.Attach(serviceIdle);

\_ctx.Entry(serviceIdle).State = EntityState.Modified;

\_ctx.SaveChanges();

}

#endregion

#region Timer

/// <summary>

/// Проверка обслуживания конфигураций серверов

/// </summary>

public void CheckServerMaintenance()

{

#region Переводнаоблуживание

var activeTasks = \_ctx.Server\_MaintenanceShedule

.Include(sms => sms.Server)

.Where(sms => sms.BeginPeriod <= DateTime.Now && sms.EndPeriod > DateTime.Now)

.ToList();

foreach (Server\_MaintenanceShedule activeTask in activeTasks)

{

var ctxServer = \_ctx.Server.Single(s => s.Id == activeTask.ServerID);

ctxServer.Active = false;

ctxServer.OnMaintenance = true;

\_ctx.SaveChanges();

List<Service> services = GetServicesByServer(activeTask.Server);

foreach (Service service in services)

{

List<ServiceIdle> serviceIdles = service.ServiceIdle

.Where(s => s.BeginPeriod >= DateTime.Now && s.EndPeriod < DateTime.Now

&& s.IdleType.Name.Equals("Плановый")

&& s.IdleReason.Name.Equals("Плановоеобслуживание"))

.ToList();

if (serviceIdles.Count == 0 && !(service.ServiceState.Name.Equals("Отправленазаявка") || service.ServiceState.Name.Equals("Отклонен")))

{

var ctxService = \_ctx.Service.Single(s => s.Id == service.Id);

ctxService.ServiceIdle.Add(new ServiceIdle

{

BeginPeriod = activeTask.BeginPeriod,

EndPeriod = activeTask.EndPeriod,

Date = DateTime.Now,

IdleReason = \_ctx.IdleReason.Single(ir => ir.Name.Equals("Плановоеобслуживание")),

IdleType = \_ctx.IdleType.Single(it => it.Name.Equals("Плановый"))

});

ctxService.ServiceState = \_ctx.ServiceState.Single(ss => ss.Name.Equals("Неактивен"));

\_ctx.SaveChanges();

}

}

}

#endregion

#region Выводсобслуживания

var completeTasks = \_ctx.Server\_MaintenanceShedule

.Include(sms => sms.Server)

.Where(sms => sms.EndPeriod < DateTime.Now)

.ToList();

foreach (Server\_MaintenanceShedule activeTask in completeTasks)

{

var ctxServer = \_ctx.Server.Single(s => s.Id == activeTask.ServerID);

ctxServer.Active = true;

ctxServer.OnMaintenance = false;

\_ctx.SaveChanges();

List<Service> services = GetServicesByServer(activeTask.Server);

foreach (Service service in services)

{

var ctxService = \_ctx.Service.Single(s => s.Id == service.Id);

if (!(ctxService.ServiceState.Name.Equals("Отправленазаявка") || ctxService.ServiceState.Name.Equals("Отклонен")))

{

ctxService.ServiceState = \_ctx.ServiceState.Single(ss => ss.Name.Equals("Активен"));

\_ctx.SaveChanges();

}

}

}

#endregion

}

/// <summary>

/// Проверкаобслуживанияхранилищданных

/// </summary>

public void CheckSANMaintenance()

{

#region Переводнаоблуживание

var activeTasks = \_ctx.SAN\_MaintenanceShedule

.Include(sms => sms.SAN)

.Where(sms => sms.BeginPeriod <= DateTime.Now && sms.EndPeriod > DateTime.Now)

.ToList();

foreach (SAN\_MaintenanceShedule activeTask in activeTasks)

{

var ctxSan = \_ctx.SAN.Single(s => s.Id == activeTask.SAN\_Id);

ctxSan.Active = false;

ctxSan.OnMaintenance = true;

\_ctx.SaveChanges();

List<Service> services = GetServicesBySAN(activeTask.SAN);

foreach (Service service in services)

{

List<ServiceIdle> serviceIdles = service.ServiceIdle

.Where(s => s.BeginPeriod >= DateTime.Now && s.EndPeriod < DateTime.Now

&& s.IdleType.Name.Equals("Плановый")

&& s.IdleReason.Name.Equals("Плановоеобслуживание"))

.ToList();

if (serviceIdles.Count == 0 && !(service.ServiceState.Name.Equals("Отправленазаявка") || service.ServiceState.Name.Equals("Отклонен")))

{

var ctxService = \_ctx.Service.Single(s => s.Id == service.Id);

ctxService.ServiceIdle.Add(new ServiceIdle

{

BeginPeriod = activeTask.BeginPeriod,

EndPeriod = activeTask.EndPeriod,

Date = DateTime.Now,

IdleReason = \_ctx.IdleReason.Single(ir => ir.Name.Equals("Плановоеобслуживание")),

IdleType = \_ctx.IdleType.Single(it => it.Name.Equals("Плановый"))

});

ctxService.ServiceState = \_ctx.ServiceState.Single(ss => ss.Name.Equals("Неактивен"));

\_ctx.SaveChanges();

}

}

}

#endregion

#region Выводсобслуживания

var completeTasks = \_ctx.SAN\_MaintenanceShedule

.Include(sms => sms.SAN)

.Where(sms => sms.EndPeriod < DateTime.Now)

.ToList();

foreach (SAN\_MaintenanceShedule activeTask in completeTasks)

{

var ctxSan = \_ctx.SAN.Single(s => s.Id == activeTask.SAN\_Id);

ctxSan.Active = true;

ctxSan.OnMaintenance = false;

\_ctx.SaveChanges();

List<Service> services = GetServicesBySAN(activeTask.SAN);

foreach (Service service in services)

{

var ctxService = \_ctx.Service.Single(s => s.Id == service.Id);

if (!(ctxService.ServiceState.Name.Equals("Отправленазаявка") || ctxService.ServiceState.Name.Equals("Отклонен")))

{

ctxService.ServiceState = \_ctx.ServiceState.Single(ss => ss.Name.Equals("Активен"));

\_ctx.SaveChanges();

}

}

}

#endregion

}

#endregion

/// <summary>

/// Получение полного списка процессоров

/// </summary>

/// <returns>Списокпроцессоров</returns>

public BindingList<CPU> GetCPUs()

{

var lst = \_ctx.CPU

.Include(c => c.CpuSocket)

.ToList();

return new BindingList<CPU>(lst);

}

}

}

Модуль «CostIdleParams.cs»

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text;

using System.Threading.Tasks;

namespace MTF\_Services.WinForms.Data

{

/// <summary>

/// Параметры для расчета простоя сервисов

/// </summary>

[Serializable]

public class CostIdleParams

{

/// <summary>

/// Путькфайлу

/// </summary>

private const string fileName = "CostIdleParams.dat";

/// <summary>

/// Годовой валовой доход компании

/// </summary>

public decimal YearIncome { get; set; }

/// <summary>

/// Числоработников

/// </summary>

public int TotalEmployeeCount { get; set; }

/// <summary>

/// Средняя часовая оплата труда администратора

/// </summary>

public decimal AvgAdminSalary { get; set; }

/// <summary>

/// Средняя часовая оплата труда работника

/// </summary>

public decimal AvgEmployeeSalary { get; set; }

/// <summary>

/// Сохранение экземпляра параметров для расчета простоя сервисов в файл

/// </summary>

/// <param name="instance">Экземлярдлясохранения</param>

public static void Save(CostIdleParams instance)

{

var bf = new BinaryFormatter();

using (FileStream fs = new FileStream(fileName, FileMode.Create,FileAccess.Write))

bf.Serialize(fs, instance);

}

/// <summary>

/// Выгрузка экземпляра параметров для расчета простоя сервисов из файла

/// </summary>

/// <returns>Выгруженныйэкземпляр</returns>

public static CostIdleParams Load()

{

if (!File.Exists(fileName))

return null;

var bf = new BinaryFormatter();

CostIdleParams deserializedInstance = null;

using (FileStream fs = new FileStream(fileName, FileMode.Open,FileAccess.Read))

deserializedInstance = (CostIdleParams) bf.Deserialize(fs);

return deserializedInstance;

}

}

}

Модуль «ServiceIdleParams.cs»

using System;

namespace MTF\_Services.WinForms.Data

{

public class ServiceIdleParams

{

public int Id { get; set; }

public string Service { get; set; }

public string Platform { get; set; }

public DateTime Begin { get; set; }

public DateTime End { get; set; }

public string DurationType { get; set; }

public string DurationValue { get; set; }

public string IdleType { get; set; }

public string IdleReason { get; set; }

public int DisabledEmployees { get; set; }

public int UsedEmployees { get; set; }

public decimal ServerCost { get; set; }

public decimal LostCost { get; set; }

public decimal TotalCost { get; set; }

}

}

Модуль «FormMode.cs»

namespace MTF\_Services.WinForms.Extentions

{

/// <summary>

/// Режимформы

/// </summary>

public enum FormMode

{

Add,

Edit,

None

}

}

Модуль «ReportType.cs»

namespace MTF\_Services.WinForms.Extentions

{

public enum ReportType

{

ServiceRequest,

ServiceCost,

ServiceIdle

}

}

Модуль «SANPartSelectionMode.cs»

namespace MTF\_Services.WinForms.Extentions

{

public enum SANPartSelectionMode

{

OK,

None,

UnSupported

}

}

Модуль «ScheduleEditType.cs»

namespace MTF\_Services.WinForms.Extentions

{

public enum ScheduleEditType

{

Server,

SAN

}

}

Модуль «ScheduleItemsShowMode.cs»

namespace MTF\_Services.WinForms.Extentions

{

public enum ScheduleItemsTimeFormat

{

Hours,

Minutes

}

}

Модуль «ServerPartSelectionMode.cs»

namespace MTF\_Services.WinForms.Extentions

{

public enum ServerPartSelectionMode

{

OK,

None,

NotSelected,

UnSupported,

CPUNotSelected,

RAMNotSelected,

StorageNotSelected,

}

}

Модуль «DirectorForm.cs»

using System;

using System.ComponentModel;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

namespace MTF\_Services.WinForms.Forms.Director

{

/// <summary>

/// Класс главной формы директора.

/// </summary>

public partial class DirectorForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Флаг совершенного выхода из учетной записи

/// </summary>

private bool \_logout;

/// <summary>

/// Выбранная заявка на создание сервиса

/// </summary>

public ServiceRequestItem SelectedServiceRequest { get; set; }

/// <summary>

/// Выбранный простой сервиса

/// </summary>

public IdleItem SelectedIdleItem { get; set; }

/// <summary>

/// Конструктор главной формы директора.

/// </summary>

public DirectorForm()

{

InitializeComponent();

SubscribeMenuItems();

FormClosing += DirectorForm\_FormClosing;

\_ctx = new Context();

BindAll();

timer.Enabled = true;

}

/// <summary>

/// Обработчик события закрытия формы.

/// </summary>

private void DirectorForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (e.CloseReason == CloseReason.ApplicationExitCall || \_logout)

return;

var result = MessageBox.Show("Выберитедальнейшеедействие:" +

"\n\tДа - сменить пользователя;" +

"\n\tНет - закрыть приложение;" +

"\n\tОтмена - остаться в текущем окне",

"Выход из основного меню",

MessageBoxButtons.YesNoCancel,

MessageBoxIcon.Asterisk,

MessageBoxDefaultButton.Button3);

switch (result)

{

case DialogResult.Cancel:

e.Cancel = true;

break;

case DialogResult.Yes:

\_ctx.Logout();

Owner?.Show();

break;

case DialogResult.No:

Application.Exit();

break;

}

}

#region Bindings

/// <summary>

/// Привязкавсехколлекций

/// </summary>

private void BindAll()

{

BindServiceInfo();

BindServiceRequest();

BindServiceCost();

BindPlatformList();

BindUsersInformation();

BindServiceIdle();

}

/// <summary>

/// Привязка коллекции с обзором сервисов

/// </summary>

private void BindServiceInfo()

{

serviceInfoBindingSource.DataSource = \_ctx.GetServiceInfo();

dg\_ServiceInfo.DataSource = serviceInfoBindingSource;

dg\_ServiceInfo.Refresh();

}

/// <summary>

/// Привязка коллекции с заявками на создание сервиса

/// </summary>

private void BindServiceRequest()

{

serviceRequestItemBindingSource.DataSource = \_ctx.GetServiceReuqests();

dg\_ServiceRequest.DataSource = serviceRequestItemBindingSource;

dg\_ServiceRequest.Refresh();

}

/// <summary>

/// Получение коллекции со стоимостью сервисов

/// </summary>

private void BindServiceCost()

{

serviceDetailInfoBindingSource.DataSource = \_ctx.GetServicesCost();

dg\_ServiceDetailInfo.DataSource = serviceDetailInfoBindingSource;

dg\_ServiceDetailInfo.Refresh();

}

/// <summary>

/// Инициализация привязки списка платформ с информацией о количестве пользователей и сервисов

/// </summary>

private void BindPlatformList()

{

platformServiceUserBindingSource.DataSource = \_ctx.GetPlatformServiceUsers();

dg\_Platforms.DataSource = platformServiceUserBindingSource;

dg\_Platforms.Refresh();

}

/// <summary>

/// Привязка списка с информацией пользователей

/// </summary>

private void BindUsersInformation()

{

userInfoBindingSource.DataSource = \_ctx.GetUsersInfo();

dg\_Users.DataSource = userInfoBindingSource;

dg\_Users.Refresh();

}

/// <summary>

/// Привязка списка с информацией о простое сервисов

/// </summary>

private void BindServiceIdle()

{

idleItemBindingSource.DataSource = \_ctx.GetIdleItems();

dg\_Idle.DataSource = idleItemBindingSource;

dg\_Idle.Refresh();

}

#endregion

/// <summary>

/// Обработчик события выхода курсора из элемента главного меню,

/// который очищает область в строке состояния.

/// </summary>

private void MenuItem\_MouseLeave(object sender, EventArgs e)

{

tip\_Label.Text = string.Empty;

}

/// <summary>

/// Обработчик события навдения курсора на элемент главного меню,

/// который выводит подсказку элемента в строку состояния.

/// </summary>

private void MenuItem\_MouseEnter(object sender, EventArgs e)

{

tip\_Label.Text = ((ToolStripItem)sender).ToolTipText;

}

/// <summary>

/// Подписка элементов главного меню и панели элементов на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

private void SubscribeMenuItems()

{

SubScribeChildMenuItems(menuBar.Items);

foreach (ToolStripItem toolBarControl in toolBar.Items)

{

toolBarControl.MouseEnter += MenuItem\_MouseEnter;

toolBarControl.MouseLeave += MenuItem\_MouseLeave;

}

}

/// <summary>

/// Подписка дочерних элементов главного меню на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

/// <param name="col">Коллекция элементов главного меню</param>

private void SubScribeChildMenuItems(ToolStripItemCollection col)

{

for (int i = 0; i < col.Count; i++)

{

var item = col[i] as ToolStripMenuItem;

if (item != null)

{

item.MouseEnter += MenuItem\_MouseEnter;

item.MouseLeave += MenuItem\_MouseLeave;

if (item.DropDownItems.Count > 0)

SubScribeChildMenuItems(item.DropDownItems);

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который открывает диалоговое окно формы просмотра и редактирования текущей учетной записи

/// </summary>

private void просмотрПрофиляToolStripMenuItem\_Click(object sender, EventArgs e)

{

new ShowUserProfileForm().ShowDialog();

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который совершает выход из текущей учетной записи

/// </summary>

private void сменитьПользователяToolStripMenuItem\_Click(object sender, EventArgs e)

{

var result = MessageBox.Show("Будет произведен выход из текущей учетной записи! Продолжить?",

"Смена пользователя",

MessageBoxButtons.YesNo,

MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

\_ctx.Logout();

Owner?.Show();

\_logout = true;

this.Close();

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который завершает работу приложения

/// </summary>

private void завершениеРаботыToolStripMenuItem\_Click(object sender, EventArgs e)

{

var result = MessageBox.Show("Приложениебудетзакрыто! Продолжить?",

"Закрытиеприложения",

MessageBoxButtons.YesNo,

MessageBoxIcon.Exclamation);

if (result == DialogResult.Yes)

Application.Exit();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы обработки выбранной заявки

/// </summary>

private void обработкаВыбраннойЗаявкиToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (SelectedServiceRequest == null)

{

MessageBox.Show("Выберитезаявкунасозданиесервиса", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

try

{

Service selectedService = \_ctx.GetServiceByID(SelectedServiceRequest.Id);

var serviceRequestTreatment = new ServiceRequestTreatment(selectedService, SelectedServiceRequest);

if (serviceRequestTreatment.ShowDialog() == DialogResult.OK)

BindAll();

}

catch

{

MessageBox.Show("Произошла ошибка при получении информации о выбранном сервисе!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных заявок на создание сервиса

/// </summary>

private void serviceRequestItemBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as ServiceRequestItem;

if (selectedItem != null)

SelectedServiceRequest = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно предварительного просмотра отчета по стоимости предоставления сервисов

/// </summary>

private void стоимостьПредоставленияСервисовToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

var headerText = dg\_ServiceDetailInfo.Columns[5].HeaderText;

BindingList<ServiceDetailInfo> sdi = (BindingList<ServiceDetailInfo>) serviceDetailInfoBindingSource.DataSource;

var reportingForm = new ReportingForm(sdi, headerText);

reportingForm.ShowDialog();

}

catch

{

MessageBox.Show("Произошлаошибкаприсозданииотчета!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных заявок простоев сервисов

/// </summary>

private void idleItemBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as IdleItem;

if (selectedItem != null)

SelectedIdleItem = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы редактирования выбранного простоя

/// </summary>

private void редактированиеВыбранногоПростояToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (SelectedIdleItem == null)

{

MessageBox.Show("Выберитепростойсервиса", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

try

{

ServiceIdle selectedServiceIdle = \_ctx.GetServiceIdleByID(SelectedIdleItem.Id);

var editIdleItemForm = new EditIdleItemForm(selectedServiceIdle);

if (editIdleItemForm.ShowDialog() == DialogResult.OK)

BindServiceIdle();

}

catch

{

MessageBox.Show("Произошла ошибка при получении информации о выбранном простое сервиса!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события выполнения такта таймера

/// </summary>

private void timer\_Tick(object sender, EventArgs e)

{

if (\_ctx == null)

return;

try

{

\_ctx.CheckServerMaintenance();

\_ctx.CheckSANMaintenance();

BindAll();

}

catch

{

}

}

}

}

Модуль «EditIdleItemForm.cs»

using System;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.Director

{

/// <summary>

/// Класс формы расчета стоимости простоя сервиса

/// </summary>

public partial class EditIdleItemForm : Form

{

// Минутвгоду

private const int minutesInYear = 525600;

// Редактируемыйэкземплярпростоя

private ServiceIdle \_serviceIdle;

// Кол-во отключенных пользователей

private int \_disabledUsersCount;

// Кол-во задействованных администраторов

private int \_usedAdminsCount;

// Продолжительность (мин.)

private double \_duration;

// Плановые расходы на администраторов

private decimal \_adminPlannedCost;

// Плановые расходы на конечных пользователей

private decimal \_employeePlannedCost;

// Плановые расходы на отключение сервера

private decimal \_servicePlannedCost;

// Потерянныйдоход

private decimal \_brokenIncome;

// Итоговая сумма простоя

private decimal \_total;

// Доходвминуту

private decimal \_incomeInMinute;

// Средний доход администратора в минуту

private decimal \_avgAdminSalaryPerMin;

// Средний доход работника в минуту

private decimal \_avgEmployeeSalaryPerMin;

// Доход на каждого работника (руб./мин)

private decimal \_incomePerEmployee;

/// <summary>

/// Объектдлядоступакконтектуданных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Конструктор формы расчета стоимости простоя сервиса

/// </summary>

/// <param name="selectedServiceIdle">Выбранныйэкземплярпростоя</param>

public EditIdleItemForm(ServiceIdle selectedServiceIdle)

{

\_serviceIdle = selectedServiceIdle;

InitializeComponent();

\_ctx = new Context();

btn\_OK.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

InitBindings(selectedServiceIdle);

comboBox\_DurationType.SelectedIndex = 0;

}

/// <summary>

/// Инициализацияпривязок

/// </summary>

private void InitBindings(ServiceIdle serviceIdle)

{

serviceIdleBindingSource.DataSource = serviceIdle;

serviceIdleBindingSource.ResumeBinding();

// длительностьпростоя

\_duration = (serviceIdle.EndPeriod - serviceIdle.BeginPeriod).TotalMinutes;

// количество пользователей, отключенных при этом

\_disabledUsersCount = serviceIdle.Service.User.Count;

// количество администраторов, задействованных для этого

\_usedAdminsCount = serviceIdle.User.Count;

// Годовойдоходкомпанни

CostIdleParams idleParams = null;

try

{

// Попытказагрузкиизфайла

idleParams = CostIdleParams.Load();

// При отсутсвтсвии файла данные берутся из БД

if (idleParams == null)

idleParams = \_ctx.GetIdleParamsFormDatabase();

}

catch

{

// в случае ошибки данные берутся из БД

idleParams = \_ctx.GetIdleParamsFormDatabase();

}

finally

{

costIdleParamsBindingSource.DataSource = idleParams;

costIdleParamsBindingSource.ResumeBinding();

}

CalcIdleCost();

}

/// <summary>

/// Расчетстоимостипростоя

/// </summary>

private void CalcIdleCost()

{

var idleParams = costIdleParamsBindingSource.DataSource as CostIdleParams;

if (idleParams == null)

return;

// Доходвминуту

\_incomeInMinute = idleParams.YearIncome / minutesInYear;

// Среднийдоходадминистраторавминуту

\_avgAdminSalaryPerMin = idleParams.AvgAdminSalary / 60;

// Среднийдоходработникавминуту

\_avgEmployeeSalaryPerMin = idleParams.AvgEmployeeSalary / 60;

// Доходнакаждогоработника (руб./мин)

\_incomePerEmployee = \_incomeInMinute / idleParams.TotalEmployeeCount;

// Плановыерасходынаадминистраторов (руб./мин.)

\_adminPlannedCost = (decimal)\_duration \* \_usedAdminsCount \* \_avgAdminSalaryPerMin;

// Плановые расходы на конечных пользователей (руб./мин.)

\_employeePlannedCost = (decimal)\_duration \* \_disabledUsersCount \* \_avgEmployeeSalaryPerMin;

// Плановыерасходынаотключениесервера

\_servicePlannedCost = \_adminPlannedCost + \_employeePlannedCost;

// Потерянныйдоход (руб./мин.)

\_brokenIncome = \_incomePerEmployee \* (decimal)\_duration;

// Итоговаястоимостьпростоя

\_total = \_servicePlannedCost + \_brokenIncome;

UpdateTextFields();

}

/// <summary>

/// Обновлениетекстовыхполей

/// </summary>

private void UpdateTextFields()

{

txt\_ServiceName.Text = \_serviceIdle.Service.ServiceType.Name;

txt\_PlarformName.Text = \_serviceIdle.Service.PaasType.Name;

txt\_IdleTypeName.Text = \_serviceIdle.IdleType.Name;

txt\_IdleReasonName.Text = \_serviceIdle.IdleReason.Name;

switch (comboBox\_DurationType.SelectedIndex)

{

case 0:

txt\_Duration.Text = \_duration.ToString();

break;

case 1:

var hours = Math.Round(\_duration / 60);

var mins = Math.Round(\_duration % 60);

txt\_Duration.Text = $"{hours:##}:{(mins > 0 ? $"{mins:##}": "00")}";

break;

}

txt\_IncomePerMinute.Text = \_incomeInMinute.ToString("C");

txt\_incomePerEmployee.Text = \_incomePerEmployee.ToString("C");

txt\_disabledUsersCount.Text = \_disabledUsersCount.ToString();

txt\_usedAdminsCount.Text = \_usedAdminsCount.ToString();

txt\_servicePlannedCost.Text = \_servicePlannedCost.ToString("C");

txt\_brokenIncome.Text = \_brokenIncome.ToString("C");

txt\_Total.Text = \_total.ToString("C");

}

/// <summary>

/// Изменение значения полей

/// </summary>

private void textBox9\_TextChanged(object sender, EventArgs e)

{

if (!string.IsNullOrEmpty(textBox9.Text))

CalcIdleCost();

}

/// <summary>

/// Обработчик события изменения текущего элемента в списке параметров простоя

/// </summary>

private void costIdleParamsBindingSource\_CurrentItemChanged(object sender, EventArgs e)

{

CalcIdleCost();

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который осуществляет отмену изменений в параметрах простоя, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который осуществляет сохранение изменений в параметрах простоя

/// </summary>

private void btn\_OK\_Click(object sender, EventArgs e)

{

var costIdleParams = costIdleParamsBindingSource.DataSource as CostIdleParams;

if (costIdleParams == null)

{

MessageBox.Show("Неудалосьполучитьпараметрыдлярасчетапростоя!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

if (costIdleParams.YearIncome <= 0)

{

MessageBox.Show("Введите годовой валовой доход компании для расчета стоимости простоя!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

return;

}

if (costIdleParams.TotalEmployeeCount <= 0)

{

MessageBox.Show("Введите общее число сотрудников для расчета стоимости простоя!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

return;

}

if (costIdleParams.TotalEmployeeCount < (\_disabledUsersCount + \_usedAdminsCount))

{

MessageBox.Show("Общее число сотрудников должно быть больше количества отключенных и задействованных!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

return;

}

if (costIdleParams.AvgAdminSalary <= 0)

{

MessageBox.Show("Введите размер средней часовой оплаты труда администратора для расчета стоимости простоя!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

return;

}

if (costIdleParams.AvgEmployeeSalary <= 0)

{

MessageBox.Show("Введите размер средней часовой оплаты труда работника для расчета стоимости простоя!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

return;

}

if (\_total <= 0)

{

MessageBox.Show("Не расчитана итоговая стоимость простоя!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

return;

}

var result = MessageBox.Show("Полученная стоимость будет сохранена! Продолжить?", "Сохранениеизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Information);

if (result == DialogResult.No)

return;

try

{

CostIdleParams.Save(costIdleParams);

}

catch

{

result = MessageBox.Show("Не удалось сохранить параметры для расчета! " +

"\nПри новом расчете простоя придется снова ввести одовой валовой доход компании!" +

"\nПродолжить сохранение?", "Ошибка", MessageBoxButtons.YesNo,

MessageBoxIcon.Error);

if (result == DialogResult.No)

return;

}

try

{

\_serviceIdle.Cost = \_total;

\_ctx.UpdateServiceIdle(\_serviceIdle);

DialogResult = DialogResult.OK;

MessageBox.Show("Полученный расчет простоя успешно сохранен!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

Close();

}

catch

{

MessageBox.Show("Не удалось сохранить полученные расчеты!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события изменения индекса формата отображения времени

/// </summary>

private void comboBox\_DurationType\_SelectedIndexChanged(object sender, EventArgs e)

{

UpdateTextFields();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы распределения задействованного персонала

/// </summary>

private void распределениеПерсоналаToolStripMenuItem\_Click(object sender, EventArgs e)

{

var staffDistributonForm = new StaffDistributonForm(\_serviceIdle);

if (staffDistributonForm.ShowDialog() == DialogResult.OK)

{

DialogResult = DialogResult.OK;

\_serviceIdle = staffDistributonForm.SelectedServiceIdle;

// количество администраторов, задействованных для этого

\_usedAdminsCount = \_serviceIdle.User.Count;

CalcIdleCost();

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который открывает диалоговое окно предварительного просмотра отчета по текущему простою

/// </summary>

private void btn\_PrintReport\_Click(object sender, EventArgs e)

{

var costIdleParams = costIdleParamsBindingSource.DataSource as CostIdleParams;

if (costIdleParams == null)

{

MessageBox.Show("Неудалосьполучитьпараметрыдлярасчетапростоя!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

if (costIdleParams.YearIncome <= 0)

{

MessageBox.Show("Введите годовой валовой доход компании для расчета стоимости простоя!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

return;

}

if (costIdleParams.TotalEmployeeCount <= 0)

{

MessageBox.Show("Введите общее число сотрудников для расчета стоимости простоя!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

return;

}

if (costIdleParams.TotalEmployeeCount < (\_disabledUsersCount + \_usedAdminsCount))

{

MessageBox.Show("Общее число сотрудников должно быть больше количества отключенных и задействованных!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

return;

}

if (costIdleParams.AvgAdminSalary <= 0)

{

MessageBox.Show("Введите размер средней часовой оплаты труда администратора для расчета стоимости простоя!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

return;

}

if (costIdleParams.AvgEmployeeSalary <= 0)

{

MessageBox.Show("Введите размер средней часовой оплаты труда работника для расчета стоимости простоя!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

return;

}

if (\_total <= 0)

{

MessageBox.Show("Не расчитана итоговая стоимость простоя!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

return;

}

var sip = new ServiceIdleParams();

sip.Id = \_serviceIdle.Id;

sip.Service = \_serviceIdle.Service.ServiceType.Name;

sip.Platform = \_serviceIdle.Service.PaasType.Name;

sip.Begin = \_serviceIdle.BeginPeriod;

sip.End = \_serviceIdle.EndPeriod;

sip.DurationType = comboBox\_DurationType.Text;

sip.DurationValue = txt\_Duration.Text;

sip.IdleType = \_serviceIdle.IdleType.Name;

sip.IdleReason = \_serviceIdle.IdleReason.Name;

sip.DisabledEmployees = \_disabledUsersCount;

sip.UsedEmployees = \_disabledUsersCount;

sip.ServerCost = \_servicePlannedCost;

sip.LostCost = \_brokenIncome;

sip.TotalCost = \_total;

var reportingForm = new ReportingForm(sip);

reportingForm.ShowDialog();

}

}

}

Модуль «ServiceRequestTreatment.cs»

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.Director

{

/// <summary>

/// Класс формы офбработки заявки на предоставление сервиса

/// </summary>

public partial class ServiceRequestTreatment : Form

{

/// <summary>

/// Текущийсервер

/// </summary>

private readonly Service \_selectedService;

private readonly ServiceRequestItem \_selectedServiceRequest;

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

public List<SoftwareInfo> InstalledSoftware { get; set; }

/// <summary>

/// Конструктор формы обработки заявки на предоставление сервиса

/// </summary>

public ServiceRequestTreatment(Service selectedService, ServiceRequestItem selectedServiceRequest)

{

\_selectedService = selectedService;

\_selectedServiceRequest = selectedServiceRequest;

InitializeComponent();

btn\_OK.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

btn\_PrintReport.Image = new Bitmap(Resources.printers\_and\_faxes, new Size(30, 20));

\_ctx = new Context();

UpdateTextFields();

comboBox\_PriceType.SelectedIndex = 0;

}

/// <summary>

/// Обновление текстовых полей формы

/// </summary>

private void UpdateTextFields()

{

InstalledSoftware = new List<SoftwareInfo>();

txt\_Name.Text = \_selectedService.ServiceType.Name;

txt\_Platform.Text = \_selectedService.PaasType.Name;

txt\_CreateDate.Text = \_selectedService.CreateDate.Value.ToString("d");

txt\_Price.Text = \_selectedService.CostPerHour.Value.ToString();

txt\_CoreCount.Text = \_selectedService.CoreCount.ToString();

txt\_RamValue.Text = \_selectedService.RamCount.Value.ToString();

txt\_HddVolume.Text = \_selectedService.HDDVolume.Value.ToString();

foreach (var software in \_selectedService.Software)

{

InstalledSoftware.Add(new SoftwareInfo

{

Id = software.Id,

Software = software.Name,

SoftType = software.SoftType.Name,

Cost = software.Cost

});

}

softwareInfoBindingSource.DataSource = InstalledSoftware;

dg\_Software.DataSource = softwareInfoBindingSource;

}

/// <summary>

/// Обработчик события изменения индекса типа вывода стоимости

/// </summary>

private void comboBox\_PriceType\_SelectedIndexChanged(object sender, EventArgs e)

{

if (\_selectedService == null)

return;

decimal initialPrice = \_selectedService.CostPerHour.Value;

switch (comboBox\_PriceType.SelectedIndex)

{

case 0:

txt\_Price.Text = initialPrice.ToString("##.00");

break;

case 1:

txt\_Price.Text = (initialPrice \* 720).ToString("##.00");

break;

case 2:

txt\_Price.Text = (initialPrice \* (720 \* 3)).ToString("##.00");

break;

case 3:

txt\_Price.Text = (initialPrice \* (720 \* 12)).ToString("##.00");

break;

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену заявки на создание сервиса, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

var result = MessageBox.Show("Заявка на создание сервиса будет отклонена! Продолжить?", "Потверждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Information);

if (result == DialogResult.Yes)

{

try

{

\_ctx.CancelServiceRequest(\_selectedService);

Close();

}

catch

{

MessageBox.Show("Произошлаошибкаприотменезаявки!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит подтверждение заявки на создание сервиса

/// </summary>

private void btn\_OK\_Click(object sender, EventArgs e)

{

var result = MessageBox.Show("Заявка на создание сервиса будет принята! Продолжить?", "Потверждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Information);

if (result == DialogResult.Yes)

{

try

{

\_ctx.ConfirmServiceRequest(\_selectedService);

Close();

}

catch

{

MessageBox.Show("Произошлаошибкаприпринятиизаявки!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который открывает диалоговое окно предварительного просмотра отчета заявки

/// </summary>

private void btn\_PrintReport\_Click(object sender, EventArgs e)

{

var reportingForm = new ReportingForm(\_selectedServiceRequest, InstalledSoftware);

reportingForm.ShowDialog();

}

}

}

Модуль «StaffDistributonForm.cs»

using System;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.Director

{

/// <summary>

/// Класс формы рапсределения персонала, задействованного в простое сервиса

/// </summary>

public partial class StaffDistributonForm : Form

{

/// <summary>

/// Выбранный экземпляр простоя сервиса

/// </summary>

public ServiceIdle SelectedServiceIdle { get; set; }

/// <summary>

/// Доступныйперсонал

/// </summary>

public BindingList<User> AvaliblePersonal { get; set; }

/// <summary>

/// Задействованныйперсонал

/// </summary>

public BindingList<User> UsedPersonal { get; set; }

/// <summary>

/// Выбранный доступный сотрудник

/// </summary>

public User SelectedAvalibleUser { get; set; }

/// <summary>

/// Выбранный задействованный сотрудник

/// </summary>

public User SelectedUsedUser { get; set; }

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Конструктор формы распределения персонала, задействованного в простое сервиса

/// </summary>

/// <param name="selectedServiceIdle">Выбранныйэкземплярпростоя</param>

public StaffDistributonForm(ServiceIdle selectedServiceIdle)

{

SelectedServiceIdle = selectedServiceIdle;

InitializeComponent();

btn\_OK.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

\_ctx = new Context();

BindCollections();

}

/// <summary>

/// Привязкаколлекций

/// </summary>

private void BindCollections()

{

var adminsList = \_ctx.GetAdminsList();

var users = SelectedServiceIdle.User.ToList();

users.ForEach(usr =>

{

if (adminsList.Contains(usr))

adminsList.Remove(usr);

});

AvaliblePersonal = new BindingList<User>(adminsList);

UsedPersonal = new BindingList<User>(users);

AvalibleUserBindingSource.DataSource = AvaliblePersonal;

UsedUserBindingSource.DataSource = UsedPersonal;

dg\_AvalibleUser.DataSource = AvalibleUserBindingSource;

dg\_UsedUser.DataSource = UsedPersonal;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных доступного персонала

/// </summary>

private void AvalibleUserBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as User;

if (selectedItem != null)

SelectedAvalibleUser = selectedItem;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных задействованного персонала

/// </summary>

private void UsedUserBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as User;

if (selectedItem != null)

SelectedUsedUser = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который осуществляет удаление из выбранных сервисов

/// </summary>

private void btn\_RemoveFromUsed\_Click(object sender, EventArgs e)

{

if (SelectedUsedUser == null || dg\_UsedUser.CurrentRow != null)

{

MessageBox.Show("Пользователь не выбран!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

AvaliblePersonal.Add(SelectedUsedUser);

UsedPersonal.Remove(SelectedUsedUser);

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который осуществляет добавление к выбранным сервисам

/// </summary>

private void btn\_AddToUsed\_Click(object sender, EventArgs e)

{

if (SelectedAvalibleUser == null || dg\_AvalibleUser.CurrentRow != null)

{

MessageBox.Show("Пользователь не выбран!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

UsedPersonal.Add(SelectedAvalibleUser);

AvaliblePersonal.Remove(SelectedAvalibleUser);

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит отмену подписки на новые сервисы, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который выполняет сохранение изменений

/// </summary>

private void btn\_OK\_Click(object sender, EventArgs e)

{

if (UsedPersonal.Count == 0)

{

MessageBox.Show("Добавьтекзадействованномуперсоналусотурдников", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

try

{

SelectedServiceIdle.User.Clear();

UsedPersonal.ToList().ForEach(up => SelectedServiceIdle.User.Add(up));

\_ctx.UpdateServiceIdle(SelectedServiceIdle);

DialogResult = DialogResult.OK;

MessageBox.Show("Изменения успешно сохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

Close();

}

catch

{

MessageBox.Show("Не удалось сохранить изменения!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void StaffDistributonForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (DialogResult != DialogResult.OK)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.No)

e.Cancel = true;

}

}

}

}

Модуль «EmployeeForm.cs»

using System;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

namespace MTF\_Services.WinForms.Forms.Employee

{

/// <summary>

/// Класс главной формы сотрудника.

/// </summary>

public partial class EmployeeForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Флаг совершенного выхода из учетной записи

/// </summary>

private bool \_logout;

/// <summary>

/// Выбранныйсервис

/// </summary>

public UserService SelectedUserService { get; set; }

/// <summary>

/// Конструктор главной формы сотрудника.

/// </summary>

public EmployeeForm()

{

InitializeComponent();

SubscribeMenuItems();

FormClosing += EmployeeForm\_FormClosing;

\_ctx = new Context();

BindUsedServices();

timer.Enabled = true;

}

/// <summary>

/// Обработчик события закрытия формы.

/// </summary>

private void EmployeeForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (e.CloseReason == CloseReason.ApplicationExitCall || \_logout)

return;

var result = MessageBox.Show("Выберитедальнейшеедействие:" +

"\n\tДа - сменить пользователя;" +

"\n\tНет - закрыть приложение;" +

"\n\tОтмена - остаться в текущем окне",

"Выход из основного меню",

MessageBoxButtons.YesNoCancel,

MessageBoxIcon.Asterisk,

MessageBoxDefaultButton.Button3);

switch (result)

{

case DialogResult.Cancel:

e.Cancel = true;

break;

case DialogResult.Yes:

\_ctx.Logout();

Owner?.Show();

break;

case DialogResult.No:

Application.Exit();

break;

}

}

/// <summary>

/// Обработчик события выхода курсора из элемента главного меню,

/// который очищает область в строке состояния.

/// </summary>

private void MenuItem\_MouseLeave(object sender, EventArgs e)

{

tip\_Label.Text = string.Empty;

}

/// <summary>

/// Обработчик события навдения курсора на элемент главного меню,

/// который выводит подсказку элемента в строку состояния.

/// </summary>

private void MenuItem\_MouseEnter(object sender, EventArgs e)

{

tip\_Label.Text = ((ToolStripItem)sender).ToolTipText;

}

/// <summary>

/// Подписка элементов главного меню и панели элементов на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

private void SubscribeMenuItems()

{

SubScribeChildMenuItems(menuBar.Items);

foreach (ToolStripItem toolBarControl in toolBar.Items)

{

toolBarControl.MouseEnter += MenuItem\_MouseEnter;

toolBarControl.MouseLeave += MenuItem\_MouseLeave;

}

}

/// <summary>

/// Подписка дочерних элементов главного меню на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

/// <param name="col">Коллекция элементов главного меню</param>

private void SubScribeChildMenuItems(ToolStripItemCollection col)

{

for (int i = 0; i < col.Count; i++)

{

var item = col[i] as ToolStripMenuItem;

if (item != null)

{

item.MouseEnter += MenuItem\_MouseEnter;

item.MouseLeave += MenuItem\_MouseLeave;

if (item.DropDownItems.Count > 0)

SubScribeChildMenuItems(item.DropDownItems);

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который открывает диалоговое окно формы просмотра и редактирования текущей учетной записи

/// </summary>

private void просмотрПрофиляToolStripMenuItem\_Click(object sender, EventArgs e)

{

new ShowUserProfileForm().ShowDialog();

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который совершает выход из текущей учетной записи

/// </summary>

private void сменитьПользователяToolStripMenuItem\_Click(object sender, EventArgs e)

{

var result = MessageBox.Show("Будет произведен выход из текущей учетной записи! Продолжить?",

"Смена пользователя",

MessageBoxButtons.YesNo,

MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

\_ctx.Logout();

Owner?.Show();

\_logout = true;

this.Close();

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который завершает работу приложения

/// </summary>

private void завершениеРаботыToolStripMenuItem\_Click(object sender, EventArgs e)

{

var result = MessageBox.Show("Приложениебудетзакрыто! Продолжить?",

"Закрытиеприложения",

MessageBoxButtons.YesNo,

MessageBoxIcon.Exclamation);

if (result == DialogResult.Yes)

Application.Exit();

}

/// <summary>

/// Привязка коллекции используемых сервисов сотрудника

/// </summary>

private void BindUsedServices()

{

userServiceBindingSource.DataSource = \_ctx.GetUsedServices();

dg\_Service.DataSource = userServiceBindingSource;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы подписки на новые сервисы

/// </summary>

private void подписатьсяНаНовыйСервисToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

var avalibleServiceTypes = \_ctx.GetAvalibleServiceTypes();

if (avalibleServiceTypes.Count == 0)

{

MessageBox.Show("Отсутствуют предоставляемые сервисы на данный момент!", "Предепреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

var subscribeServicesForm = new SubscribeServicesForm(avalibleServiceTypes);

if (subscribeServicesForm.ShowDialog() == DialogResult.OK)

BindUsedServices();

}

catch

{

MessageBox.Show("Произошла ошибка при получении списка доступных сервисов!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который совершает отписку от выбранного сервиса

/// </summary>

private void отписатьсяОтВыбранногоСервисаToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (SelectedUserService == null)

{

MessageBox.Show("Выберитесервисдляотписки!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

var result = MessageBox.Show("Выбудетеотписанныотвыбранногосервиса! Продолжить?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.Yes)

{

try

{

ServiceType serviceType = \_ctx.GetServiceTypeById(SelectedUserService.ServiceTypeId);

\_ctx.UnSubscribeSelectedService(serviceType);

MessageBox.Show("Измененияуспешносохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

BindUsedServices();

}

catch

{

MessageBox.Show("Произошлаошибкаприотпискеотвыбранногосервиса!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных подписанных серверов

/// </summary>

private void userServiceBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as UserService;

if (selectedItem != null)

SelectedUserService = selectedItem;

}

/// <summary>

/// Обработчик события выполнения такта таймера

/// </summary>

private void timer\_Tick(object sender, EventArgs e)

{

if (\_ctx == null)

return;

try

{

\_ctx.CheckServerMaintenance();

\_ctx.CheckSANMaintenance();

BindUsedServices();

}

catch

{

}

}

}

}

Модуль «SubscribeServicesForm.cs»

using System;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.Employee

{

/// <summary>

/// Класс формы подписки сотрудника на новые сервисы

/// </summary>

public partial class SubscribeServicesForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Доступныесервисы

/// </summary>

public BindingList<ServiceType> AvalibleServices { get; set; }

/// <summary>

/// Выбранные сервисы на подписку

/// </summary>

public BindingList<ServiceType> UsedServices { get; set; }

/// <summary>

/// Выбранный использованный сервис

/// </summary>

public ServiceType SelectedUsedServiceType { get; set; }

/// <summary>

/// Выбранный доступный сервис

/// </summary>

public ServiceType SelectedAvalibleServiceType { get; set; }

/// <summary>

/// Конструктор формы подписки сотрудника на новые сервисы

/// </summary>

/// <param name="avalibleServiceTypes">Коллекциядоступныхтиповсервисов</param>

public SubscribeServicesForm(BindingList<ServiceType> avalibleServiceTypes)

{

InitializeComponent();

btn\_OK.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

\_ctx = new Context();

BindCollections(avalibleServiceTypes);

}

/// <summary>

/// Привязкаколлекций

/// </summary>

private void BindCollections(BindingList<ServiceType> avalibleServiceTypes)

{

AvalibleServices = avalibleServiceTypes;

AvalibleServiceTypeBS.DataSource = avalibleServiceTypes;

lst\_AvalibleServices.DataSource = AvalibleServiceTypeBS;

UsedServices = new BindingList<ServiceType>();

UsedServiceTypeBS.DataSource = UsedServices;

lst\_UsedServices.DataSource = UsedServiceTypeBS;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных доступных сервисов

/// </summary>

private void AvalibleServiceTypeBS\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as ServiceType;

if (selectedItem != null)

SelectedAvalibleServiceType = selectedItem;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных выбранных сервисов

/// </summary>

private void UsedServiceTypeBS\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as ServiceType;

if (selectedItem != null)

SelectedUsedServiceType = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который совершает добавление к выбранным сервисам

/// </summary>

private void btn\_AddToUsed\_Click(object sender, EventArgs e)

{

if (SelectedAvalibleServiceType == null || lst\_AvalibleServices.SelectedIndex < 0)

{

MessageBox.Show("Доступный сервис не выбран!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

UsedServices.Add(SelectedAvalibleServiceType);

AvalibleServices.Remove(SelectedAvalibleServiceType);

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который совершает удаление из выбранных сервисов

/// </summary>

private void btn\_RemoveFromUsed\_Click(object sender, EventArgs e)

{

if (SelectedUsedServiceType == null || lst\_UsedServices.SelectedIndex < 0)

{

MessageBox.Show("Используемый сервис не выбран!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

AvalibleServices.Add(SelectedUsedServiceType);

UsedServices.Remove(SelectedUsedServiceType);

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит отмену изменений, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void SubscribeServicesForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (UsedServices.Count > 0 && DialogResult != DialogResult.OK)

{

var result = MessageBox.Show("Изменения не будут сохранены! Продолжить?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который выполняет сохранение изменений

/// </summary>

private void btn\_OK\_Click(object sender, EventArgs e)

{

if (UsedServices.Count == 0)

{

MessageBox.Show("Отсутствуютвыбранныесервисы!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

try

{

\_ctx.SubscribeToSelectedServices(UsedServices.ToList());

UsedServices.Clear();

MessageBox.Show("Новыеподпискинасервисыуспешносохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

DialogResult = DialogResult.OK;

Close();

}

catch

{

MessageBox.Show("Неудалосьподписатьсянавыбранныесервисы!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

Модуль «EditScheduleItem.cs»

using System;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.EquipmentIdle

{

/// <summary>

/// Класс формы редактирования позиции расписания обслуживания

/// </summary>

public partial class EditScheduleItem : Form

{

private readonly SAN \_selectedSan;

private readonly Server \_selectedServer;

private readonly SAN\_MaintenanceShedule \_sanMaintenanceShedule;

private readonly Server\_MaintenanceShedule \_serverMaintenanceShedule;

/// <summary>

/// Тип редактируемого расписания

/// </summary>

private readonly ScheduleEditType \_scheduleEditType;

/// <summary>

/// Режимиспользованияформы

/// </summary>

private readonly FormMode \_formMode;

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Дата начала обслуживания

/// </summary>

private DateTime \_beginDateTime;

/// <summary>

/// Датаокончанияобслуживания

/// </summary>

private DateTime \_endDateTime;

/// <summary>

/// Конструктор формы редактирования позиции расписания обслуживания

/// </summary>

private EditScheduleItem(ScheduleEditType scheduleEditType)

{

\_scheduleEditType = scheduleEditType;

InitializeComponent();

\_ctx = new Context();

btn\_OK.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

comboBox\_DurationType.SelectedIndex = 0;

dateTimePicker\_Begin.MinDate = DateTime.Now;

dateTimePicker\_End.MinDate = DateTime.Now;

numericUpDown\_BeginHours.Minimum = DateTime.Now.Hour;

numericUpDown\_BeginMinutes.Minimum = DateTime.Now.Minute;

numericUpDown\_EndHours.Minimum = DateTime.Now.Hour;

numericUpDown\_EndMinutes.Minimum = DateTime.Now.Minute;

SetEndDate();

\_formMode = FormMode.Add;

switch (scheduleEditType)

{

case ScheduleEditType.Server:

Text = "Планированиеобслуживаниясервера";

break;

case ScheduleEditType.SAN:

Text = "Планированиеобслуживанияхранилищаданных";

break;

}

}

/// <summary>

/// Конструктор добавления новой позиции расписания обслуживания севера

/// </summary>

/// <param name="selectedServer">Экземпляр выбранной конфигурации сервера</param>

public EditScheduleItem(Server selectedServer) : this(ScheduleEditType.Server)

{

\_selectedServer = selectedServer;

}

/// <summary>

/// Конструктор добавления новой позиции расписания обслуживания хранилища данных

/// </summary>

/// <param name="selectedSan">Экземпляр выбранного хранилища данных</param>

public EditScheduleItem(SAN selectedSan) : this(ScheduleEditType.SAN)

{

\_selectedSan = selectedSan;

}

/// <summary>

/// Конструктор редактирования позиции расписания обслуживания севера

/// </summary>

/// <param name="serverMaintenanceShedule">Выбранная позиция расписания обслуживания севера</param>

public EditScheduleItem(Server\_MaintenanceShedule serverMaintenanceShedule) : this(ScheduleEditType.Server)

{

\_serverMaintenanceShedule = serverMaintenanceShedule;

UnSubscribeEvents();

SetMinimumValuesToDefault();

dateTimePicker\_Begin.Value = \_serverMaintenanceShedule.BeginPeriod;

numericUpDown\_BeginHours.Value = \_serverMaintenanceShedule.BeginPeriod.Hour;

numericUpDown\_BeginMinutes.Value = \_serverMaintenanceShedule.BeginPeriod.Minute;

dateTimePicker\_End.Value = \_serverMaintenanceShedule.EndPeriod;

numericUpDown\_EndHours.Value = \_serverMaintenanceShedule.EndPeriod.Hour;

numericUpDown\_EndMinutes.Value = \_serverMaintenanceShedule.EndPeriod.Minute;

CalculateDuration();

SubscribeEvents();

\_formMode = FormMode.Edit;

}

/// <summary>

/// Конструктор редактирования позиции расписания обслуживания хранилища данных

/// </summary>

/// <param name="sanMaintenanceShedule">Выбранная позиция расписания обслуживания хранилища данных</param>

public EditScheduleItem(SAN\_MaintenanceShedule sanMaintenanceShedule) : this(ScheduleEditType.Server)

{

\_sanMaintenanceShedule = sanMaintenanceShedule;

UnSubscribeEvents();

SetMinimumValuesToDefault();

dateTimePicker\_Begin.Value = \_sanMaintenanceShedule.BeginPeriod;

numericUpDown\_BeginHours.Value = \_sanMaintenanceShedule.BeginPeriod.Hour;

numericUpDown\_BeginMinutes.Value = \_sanMaintenanceShedule.BeginPeriod.Minute;

dateTimePicker\_End.Value = \_sanMaintenanceShedule.EndPeriod;

numericUpDown\_EndHours.Value = \_sanMaintenanceShedule.EndPeriod.Hour;

numericUpDown\_EndMinutes.Value = \_sanMaintenanceShedule.EndPeriod.Minute;

CalculateDuration();

SubscribeEvents();

\_formMode = FormMode.Edit;

}

/// <summary>

/// Установка минимальных значений для компонентов по умолчанию

/// </summary>

private void SetMinimumValuesToDefault()

{

numericUpDown\_BeginHours.Minimum = 0;

numericUpDown\_BeginMinutes.Minimum = 0;

numericUpDown\_EndHours.Minimum = 0;

numericUpDown\_EndMinutes.Minimum = 0;

}

/// <summary>

/// Отпискаобработчиковсобытийизмененийзначенийкомпонентов

/// </summary>

private void UnSubscribeEvents()

{

dateTimePicker\_Begin.ValueChanged -= dateTimePicker\_Begin\_ValueChanged;

dateTimePicker\_End.ValueChanged -= dateTimePicker\_End\_ValueChanged;

numericUpDown\_BeginHours.ValueChanged -= numericUpDown\_BeginHours\_ValueChanged;

numericUpDown\_BeginMinutes.ValueChanged -= numericUpDown\_BeginMinutes\_ValueChanged;

numericUpDown\_EndHours.ValueChanged -= numericUpDown\_EndHours\_ValueChanged;

numericUpDown\_EndMinutes.ValueChanged -= numericUpDown\_EndMinutes\_ValueChanged;

}

/// <summary>

/// Подписка обработчиков событий изменений значений компонентов

/// </summary>

private void SubscribeEvents()

{

dateTimePicker\_Begin.ValueChanged += dateTimePicker\_Begin\_ValueChanged;

dateTimePicker\_End.ValueChanged += dateTimePicker\_End\_ValueChanged;

numericUpDown\_BeginHours.ValueChanged += numericUpDown\_BeginHours\_ValueChanged;

numericUpDown\_BeginMinutes.ValueChanged += numericUpDown\_BeginMinutes\_ValueChanged;

numericUpDown\_EndHours.ValueChanged += numericUpDown\_EndHours\_ValueChanged;

numericUpDown\_EndMinutes.ValueChanged += numericUpDown\_EndMinutes\_ValueChanged;

}

/// <summary>

/// Обработчик события изменения даты начала обслуживания,

/// который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности

/// </summary>

private void dateTimePicker\_Begin\_ValueChanged(object sender, EventArgs e)

{

if (dateTimePicker\_Begin.Value > DateTime.Now)

{

numericUpDown\_BeginHours.Minimum = 0;

numericUpDown\_BeginMinutes.Minimum = 0;

if (numericUpDown\_BeginHours.Value == 23 && numericUpDown\_BeginMinutes.Value == 59)

dateTimePicker\_End.MinDate = dateTimePicker\_Begin.Value.AddDays(1);

else

SetEndDate();

}

else

{

dateTimePicker\_Begin.MinDate = DateTime.Now;

dateTimePicker\_End.MinDate = DateTime.Now;

numericUpDown\_BeginHours.Minimum = DateTime.Now.Hour;

numericUpDown\_BeginMinutes.Minimum = DateTime.Now.Minute;

numericUpDown\_EndHours.Minimum = DateTime.Now.Hour;

numericUpDown\_EndMinutes.Minimum = DateTime.Now.Minute;

CalculateDuration();

}

}

/// <summary>

/// Корректировкадопустимойдаты

/// </summary>

private void SetEndDate()

{

if (numericUpDown\_BeginHours.Value == 23 && numericUpDown\_BeginMinutes.Value == 59)

dateTimePicker\_End.MinDate = dateTimePicker\_Begin.Value.AddDays(1);

else

{

dateTimePicker\_End.MinDate = dateTimePicker\_Begin.Value;

if (numericUpDown\_BeginMinutes.Value == 59)

{

numericUpDown\_EndMinutes.Minimum = 0;

numericUpDown\_EndHours.Minimum = numericUpDown\_BeginHours.Value + 1;

}

else if (dateTimePicker\_End.Value > dateTimePicker\_Begin.Value)

{

numericUpDown\_EndHours.Minimum = 0;

numericUpDown\_EndMinutes.Minimum = 0;

}

else

{

if (numericUpDown\_EndHours.Value == numericUpDown\_BeginHours.Value)

numericUpDown\_EndMinutes.Minimum = numericUpDown\_BeginMinutes.Value;

numericUpDown\_EndHours.Minimum = numericUpDown\_BeginHours.Value;

}

}

CalculateDuration();

}

/// <summary>

/// Расчет разницы между окончанием и началом обслуживания

/// </summary>

private void CalculateDuration()

{

\_beginDateTime = new DateTime

(

dateTimePicker\_Begin.Value.Year,

dateTimePicker\_Begin.Value.Month,

dateTimePicker\_Begin.Value.Day,

(int)numericUpDown\_BeginHours.Value,

(int)numericUpDown\_BeginMinutes.Value,

0

);

\_endDateTime = new DateTime

(

dateTimePicker\_End.Value.Year,

dateTimePicker\_End.Value.Month,

dateTimePicker\_End.Value.Day,

(int)numericUpDown\_EndHours.Value,

(int)numericUpDown\_EndMinutes.Value,

0

);

TimeSpan difference = \_endDateTime - \_beginDateTime;

switch (comboBox\_DurationType.SelectedIndex)

{

case 0:

textBox\_Duration.Text = difference.TotalMinutes.ToString();

break;

case 1:

textBox\_Duration.Text = TimeSpan.FromHours(difference.TotalHours).ToString(@"hh\:mm");

break;

}

}

/// <summary>

/// Обработчик события изменения часа начала обслуживания,

/// который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности

/// </summary>

private void numericUpDown\_BeginHours\_ValueChanged(object sender, EventArgs e)

{

SetEndDate();

}

/// <summary>

/// Обработчик события изменения минуты начала обслуживания,

/// который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности

/// </summary>

private void numericUpDown\_BeginMinutes\_ValueChanged(object sender, EventArgs e)

{

SetEndDate();

}

/// <summary>

/// Обработчик события изменения даты окончания обслуживания,

/// который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности

/// </summary>

private void dateTimePicker\_End\_ValueChanged(object sender, EventArgs e)

{

SetEndDate();

}

/// <summary>

/// Обработчик события изменения часа окончания обслуживания,

/// который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности

/// </summary>

private void numericUpDown\_EndHours\_ValueChanged(object sender, EventArgs e)

{

SetEndDate();

}

/// <summary>

/// Обработчик события изменения минуты окончания обслуживания,

/// который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности

/// </summary>

private void numericUpDown\_EndMinutes\_ValueChanged(object sender, EventArgs e)

{

SetEndDate();

}

/// <summary>

/// Обработчик события изменения индекса формата отображения длительности обслуживания

/// </summary>

private void comboBox\_DurationType\_SelectedIndexChanged(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(textBox\_Duration.Text))

return;

double result = 0;

var tryParse = double.TryParse(textBox\_Duration.Text, out result);

switch (comboBox\_DurationType.SelectedIndex)

{

case 0:

if (tryParse)

textBox\_Duration.Text = (result \* 60).ToString();

else

{

string[] strings = textBox\_Duration.Text.Split(':');

var hours = int.Parse(strings[0]);

var minutes = int.Parse(strings[1]);

var timeSpan = new TimeSpan(hours, minutes, 0);

textBox\_Duration.Text = textBox\_Duration.Text = timeSpan.TotalMinutes.ToString();

}

break;

case 1:

if (tryParse)

textBox\_Duration.Text = TimeSpan.FromHours(TimeSpan.FromMinutes(result).TotalHours)

.ToString(@"hh\:mm");

break;

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который выполняет сохранение изменений

/// </summary>

private async void btn\_OK\_Click(object sender, EventArgs e)

{

if (\_beginDateTime >= \_endDateTime)

{

MessageBox.Show("Конечнаядатадолжнабытьбольшеначальной!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

if (\_beginDateTime <= DateTime.Now)

{

MessageBox.Show("Начальнаядатадолжнабытьбольшетекущей!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

try

{

bool checkResult = false;

switch (\_scheduleEditType)

{

case ScheduleEditType.Server:

checkResult = await \_ctx.CheckMaintenanceDateRange(

\_serverMaintenanceShedule?.Id ?? -1,

\_selectedServer?.Id ?? \_serverMaintenanceShedule.ServerID,

true,

\_beginDateTime,

\_endDateTime);

break;

case ScheduleEditType.SAN:

checkResult = await \_ctx.CheckMaintenanceDateRange(

\_sanMaintenanceShedule?.Id ?? -1,

\_selectedSan?.Id ?? \_sanMaintenanceShedule.SAN\_Id,

false,

\_beginDateTime,

\_endDateTime);

break;

}

if (checkResult)

{

MessageBox.Show(

"Уже существуют запланнированые позиции обслуживания в указанном временном промежутке!" +

"\nПозиция не будет сохранена!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

}

catch

{

MessageBox.Show("Произошла ошибка при проверке уже запланированных позиций обсулживания в установленном диапазоне дат!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (\_formMode == FormMode.Add)

{

try

{

switch (\_scheduleEditType)

{

case ScheduleEditType.Server:

await \_ctx.AddNewServerSchedulePosition(new Server\_MaintenanceShedule

{

Server = \_selectedServer,

BeginPeriod = \_beginDateTime,

EndPeriod = \_endDateTime

});

break;

case ScheduleEditType.SAN:

await \_ctx.AddNewSANSchedulePosition(new SAN\_MaintenanceShedule

{

SAN = \_selectedSan,

BeginPeriod = \_beginDateTime,

EndPeriod = \_endDateTime

});

break;

}

MessageBox.Show("Новая позиция в расписании обслуживания успешно создана!", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

DialogResult = DialogResult.OK;

Close();

}

catch

{

MessageBox.Show("Не удалось сохранить новую позицию в расписании обслуживания!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else

{

try

{

switch (\_scheduleEditType)

{

case ScheduleEditType.Server:

\_serverMaintenanceShedule.BeginPeriod = \_beginDateTime;

\_serverMaintenanceShedule.EndPeriod = \_endDateTime;

await \_ctx.EditServerSchedulePosition(\_serverMaintenanceShedule);

break;

case ScheduleEditType.SAN:

\_sanMaintenanceShedule.BeginPeriod = \_beginDateTime;

\_sanMaintenanceShedule.EndPeriod = \_endDateTime;

await \_ctx.EditSANSchedulePosition(\_sanMaintenanceShedule);

break;

}

MessageBox.Show("Изменения в позиции расписания обслуживания успешно сохранены!", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

DialogResult = DialogResult.OK;

Close();

}

catch

{

MessageBox.Show("Не удалось сохранить изменения в позиции расписания обслуживания!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену создания / редактирования позиции расписания обслуживания, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditScheduleItem\_FormClosing(object sender, FormClosingEventArgs e)

{

if (DialogResult != DialogResult.OK)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.No)

e.Cancel = true;

}

}

}

}

Модуль «EquipmentIdleMenu.cs»

using System;

using System.Windows.Forms;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

namespace MTF\_Services.WinForms.Forms.SysAdmin.EquipmentIdle

{

/// <summary>

/// Класс формы меню простоя оборудования

/// </summary>

public partial class EquipmentIdleMenu : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Выбранныйсервер

/// </summary>

public ServerIdleItem CurrentServerIdleItem { get; set; }

/// <summary>

/// Выбранное хранилище данных

/// </summary>

public SANIdleItem CurrentSanIdleItem { get; set; }

/// <summary>

/// Конструктор формы меню простоя оборудования

/// </summary>

public EquipmentIdleMenu()

{

InitializeComponent();

SubscribeMenuItems();

\_ctx = new Context();

BindCollections();

timer.Enabled = true;

}

/// <summary>

/// Обработчик события выхода курсора из элемента главного меню,

/// который очищает область в строке состояния.

/// </summary>

private void MenuItem\_MouseLeave(object sender, EventArgs e)

{

tip\_Label.Text = string.Empty;

}

/// <summary>

/// Обработчик события навдения курсора на элемент главного меню,

/// который выводит подсказку элемента в строку состояния.

/// </summary>

private void MenuItem\_MouseEnter(object sender, EventArgs e)

{

tip\_Label.Text = ((ToolStripItem)sender).ToolTipText;

}

/// <summary>

/// Подписка элементов главного меню и панели элементов на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

private void SubscribeMenuItems()

{

SubScribeChildMenuItems(menuBar.Items);

}

/// <summary>

/// Подписка дочерних элементов главного меню на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

/// <param name="col">Коллекция элементов главного меню</param>

private void SubScribeChildMenuItems(ToolStripItemCollection col)

{

for (int i = 0; i < col.Count; i++)

{

var item = col[i] as ToolStripMenuItem;

if (item != null)

{

item.MouseEnter += MenuItem\_MouseEnter;

item.MouseLeave += MenuItem\_MouseLeave;

if (item.DropDownItems.Count > 0)

SubScribeChildMenuItems(item.DropDownItems);

}

}

}

/// <summary>

/// Привязкаколлекций

/// </summary>

private void BindCollections()

{

UpdateServerIdleItems();

sANIdleItemBindingSource.DataSource = \_ctx.GetSanIdleItems();

dg\_Sans.DataSource = sANIdleItemBindingSource;

serviceIdleItemBindingSource.DataSource = \_ctx.GetServiceIdleItems();

dg\_Services.DataSource = serviceIdleItemBindingSource;

}

/// <summary>

/// Обновление коллекции простоя серверов

/// </summary>

public void UpdateServerIdleItems()

{

serverIdleItemBindingSource.DataSource = \_ctx.GetServerIdleItems();

dg\_Servers.DataSource = serverIdleItemBindingSource;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных информации о простое сервера

/// </summary>

private void serverIdleItemBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as ServerIdleItem;

if (selectedItem != null)

CurrentServerIdleItem = selectedItem;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных информации о простое хранилища данных

/// </summary>

private void sANIdleItemBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as SANIdleItem;

if (selectedItem != null)

CurrentSanIdleItem = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы планирования расписания обслуживания выбранного сервера

/// </summary>

private async void tsmi\_ServerPlanningMaintenance\_Click(object sender, EventArgs e)

{

if (CurrentServerIdleItem == null)

{

MessageBox.Show("Выберите сервер из списка для планирования расписания обслуживания!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

try

{

var server = await \_ctx.GetServerByID(CurrentServerIdleItem.ID);

var planningSchedule = new PlanningSchedule(server) { Owner = this };

planningSchedule.ShowDialog();

}

catch

{

MessageBox.Show("Произошла ошибка при получении информации о выбранной конфигурации сервера!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы регистрации нового простоя сервера

/// </summary>

private async void tsmi\_ServerRegisterIdle\_Click(object sender, EventArgs e)

{

if (CurrentServerIdleItem == null)

{

MessageBox.Show("Выберите сервер из списка для регистрации нового простоя!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

try

{

var server = await \_ctx.GetServerByID(CurrentServerIdleItem.ID);

if (\_ctx.GetServicesByServer(server).Count == 0)

{

MessageBox.Show("Данный сервер не используется на данный момент! " +

"\nРегистрация простоя невозможна!",

"Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

var registerNewIdle = new RegisterNewIdle(server);

if (registerNewIdle.ShowDialog() == DialogResult.OK)

BindCollections();

}

catch

{

MessageBox.Show("Произошла ошибка при получении информации о выбранной конфигурации сервера!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы планирования расписания обслуживания выбранного хранилища данных

/// </summary>

private async void tsmi\_SanPlanningMaintenance\_Click(object sender, EventArgs e)

{

if (CurrentSanIdleItem == null)

{

MessageBox.Show("Выберите хранилище данных из списка для планирования расписания обслуживания!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

try

{

var san = await \_ctx.GetSANByID(CurrentSanIdleItem.ID);

var planningSchedule = new PlanningSchedule(san) { Owner = this };

planningSchedule.ShowDialog();

}

catch

{

MessageBox.Show("Произошлаошибкаприполученииинформацииовыбранномхранилищеданных!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы регистрации нового простоя хранилища данных

/// </summary>

private async void tsmi\_SanRegisterIdle\_Click(object sender, EventArgs e)

{

if (CurrentSanIdleItem == null)

{

MessageBox.Show("Выберите хранилище данных из списка для регистрации нового простоя!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

try

{

var san = await \_ctx.GetSANByID(CurrentSanIdleItem.ID);

if (\_ctx.GetServicesBySAN(san).Count == 0)

{

MessageBox.Show("Данное хранилище данных не используется на данный момент! " +

"\nРегистрация простоя невозможна!",

"Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

varregisterNewIdle = newRegisterNewIdle(san);

if (registerNewIdle.ShowDialog() == DialogResult.OK)

BindCollections();

}

catch

{

MessageBox.Show("Произошла ошибка при получении информации о выбранной конфигурации сервера!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EquipmentIdleMenu\_FormClosing(object sender, FormClosingEventArgs e)

{

Owner?.Show();

}

/// <summary>

/// Обработчик события выполнения такта таймера

/// </summary>

private void timer\_Tick(object sender, EventArgs e)

{

if (\_ctx == null)

return;

try

{

\_ctx.CheckServerMaintenance();

\_ctx.CheckSANMaintenance();

BindCollections();

}

catch

{

}

}

}

}

Модуль «PlanningSchedule.cs»

using System;

using System.ComponentModel;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

namespace MTF\_Services.WinForms.Forms.SysAdmin.EquipmentIdle

{

/// <summary>

/// Класс формы планирования расписания обслуживания

/// </summary>

public partial class PlanningSchedule : Form

{

private readonly SAN \_selectedSan;

private readonly Server \_selectedServer;

private readonly ScheduleEditType \_scheduleEditType;

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Список поизций расписания обслуживания

/// </summary>

public BindingList<ScheduleItem> ScheduleItemsMain { get; set; }

/// <summary>

/// Отобранный список поизций расписания обслуживания

/// </summary>

public BindingList<ScheduleItem> ScheduleItemsToShow { get; set; }

/// <summary>

/// Выбранная позиция расписания обслуживания

/// </summary>

public ScheduleItem CurrentScheduleItem { get; set; }

/// <summary>

/// Режим просмотра списка

/// </summary>

private ScheduleItemsShowMode \_scheduleItemsShowMode;

private ScheduleItemsTimeFormat \_scheduleItemsTimeFormat;

/// <summary>

/// Конструктор формы планирования расписания обслуживания

/// </summary>

private PlanningSchedule(ScheduleEditType scheduleEditType)

{

\_scheduleEditType = scheduleEditType;

InitializeComponent();

SubscribeMenuItems();

\_ctx = new Context();

dg\_Schedule.Columns[3].HeaderText = "Длительность (минуты)";

\_scheduleItemsShowMode = ScheduleItemsShowMode.All;

\_scheduleItemsTimeFormat = ScheduleItemsTimeFormat.Minutes;

switch (\_scheduleEditType)

{

case ScheduleEditType.Server:

Text = "Планирование расписания обслуживания сервера";

break;

case ScheduleEditType.SAN:

Text = "Планирование расписания обслуживания хранилища данных";

break;

}

}

/// <summary>

/// Конструктор формы планирования расписания обслуживания для сервера

/// </summary>

/// <param name="selectedServer">Выбраннаяконфигурациясервера</param>

public PlanningSchedule(Server selectedServer) : this(ScheduleEditType.Server)

{

\_selectedServer = selectedServer;

BindCollection();

UpdateStatBar();

}

/// <summary>

/// Конструктор формы планирования расписания обслуживания для хранилища данных

/// </summary>

/// <param name="selectedSan">Выбранноехранилищеданных</param>

public PlanningSchedule(SAN selectedSan) : this(ScheduleEditType.SAN)

{

\_selectedSan = selectedSan;

BindCollection();

UpdateStatBar();

}

/// <summary>

/// Привязкаколлекции

/// </summary>

private void BindCollection()

{

switch (\_scheduleEditType)

{

case ScheduleEditType.Server:

ScheduleItemsMain = \_ctx.GetScheduleItemsOfServer(\_selectedServer);

break;

case ScheduleEditType.SAN:

ScheduleItemsMain = \_ctx.GetScheduleItemsOfSAN(\_selectedSan);

break;

}

switch (\_scheduleItemsTimeFormat)

{

case ScheduleItemsTimeFormat.Hours:

часыToolStripMenuItem\_Click(null, EventArgs.Empty);

break;

case ScheduleItemsTimeFormat.Minutes:

минутыToolStripMenuItem\_Click(null, EventArgs.Empty);

break;

}

}

/// <summary>

/// Обновление строки состояния

/// </summary>

private void UpdateStatBar()

{

lbl\_PositionTotalCount.Text = $"Общеекол-вопозиций: {ScheduleItemsMain?.Count ?? 0}";

lbl\_PositionSelectedCount.Text = $"Отобранопозиций: {dg\_Schedule.Rows.Count}";

}

/// <summary>

/// Обработчик события выхода курсора из элемента главного меню,

/// который очищает область в строке состояния.

/// </summary>

private void MenuItem\_MouseLeave(object sender, EventArgs e)

{

tip\_Label.Text = string.Empty;

}

/// <summary>

/// Обработчик события навдения курсора на элемент главного меню,

/// который выводит подсказку элемента в строку состояния.

/// </summary>

private void MenuItem\_MouseEnter(object sender, EventArgs e)

{

tip\_Label.Text = ((ToolStripItem)sender).ToolTipText;

}

/// <summary>

/// Подписка элементов главного меню и панели элементов на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

private void SubscribeMenuItems()

{

SubScribeChildMenuItems(menuBar.Items);

foreach (ToolStripItem toolBarControl in toolBar.Items)

{

toolBarControl.MouseEnter += MenuItem\_MouseEnter;

toolBarControl.MouseLeave += MenuItem\_MouseLeave;

}

}

/// <summary>

/// Подписка дочерних элементов главного меню на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

/// <param name="col">Коллекция элементов главного меню</param>

private void SubScribeChildMenuItems(ToolStripItemCollection col)

{

for (int i = 0; i < col.Count; i++)

{

var item = col[i] as ToolStripMenuItem;

if (item != null)

{

item.MouseEnter += MenuItem\_MouseEnter;

item.MouseLeave += MenuItem\_MouseLeave;

if (item.DropDownItems.Count > 0)

SubScribeChildMenuItems(item.DropDownItems);

}

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных позиций расписания обслуживания

/// </summary>

private void scheduleItemBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as ScheduleItem;

if (selectedItem != null)

CurrentScheduleItem = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который выполняет переход на форму добавления новой позиции расписания обслуживания

/// </summary>

private void новаяПозицияToolStripMenuItem\_Click(object sender, EventArgs e)

{

EditScheduleItem editScheduleItem = null;

switch (\_scheduleEditType)

{

case ScheduleEditType.Server:

editScheduleItem = new EditScheduleItem(\_selectedServer);

break;

case ScheduleEditType.SAN:

editScheduleItem = new EditScheduleItem(\_selectedSan);

break;

}

if (editScheduleItem.ShowDialog() == DialogResult.OK)

{

BindCollection();

try

{

var ownForm = Owner as EquipmentIdleMenu;

ownForm.UpdateServerIdleItems();

}

catch

{

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который выполняет переход на форму редактирования выбранной позиции расписания обслуживания

/// </summary>

private async void редактироватьВыбраннуюПозициюToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (CurrentScheduleItem == null)

{

MessageBox.Show("Выберитепозициюизспискадляеередактирования!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

if (CurrentScheduleItem.BeginDate <= DateTime.Now)

{

MessageBox.Show("Редактировать можно только запланированные позиции!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

Server\_MaintenanceShedule selectedServerMaintenanceShedule = null;

SAN\_MaintenanceShedule selectedSanMaintenanceShedule = null;

try

{

switch (\_scheduleEditType)

{

case ScheduleEditType.Server:

selectedServerMaintenanceShedule =

await \_ctx.GetServerMaintenanceSheduleByID(CurrentScheduleItem.Id);

break;

case ScheduleEditType.SAN:

selectedSanMaintenanceShedule =

await \_ctx.GetSANMaintenanceSheduleByID(CurrentScheduleItem.Id);

break;

}

}

catch

{

MessageBox.Show("Не удалось получить информацию о выбранной позиции расписания обслуживания", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

EditScheduleItem editScheduleItem = null;

switch (\_scheduleEditType)

{

case ScheduleEditType.Server:

editScheduleItem = new EditScheduleItem(selectedServerMaintenanceShedule);

break;

case ScheduleEditType.SAN:

editScheduleItem = new EditScheduleItem(selectedSanMaintenanceShedule);

break;

}

if (editScheduleItem.ShowDialog() == DialogResult.OK)

{

BindCollection();

try

{

var ownForm = Owner as EquipmentIdleMenu;

ownForm.UpdateServerIdleItems();

}

catch

{

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который совершает показ всех позиций расписания обслуживания

/// </summary>

private void всеЗаписиToolStripMenuItem\_Click(object sender, EventArgs e)

{

scheduleItemBindingSource.DataSource = ScheduleItemsMain;

dg\_Schedule.DataSource = scheduleItemBindingSource;

UpdateStatBar();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который совершает отбор запланированных позиций расписания обслуживания

/// </summary>

private void запланированныеToolStripMenuItem\_Click(object sender, EventArgs e)

{

\_scheduleItemsShowMode = ScheduleItemsShowMode.Planned;

ScheduleItemsToShow = new BindingList<ScheduleItem>(ScheduleItemsMain.Where(sim => sim.BeginDate > DateTime.Now).ToList());

scheduleItemBindingSource.DataSource = ScheduleItemsToShow;

dg\_Schedule.DataSource = scheduleItemBindingSource;

UpdateStatBar();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который совершает отбор завершенных позиций расписания обслуживания

/// </summary>

private void завершенныеToolStripMenuItem\_Click(object sender, EventArgs e)

{

\_scheduleItemsShowMode = ScheduleItemsShowMode.Completed;

ScheduleItemsToShow = new BindingList<ScheduleItem>(ScheduleItemsMain.Where(sim => sim.EndDate < DateTime.Now).ToList());

scheduleItemBindingSource.DataSource = ScheduleItemsToShow;

dg\_Schedule.DataSource = scheduleItemBindingSource;

UpdateStatBar();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который меняет формат на минуты длительности позиции обслуживания

/// </summary>

private void минутыToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

var scheduleItems = ScheduleItemsMain.ToList();

scheduleItems.ForEach(si => si.Duration = (si.EndDate - si.BeginDate).TotalMinutes.ToString());

ScheduleItemsMain = new BindingList<ScheduleItem>(scheduleItems.ToList());

switch (\_scheduleItemsShowMode)

{

case ScheduleItemsShowMode.All:

scheduleItemBindingSource.DataSource = ScheduleItemsMain;

dg\_Schedule.DataSource = scheduleItemBindingSource;

break;

case ScheduleItemsShowMode.Planned:

запланированныеToolStripMenuItem\_Click(null, EventArgs.Empty);

break;

case ScheduleItemsShowMode.Completed:

завершенныеToolStripMenuItem\_Click(null, EventArgs.Empty);

break;

}

dg\_Schedule.Columns[3].HeaderText = "Длительность (минуты)";

минутыToolStripMenuItem.Checked = true;

toolStripMenuItem5.Checked = true;

часыToolStripMenuItem.Checked = false;

toolStripMenuItem6.Checked = false;

\_scheduleItemsTimeFormat = ScheduleItemsTimeFormat.Minutes;

}

catch

{

MessageBox.Show("Произошлаошибкаприпереводеформата!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который меняет формат на часы длительности позиции обслуживания

/// </summary>

private void часыToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

var scheduleItems = ScheduleItemsMain.ToList();

scheduleItems.ForEach(si => si.Duration = TimeSpan.FromHours((si.EndDate - si.BeginDate).TotalHours).ToString(@"hh\:mm"));

ScheduleItemsMain = new BindingList<ScheduleItem>(scheduleItems.ToList());

switch (\_scheduleItemsShowMode)

{

case ScheduleItemsShowMode.All:

scheduleItemBindingSource.DataSource = ScheduleItemsMain;

dg\_Schedule.DataSource = scheduleItemBindingSource;

UpdateStatBar();

break;

case ScheduleItemsShowMode.Planned:

запланированныеToolStripMenuItem\_Click(null, EventArgs.Empty);

break;

case ScheduleItemsShowMode.Completed:

завершенныеToolStripMenuItem\_Click(null, EventArgs.Empty);

break;

}

dg\_Schedule.Columns[3].HeaderText = "Длительность (часы)";

минутыToolStripMenuItem.Checked = false;

toolStripMenuItem5.Checked = false;

часыToolStripMenuItem.Checked = true;

toolStripMenuItem6.Checked = true;

\_scheduleItemsTimeFormat = ScheduleItemsTimeFormat.Hours;

}

catch

{

MessageBox.Show("Произошлаошибкаприпереводеформата!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который осуществляет удаление выбранной позиции расписания обслуживания

/// </summary>

private async void удалитьВыбраннуюПозициюToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (CurrentScheduleItem == null)

{

MessageBox.Show("Выберитепозициюизспискадляееудаленияя!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

if (CurrentScheduleItem.BeginDate <= DateTime.Now)

{

MessageBox.Show("Удалять можно только запланированные позиции!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

Server\_MaintenanceShedule selectedServerMaintenanceShedule = null;

SAN\_MaintenanceShedule selectedSanMaintenanceShedule = null;

try

{

switch (\_scheduleEditType)

{

case ScheduleEditType.Server:

selectedServerMaintenanceShedule =

await \_ctx.GetServerMaintenanceSheduleByID(CurrentScheduleItem.Id);

break;

case ScheduleEditType.SAN:

selectedSanMaintenanceShedule =

await \_ctx.GetSANMaintenanceSheduleByID(CurrentScheduleItem.Id);

break;

}

}

catch

{

MessageBox.Show("Не удалось получить информацию о выбранной позиции расписания обслуживания", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

var result = MessageBox.Show("Выбранная позиция будет удалена! Продолжить?",

"Удаление позиции расписания обслуживания", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.Yes)

{

try

{

switch (\_scheduleEditType)

{

case ScheduleEditType.Server:

await \_ctx.DeleteSelectedServiceScheduleMaintenance(selectedServerMaintenanceShedule);

break;

case ScheduleEditType.SAN:

await \_ctx.DeleteSelectedSANScheduleMaintenance(selectedSanMaintenanceShedule);

break;

}

BindCollection();

try

{

var ownForm = Owner as EquipmentIdleMenu;

ownForm.UpdateServerIdleItems();

}

catch

{

}

}

catch

{

MessageBox.Show("Произошлаошибкаприудалениивыбраннойпозициирасписанияобслуживания!",

"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

}

Модуль «RegisterNewIdle.cs»

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.EquipmentIdle

{

/// <summary>

/// Класс формы регистрации новго простоя

/// </summary>

public partial class RegisterNewIdle : Form

{

private readonly SAN \_san;

private readonly Server \_server;

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Дата начала обслуживания

/// </summary>

private DateTime \_beginDateTime;

/// <summary>

/// Датаокончанияобслуживания

/// </summary>

private DateTime \_endDateTime;

/// <summary>

/// Доступныйперсонал

/// </summary>

public BindingList<User> AvaliblePersonal { get; set; }

/// <summary>

/// Задействованныйперсонал

/// </summary>

public BindingList<User> UsedPersonal { get; set; }

/// <summary>

/// Выбранный доступный сотрудник

/// </summary>

public User SelectedAvalibleUser { get; set; }

/// <summary>

/// Выбранный задействованный сотрудник

/// </summary>

public User SelectedUsedUser { get; set; }

/// <summary>

/// Выбранный тип простоя

/// </summary>

public IdleType SelectedIdleType { get; set; }

/// <summary>

/// Выбранная причина простоя

/// </summary>

public IdleReason SelectedIdleReason { get; set; }

/// <summary>

/// Конструктор формы регистрации нового простоя для сервера

/// </summary>

public RegisterNewIdle(Server server)

{

\_server = server;

InitializeComponent();

\_ctx = new Context();

btn\_OK.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

comboBox\_DurationType.SelectedIndex = 0;

SetEndDate();

BindCollections();

}

/// <summary>

/// Конструктор формы регистрации нового простоя для хранилища данных

/// </summary>

public RegisterNewIdle(SAN san)

{

\_san = san;

InitializeComponent();

\_ctx = new Context();

btn\_OK.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

comboBox\_DurationType.SelectedIndex = 0;

SetEndDate();

BindCollections();

}

/// <summary>

/// Привязкаколлекций

/// </summary>

private void BindCollections()

{

idleTypeBindingSource.DataSource = \_ctx.GetIdleTypes();

idleReasonBindingSource.DataSource = \_ctx.GetIdleReasons();

AvaliblePersonal = new BindingList<User>(\_ctx.GetAdminsList());

UsedPersonal = new BindingList<User>();

AvalibleUserBindingSource.DataSource = AvaliblePersonal;

UsedUserBindingSource.DataSource = UsedPersonal;

dg\_AvalibleUser.DataSource = AvalibleUserBindingSource;

dg\_UsedUser.DataSource = UsedPersonal;

}

/// <summary>

/// Обработчик события изменения даты начала обслуживания,

/// который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности

/// </summary>

private void dateTimePicker\_Begin\_ValueChanged(object sender, EventArgs e)

{

if (numericUpDown\_BeginHours.Value == 23 && numericUpDown\_BeginMinutes.Value == 59)

dateTimePicker\_End.MinDate = dateTimePicker\_Begin.Value.AddDays(1);

else

SetEndDate();

}

/// <summary>

/// Корректировка допустимой даты

/// </summary>

private void SetEndDate()

{

if (numericUpDown\_BeginHours.Value == 23 && numericUpDown\_BeginMinutes.Value == 59)

dateTimePicker\_End.MinDate = dateTimePicker\_Begin.Value.AddDays(1);

else

{

dateTimePicker\_End.MinDate = dateTimePicker\_Begin.Value;

if (numericUpDown\_BeginMinutes.Value == 59)

{

numericUpDown\_EndMinutes.Minimum = 0;

numericUpDown\_EndHours.Minimum = numericUpDown\_BeginHours.Value + 1;

}

else if (dateTimePicker\_End.Value > dateTimePicker\_Begin.Value)

{

numericUpDown\_EndHours.Minimum = 0;

numericUpDown\_EndMinutes.Minimum = 0;

}

else

{

if (numericUpDown\_EndHours.Value == numericUpDown\_BeginHours.Value)

numericUpDown\_EndMinutes.Minimum = numericUpDown\_BeginMinutes.Value;

numericUpDown\_EndHours.Minimum = numericUpDown\_BeginHours.Value;

}

}

CalculateDuration();

}

/// <summary>

/// Расчет разницы между окончанием и началом обслуживания

/// </summary>

private void CalculateDuration()

{

\_beginDateTime = new DateTime

(

dateTimePicker\_Begin.Value.Year,

dateTimePicker\_Begin.Value.Month,

dateTimePicker\_Begin.Value.Day,

(int)numericUpDown\_BeginHours.Value,

(int)numericUpDown\_BeginMinutes.Value,

0

);

\_endDateTime = new DateTime

(

dateTimePicker\_End.Value.Year,

dateTimePicker\_End.Value.Month,

dateTimePicker\_End.Value.Day,

(int)numericUpDown\_EndHours.Value,

(int)numericUpDown\_EndMinutes.Value,

0

);

TimeSpan difference = \_endDateTime - \_beginDateTime;

switch (comboBox\_DurationType.SelectedIndex)

{

case 0:

textBox\_Duration.Text = difference.TotalMinutes.ToString();

break;

case 1:

textBox\_Duration.Text = TimeSpan.FromHours(difference.TotalHours).ToString(@"hh\:mm");

break;

}

}

/// <summary>

/// Обработчик события изменения часа начала обслуживания,

/// который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности

/// </summary>

private void numericUpDown\_BeginHours\_ValueChanged(object sender, EventArgs e)

{

SetEndDate();

}

/// <summary>

/// Обработчик события изменения минуты начала обслуживания,

/// который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности

/// </summary>

private void numericUpDown\_BeginMinutes\_ValueChanged(object sender, EventArgs e)

{

SetEndDate();

}

/// <summary>

/// Обработчик события изменения даты окончания обслуживания,

/// который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности

/// </summary>

private void dateTimePicker\_End\_ValueChanged(object sender, EventArgs e)

{

SetEndDate();

}

/// <summary>

/// Обработчик события изменения часа окончания обслуживания,

/// который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности

/// </summary>

private void numericUpDown\_EndHours\_ValueChanged(object sender, EventArgs e)

{

SetEndDate();

}

/// <summary>

/// Обработчик события изменения минуты окончания обслуживания,

/// который осуществляет корректировку допустимой даты окончания обслуживания и расчет длительности

/// </summary>

private void numericUpDown\_EndMinutes\_ValueChanged(object sender, EventArgs e)

{

SetEndDate();

}

/// <summary>

/// Обработчик события изменения индекса формата отображения длительности обслуживания

/// </summary>

private void comboBox\_DurationType\_SelectedIndexChanged(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(textBox\_Duration.Text))

return;

double result = 0;

var tryParse = double.TryParse(textBox\_Duration.Text, out result);

switch (comboBox\_DurationType.SelectedIndex)

{

case 0:

if (tryParse)

textBox\_Duration.Text = (result \* 60).ToString();

else

{

string[] strings = textBox\_Duration.Text.Split(':');

var hours = int.Parse(strings[0]);

var minutes = int.Parse(strings[1]);

var timeSpan = new TimeSpan(hours, minutes, 0);

textBox\_Duration.Text = textBox\_Duration.Text = timeSpan.TotalMinutes.ToString();

}

break;

case 1:

if (tryParse)

textBox\_Duration.Text = TimeSpan.FromHours(TimeSpan.FromMinutes(result).TotalHours)

.ToString(@"hh\:mm");

break;

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных доступного персонала

/// </summary>

private void AvalibleUserBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as User;

if (selectedItem != null)

SelectedAvalibleUser = selectedItem;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных задействованного персонала

/// </summary>

private void UsedUserBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as User;

if (selectedItem != null)

SelectedUsedUser = selectedItem;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных типа простоя

/// </summary>

private void idleTypeBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as IdleType;

if (selectedItem != null)

SelectedIdleType = selectedItem;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных причины простоя

/// </summary>

private void idleReasonBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as IdleReason;

if (selectedItem != null)

SelectedIdleReason = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит удаление выбранного сотрудника из задействованных сотрудников

/// </summary>

private void btn\_RemoveFromUsed\_Click(object sender, EventArgs e)

{

if (SelectedUsedUser == null || dg\_UsedUser.CurrentRow == null)

{

MessageBox.Show("Пользователь не выбран!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

AvaliblePersonal.Add(SelectedUsedUser);

UsedPersonal.Remove(SelectedUsedUser);

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит добавление выбранного сотрудника из задействованных сотрудников

/// </summary>

private void btn\_AddToUsed\_Click(object sender, EventArgs e)

{

if (SelectedAvalibleUser == null || dg\_AvalibleUser.CurrentRow == null)

{

MessageBox.Show("Пользователь не выбран!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

UsedPersonal.Add(SelectedAvalibleUser);

AvaliblePersonal.Remove(SelectedAvalibleUser);

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который выполняет сохранение изменений

/// </summary>

private void btn\_OK\_Click(object sender, EventArgs e)

{

if (\_beginDateTime >= \_endDateTime)

{

MessageBox.Show("Конечнаядатадолжнабытьбольшеначальной!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

if (\_endDateTime >= DateTime.Now)

{

MessageBox.Show("Конечнаядатадолжнабытьменьшетекущей!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

if (SelectedIdleType == null || comboBox1.SelectedIndex < 0)

{

MessageBox.Show("Выберитетиппростоя!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

if (SelectedIdleReason == null || comboBox2.SelectedIndex < 0)

{

MessageBox.Show("Выберитепричинупростоя!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

if (UsedPersonal.Count == 0)

{

MessageBox.Show("Добавьте к задействованному персоналу сотрудников", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

try

{

List<Service> services = null;

if (\_server != null)

services = \_ctx.GetServicesByServer(\_server);

else

services = \_ctx.GetServicesBySAN(\_san);

\_ctx.RegisterNewIdle(services, UsedPersonal.ToList(), \_beginDateTime, \_endDateTime,

SelectedIdleType, SelectedIdleReason);

MessageBox.Show("Простойоборудованияуспешнозарегистрирован!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

DialogResult = DialogResult.OK;

Close();

}

catch

{

MessageBox.Show("Произошла ошибка при регистрации нового простоя! Изменения не сохранены!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену регистрации простоя, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

}

}

Модуль «SAN\_ConditionConstructorForm.cs»

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Common

{

/// <summary>

/// Класс формы отбора конфигураций хранилищ данных

/// </summary>

public partial class SAN\_ConditionConstructorForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Флаг активации привязки данных после ее приоставновки в источнике данных производителей

/// </summary>

private bool \_afterManufacturerSuspend;

/// <summary>

/// Выбранныйпроизводитель

/// </summary>

public Manufacturer SelectedManufacturer { get; set; }

/// <summary>

/// Список конфигураций хранилищ данных

/// </summary>

public BindingList<SAN\_Info> SansInfoMain { get; set; }

/// <summary>

/// Отобранный список конфигураций хранилищ данных

/// </summary>

public BindingList<SAN\_Info> SansInfoToShow { get; set; }

/// <summary>

/// Конструктор формы отбора конфигураций хранилищ данных

/// </summary>

public SAN\_ConditionConstructorForm(BindingList<SAN\_Info> sansInfoMain)

{

InitializeComponent();

btn\_Cancel.Image = new Bitmap(Resources.no, 20, 20);

btn\_OK.Image = new Bitmap(Resources.camera\_test, 20, 20);

\_ctx = new Context();

SansInfoMain = sansInfoMain;

InitBindings();

pictureBox1\_Click(null, EventArgs.Empty);

pictureBox2\_Click(null, EventArgs.Empty);

}

/// <summary>

/// Инициализацияпривязок

/// </summary>

private void InitBindings()

{

manufacturerBindingSource.DataSource = \_ctx.GetManufacturers();

comboBox2.Items.AddRange(SansInfoMain.Select(x => x.Model).Distinct().ToArray());

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных уровней производителей

/// </summary>

private void manufacturerBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (manufacturerBindingSource.IsBindingSuspended)

{

\_afterManufacturerSuspend = true;

manufacturerBindingSource.ResumeBinding();

}

else if (\_afterManufacturerSuspend)

{

\_afterManufacturerSuspend = false;

manufacturerBindingSource.Position = comboBox1.SelectedIndex;

}

if (manufacturerBindingSource.Position > -1)

{

var manufacturer = manufacturerBindingSource.Current as Manufacturer;

if (manufacturer != null)

{

SelectedManufacturer = manufacturer;

lbl\_SelectedConfigsStatus.Text = $"Отобраноконфигураций: {RunSelection()}";

}

}

}

/// <summary>

/// Выполнение отбора конфигураций хранилищ данных по условию

/// </summary>

private int RunSelection()

{

List<SAN\_Info> selectedSan = null;

try

{

if (SelectedManufacturer != null && comboBox2.SelectedIndex > -1)

selectedSan = SansInfoMain.Where(si =>

si.Manufacturer.Equals(SelectedManufacturer.Name) && si.Model.Equals(comboBox2.Items[comboBox2.SelectedIndex]))

.ToList();

else if (SelectedManufacturer != null)

selectedSan = SansInfoMain.Where(si =>si.Manufacturer.Equals(SelectedManufacturer.Name)).ToList();

else if (comboBox2.SelectedIndex > -1)

selectedSan = SansInfoMain.Where(si => si.Model.Equals(comboBox2.Items[comboBox2.SelectedIndex])).ToList();

else

selectedSan = SansInfoMain.ToList();

SansInfoToShow = new BindingList<SAN\_Info>(selectedSan);

return SansInfoToShow.Count;

}

catch

{

SansInfoToShow = null;

return 0;

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет отмену выбора производителя

/// </summary>

private void pictureBox1\_Click(object sender, EventArgs e)

{

comboBox1.SelectedIndex = -1;

SelectedManufacturer = null;

manufacturerBindingSource.SuspendBinding();

lbl\_SelectedConfigsStatus.Text = $"Отобраноконфигураций: {RunSelection()}";

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет отмену выбора модели

/// </summary>

private void pictureBox2\_Click(object sender, EventArgs e)

{

comboBox2.SelectedIndex = -1;

lbl\_SelectedConfigsStatus.Text = $"Отобраноконфигураций: {RunSelection()}";

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену совершенного отбора записей

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке производителей

/// </summary>

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox1.SelectedIndex > -1)

manufacturerBindingSource\_CurrentChanged(manufacturerBindingSource, EventArgs.Empty);

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке моделей

/// </summary>

private void comboBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox2.SelectedIndex > -1)

lbl\_SelectedConfigsStatus.Text = $"Отобраноконфигураций: {RunSelection()}";

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет подтверждение совершенного отбора конфигураций хранилищ данных

/// </summary>

private void btn\_OK\_Click(object sender, EventArgs e)

{

if (SansInfoToShow != null && SansInfoToShow.Count > 0)

{

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Ненайденыконфигурациихранилищданныхпоуказанномуусловию!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

}

Модуль «ServersConditionConstructorForm.cs»

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Common

{

/// <summary>

/// Класс формы отбора конфигураций серверов

/// </summary>

public partial class ServersConditionConstructorForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Флаг активации привязки данных после ее приоставновки в источнике данных платформ

/// </summary>

private bool \_afterPlatformSuspend;

/// <summary>

/// Флаг активации привязки данных после ее приоставновки в источнике данных процессоров

/// </summary>

private bool \_afterCPUSuspend;

/// <summary>

/// Выбранная платформа

/// </summary>

public Platform SelectedPlatform { get; set; }

/// <summary>

/// Выбранный процессор

/// </summary>

public CPU SelectedCPU { get; set; }

/// <summary>

/// Список конфигураций серверов

/// </summary>

public BindingList<ServerInfo> ServersInfoMain { get; set; }

/// <summary>

/// Отобранный список конфигураций серверов

/// </summary>

public BindingList<ServerInfo> ServersInfoToShow { get; set; }

/// <summary>

/// Конструктор формы отбора конфигураций серверов

/// </summary>

public ServersConditionConstructorForm(BindingList<ServerInfo> serversInfoMain)

{

InitializeComponent();

btn\_Cancel.Image = new Bitmap(Resources.no, 20, 20);

btn\_OK.Image = new Bitmap(Resources.camera\_test, 20, 20);

\_ctx = new Context();

ServersInfoMain = serversInfoMain;

InitBindings();

}

/// <summary>

/// Инициализация привязок

/// </summary>

private void InitBindings()

{

platformBindingSource.DataSource = \_ctx.GetPlatformsList();

cPUBindingSource.DataSource = \_ctx.GetCPUs();

}

/// <summary>

/// Обработчик события форматирования отображаемого значения в combobox для платформы

/// </summary>

private void comboBox1\_Format(object sender, ListControlConvertEventArgs e)

{

var platformToFormat = e.ListItem as Platform;

if (platformToFormat != null)

e.Value = $"{platformToFormat.Manufacturer.Name} {platformToFormat.Model}";

}

/// <summary>

/// Обработчик события форматирования отображаемого значения в combobox для процессора

/// </summary>

private void comboBox2\_Format(object sender, ListControlConvertEventArgs e)

{

var cpuToFormat = e.ListItem as CPU;

if (cpuToFormat != null)

e.Value = $"{cpuToFormat.Manufacturer.Name} {cpuToFormat.Model} ({cpuToFormat.CpuSocket.Name})";

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных платформ

/// </summary>

private void platformBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (platformBindingSource.IsBindingSuspended)

{

\_afterPlatformSuspend = true;

platformBindingSource.ResumeBinding();

}

else if (\_afterPlatformSuspend)

{

\_afterPlatformSuspend = false;

platformBindingSource.Position = comboBox1.SelectedIndex;

}

if (platformBindingSource.Position > -1)

{

var platform = platformBindingSource.Current as Platform;

if (platform != null)

{

SelectedPlatform = platform;

lbl\_SelectedConfigsStatus.Text = $"Отобрано конфигураций: {RunSelection()}";

}

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных процессоров

/// </summary>

private void cPUBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (cPUBindingSource.IsBindingSuspended)

{

\_afterCPUSuspend = true;

cPUBindingSource.ResumeBinding();

}

else if (\_afterCPUSuspend)

{

\_afterCPUSuspend = false;

cPUBindingSource.Position = comboBox2.SelectedIndex;

}

if (cPUBindingSource.Position > -1)

{

var cpu = cPUBindingSource.Current as CPU;

if (cpu != null)

{

SelectedCPU = cpu;

lbl\_SelectedConfigsStatus.Text = $"Отобрано конфигураций: {RunSelection()}";

}

}

}

/// <summary>

/// Выполнение отбора конфигураций серверов данных по условию

/// </summary>

private int RunSelection()

{

List<ServerInfo> selectedServers = null;

try

{

if (SelectedPlatform != null && SelectedCPU != null)

selectedServers = ServersInfoMain.Where(si =>

si.Platform.Equals(SelectedPlatform.Manufacturer.Name + " " + SelectedPlatform.Model) && si.CPU.Equals(SelectedCPU.Manufacturer.Name + " " + SelectedCPU.Model))

.ToList();

else if (SelectedPlatform != null)

selectedServers = ServersInfoMain.Where(si => si.Platform.Equals(SelectedPlatform.Manufacturer.Name + " " + SelectedPlatform.Model)).ToList();

else if (SelectedCPU != null)

selectedServers = ServersInfoMain.Where(si => si.CPU.Equals(SelectedCPU.Manufacturer.Name + " " + SelectedCPU.Model)).ToList();

else

selectedServers = ServersInfoMain.ToList();

ServersInfoToShow = new BindingList<ServerInfo>(selectedServers);

return ServersInfoToShow.Count;

}

catch

{

ServersInfoToShow = null;

return 0;

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет отмену выбора платформы

/// </summary>

private void pictureBox1\_Click(object sender, EventArgs e)

{

comboBox1.SelectedIndex = -1;

SelectedPlatform = null;

platformBindingSource.SuspendBinding();

lbl\_SelectedConfigsStatus.Text = $"Отобрано конфигураций: {RunSelection()}";

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет отмену выбора процессора

/// </summary>

private void pictureBox2\_Click(object sender, EventArgs e)

{

comboBox2.SelectedIndex = -1;

SelectedCPU = null;

cPUBindingSource.SuspendBinding();

lbl\_SelectedConfigsStatus.Text = $"Отобрано конфигураций: {RunSelection()}";

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке платформ

/// </summary>

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox1.SelectedIndex > -1)

platformBindingSource\_CurrentChanged(platformBindingSource, EventArgs.Empty);

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке процессоров

/// </summary>

private void comboBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox2.SelectedIndex > -1)

cPUBindingSource\_CurrentChanged(cPUBindingSource, EventArgs.Empty);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену совершенного отбора записей

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет подтверждение совершенного отбора конфигураций серверов

/// </summary>

private void btn\_OK\_Click(object sender, EventArgs e)

{

if (ServersInfoToShow != null && ServersInfoToShow.Count > 0)

{

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Не найдены конфигурации серверов по указанному условию!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

}

Модуль «EditCPUSocketForm.cs»

using System;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Dictionary

{

/// <summary>

/// Класс формы редактирования разъемов процессоров

/// </summary>

public partial class EditCPUSocketForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Наименование разъема перед изменением

/// </summary>

private string \_socketNameBeforeEdit;

/// <summary>

/// Наименование производителя разъема перед изменением

/// </summary>

private string \_socketManufacturerBeforeEdit;

/// <summary>

/// Списокраъемов

/// </summary>

public BindingList<CPUSocketInfo> CpuSockets { get; set; }

/// <summary>

/// Текущийразъем

/// </summary>

public CpuSocket CurrentSocket { get; set; }

/// <summary>

/// Режим работы формы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Индекс последнего выбранного элемента перед добавлением новой записи

/// </summary>

private int \_lastSelectedIndex;

/// <summary>

/// Флагредактированияразъемов

/// </summary>

public bool Edited { get; set; }

/// <summary>

/// Конструктор формы редактирования разъемов процессоров

/// </summary>

public EditCPUSocketForm()

{

InitializeComponent();

\_formMode = FormMode.None;

\_ctx = new Context();

EnDisFields(false);

BindManufacturersToFind();

BindCollection();

}

/// <summary>

/// Конструктордлявыбораразъемапроцессораизсписка

/// </summary>

/// <param name="findMode">Флагпоискаразъема</param>

public EditCPUSocketForm(bool findMode) : this()

{

if (!findMode)

throw new ArgumentException("Неверноезначениефлага!");

\_formMode = FormMode.None;

but\_Select.Visible = findMode;

but\_Select.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

}

/// <summary>

/// Привязка коллекции разъемов процессора

/// </summary>

private void BindCollection()

{

manufacturerBindingSource.DataSource = \_ctx.GetManufacturers();

comboBox2.DataSource = manufacturerBindingSource;

CpuSockets = \_ctx.GetCPUSocketsInfo();

cPUSocketInfoBindingSource.DataSource = CpuSockets;

dataGridView1.DataSource = cPUSocketInfoBindingSource;

}

/// <summary>

/// Привязка коллекции производителей для поиска записей

/// </summary>

private void BindManufacturersToFind()

{

manufaturerToFindBS.DataSource = \_ctx.GetManufacturers();

comboBox\_ManufacturersToFind.DataSource = manufaturerToFindBS;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных разъемов процессоров

/// </summary>

private async void cPUSocketInfoBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as CPUSocketInfo;

if (selectedItem != null)

{

try

{

\_lastSelectedIndex = CpuSockets.IndexOf(selectedItem);

CurrentSocket = await \_ctx.GetCPUSocketByInfo(selectedItem);

cpuSocketBindingSource.DataSource = CurrentSocket;

}

catch

{

}

}

}

/// <summary>

/// Активация/деактивация полей для редактирования

/// </summary>

/// <param name="enabled">Флаг активации полей редактирования</param>

private void EnDisFields(bool enabled)

{

textBox2.Enabled = enabled;

comboBox2.Enabled = enabled;

btn\_Cancel.Enabled = enabled;

btn\_Save.Enabled = enabled;

textBox\_ConditionToFind.Enabled = !enabled;

comboBox\_ManufacturersToFind.Enabled = !enabled;

dataGridView1.Enabled = !enabled;

picBtn\_FindByCondition.Enabled = !enabled;

picBtn\_ClearCondition.Enabled = !enabled;

button2.Enabled = !enabled;

button3.Enabled = !enabled;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который осуществляет добавление нового разъема

/// </summary>

private void button2\_Click(object sender, EventArgs e)

{

CurrentSocket = new CpuSocket();

cpuSocketBindingSource.DataSource = CurrentSocket;

EnDisFields(true);

\_formMode = FormMode.Add;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который осуществляет редактирование выбранного разъема процессора

/// </summary>

private void button3\_Click(object sender, EventArgs e)

{

if (CurrentSocket != null)

{

EnDisFields(true);

\_socketManufacturerBeforeEdit = CurrentSocket.Manufacturer.Name;

\_socketNameBeforeEdit = CurrentSocket.Name;

\_formMode = FormMode.Edit;

}

else

MessageBox.Show("Выберите элемент из списка или добавьте новый!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену последнего добавления/редактирования

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

try

{

if (\_formMode == FormMode.Edit)

{

CurrentSocket = \_ctx.CancelChanges(CurrentSocket);

cpuSocketBindingSource.DataSource = CurrentSocket;

cpuSocketBindingSource.ResetBindings(true);

}

else

{

if (\_lastSelectedIndex > -1)

cPUSocketInfoBindingSource.Position = \_lastSelectedIndex;

}

EnDisFields(false);

\_formMode = FormMode.None;

}

catch

{

MessageBox.Show("Произошлаошибкаприотменеизменений!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который выполняет сохранение изменений

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

try

{

var selectedManufacturer = manufacturerBindingSource.Current as Manufacturer;

if (selectedManufacturer == null || string.IsNullOrWhiteSpace(CurrentSocket.Name))

{

MessageBox.Show("Заполните все поля перед сохранением!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

bool existed = false;

switch (\_formMode)

{

case FormMode.Add:

existed = await \_ctx.CheckCpuSocketForDublicate(CurrentSocket.Name, selectedManufacturer.Name);

break;

case FormMode.Edit:

existed = await \_ctx.CheckCpuSocketForDublicate(CurrentSocket.Name, selectedManufacturer.Name)

&& (!CurrentSocket.Name.Equals(\_socketNameBeforeEdit) || !selectedManufacturer.Name.Equals(\_socketManufacturerBeforeEdit));

break;

}

if (existed)

{

MessageBox.Show("Разъем с таким производителем и наименованием уже существует!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

switch (\_formMode)

{

case FormMode.Add:

await \_ctx.AddNewCPUSocket(CurrentSocket);

break;

case FormMode.Edit:

await \_ctx.EditCPUSocket(CurrentSocket);

break;

}

EnDisFields(false);

if (\_formMode == FormMode.Add)

CpuSockets.Add(new CPUSocketInfo

{

Socket = CurrentSocket.Name,

Manufacturer = selectedManufacturer.Name

});

else

BindCollection();

\_formMode = FormMode.None;

Edited = true;

MessageBox.Show("Изменения успешно сохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

catch

{

MessageBox.Show("Изменения не удалось сохранить!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditCPUSocketForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который совершает подтверждение выбранного разъема

/// </summary>

private void but\_Select\_Click(object sender, EventArgs e)

{

if (CurrentSocket != null)

{

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Выберите элемент из списка или добавьте новый!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит очистку строки для поиска

/// </summary>

private void picBtn\_ClearCondition\_Click(object sender, EventArgs e)

{

textBox\_ConditionToFind.Text = string.Empty;

comboBox\_ManufacturersToFind.SelectedIndex = -1;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаграфическийобъект,

/// который производит поиск разъема по наименованию и/или производителю

/// </summary>

private void picBtn\_FindByCondition\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(textBox\_ConditionToFind.Text) && comboBox\_ManufacturersToFind.SelectedIndex < 0)

{

MessageBox.Show("Введите условие для поиска разъема процессора!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

CPUSocketInfo firstFoundedElement = null;

if (!string.IsNullOrWhiteSpace(textBox\_ConditionToFind.Text) && comboBox\_ManufacturersToFind.SelectedIndex >= 0)

firstFoundedElement = CpuSockets.

FirstOrDefault(x => x.Socket.ToUpper().Contains(textBox\_ConditionToFind.Text.ToUpper())

&& x.Manufacturer.Equals((manufaturerToFindBS.Current as Manufacturer).Name));

else if (!string.IsNullOrWhiteSpace(textBox\_ConditionToFind.Text))

firstFoundedElement = CpuSockets.FirstOrDefault(x => x.Socket.ToUpper().Contains(textBox\_ConditionToFind.Text.ToUpper()));

else if (comboBox\_ManufacturersToFind.SelectedIndex >= 0)

firstFoundedElement = CpuSockets.

FirstOrDefault(x => x.Manufacturer.Equals((manufaturerToFindBS.Current as Manufacturer).Name));

if (firstFoundedElement != null)

cPUSocketInfoBindingSource.Position = CpuSockets.IndexOf(firstFoundedElement);

else

MessageBox.Show("Не удалось найти разъем процессора по указанному условию!", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных разъемов для поиска

/// </summary>

private void manufaturerToFindBS\_CurrentChanged(object sender, EventArgs e)

{

if (comboBox\_ManufacturersToFind.SelectedIndex == -1)

((BindingSource)sender).SuspendBinding();

else

((BindingSource)sender).ResumeBinding();

}

/// <summary>

/// Ограничение ввода в наименовании добавляемого/редактируемого разъема процессора

/// </summary>

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!Char.IsControl(e.KeyChar) && !Char.IsLetter(e.KeyChar) && e.KeyChar != (int)Keys.Space && !Char.IsDigit(e.KeyChar))

e.Handled = true;

}

}

}

Модуль «EditManufacturerForm.cs»

using System;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using Resources = MTF\_Services.WinForms.Properties.Resources;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Dictionary

{

/// <summary>

/// Класс формы редактирования производителей

/// </summary>

public partial class EditManufacturerForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Списокпроизводителей

/// </summary>

public BindingList<Manufacturer> Manufacturers { get; set; }

/// <summary>

/// Текущийпроизводитель

/// </summary>

public Manufacturer CurrentManufacturer { get; set; }

/// <summary>

/// Наименование производителя до изменения

/// </summary>

private string \_manufacturerNameBeforeEditing;

/// <summary>

/// Режимработыформы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Флагредактированияпроизводителей

/// </summary>

public bool Edited { get; set; }

/// <summary>

/// Конструктор формы редактирования производителей

/// </summary>

public EditManufacturerForm()

{

InitializeComponent();

\_formMode = FormMode.None;

\_ctx = new Context();

EnDisFields(false);

BindCollection();

}

/// <summary>

/// Конструктор для выбора производителя из списка

/// </summary>

/// <param name="findMode">Флагпоискапроизводителя</param>

public EditManufacturerForm(bool findMode) : this()

{

if (!findMode)

throw new ArgumentException("Неверноезначениефлага!");

\_formMode = FormMode.None;

but\_Select.Visible = findMode;

but\_Select.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

}

/// <summary>

/// Привязка коллекции производителей

/// </summary>

private void BindCollection()

{

Manufacturers = \_ctx.GetManufacturers();

manufacturerBindingSource.DataSource = Manufacturers;

listBox1.DataSource = manufacturerBindingSource;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который осуществляет добавление нового производителя

/// </summary>

private void button2\_Click(object sender, EventArgs e)

{

Manufacturers.AddNew();

manufacturerBindingSource.MoveLast();

EnDisFields(true);

\_formMode = FormMode.Add;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который осуществляет редактирование выбранного производителя

/// </summary>

private void button3\_Click(object sender, EventArgs e)

{

if (CurrentManufacturer != null)

{

EnDisFields(true);

\_manufacturerNameBeforeEditing = CurrentManufacturer.Name;

\_formMode = FormMode.Edit;

}

else

MessageBox.Show("Выберите производителя из списка или добавьте нового!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Активация/деактивация полей для редактирования

/// </summary>

/// <param name="enabled">Флаг активации полей редактирования</param>

private void EnDisFields(bool enabled)

{

textBox2.Enabled = enabled;

btn\_Save.Enabled = enabled;

btn\_Cancel.Enabled = enabled;

textBox\_ConditionToFind.Enabled = !enabled;

listBox1.Enabled = !enabled;

picBtn\_FindByCondition.Enabled = !enabled;

picBtn\_ClearCondition.Enabled = !enabled;

button2.Enabled = !enabled;

button3.Enabled = !enabled;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных производителей

/// </summary>

private void manufacturerBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as Manufacturer;

if (selectedItem != null)

CurrentManufacturer = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который выполняет сохранение изменений

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

try

{

if (string.IsNullOrWhiteSpace(CurrentManufacturer.Name))

{

MessageBox.Show("Введите наименование производителя!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

bool existed = false;

switch (\_formMode)

{

case FormMode.Add:

existed = await \_ctx.CheckManufacturerForDublicate(CurrentManufacturer.Name);

break;

case FormMode.Edit:

existed = await \_ctx.CheckManufacturerForDublicate(CurrentManufacturer.Name) && !CurrentManufacturer.Name.Equals(\_manufacturerNameBeforeEditing);

break;

}

if (existed)

{

MessageBox.Show("Производитель с таким наименованием уже существует!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (\_formMode == FormMode.Add)

Manufacturers.EndNew(Manufacturers.IndexOf(CurrentManufacturer));

await \_ctx.SaveChangesAsync();

EnDisFields(false);

Edited = true;

\_formMode = FormMode.None;

DialogResult = DialogResult.OK;

MessageBox.Show("Измененияуспешносохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

catch

{

MessageBox.Show("Изменениянеудалосьсохранить!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит отмену последнего добавления/редактирования

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

try

{

if (\_formMode == FormMode.Edit)

{

int indexOfElement = Manufacturers.IndexOf(CurrentManufacturer);

CurrentManufacturer = \_ctx.CancelChanges(CurrentManufacturer);

Manufacturers[indexOfElement] = CurrentManufacturer;

}

else

Manufacturers.CancelNew(Manufacturers.IndexOf(CurrentManufacturer));

EnDisFields(false);

\_formMode = FormMode.None;

}

catch

{

MessageBox.Show("Произошлаошибкаприотменеизменений!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который совершает подтверждение выбранного производителя

/// </summary>

private void but\_Select\_Click(object sender, EventArgs e)

{

if (CurrentManufacturer != null)

{

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Выберите элемент из списка или добавьте новый!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditManufacturerForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Ограничение ввода в наименовании добавляемого/редактируемого производителя

/// </summary>

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!Char.IsControl(e.KeyChar) && !Char.IsLetter(e.KeyChar) && e.KeyChar != (int) Keys.Space)

e.Handled = true;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаграфическийобъект,

/// который производит очистку строки для поиска

/// </summary>

private void picBtn\_ClearCondition\_Click(object sender, EventArgs e)

{

textBox\_ConditionToFind.Text = string.Empty;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит поиск производителя по наименованию

/// </summary>

private void picBtn\_FindByCondition\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(textBox\_ConditionToFind.Text))

{

MessageBox.Show("Введите условие для поиска производителя!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

var firstFoundedElement = Manufacturers.FirstOrDefault(x => x.Name.ToUpper().Contains(textBox\_ConditionToFind.Text.ToUpper()));

if (firstFoundedElement != null)

manufacturerBindingSource.Position = Manufacturers.IndexOf(firstFoundedElement);

else

MessageBox.Show("Не удалось найти производителя по указанному условию!", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

}

Модуль «EditRAMTypeForm.cs»

using System;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Dictionary

{

/// <summary>

/// Класс формы редактирования типов оперативной памяти

/// </summary>

public partial class EditRAMTypeForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private Context \_ctx;

/// <summary>

/// Список типов оперативной памяти

/// </summary>

public BindingList<RamType> RamTypes { get; set; }

/// <summary>

/// Текущий тип оперативной памяти

/// </summary>

public RamType CurrentRamType { get; set; }

/// <summary>

/// Наименование типа оперативной памяти до изменения

/// </summary>

private string \_ramTypeNameBeforeEditing;

/// <summary>

/// Режимработыформы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Флаг редактирования типов оперативной памяти

/// </summary>

public bool Edited { get; set; }

/// <summary>

/// Конструктор формы редактирования типов оперативной памяти

/// </summary>

public EditRAMTypeForm()

{

InitializeComponent();

\_formMode = FormMode.None;

\_ctx = new Context();

EnDisFields(false);

BindCollection();

}

/// <summary>

/// Конструктор для выбора типа оперативной памяти из списка

/// </summary>

/// <param name="findMode">Флаг поиска типа оперативной памяти</param>

public EditRAMTypeForm(bool findMode) : this()

{

if (!findMode)

throw new ArgumentException("Неверноезначениефлага!");

\_formMode = FormMode.None;

but\_Select.Visible = findMode;

but\_Select.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

}

/// <summary>

/// Привязка коллекции типов оперативной памяти

/// </summary>

private void BindCollection()

{

RamTypes = \_ctx.GetRAMTypesBS();

ramTypeBindingSource.DataSource = RamTypes;

listBox1.DataSource = ramTypeBindingSource;

}

/// <summary>

/// Активация/деактивация полей для редактирования

/// </summary>

/// <param name="enabled">Флаг активации полей редактирования</param>

private void EnDisFields(bool enabled)

{

textBox2.Enabled = enabled;

btn\_Save.Enabled = enabled;

btn\_Cancel.Enabled = enabled;

textBox\_ConditionToFind.Enabled = !enabled;

listBox1.Enabled = !enabled;

picBtn\_FindByCondition.Enabled = !enabled;

picBtn\_ClearCondition.Enabled = !enabled;

button2.Enabled = !enabled;

button3.Enabled = !enabled;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который осуществляет добавление нового типа оперативной памяти

/// </summary>

private void button2\_Click(object sender, EventArgs e)

{

RamTypes.AddNew();

ramTypeBindingSource.MoveLast();

EnDisFields(true);

\_formMode = FormMode.Add;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который осуществляет редактирование выбранного типа оперативной памяти

/// </summary>

private void button3\_Click(object sender, EventArgs e)

{

if (CurrentRamType != null)

{

EnDisFields(true);

\_ramTypeNameBeforeEditing = CurrentRamType.Name;

\_formMode = FormMode.Edit;

}

else

MessageBox.Show("Выберите тип оперативной памяти из списка или добавьте новый!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных типов оперативной памяти

/// </summary>

private void ramTypeBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as RamType;

if (selectedItem != null)

CurrentRamType = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который выполняет сохранение изменений

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

try

{

if (string.IsNullOrWhiteSpace(CurrentRamType.Name))

{

MessageBox.Show("Введите наименование типа оперативной памяти!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

bool existed = false;

switch (\_formMode)

{

case FormMode.Add:

existed = await \_ctx.CheckRAMTypeForDublicate(CurrentRamType.Name);

break;

case FormMode.Edit:

existed = await \_ctx.CheckRAMTypeForDublicate(CurrentRamType.Name) && !CurrentRamType.Name.Equals(\_ramTypeNameBeforeEditing);

break;

}

if (existed)

{

MessageBox.Show("Тип оперативной памяти с таким наименованием уже существует!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (\_formMode == FormMode.Add)

RamTypes.EndNew(RamTypes.IndexOf(CurrentRamType));

await \_ctx.SaveChangesAsync();

EnDisFields(false);

Edited = true;

\_formMode = FormMode.None;

DialogResult = DialogResult.OK;

MessageBox.Show("Измененияуспешносохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

catch

{

MessageBox.Show("Изменениянеудалосьсохранить!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит отмену последнего добавления/редактирования

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

try

{

if (\_formMode == FormMode.Edit)

{

int indexOfElement = RamTypes.IndexOf(CurrentRamType);

CurrentRamType = \_ctx.CancelChanges(CurrentRamType);

RamTypes[indexOfElement] = CurrentRamType;

}

else

RamTypes.CancelNew(RamTypes.IndexOf(CurrentRamType));

EnDisFields(false);

\_formMode = FormMode.None;

}

catch

{

MessageBox.Show("Произошлаошибкаприотменеизменений!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который совершает подтверждение выбранного типа оперативной памяти

/// </summary>

private void but\_Select\_Click(object sender, EventArgs e)

{

if (CurrentRamType != null)

{

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Выберите тип оперативной памяти из списка или добавьте новый!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditRAMTypeForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Ограничение ввода в наименовании добавляемого/редактируемого типа оперативной памяти

/// </summary>

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!Char.IsControl(e.KeyChar) && !Char.IsLetter(e.KeyChar) && e.KeyChar != (int)Keys.Space && !Char.IsDigit(e.KeyChar))

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит очистку строки для поиска

/// </summary>

private void picBtn\_ClearCondition\_Click(object sender, EventArgs e)

{

textBox\_ConditionToFind.Text = string.Empty;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит поиск типа оперативной памяти по наименованию

/// </summary>

private void picBtn\_FindByCondition\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(textBox\_ConditionToFind.Text))

{

MessageBox.Show("Введите условие для поиска типа оперативной памяти!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

var firstFoundedElement = RamTypes.FirstOrDefault(x => x.Name.ToUpper().Contains(textBox\_ConditionToFind.Text.ToUpper()));

if (firstFoundedElement != null)

ramTypeBindingSource.Position = RamTypes.IndexOf(firstFoundedElement);

else

MessageBox.Show("Не удалось найти тип оперативной памяти по указанному условию!", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

}

Модуль «EditStorageInterfaceForm.cs»

using System;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Dictionary

{

/// <summary>

/// Класс формы редактирования интерфейсов накопителей

/// </summary>

public partial class EditStorageInterfaceForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Список интерфейсов накопителей

/// </summary>

public BindingList<StrorageInterface> StorageInterfaces { get; set; }

/// <summary>

/// Текущий интерфейс накопителя

/// </summary>

public StrorageInterface CurrentStorageInterface { get; set; }

/// <summary>

/// Наименование интерфейса накопителя до изменения

/// </summary>

private string \_storageInterfaceNameBeforeEditing;

/// <summary>

/// Режимработыформы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Флаг редактирования интерфейсов накопителей

/// </summary>

public bool Edited { get; set; }

/// <summary>

/// Конструктор формы редактирования интерфейсов накопителей

/// </summary>

public EditStorageInterfaceForm()

{

InitializeComponent();

\_formMode = FormMode.None;

\_ctx = new Context();

EnDisFields(false);

BindCollection();

}

/// <summary>

/// Конструктор для выбора интерфейса накопителя из списка

/// </summary>

/// <param name="findMode">Флаг поиска интерфейса накопителя</param>

public EditStorageInterfaceForm(bool findMode) : this()

{

if (!findMode)

throw new ArgumentException("Неверноезначениефлага!");

\_formMode = FormMode.None;

but\_Select.Visible = findMode;

but\_Select.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

}

/// <summary>

/// Привязка коллекции интерфейсов накопителей

/// </summary>

private void BindCollection()

{

StorageInterfaces = \_ctx.GetStorageInterfacesBS();

strorageInterfaceBindingSource.DataSource = StorageInterfaces;

listBox1.DataSource = strorageInterfaceBindingSource;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который осуществляет добавление нового интерфейса накопителя

/// </summary>

private void button2\_Click(object sender, EventArgs e)

{

StorageInterfaces.AddNew();

strorageInterfaceBindingSource.MoveLast();

EnDisFields(true);

\_formMode = FormMode.Add;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который осуществляет редактирование выбранного интерфейса накопителя

/// </summary>

private void button3\_Click(object sender, EventArgs e)

{

if (CurrentStorageInterface != null)

{

EnDisFields(true);

\_storageInterfaceNameBeforeEditing = CurrentStorageInterface.Name;

\_formMode = FormMode.Edit;

}

else

MessageBox.Show("Выберите интерфейс накопителя из списка или добавьте новый!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Активация/деактивация полей для редактирования

/// </summary>

/// <param name="enabled">Флаг активации полей редактирования</param>

private void EnDisFields(bool enabled)

{

textBox2.Enabled = enabled;

btn\_Save.Enabled = enabled;

btn\_Cancel.Enabled = enabled;

textBox\_ConditionToFind.Enabled = !enabled;

listBox1.Enabled = !enabled;

picBtn\_FindByCondition.Enabled = !enabled;

picBtn\_ClearCondition.Enabled = !enabled;

button2.Enabled = !enabled;

button3.Enabled = !enabled;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных интерфейсов накопителя

/// </summary>

private void strorageInterfaceBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as StrorageInterface;

if (selectedItem != null)

CurrentStorageInterface = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который выполняет сохранение изменений

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

try

{

if (string.IsNullOrWhiteSpace(CurrentStorageInterface.Name))

{

MessageBox.Show("Введите наименование интерфейса накопителя!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

bool existed = false;

switch (\_formMode)

{

case FormMode.Add:

existed = await \_ctx.CheckStorageInterfaceForDublicate(CurrentStorageInterface.Name);

break;

case FormMode.Edit:

existed = await \_ctx.CheckStorageInterfaceForDublicate(CurrentStorageInterface.Name) && !CurrentStorageInterface.Name.Equals(\_storageInterfaceNameBeforeEditing);

break;

}

if (existed)

{

MessageBox.Show("Интерфейс накопителя с таким наименованием уже существует!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (\_formMode == FormMode.Add)

StorageInterfaces.EndNew(StorageInterfaces.IndexOf(CurrentStorageInterface));

await \_ctx.SaveChangesAsync();

EnDisFields(false);

Edited = true;

\_formMode = FormMode.None;

DialogResult = DialogResult.OK;

MessageBox.Show("Измененияуспешносохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

catch

{

MessageBox.Show("Изменениянеудалосьсохранить!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит отмену последнего добавления/редактирования

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

try

{

if (\_formMode == FormMode.Edit)

{

int indexOfElement = StorageInterfaces.IndexOf(CurrentStorageInterface);

CurrentStorageInterface = \_ctx.CancelChanges(CurrentStorageInterface);

StorageInterfaces[indexOfElement] = CurrentStorageInterface;

}

else

StorageInterfaces.CancelNew(StorageInterfaces.IndexOf(CurrentStorageInterface));

EnDisFields(false);

\_formMode = FormMode.None;

}

catch

{

MessageBox.Show("Произошлаошибкаприотменеизменений!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который совершает подтверждение выбранного интерфейса накопителя

/// </summary>

private void but\_Select\_Click(object sender, EventArgs e)

{

if (CurrentStorageInterface != null)

{

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Выберите элемент из списка или добавьте новый!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditStorageInterfaceForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Ограничение ввода в наименовании добавляемого/редактируемого интерфейса накопителя

/// </summary>

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!Char.IsControl(e.KeyChar) && !Char.IsLetter(e.KeyChar) && e.KeyChar != (int)Keys.Space && !Char.IsDigit(e.KeyChar))

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит очистку строки для поиска

/// </summary>

private void picBtn\_ClearCondition\_Click(object sender, EventArgs e)

{

textBox\_ConditionToFind.Text = string.Empty;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит поиск интерфейса накопителя по наименованию

/// </summary>

private void picBtn\_FindByCondition\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(textBox\_ConditionToFind.Text))

{

MessageBox.Show("Введите условие для поиска интерфейса накопителя!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

var firstFoundedElement = StorageInterfaces.FirstOrDefault(x => x.Name.ToUpper().Contains(textBox\_ConditionToFind.Text.ToUpper()));

if (firstFoundedElement != null)

strorageInterfaceBindingSource.Position = StorageInterfaces.IndexOf(firstFoundedElement);

else

MessageBox.Show("Не удалось найти интерфейс накопителя по указанному условию!", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

}

Модуль «EditCPUForm.cs»

using System;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Dictionary;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Parts

{

/// <summary>

/// Класс формы редактирования процессора

/// </summary>

public partial class EditCPUForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Редактируемый накопитель

/// </summary>

public CPU CurrentCPU { get; set; }

/// <summary>

/// Наименование модели процессора перед редактированием

/// </summary>

private string \_cpuModelBeforeEdit;

/// <summary>

/// Наименование производителя процессора перед редактированием

/// </summary>

private string \_cpuManufacturerBeforeEdit;

/// <summary>

/// Режим работы формы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Флаг активации привязки данных после ее приоставновке в источнике данных производителей

/// </summary>

private bool \_afterManufacturerSuspend;

/// <summary>

/// Флаг активации привязки данных после ее приоставновке в источнике данных разъемов процессоров

/// </summary>

private bool \_afterCpuSocketSuspend;

/// <summary>

/// Конструктор формы создания нового процессора

/// </summary>

public EditCPUForm()

{

InitializeComponent();

btn\_Cancel.Image = new Bitmap(Resources.no, 20, 20);

btn\_Save.Image = new Bitmap(Resources.camera\_test, 20, 20);

\_ctx = new Context();

BindAll();

CurrentCPU = new CPU();

cPUBindingSource.DataSource = CurrentCPU;

\_formMode = FormMode.Add;

Text = "Создание процессора";

}

/// <summary>

/// Конструктор формы редактирования выбранного процессора

/// </summary>

/// <param name="selectedCPU">Выбранный процессор</param>

public EditCPUForm(CPU selectedCPU) : this()

{

\_formMode = FormMode.Edit;

Text = "Редактирование процессора";

InitEditCPU(selectedCPU);

}

#region Bindings

/// <summary>

/// Инициализация привязок всех коллекций

/// </summary>

private void BindAll()

{

BindManufacturer();

BindCPUSockets();

comboBox1.SelectedIndex = -1;

comboBox2.SelectedIndex = -1;

}

/// <summary>

/// Инициализация привязки коллекции с производителями

/// </summary>

private void BindManufacturer()

{

manufacturerBindingSource.DataSource = \_ctx.GetManufacturers();

comboBox1.DataSource = manufacturerBindingSource;

}

/// <summary>

/// Инициализация привязки коллекции с разъемами процессоров

/// </summary>

private void BindCPUSockets()

{

cpuSocketBindingSource.DataSource = \_ctx.GetCPUSockets();

comboBox2.DataSource = cpuSocketBindingSource;

}

/// <summary>

/// Инициализация привязок для редактирования выбраного процессора

/// </summary>

private void InitEditCPU(CPU selectedCPU)

{

CurrentCPU = selectedCPU;

cPUBindingSource.DataSource = CurrentCPU;

\_cpuManufacturerBeforeEdit = CurrentCPU.Manufacturer.Name;

\_cpuModelBeforeEdit = CurrentCPU.Model;

cPUBindingSource.ResumeBinding();

}

#endregion

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных производителей

/// </summary>

private void manufacturerBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (manufacturerBindingSource.IsBindingSuspended)

{

\_afterManufacturerSuspend = true;

manufacturerBindingSource.ResumeBinding();

}

else if (\_afterManufacturerSuspend)

{

\_afterManufacturerSuspend = false;

manufacturerBindingSource.Position = comboBox1.SelectedIndex;

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных разъемов процессоров

/// </summary>

private void cpuSocketBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (cpuSocketBindingSource.IsBindingSuspended)

{

\_afterCpuSocketSuspend = true;

cpuSocketBindingSource.ResumeBinding();

}

else if (\_afterCpuSocketSuspend)

{

\_afterCpuSocketSuspend = false;

cpuSocketBindingSource.Position = comboBox2.SelectedIndex;

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление выбранного производителя

/// </summary>

private void pictureBox1\_Click(object sender, EventArgs e)

{

comboBox1.SelectedIndex = -1;

manufacturerBindingSource.SuspendBinding();

if (CurrentCPU != null)

CurrentCPU.ManufacturerId = 0;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление выбранного разъема процессора

/// </summary>

private void pictureBox2\_Click(object sender, EventArgs e)

{

comboBox2.SelectedIndex = -1;

cpuSocketBindingSource.SuspendBinding();

if (CurrentCPU != null)

CurrentCPU.CpuSocketId = 0;

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditCPUForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменения не будут сохранены! Продолжить?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит сохранение добавляемой / редактируемой записи об процессоре

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит добавление выбранного интерфейса к доступным

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (comboBox1.SelectedIndex == -1 || comboBox2.SelectedIndex == -1 ||

string.IsNullOrWhiteSpace(CurrentCPU.Model) ||

CurrentCPU.Price <= 0 || CurrentCPU.Frequency <= 0 || CurrentCPU.Frequency <= 0)

{

MessageBox.Show("Заполните все поля для сохранения изменений!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

try

{

var selectedManufacturer = manufacturerBindingSource.Current as Manufacturer;

var selectedCPUSocket = cpuSocketBindingSource.Current as CpuSocket;

bool existed = false;

if (!selectedManufacturer.Name.Equals(selectedCPUSocket.Manufacturer.Name))

{

MessageBox.Show("Производитель процессора не совпадает с производителем разъема!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

switch (\_formMode)

{

case FormMode.Add:

existed = await \_ctx.CheckCPUForDublicate(CurrentCPU.Model, selectedManufacturer.Name);

break;

case FormMode.Edit:

existed = await \_ctx.CheckCPUForDublicate(CurrentCPU.Model, selectedManufacturer.Name)

&& (!CurrentCPU.Model.Equals(\_cpuModelBeforeEdit) ||

!selectedManufacturer.Name.Equals(\_cpuManufacturerBeforeEdit));

break;

}

if (existed)

{

MessageBox.Show("Процессор с таким производителем и моделью уже существует!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

switch (\_formMode)

{

case FormMode.Add:

await \_ctx.AddNewCPU(CurrentCPU);

MessageBox.Show("Новый процессор успешно сохранен!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

break;

case FormMode.Edit:

await \_ctx.EditCPU(CurrentCPU);

MessageBox.Show("Изменения в процессоре успешно сохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

break;

}

\_formMode = FormMode.None;

DialogResult = DialogResult.OK;

Close();

}

catch

{

MessageBox.Show("Изменения не удалось сохранить!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно редактирования производителей

/// </summary>

private void pictureBox4\_Click(object sender, EventArgs e)

{

var editManufacturer = new EditManufacturerForm();

if (editManufacturer.ShowDialog() == DialogResult.OK)

{

var selectedManufacturer = manufacturerBindingSource.Current as Manufacturer;

BindManufacturer();

if (selectedManufacturer != null)

{

int pos = manufacturerBindingSource.IndexOf(selectedManufacturer);

if (pos > -1)

manufacturerBindingSource.Position = pos;

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска производителя

/// </summary>

private void pictureBox17\_Click(object sender, EventArgs e)

{

var editManufacturer = new EditManufacturerForm(true);

if (editManufacturer.ShowDialog() == DialogResult.OK)

{

if (editManufacturer.Edited)

BindManufacturer();

manufacturerBindingSource\_CurrentChanged(manufacturerBindingSource, EventArgs.Empty);

manufacturerBindingSource.Position =

manufacturerBindingSource.IndexOf(editManufacturer.CurrentManufacturer);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно редактирования разъемов процессора

/// </summary>

private void pictureBox5\_Click(object sender, EventArgs e)

{

var editCpuSocket = new EditCPUSocketForm();

if (editCpuSocket.ShowDialog() == DialogResult.OK)

{

var cpuSocket = cpuSocketBindingSource.Current as CpuSocket;

BindCPUSockets();

if (cpuSocket != null)

{

int pos = cpuSocketBindingSource.IndexOf(cpuSocket);

if (pos > -1)

cpuSocketBindingSource.Position = pos;

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска разъема процессора

/// </summary>

private void pictureBox3\_Click(object sender, EventArgs e)

{

var editCpuSocket = new EditCPUSocketForm(true);

if (editCpuSocket.ShowDialog() == DialogResult.OK)

{

if (editCpuSocket.Edited)

BindCPUSockets();

cpuSocketBindingSource\_CurrentChanged(cpuSocketBindingSource, EventArgs.Empty);

cpuSocketBindingSource.Position =

cpuSocketBindingSource.IndexOf(editCpuSocket.CurrentSocket);

}

}

/// <summary>

/// Обработчик события форматирования отображаемого значения в списке разъемов процессоров

/// </summary>

private void comboBox2\_Format(object sender, ListControlConvertEventArgs e)

{

var socket = e.ListItem as CpuSocket;

if (socket != null)

e.Value = $"{socket.Manufacturer.Name} {socket.Name}";

}

/// <summary>

/// Обработчик нажатий клавиш на форме

/// </summary>

private void EditCPUForm\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.Escape:

Close();

break;

case Keys.Enter:

btn\_Save\_Click(null, EventArgs.Empty);

break;

}

}

/// <summary>

/// Обработчик события нажатия клавиши в текстовом поле с наименованием,

/// который ограничивает ввод только русских букв, пробела и управляющих символов

/// </summary>

private void textBox1\_KeyPress(object sender, KeyPressEventArgs e)

{

var l = e.KeyChar;

if (!

((l >= 'A' && l <= 'z')

|| l == (int)Keys.Space

|| char.IsDigit(e.KeyChar)

|| l == '\b'

|| char.IsControl(l)))

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатия клавиши в numericupdown с количеством ядер,

/// который ограничивает ввод только цифр и управляющих символов

/// </summary>

private void numericUpDown1\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!(char.IsDigit(e.KeyChar) || char.IsControl(e.KeyChar)))

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатия клавиши в текстовом поле с тактовой частотой и ценой,

/// который ограничивает ввод только цифр, управляющих символов, и 1 запятой

/// </summary>

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)

{

var txtBox = (TextBox)sender;

if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))

{

if (txtBox.Text.Length == 10 && txtBox.Text.Length <= 10)

e.Handled = true;

return;

}

// Точку заменим запятой

if (e.KeyChar == '.')

e.KeyChar = ',';

if (e.KeyChar == ',')

{

// Не более одной запятой и запятая не может быть первым символом.

if ((txtBox.Text.IndexOf(',') != -1) || (txtBox.Text.Length == 0))

e.Handled = true;

return;

}

if (Char.IsControl(e.KeyChar))

if (e.KeyChar == (char)Keys.Back)

return;

e.Handled = true;

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке производителей

/// </summary>

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox1.SelectedIndex > -1)

manufacturerBindingSource\_CurrentChanged(manufacturerBindingSource, EventArgs.Empty);

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке разъемов процессора

/// </summary>

private void comboBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox2.SelectedIndex > -1)

cpuSocketBindingSource\_CurrentChanged(cpuSocketBindingSource, EventArgs.Empty);

}

}

}

Модуль «EditPlatformForm.cs»

using System;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Dictionary;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Parts

{

/// <summary>

/// Класс формы редактирования платформы

/// </summary>

public partial class EditPlatformForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Наименование модели платформы перед редактированием

/// </summary>

private string \_modelNameBeforeEdit;

/// <summary>

/// Наименование производителя платформы перед редактированием

/// </summary>

private string \_manufacturerBeforeEdit;

/// <summary>

/// Редактируемуяплатформа

/// </summary>

public Platform CurrentPlatform { get; set; }

/// <summary>

/// Список поддерживаемых интерфейсов

/// </summary>

public BindingList<AvalibleInterface> AvalibleInterfaces { get; set; }

/// <summary>

/// Режим работы формы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Флагсохраненияизменений

/// </summary>

private bool \_saved;

/// <summary>

/// Флаг редактирования справочников или добавления/изменения платформ

/// </summary>

public bool Edited { get; set; }

/// <summary>

/// Флаг активации привязки данных после ее приоставновке в источнике данных производителей

/// </summary>

private bool \_afterManufacturerSuspend;

/// <summary>

/// Флаг активации привязки данных после ее приоставновке в источнике данных разъемов процессоров

/// </summary>

private bool \_afterCpuSocketSuspend;

/// <summary>

/// Флаг активации привязки данных после ее приоставновке в источнике данных типов оперативной памяти

/// </summary>

private bool \_afterRamTypeSuspend;

/// <summary>

/// Конструктор формы создания новой платформы

/// </summary>

public EditPlatformForm()

{

InitializeComponent();

btn\_Cancel.Image = new Bitmap(Resources.no, 20, 20);

btn\_Save.Image = new Bitmap(Resources.camera\_test, 20, 20);

\_ctx = new Context();

BindAll();

CurrentPlatform = new Platform();

platformBindingSource.DataSource = CurrentPlatform;

\_formMode = FormMode.Add;

Text = "Создание новой платформы";

}

/// <summary>

/// Конструктор формы редактирования выбранной платформы

/// </summary>

/// <param name="selectedPlatform">Выбраннаяплатформа</param>

public EditPlatformForm(PlatformInfo selectedPlatform) : this()

{

\_formMode = FormMode.Edit;

Text = "Редактированиеплатформы";

InitEditPlatform(selectedPlatform);

}

#region Bindings

/// <summary>

/// Инициализация привязок для редактирования выбранной платформы

/// </summary>

private void InitEditPlatform(PlatformInfo selectedPlatform)

{

CurrentPlatform = \_ctx.GetPlatformByPlatformInfo(selectedPlatform);

platformBindingSource.DataSource = CurrentPlatform;

\_manufacturerBeforeEdit = CurrentPlatform.Manufacturer.Name;

\_modelNameBeforeEdit = selectedPlatform.Model;

BindAvalibleInterfacesForEdit();

platformBindingSource.ResumeBinding();

}

/// <summary>

/// Инициализацияпривязоквсехколлекций

/// </summary>

private void BindAll()

{

BindCpuSockets();

BindManufacturer();

BindRamTypes();

BindStorageInterfaces();

BindAvalibleInterfacesNew();

comboBox1.SelectedIndex = -1;

comboBox2.SelectedIndex = -1;

comboBox3.SelectedIndex = -1;

}

/// <summary>

/// Инициализация привязки коллекции с производителями

/// </summary>

private void BindManufacturer()

{

manufacturerBindingSource.DataSource = \_ctx.GetManufacturers();

comboBox1.DataSource = manufacturerBindingSource;

}

/// <summary>

/// Инициализацияпривязкиколлекциисразъемамипроцессоров

/// </summary>

private void BindCpuSockets()

{

cpuSocketBindingSource.DataSource = \_ctx.GetCPUSockets();

comboBox2.DataSource = cpuSocketBindingSource;

}

/// <summary>

/// Инициализацияпривязкиколлекциистипамиоперативнойпамяти

/// </summary>

private void BindRamTypes()

{

ramTypeBindingSource.DataSource = \_ctx.GetRamsTypes();

comboBox3.DataSource = ramTypeBindingSource;

}

/// <summary>

/// Инициализацияпривязкиколлекциисинтерфейсаминакопителей

/// </summary>

private void BindStorageInterfaces()

{

strorageInterfaceBindingSource.DataSource = \_ctx.GetStorageInterfaces();

listBox1.DataSource = strorageInterfaceBindingSource;

}

/// <summary>

/// Привязкаколлекциидоступныхинтерфейсовнакопителейдляновойплатформы

/// </summary>

private void BindAvalibleInterfacesNew()

{

AvalibleInterfaces = new BindingList<AvalibleInterface>();

avalibleInterfaceBindingSource.DataSource = AvalibleInterfaces;

dataGridView1.DataSource = avalibleInterfaceBindingSource;

}

/// <summary>

/// Привязка коллекции доступных интерфейсов накопителей для редактируемой платформы

/// </summary>

private void BindAvalibleInterfacesForEdit()

{

AvalibleInterfaces = \_ctx.GetAvalibleInterfacesOfPlarformBS(CurrentPlatform.Platform\_StorageInt);

avalibleInterfaceBindingSource.DataSource = AvalibleInterfaces;

dataGridView1.DataSource = avalibleInterfaceBindingSource;

}

#endregion

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска производителя

/// </summary>

private void pictureBox17\_Click(object sender, EventArgs e)

{

var findManufacturer = new EditManufacturerForm(true);

if (findManufacturer.ShowDialog() == DialogResult.OK)

{

if (findManufacturer.Edited)

{

BindManufacturer();

Edited = true;

}

manufacturerBindingSource\_CurrentChanged(manufacturerBindingSource, EventArgs.Empty);

manufacturerBindingSource.Position =

((BindingList<Manufacturer>)manufacturerBindingSource.DataSource).IndexOf(findManufacturer.CurrentManufacturer);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска разъема процессора

/// </summary>

private void pictureBox4\_Click(object sender, EventArgs e)

{

var cpuSocket = new EditCPUSocketForm(true);

if (cpuSocket.ShowDialog() == DialogResult.OK)

{

if (cpuSocket.Edited)

{

BindCpuSockets();

Edited = true;

}

cpuSocketBindingSource\_CurrentChanged(cpuSocketBindingSource, EventArgs.Empty);

cpuSocketBindingSource.Position =

((BindingList<CpuSocket>)cpuSocketBindingSource.DataSource).IndexOf(cpuSocket.CurrentSocket);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска типа оперативной памяти

/// </summary>

private void pictureBox5\_Click(object sender, EventArgs e)

{

var ramType = new EditRAMTypeForm(true);

if (ramType.ShowDialog() == DialogResult.OK)

{

if (ramType.Edited)

{

BindRamTypes();

Edited = true;

}

ramTypeBindingSource\_CurrentChanged(ramTypeBindingSource, EventArgs.Empty);

ramTypeBindingSource.Position =

((BindingList<RamType>)ramTypeBindingSource.DataSource).IndexOf(ramType.CurrentRamType);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление выбранного производителя

/// </summary>

private void pictureBox1\_Click(object sender, EventArgs e)

{

comboBox1.SelectedIndex = -1;

manufacturerBindingSource.SuspendBinding();

if (CurrentPlatform != null)

CurrentPlatform.ManufacturerId = 0;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаграфическийобъект,

/// который производит удаление выбранного разъема процессора

/// </summary>

private void pictureBox2\_Click(object sender, EventArgs e)

{

comboBox2.SelectedIndex = -1;

cpuSocketBindingSource.SuspendBinding();

if (CurrentPlatform != null)

CurrentPlatform.CpuSocketId = 0;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление выбранного типа оперативной памяти

/// </summary>

private void pictureBox3\_Click(object sender, EventArgs e)

{

comboBox3.SelectedIndex = -1;

ramTypeBindingSource.SuspendBinding();

if (CurrentPlatform != null)

CurrentPlatform.RamTypeId = 0;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных производителей

/// </summary>

private void manufacturerBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (manufacturerBindingSource.IsBindingSuspended)

{

\_afterManufacturerSuspend = true;

manufacturerBindingSource.ResumeBinding();

}

else if (\_afterManufacturerSuspend)

{

\_afterManufacturerSuspend = false;

manufacturerBindingSource.Position = comboBox1.SelectedIndex;

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных разъемов процессоров

/// </summary>

private void cpuSocketBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (cpuSocketBindingSource.IsBindingSuspended)

{

\_afterCpuSocketSuspend = true;

cpuSocketBindingSource.ResumeBinding();

}

else if (\_afterCpuSocketSuspend)

{

\_afterCpuSocketSuspend = false;

cpuSocketBindingSource.Position = comboBox2.SelectedIndex;

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных типов оперативной памяти

/// </summary>

private void ramTypeBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (ramTypeBindingSource.IsBindingSuspended)

{

\_afterRamTypeSuspend = true;

ramTypeBindingSource.ResumeBinding();

}

else if (\_afterRamTypeSuspend)

{

\_afterRamTypeSuspend = false;

ramTypeBindingSource.Position = comboBox3.SelectedIndex;

}

}

/// <summary>

/// Обработчик события форматирования отображаемого значения в списке разъемов процессоров

/// </summary>

private void comboBox2\_Format(object sender, ListControlConvertEventArgs e)

{

var cpuToFormat = e.ListItem as CpuSocket;

if (cpuToFormat != null)

e.Value = $"{cpuToFormat.Manufacturer.Name} {cpuToFormat.Name}";

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит добавление выбранного интерфейса к доступным

/// </summary>

private void pictureBox7\_Click(object sender, EventArgs e)

{

var selectedInterface = strorageInterfaceBindingSource.Current as StrorageInterface;

if (selectedInterface != null)

{

var existedInt = AvalibleInterfaces.SingleOrDefault(ai => ai.Name.Equals(selectedInterface.Name));

if (existedInt != null)

{

existedInt.Slot\_Count++;

AvalibleInterfaces.Remove(existedInt);

AvalibleInterfaces.Add(existedInt);

}

else

AvalibleInterfaces.Add(new AvalibleInterface

{

Name = selectedInterface.Name,

Slot\_Count = 1

});

}

else

MessageBox.Show("Выберитеинтерфейсизсписка!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит увеличение количества выбранного интерфейса

/// </summary>

private void pictureBox21\_Click(object sender, EventArgs e)

{

var selectedInterface = avalibleInterfaceBindingSource.Current as AvalibleInterface;

if (selectedInterface != null)

{

selectedInterface.Slot\_Count++;

AvalibleInterfaces.Remove(selectedInterface);

AvalibleInterfaces.Add(selectedInterface);

}

else

MessageBox.Show("Выберите интерфейс из списка или добавьте новый!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит уменьшение количества выбранного интерфейса

/// </summary>

private void pictureBox20\_Click(object sender, EventArgs e)

{

var selectedInterface = avalibleInterfaceBindingSource.Current as AvalibleInterface;

if (selectedInterface != null)

{

selectedInterface.Slot\_Count--;

AvalibleInterfaces.Remove(selectedInterface);

if (selectedInterface.Slot\_Count > 0)

AvalibleInterfaces.Add(selectedInterface);

}

else

MessageBox.Show("Выберитеинтерфейсизспискаилидобавьтеновый!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление выбранного интерфейса

/// </summary>

private void pictureBox19\_Click(object sender, EventArgs e)

{

var selectedInterface = avalibleInterfaceBindingSource.Current as AvalibleInterface;

if (selectedInterface != null)

{

var result = MessageBox.Show("Выбранныйинтерфейсбудетудален! Продолжить?",

"Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

if (result == DialogResult.Yes)

AvalibleInterfaces.Remove(selectedInterface);

}

else

MessageBox.Show("Выберитеинтерфейсизспискаилидобавьтеновый!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит сохранение добавляемой / редактируемой записи об платформе

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (comboBox1.SelectedIndex == -1 || comboBox2.SelectedIndex == -1 || comboBox3.SelectedIndex == -1 ||

string.IsNullOrWhiteSpace(CurrentPlatform.Model) || CurrentPlatform.RamVolumeMax <= 0 ||

CurrentPlatform.RamSocketCount <= 0 ||

CurrentPlatform.Price <= 0 || CurrentPlatform.CPUCount <= 0)

{

MessageBox.Show("Заполните все поля для сохранения изменений!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

try

{

var selectedManufacturer = manufacturerBindingSource.Current as Manufacturer;

bool existed = false;

switch (\_formMode)

{

case FormMode.Add:

existed = await \_ctx.CheckPlatformForDublicate(CurrentPlatform.Model, selectedManufacturer.Name);

break;

case FormMode.Edit:

existed = await \_ctx.CheckPlatformForDublicate(CurrentPlatform.Model, selectedManufacturer.Name)

&& (!CurrentPlatform.Model.Equals(\_modelNameBeforeEdit) || !selectedManufacturer.Name.Equals(\_manufacturerBeforeEdit));

break;

}

if (existed)

{

MessageBox.Show("Платформа с таким производителем и моделью уже существует!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

CurrentPlatform.Platform\_StorageInt = \_ctx.GetPlatformStorageIntFromAvalible(CurrentPlatform, AvalibleInterfaces.ToList());

switch (\_formMode)

{

case FormMode.Add:

await \_ctx.AddNewPlatform(CurrentPlatform);

MessageBox.Show("Новаяплатформауспешносохранена!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

break;

case FormMode.Edit:

await \_ctx.EditPlatform(CurrentPlatform);

MessageBox.Show("Изменениявплатформеуспешносохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

break;

}

\_saved = true;

DialogResult = DialogResult.OK;

Close();

}

catch

{

MessageBox.Show("Изменения не удалось сохранить!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditPlatformForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (!\_saved)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит отмену создания / редактирования платформы, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке производителей

/// </summary>

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox1.SelectedIndex > -1)

manufacturerBindingSource\_CurrentChanged(manufacturerBindingSource, EventArgs.Empty);

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке разъемов процессора

/// </summary>

private void comboBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox2.SelectedIndex > -1)

cpuSocketBindingSource\_CurrentChanged(cpuSocketBindingSource, EventArgs.Empty);

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке типов оперативной памяти

/// </summary>

private void comboBox3\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox3.SelectedIndex > -1)

ramTypeBindingSource\_CurrentChanged(ramTypeBindingSource, EventArgs.Empty);

}

/// <summary>

/// Обработчик события нажатия клавиши в текстовом поле с наименованием,

/// который ограничивает ввод только русских букв, пробела и управляющих символов

/// </summary>

private void textBox1\_KeyPress(object sender, KeyPressEventArgs e)

{

var l = e.KeyChar;

if (!

((l >= 'A' && l <= 'z')

|| l == (int)Keys.Space

|| char.IsDigit(e.KeyChar)

|| l == '\b'

|| char.IsControl(l)))

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатия клавиши в текстовом поле с макс. количеством ОЗУ,

/// который ограничивает ввод только цифр и управляющих символов

/// </summary>

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!(char.IsDigit(e.KeyChar) || char.IsControl(e.KeyChar)))

e.Handled = true;

}

/// <summary>

/// Обработчиксобытиянажатияклавишив numericupdown сколичествомразъемов,

/// который ограничивает ввод только цифр и управляющих символов

/// </summary>

private void numericUpDown1\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!(char.IsDigit(e.KeyChar) || char.IsControl(e.KeyChar)))

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатия клавиши в текстовом поле с ценой,

/// который ограничивает ввод только цифр, управляющих символов, и 1 запятой

/// </summary>

private void textBox3\_KeyPress(object sender, KeyPressEventArgs e)

{

var txtBox = (TextBox)sender;

if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))

{

if (txtBox.Text.Length == 10 && txtBox.Text.Length <= 10)

e.Handled = true;

return;

}

// Точкузаменимзапятой

if (e.KeyChar == '.')

e.KeyChar = ',';

if (e.KeyChar == ',')

{

// Не более одной запятой и запятая не может быть первым символом.

if ((txtBox.Text.IndexOf(',') != -1) || (txtBox.Text.Length == 0))

e.Handled = true;

return;

}

if (Char.IsControl(e.KeyChar))

if (e.KeyChar == (char)Keys.Back)

return;

e.Handled = true;

}

}

}

Модуль «EditRAMForm.cs»

using System;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Dictionary;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Parts

{

/// <summary>

/// Класс формы редактирования оперативной памяти

/// </summary>

public partial class EditRAMForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Редактируемуя оперативная память

/// </summary>

public RAM CurrentRAM { get; set; }

/// <summary>

/// Наименование модели оперативной памяти перед редактированием

/// </summary>

private string \_ramModelBeforeEdit;

/// <summary>

/// Наименование производителя оперативной памяти перед редактированием

/// </summary>

private string \_ramManufacturerBeforeEdit;

/// <summary>

/// Режимработыформы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Флаг активации привязки данных после ее приоставновке в источнике данных производителей

/// </summary>

private bool \_afterManufacturerSuspend;

/// <summary>

/// Флаг активации привязки данных после ее приоставновке в источнике данных типов оперативной памяти

/// </summary>

private bool \_afterRamTypeSuspend;

/// <summary>

/// Конструктор формы создания новой оперативной памяти

/// </summary>

public EditRAMForm()

{

InitializeComponent();

btn\_Cancel.Image = new Bitmap(Resources.no, 20, 20);

btn\_Save.Image = new Bitmap(Resources.camera\_test, 20, 20);

\_ctx = new Context();

BindAll();

CurrentRAM = new RAM();

rAMBindingSource.DataSource = CurrentRAM;

\_formMode = FormMode.Add;

Text = "Создание оперативной памяти";

}

/// <summary>

/// Конструктор формы редактирования выбранной оперативной памяти

/// </summary>

public EditRAMForm(RAM selectedRAM) : this()

{

\_formMode = FormMode.Edit;

Text = "Редактирование оперативной памяти";

InitEditRAM(selectedRAM);

}

#region Bindings

/// <summary>

/// Инициализация привязок всех коллекций

/// </summary>

private void BindAll()

{

BindManufacturer();

BindRAMTypes();

comboBox1.SelectedIndex = -1;

comboBox2.SelectedIndex = -1;

}

/// <summary>

/// Инициализация привязки коллекции с производителями

/// </summary>

private void BindManufacturer()

{

manufacturerBindingSource.DataSource = \_ctx.GetManufacturers();

comboBox1.DataSource = manufacturerBindingSource;

}

/// <summary>

/// Инициализацияпривязкиколлекциистипамиоперативнойпамяти

/// </summary>

private void BindRAMTypes()

{

ramTypeBindingSource.DataSource = \_ctx.GetRamsTypes();

comboBox2.DataSource = ramTypeBindingSource;

}

/// <summary>

/// Инициализацияпривязокдляредактированиявыбраннойоперативнойпамяти

/// </summary>

private void InitEditRAM(RAM selectedRAM)

{

CurrentRAM = selectedRAM;

rAMBindingSource.DataSource = CurrentRAM;

\_ramManufacturerBeforeEdit = CurrentRAM.Manufacturer.Name;

\_ramModelBeforeEdit = CurrentRAM.Model;

rAMBindingSource.ResumeBinding();

}

#endregion

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных производителей

/// </summary>

private void manufacturerBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (manufacturerBindingSource.IsBindingSuspended)

{

\_afterManufacturerSuspend = true;

manufacturerBindingSource.ResumeBinding();

}

else if (\_afterManufacturerSuspend)

{

\_afterManufacturerSuspend = false;

manufacturerBindingSource.Position = comboBox1.SelectedIndex;

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных типов оперативной памяти

/// </summary>

private void ramTypeBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (ramTypeBindingSource.IsBindingSuspended)

{

\_afterRamTypeSuspend = true;

ramTypeBindingSource.ResumeBinding();

}

else if (\_afterRamTypeSuspend)

{

\_afterRamTypeSuspend = false;

ramTypeBindingSource.Position = comboBox2.SelectedIndex;

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление выбранного производителя

/// </summary>

private void pictureBox1\_Click(object sender, EventArgs e)

{

comboBox1.SelectedIndex = -1;

manufacturerBindingSource.SuspendBinding();

if (CurrentRAM != null)

CurrentRAM.ManufacturerId = 0;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление выбранного типа оперативной памяти

/// </summary>

private void pictureBox2\_Click(object sender, EventArgs e)

{

comboBox2.SelectedIndex = -1;

ramTypeBindingSource.SuspendBinding();

if (CurrentRAM != null)

CurrentRAM.RamTypeId = 0;

}

/// <summary>

/// Обработчиксобытиязакрытияформы

/// </summary>

private void EditRAMForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит отмену создания / редактирования оперативной памяти, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит сохранение добавляемой / редактируемой записи об оперативной памяти

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (comboBox1.SelectedIndex == -1 || comboBox2.SelectedIndex == -1 || string.IsNullOrWhiteSpace(CurrentRAM.Model) ||

CurrentRAM.Price <= 0 || CurrentRAM.Volume <= 0)

{

MessageBox.Show("Заполнитевсеполядлясохраненияизменений!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

try

{

var selectedManufacturer = manufacturerBindingSource.Current as Manufacturer;

bool existed = false;

switch (\_formMode)

{

case FormMode.Add:

existed = await \_ctx.CheckRAMForDublicate(CurrentRAM.Model, selectedManufacturer.Name);

break;

case FormMode.Edit:

existed = await \_ctx.CheckRAMForDublicate(CurrentRAM.Model, selectedManufacturer.Name)

&& (!CurrentRAM.Model.Equals(\_ramModelBeforeEdit) || !selectedManufacturer.Name.Equals(\_ramManufacturerBeforeEdit));

break;

}

if (existed)

{

MessageBox.Show("Оперативная память с таким производителем и моделью уже существует!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

switch (\_formMode)

{

case FormMode.Add:

await \_ctx.AddNewRAM(CurrentRAM);

MessageBox.Show("Новаяоперативнаяпамятьуспешносохранена!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

break;

case FormMode.Edit:

await \_ctx.EditRAM(CurrentRAM);

MessageBox.Show("Изменениявоперативнойпамятиуспешносохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

break;

}

\_formMode = FormMode.None;

DialogResult = DialogResult.OK;

Close();

}

catch

{

MessageBox.Show("Изменения не удалось сохранить!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно формы редактирования производителей

/// </summary>

private void pictureBox4\_Click(object sender, EventArgs e)

{

var editManufacturer = new EditManufacturerForm();

if (editManufacturer.ShowDialog() == DialogResult.OK)

{

var selectedManufacturer = manufacturerBindingSource.Current as Manufacturer;

BindManufacturer();

if (selectedManufacturer != null)

{

int pos = manufacturerBindingSource.IndexOf(selectedManufacturer);

if (pos > -1)

manufacturerBindingSource.Position = pos;

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска производителя

/// </summary>

private void pictureBox17\_Click(object sender, EventArgs e)

{

var editManufacturer = new EditManufacturerForm(true);

if (editManufacturer.ShowDialog() == DialogResult.OK)

{

if (editManufacturer.Edited)

BindManufacturer();

manufacturerBindingSource\_CurrentChanged(manufacturerBindingSource, EventArgs.Empty);

manufacturerBindingSource.Position =

manufacturerBindingSource.IndexOf(editManufacturer.CurrentManufacturer);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно редактирования типов оперативной памяти

/// </summary>

private void pictureBox5\_Click(object sender, EventArgs e)

{

var editRamType = new EditRAMTypeForm();

if (editRamType.ShowDialog() == DialogResult.OK)

{

var selectedRamType = ramTypeBindingSource.Current as RamType;

BindRAMTypes();

if (selectedRamType != null)

{

int pos = ramTypeBindingSource.IndexOf(selectedRamType);

if (pos > -1)

ramTypeBindingSource.Position = pos;

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска типа оперативной памяти

/// </summary>

private void pictureBox3\_Click(object sender, EventArgs e)

{

var editRamType = new EditRAMTypeForm(true);

if (editRamType.ShowDialog() == DialogResult.OK)

{

if (editRamType.Edited)

BindRAMTypes();

ramTypeBindingSource\_CurrentChanged(ramTypeBindingSource, EventArgs.Empty);

ramTypeBindingSource.Position =

ramTypeBindingSource.IndexOf(editRamType.CurrentRamType);

}

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке производителей

/// </summary>

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox1.SelectedIndex > -1)

manufacturerBindingSource\_CurrentChanged(manufacturerBindingSource, EventArgs.Empty);

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке типов оперативной памяти

/// </summary>

private void comboBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox2.SelectedIndex > -1)

ramTypeBindingSource\_CurrentChanged(ramTypeBindingSource, EventArgs.Empty);

}

/// <summary>

/// Обработчик события нажатия клавиши в текстовом поле с наименованием,

/// который ограничивает ввод только русских букв, пробела и управляющих символов

/// </summary>

private void textBox1\_KeyPress(object sender, KeyPressEventArgs e)

{

var l = e.KeyChar;

if (!

((l >= 'A' && l <= 'z')

|| l == (int)Keys.Space

|| char.IsDigit(e.KeyChar)

|| l == '\b'

|| char.IsControl(l)))

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатия клавиши в numericupdown с объемом модуля,

/// который ограничивает ввод только цифр и управляющих символов

/// </summary>

private void numericUpDown1\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!(char.IsDigit(e.KeyChar) || char.IsControl(e.KeyChar)))

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатия клавиши в текстовом поле с ценой,

/// который ограничивает ввод только цифр, управляющих символов, и 1 запятой

/// </summary>

private void textBox3\_KeyPress(object sender, KeyPressEventArgs e)

{

var txtBox = (TextBox)sender;

if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))

{

if (txtBox.Text.Length == 10 && txtBox.Text.Length <= 10)

e.Handled = true;

return;

}

// Точкузаменимзапятой

if (e.KeyChar == '.')

e.KeyChar = ',';

if (e.KeyChar == ',')

{

// Не более одной запятой и запятая не может быть первым символом.

if ((txtBox.Text.IndexOf(',') != -1) || (txtBox.Text.Length == 0))

e.Handled = true;

return;

}

if (Char.IsControl(e.KeyChar))

if (e.KeyChar == (char)Keys.Back)

return;

e.Handled = true;

}

}

}

Модуль «EditStorageForm.cs»

using System;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Dictionary;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Parts

{

/// <summary>

/// Класс формы редактирования накопителя

/// </summary>

public partial class EditStorageForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Редактируемыйнакопитель

/// </summary>

public Strorage CurrentStrorage { get; set; }

/// <summary>

/// Наименование модели накопителя перед редактированием

/// </summary>

private string \_storageModelBeforeEdit;

/// <summary>

/// Наименование производителя накопителя перед редактированием

/// </summary>

private string \_storageManufacturerBeforeEdit;

/// <summary>

/// Режимработыформы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Флаг активации привязки данных после ее приоставновке в источнике данных производителей

/// </summary>

private bool \_afterManufacturerSuspend;

/// <summary>

/// Флаг активации привязки данных после ее приоставновке в источнике данных типов интерфейса

/// </summary>

private bool \_afterStorageInterfaceSuspend;

/// <summary>

/// Конструктор формы создания нового накопителя

/// </summary>

public EditStorageForm()

{

InitializeComponent();

btn\_Cancel.Image = new Bitmap(Resources.no, 20, 20);

btn\_Save.Image = new Bitmap(Resources.camera\_test, 20, 20);

\_ctx = new Context();

BindAll();

CurrentStrorage = new Strorage();

strorageBindingSource.DataSource = CurrentStrorage;

\_formMode = FormMode.Add;

Text = "Создание накопителя";

}

/// <summary>

/// Конструктор формы редактирования выбранного накопителя

/// </summary>

public EditStorageForm(Strorage selectedStorage) : this()

{

\_formMode = FormMode.Edit;

Text = "Редактированиенакопителя";

InitEditStorage(selectedStorage);

}

#region Bindings

/// <summary>

/// Инициализация привязок всех коллекций

/// </summary>

private void BindAll()

{

BindManufacturer();

BindStorageInterfaces();

comboBox1.SelectedIndex = -1;

comboBox2.SelectedIndex = -1;

}

/// <summary>

/// Инициализация привязки коллекции с производителями

/// </summary>

private void BindManufacturer()

{

manufacturerBindingSource.DataSource = \_ctx.GetManufacturers();

comboBox1.DataSource = manufacturerBindingSource;

}

/// <summary>

/// Инициализацияпривязкиколлекциисинтерфейсамиобменаданных

/// </summary>

private void BindStorageInterfaces()

{

strorageInterfaceBindingSource.DataSource = \_ctx.GetStorageInterfaces();

comboBox2.DataSource = strorageInterfaceBindingSource;

}

/// <summary>

/// Инициализацияпривязокдляредактированиявыбранногонакопителя

/// </summary>

private void InitEditStorage(Strorage selectedStrorage)

{

CurrentStrorage = selectedStrorage;

strorageBindingSource.DataSource = CurrentStrorage;

\_storageManufacturerBeforeEdit = CurrentStrorage.Manufacturer.Name;

\_storageModelBeforeEdit = CurrentStrorage.Model;

strorageBindingSource.ResumeBinding();

}

#endregion

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных производителей

/// </summary>

private void manufacturerBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (manufacturerBindingSource.IsBindingSuspended)

{

\_afterManufacturerSuspend = true;

manufacturerBindingSource.ResumeBinding();

}

else if (\_afterManufacturerSuspend)

{

\_afterManufacturerSuspend = false;

manufacturerBindingSource.Position = comboBox1.SelectedIndex;

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных типа интерфейса накопителя

/// </summary>

private void strorageInterfaceBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (strorageInterfaceBindingSource.IsBindingSuspended)

{

\_afterStorageInterfaceSuspend = true;

strorageInterfaceBindingSource.ResumeBinding();

}

else if (\_afterStorageInterfaceSuspend)

{

\_afterStorageInterfaceSuspend = false;

strorageInterfaceBindingSource.Position = comboBox2.SelectedIndex;

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление выбранного производителя

/// </summary>

private void pictureBox1\_Click(object sender, EventArgs e)

{

comboBox1.SelectedIndex = -1;

manufacturerBindingSource.SuspendBinding();

if (CurrentStrorage != null)

CurrentStrorage.ManufacturerId = 0;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаграфическийобъект,

/// который производит удаление выбранного интерфейса

/// </summary>

private void pictureBox2\_Click(object sender, EventArgs e)

{

comboBox2.SelectedIndex = -1;

strorageInterfaceBindingSource.SuspendBinding();

if (CurrentStrorage != null)

CurrentStrorage.StrorageInterfaceId = 0;

}

/// <summary>

/// Обработчиксобытиязакрытияформы

/// </summary>

private void EditStorageForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит отмену создания / редактирования накопителя, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит сохранение добавляемой / редактируемой записи об накопителе

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (comboBox1.SelectedIndex == -1 || comboBox2.SelectedIndex == -1 || string.IsNullOrWhiteSpace(CurrentStrorage.Model) ||

CurrentStrorage.Price <= 0 || CurrentStrorage.Volume <= 0)

{

MessageBox.Show("Заполнитевсеполядлясохраненияизменений!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

try

{

var selectedManufacturer = manufacturerBindingSource.Current as Manufacturer;

bool existed = false;

switch (\_formMode)

{

case FormMode.Add:

existed = await \_ctx.CheckStorageForDublicate(CurrentStrorage.Model, selectedManufacturer.Name);

break;

case FormMode.Edit:

existed = await \_ctx.CheckStorageForDublicate(CurrentStrorage.Model, selectedManufacturer.Name)

&& (!CurrentStrorage.Model.Equals(\_storageModelBeforeEdit) || !selectedManufacturer.Name.Equals(\_storageManufacturerBeforeEdit));

break;

}

if (existed)

{

MessageBox.Show("Накопитель с таким производителем и моделью уже существует!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

switch (\_formMode)

{

case FormMode.Add:

await \_ctx.AddNewStorage(CurrentStrorage);

MessageBox.Show("Новыйнакопительуспешносохранен!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

break;

case FormMode.Edit:

await \_ctx.EditStorage(CurrentStrorage);

MessageBox.Show("Изменениявнакопителеуспешносохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

break;

}

\_formMode = FormMode.None;

DialogResult = DialogResult.OK;

Close();

}

catch

{

MessageBox.Show("Изменения не удалось сохранить!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно формы редактирования производителей

/// </summary>

private void pictureBox4\_Click(object sender, EventArgs e)

{

var editManufacturer = new EditManufacturerForm();

if (editManufacturer.ShowDialog() == DialogResult.OK)

{

var selectedManufacturer = manufacturerBindingSource.Current as Manufacturer;

BindManufacturer();

if (selectedManufacturer != null)

{

int pos = manufacturerBindingSource.IndexOf(selectedManufacturer);

if (pos > -1)

manufacturerBindingSource.Position = pos;

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска производителя

/// </summary>

private void pictureBox17\_Click(object sender, EventArgs e)

{

var editManufacturer = new EditManufacturerForm(true);

if (editManufacturer.ShowDialog() == DialogResult.OK)

{

if (editManufacturer.Edited)

BindManufacturer();

manufacturerBindingSource\_CurrentChanged(manufacturerBindingSource, EventArgs.Empty);

manufacturerBindingSource.Position =

manufacturerBindingSource.IndexOf(editManufacturer.CurrentManufacturer);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска интерфейса накопителя

/// </summary>

private void pictureBox3\_Click(object sender, EventArgs e)

{

var editStorageInterface = new EditStorageInterfaceForm(true);

if (editStorageInterface.ShowDialog() == DialogResult.OK)

{

if (editStorageInterface.Edited)

BindStorageInterfaces();

strorageInterfaceBindingSource\_CurrentChanged(strorageInterfaceBindingSource, EventArgs.Empty);

strorageInterfaceBindingSource.Position =

strorageInterfaceBindingSource.IndexOf(editStorageInterface.CurrentStorageInterface);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно редактирования интерфейсов накопителя

/// </summary>

private void pictureBox5\_Click(object sender, EventArgs e)

{

var editStorageInterface = new EditStorageInterfaceForm();

if (editStorageInterface.ShowDialog() == DialogResult.OK)

{

var strorageInterface = strorageInterfaceBindingSource.Current as StrorageInterface;

BindStorageInterfaces();

if (strorageInterface != null)

{

int pos = strorageInterfaceBindingSource.IndexOf(strorageInterface);

if (pos > -1)

strorageInterfaceBindingSource.Position = pos;

}

}

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке производителей

/// </summary>

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox1.SelectedIndex > -1)

manufacturerBindingSource\_CurrentChanged(manufacturerBindingSource, EventArgs.Empty);

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке интерфейсов накопителя

/// </summary>

private void comboBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox2.SelectedIndex > -1)

strorageInterfaceBindingSource\_CurrentChanged(strorageInterfaceBindingSource, EventArgs.Empty);

}

/// <summary>

/// Обработчик события нажатия клавиши в текстовом поле с наименованием,

/// который ограничивает ввод только русских букв, пробела и управляющих символов

/// </summary>

private void textBox1\_KeyPress(object sender, KeyPressEventArgs e)

{

var l = e.KeyChar;

if (!

((l >= 'A' && l <= 'z')

|| l == (int)Keys.Space

|| char.IsDigit(e.KeyChar)

|| l == '\b'

|| char.IsControl(l)))

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатия клавиши в текстовом поле с объемом,

/// который ограничивает ввод только цифр и управляющих символов

/// </summary>

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!(char.IsDigit(e.KeyChar) || char.IsControl(e.KeyChar)))

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатия клавиши в текстовом поле с ценой,

/// который ограничивает ввод только цифр, управляющих символов, и 1 запятой

/// </summary>

private void textBox3\_KeyPress(object sender, KeyPressEventArgs e)

{

var txtBox = (TextBox)sender;

if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))

{

if (txtBox.Text.Length == 10 && txtBox.Text.Length <= 10)

e.Handled = true;

return;

}

// Точкузаменимзапятой

if (e.KeyChar == '.')

e.KeyChar = ',';

if (e.KeyChar == ',')

{

// Не более одной запятой и запятая не может быть первым символом.

if ((txtBox.Text.IndexOf(',') != -1) || (txtBox.Text.Length == 0))

e.Handled = true;

return;

}

if (Char.IsControl(e.KeyChar))

if (e.KeyChar == (char)Keys.Back)

return;

e.Handled = true;

}

}

}

Модуль «PlatformInterfacesForm.cs»

using System;

using System.ComponentModel;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Parts

{

/// <summary>

/// Класс формы просмотра списка доступных интерфейсов хранения выбранной платформы

/// </summary>

public partial class PlatformInterfacesForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Конструктор формы просмотра списка доступных интерфейсов хранения выбранной платформы

/// </summary>

/// <param name="getAvalibleInterfacesOfPlarformBs"></param>

public PlatformInterfacesForm(BindingList<AvalibleInterface> avalibleInterfaces)

{

InitializeComponent();

btn\_OK.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

\_ctx = new Context();

avalibleInterfaceBindingSource.DataSource = avalibleInterfaces;

dataGridView1.DataSource = avalibleInterfaceBindingSource;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит закрытие формы

/// </summary>

private void btn\_OK\_Click(object sender, EventArgs e)

{

Close();

}

}

}

Модуль «SelectCPUForm.cs»

using System;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Parts

{

/// <summary>

/// Класс формы выбора процессора

/// </summary>

public partial class SelectCPUForm : Form

{

/// <summary>

/// Текущий разъем процессора

/// </summary>

private readonly CpuSocket \_currentCpuSocket;

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Текущийпроцессор

/// </summary>

public CPUInfo CurrentCPU { get; set; }

/// <summary>

/// Выбранныйпроцессор

/// </summary>

public CPU SelectedCPU { get; set; }

/// <summary>

/// Флаг отмены выбора процессора

/// </summary>

private bool \_canceled;

/// <summary>

/// Конструктор формы выбора процессора

/// </summary>

/// <param name="currentCpuSocket">Текущийразъемпроцессора</param>

public SelectCPUForm(CpuSocket currentCpuSocket)

{

\_currentCpuSocket = currentCpuSocket;

InitializeComponent();

SubscribeMenuItems();

btn\_OK.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

созданиеToolStripMenuItem1.Click += созданиеToolStripMenuItem\_Click;

редактированиеToolStripMenuItem1.Click += редактированиеToolStripMenuItem\_Click;

удалениеToolStripMenuItem1.Click += удалениеToolStripMenuItem\_Click;

\_ctx = new Context();

InitBindings();

}

/// <summary>

/// Инициализацияпривязокданных

/// </summary>

private void InitBindings()

{

cPUInfoBindingSource.DataSource = \_ctx.GetCPUsInfo();

dataGridView1.DataSource = cPUInfoBindingSource;

txt\_RecCount.Text = $"Общееколичество: {cPUInfoBindingSource.Count}";

}

/// <summary>

/// Обработчик события выхода курсора из элемента главного меню,

/// который очищает область в строке состояния.

/// </summary>

private void MenuItem\_MouseLeave(object sender, EventArgs e)

{

tip\_Label.Text = string.Empty;

}

/// <summary>

/// Обработчик события навдения курсора на элемент главного меню,

/// который выводит подсказку элемента в строку состояния.

/// </summary>

private void MenuItem\_MouseEnter(object sender, EventArgs e)

{

tip\_Label.Text = ((ToolStripItem)sender).ToolTipText;

}

/// <summary>

/// Подписка элементов главного меню и панели элементов на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

private void SubscribeMenuItems()

{

SubScribeChildMenuItems(menuBar.Items);

}

/// <summary>

/// Подписка дочерних элементов главного меню на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

/// <param name="col">Коллекция элементов главного меню</param>

private void SubScribeChildMenuItems(ToolStripItemCollection col)

{

for (int i = 0; i < col.Count; i++)

{

var item = col[i] as ToolStripMenuItem;

if (item != null)

{

item.MouseEnter += MenuItem\_MouseEnter;

item.MouseLeave += MenuItem\_MouseLeave;

if (item.DropDownItems.Count > 0)

SubScribeChildMenuItems(item.DropDownItems);

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который открывает диалоговое окно формы создания нового процессора

/// </summary>

private void созданиеToolStripMenuItem\_Click(object sender, EventArgs e)

{

var editCpuForm = new EditCPUForm();

if (editCpuForm.ShowDialog() == DialogResult.OK)

InitBindings();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы редактирования выбранного процессора

/// </summary>

private async void редактированиеToolStripMenuItem\_Click(object sender, EventArgs e)

{

SelectedCPU = await \_ctx.GetCPUByCPUInfo(CurrentCPU);

if (SelectedCPU != null)

{

var editCpuForm = new EditCPUForm(SelectedCPU);

if (editCpuForm.ShowDialog() == DialogResult.OK)

{

InitBindings();

if (CurrentCPU != null)

{

int pos = cPUInfoBindingSource.IndexOf(CurrentCPU);

if (pos > -1)

cPUInfoBindingSource.Position = pos;

}

}

}

else

MessageBox.Show("Выберите процессор из списка для его редактирования!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который осуществляет удаление выбранного процессора

/// </summary>

private async void удалениеToolStripMenuItem\_Click(object sender, EventArgs e)

{

SelectedCPU = await \_ctx.GetCPUByCPUInfo(CurrentCPU);

if (SelectedCPU != null)

{

var result = MessageBox.Show("Выбранныйпроцессорбудетудален! Продолжить?",

"Удалениепроцессора", MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

try

{

await \_ctx.DeleteCPU(SelectedCPU);

InitBindings();

}

catch

{

MessageBox.Show("Произошлаошибкаприудалениипроцессора!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

else

MessageBox.Show("Выберитепроцессоризспискадляегоудаления!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных процессоров

/// </summary>

private void cPUInfoBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as CPUInfo;

if (selectedItem != null)

CurrentCPU = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит отмену выбора процессора, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

\_canceled = true;

Close();

}

/// <summary>

/// Обработчиксобытиязакрытияформы

/// </summary>

private void SelectCPUForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_canceled)

{

var resut = MessageBox.Show("Процессорнебудетвыбран! Продолжить?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Hand);

if (resut == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит подтверждение выбранного процессора

/// </summary>

private async void btn\_OK\_Click(object sender, EventArgs e)

{

SelectedCPU = await \_ctx.GetCPUByCPUInfo(CurrentCPU);

if (SelectedCPU != null)

{

if (SelectedCPU.CpuSocket != \_currentCpuSocket)

{

MessageBox.Show("Выбранный процессор не подходит к разъему платформы", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Выберитепроцессоризсписка!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

}

}

Модуль «SelectPlatformForm.cs»

using System;

using System.ComponentModel;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Parts

{

/// <summary>

/// Класс формы выбора платформы

/// </summary>

public partial class SelectPlatformForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Выбранная платформа

/// </summary>

public PlatformInfo SelectedPlatformInfo { get; set; }

/// <summary>

/// Флаг редактирования платформ

/// </summary>

public bool Edited { get; set; }

/// <summary>

/// Конструктор формы выбора платформы

/// </summary>

public SelectPlatformForm()

{

InitializeComponent();

SubscribeMenuItems();

btn\_ShowSupportedInterfacesList.Image = new Bitmap(Resources.hdd, new Size(20, 20));

btn\_OK.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

созданиеToolStripMenuItem1.Click += созданиеToolStripMenuItem\_Click;

редактированиеToolStripMenuItem1.Click += редактированиеToolStripMenuItem\_Click;

удалениеToolStripMenuItem1.Click += удалениеToolStripMenuItem\_Click;

toolStripButton4.Click += btn\_ShowSupportedInterfacesList\_Click;

toolStripButton5.Click += созданиеToolStripMenuItem\_Click;

toolStripButton6.Click += редактированиеToolStripMenuItem\_Click;

toolStripButton7.Click += удалениеToolStripMenuItem\_Click;

\_ctx = new Context();

InitCollection();

}

/// <summary>

/// Инициализация привязки коллекции

/// </summary>

private void InitCollection()

{

platformInfoBindingSource.DataSource = \_ctx.GetPlatformsInfo();

dataGridView1.DataSource = platformInfoBindingSource;

}

/// <summary>

/// Обработчик события выхода курсора из элемента главного меню,

/// который очищает область в строке состояния.

/// </summary>

private void MenuItem\_MouseLeave(object sender, EventArgs e)

{

tip\_Label.Text = string.Empty;

}

/// <summary>

/// Обработчик события навдения курсора на элемент главного меню,

/// который выводит подсказку элемента в строку состояния.

/// </summary>

private void MenuItem\_MouseEnter(object sender, EventArgs e)

{

tip\_Label.Text = ((ToolStripItem)sender).ToolTipText;

}

/// <summary>

/// Подписка элементов главного меню и панели элементов на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

private void SubscribeMenuItems()

{

SubScribeChildMenuItems(menuBar.Items);

foreach (ToolStripItem toolBarControl in toolBar.Items)

{

toolBarControl.MouseEnter += MenuItem\_MouseEnter;

toolBarControl.MouseLeave += MenuItem\_MouseLeave;

}

}

/// <summary>

/// Подписка дочерних элементов главного меню на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

/// <paramname="col">Коллекция элементов главного меню</param>

private void SubScribeChildMenuItems(ToolStripItemCollection col)

{

for (int i = 0; i < col.Count; i++)

{

var item = col[i] as ToolStripMenuItem;

if (item != null)

{

item.MouseEnter += MenuItem\_MouseEnter;

item.MouseLeave += MenuItem\_MouseLeave;

if (item.DropDownItems.Count > 0)

SubScribeChildMenuItems(item.DropDownItems);

}

}

}

/// <summary>

/// Обработчик события изменения текущего источника данных платформ

/// </summary>

private void platformInfoBindingSource\_DataSourceChanged(object sender, EventArgs e)

{

var bs = sender as BindingSource;

if (bs != null && bs.DataSource is BindingList<PlatformInfo>)

lbl\_RecordsCount.Text = $"Кол-во: {((BindingList<PlatformInfo>)bs.DataSource).Count}";

else

lbl\_RecordsCount.Text = "Кол-во: 0";

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных платформ

/// </summary>

private void platformInfoBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as PlatformInfo;

if (selectedItem != null)

SelectedPlatformInfo = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену выбора платформы, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит подтверждение выбранной платформы

/// </summary>

private void btn\_OK\_Click(object sender, EventArgs e)

{

if (SelectedPlatformInfo != null)

{

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Выберите платформу из списка!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы создания новой платформы

/// </summary>

private void созданиеToolStripMenuItem\_Click(object sender, EventArgs e)

{

var editPlatformForm = new EditPlatformForm();

if (editPlatformForm.ShowDialog() == DialogResult.OK)

{

var selectedElement = platformInfoBindingSource.Current as PlatformInfo;

Edited = true;

InitCollection();

if (selectedElement != null)

platformInfoBindingSource.Position = platformInfoBindingSource.IndexOf(selectedElement);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы редактирования выбранной платформы

/// </summary>

private void редактированиеToolStripMenuItem\_Click(object sender, EventArgs e)

{

var selectedElement = platformInfoBindingSource.Current as PlatformInfo;

if (selectedElement == null)

{

MessageBox.Show("Выберите платформу из списка или добавьте новую!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

}

else

{

var editPlatformForm = new EditPlatformForm(selectedElement);

if (editPlatformForm.ShowDialog() == DialogResult.OK)

{

InitCollection();

Edited = true;

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который осуществляет удаление выбранной платформы

/// </summary>

private async void удалениеToolStripMenuItem\_Click(object sender, EventArgs e)

{

var selectedElement = platformInfoBindingSource.Current as PlatformInfo;

if (selectedElement == null)

{

MessageBox.Show("Выберите платформу из списка или добавьте новую!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

try

{

var platform = \_ctx.GetPlatformByPlatformInfo(selectedElement);

varresult = MessageBox.Show("Выбранная платформа будет удалена! Продолжить?",

"Удаление платформы", MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

try

{

await \_ctx.DeletePlatform(platform);

InitCollection();

}

catch

{

MessageBox.Show("Произошла ошибка при удалении оперативной памяти!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

catch

{

MessageBox.Show("Не удалось получить информацию о выбранной платформе!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который открывает диалоговое окно формы просмотра списка доступных интерфейсов хранения выбранной платформы

/// </summary>

private void btn\_ShowSupportedInterfacesList\_Click(object sender, EventArgs e)

{

if (SelectedPlatformInfo != null)

{

try

{

var selPlatform = \_ctx.GetPlatformByPlatformInfo(SelectedPlatformInfo);

var platformInterfacesForm = new PlatformInterfacesForm(\_ctx.GetAvalibleInterfacesOfPlarformBS(selPlatform.Platform\_StorageInt));

platformInterfacesForm.ShowDialog();

}

catch

{

MessageBox.Show("Ошибка при получении данных об выбранной платформе!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

}

}

else

MessageBox.Show("Выберите платформу из списка!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

}

}

}

Модуль «SelectRAMForm.cs»

using System;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Parts

{

/// <summary>

/// Класс формы выбора оперативной памяти

/// </summary>

public partial class SelectRAMForm : Form

{

private readonly RamType \_currentRamType;

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Текущий оперативная память

/// </summary>

public RAMInfo CurrentRAM { get; set; }

/// <summary>

/// Выбранная оперативная память

/// </summary>

public RAM SelectedRAM { get; set; }

/// <summary>

/// Флаг отмены выбора оперативной памяти

/// </summary>

private bool \_canceled;

/// <summary>

/// Конструктор формы выбора оперативной памяти

/// </summary>

/// <param name="currentRamType">Текущий тип памяти</param>

public SelectRAMForm(RamType currentRamType)

{

\_currentRamType = currentRamType;

InitializeComponent();

SubscribeMenuItems();

btn\_OK.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

созданиеToolStripMenuItem1.Click += созданиеToolStripMenuItem\_Click;

редактированиеToolStripMenuItem1.Click += редактированиеToolStripMenuItem\_Click;

удалениеToolStripMenuItem1.Click += удалениеToolStripMenuItem\_Click;

\_ctx = new Context();

InitBindings();

}

/// <summary>

/// Инициализация привязок данных

/// </summary>

private void InitBindings()

{

rAMInfoBindingSource.DataSource = \_ctx.GetRAMsInfo();

dataGridView1.DataSource = rAMInfoBindingSource;

txt\_RecCount.Text = $"Общее количество: {rAMInfoBindingSource.Count}";

}

/// <summary>

/// Обработчик события выхода курсора из элемента главного меню,

/// который очищает область в строке состояния.

/// </summary>

private void MenuItem\_MouseLeave(object sender, EventArgs e)

{

tip\_Label.Text = string.Empty;

}

/// <summary>

/// Обработчик события навдения курсора на элемент главного меню,

/// который выводит подсказку элемента в строку состояния.

/// </summary>

private void MenuItem\_MouseEnter(object sender, EventArgs e)

{

tip\_Label.Text = ((ToolStripItem)sender).ToolTipText;

}

/// <summary>

/// Подписка элементов главного меню и панели элементов на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

private void SubscribeMenuItems()

{

SubScribeChildMenuItems(menuBar.Items);

}

/// <summary>

/// Подписка дочерних элементов главного меню на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

/// <paramname="col">Коллекция элементов главного меню</param>

private void SubScribeChildMenuItems(ToolStripItemCollection col)

{

for (int i = 0; i < col.Count; i++)

{

var item = col[i] as ToolStripMenuItem;

if (item != null)

{

item.MouseEnter += MenuItem\_MouseEnter;

item.MouseLeave += MenuItem\_MouseLeave;

if (item.DropDownItems.Count > 0)

SubScribeChildMenuItems(item.DropDownItems);

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы создания новой оперативной памяти

/// </summary>

private void созданиеToolStripMenuItem\_Click(object sender, EventArgs e)

{

var editRamForm = new EditRAMForm();

if (editRamForm.ShowDialog() == DialogResult.OK)

InitBindings();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы редактирования выбранной оперативной памяти

/// </summary>

private async void редактированиеToolStripMenuItem\_Click(object sender, EventArgs e)

{

SelectedRAM = await \_ctx.GetRAMByRAMInfo(CurrentRAM);

if (SelectedRAM != null)

{

var editRamForm = new EditRAMForm(SelectedRAM);

if (editRamForm.ShowDialog() == DialogResult.OK)

{

InitBindings();

if (CurrentRAM != null)

{

int pos = rAMInfoBindingSource.IndexOf(CurrentRAM);

if (pos > -1)

rAMInfoBindingSource.Position = pos;

}

}

}

else

MessageBox.Show("Выберите оперативную память из списка для ее редактирования!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который осуществляет удаление выбранной оперативной памяти

/// </summary>

private async void удалениеToolStripMenuItem\_Click(object sender, EventArgs e)

{

SelectedRAM = await \_ctx.GetRAMByRAMInfo(CurrentRAM);

if (SelectedRAM != null)

{

var result = MessageBox.Show("Выбранная оперативная память будет удалена! Продолжить?",

"Удаление оперативной памяти", MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

try

{

await \_ctx.DeleteRAM(SelectedRAM);

InitBindings();

}

catch

{

MessageBox.Show("Произошла ошибка при удалении оперативной памяти!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

else

MessageBox.Show("Выберите оперативную память из списка для ее удаления!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных оперативной памяти

/// </summary>

private void rAMInfoBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as RAMInfo;

if (selectedItem != null)

CurrentRAM = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит отмену выбора оперативной памяти, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

\_canceled = true;

Close();

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void SelectRAMForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_canceled)

{

var resut = MessageBox.Show("Оперативная память не будет выбрана! Продолжить?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Hand);

if (resut == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит подтверждение выбранной оперативной памяти

/// </summary>

private async void btn\_OK\_Click(object sender, EventArgs e)

{

SelectedRAM = await \_ctx.GetRAMByRAMInfo(CurrentRAM);

if (SelectedRAM != null)

{

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Выберите оперативную память из списка!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

}

}

Модуль «SelectStorageForm.cs»

using System;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Parts

{

/// <summary>

/// Класс формы выбора накопителя

/// </summary>

public partial class SelectStorageForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Текущийнакопитель

/// </summary>

public StorageInfo CurrentStorage { get; set; }

/// <summary>

/// Выбранныйнакопитель

/// </summary>

public Strorage SelectedStorage { get; set; }

/// <summary>

/// Флаг отмены выбора накопителя

/// </summary>

private bool \_canceled;

/// <summary>

/// Конструктор формы выбора накопителя

/// </summary>

public SelectStorageForm()

{

InitializeComponent();

SubscribeMenuItems();

btn\_OK.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

созданиеToolStripMenuItem1.Click += созданиеToolStripMenuItem\_Click;

редактированиеToolStripMenuItem1.Click += редактированиеToolStripMenuItem\_Click;

удалениеToolStripMenuItem1.Click += удалениеToolStripMenuItem\_Click;

\_ctx = new Context();

InitBindings();

}

/// <summary>

/// Инициализацияпривязокданных

/// </summary>

private void InitBindings()

{

storageInfoBindingSource.DataSource = \_ctx.GetStoragesInfo();

dataGridView1.DataSource = storageInfoBindingSource;

txt\_RecCount.Text = $"Общееколичество: {storageInfoBindingSource.Count}";

}

/// <summary>

/// Обработчик события выхода курсора из элемента главного меню,

/// который очищает область в строке состояния.

/// </summary>

private void MenuItem\_MouseLeave(object sender, EventArgs e)

{

tip\_Label.Text = string.Empty;

}

/// <summary>

/// Обработчик события навдения курсора на элемент главного меню,

/// который выводит подсказку элемента в строку состояния.

/// </summary>

private void MenuItem\_MouseEnter(object sender, EventArgs e)

{

tip\_Label.Text = ((ToolStripItem)sender).ToolTipText;

}

/// <summary>

/// Подписка элементов главного меню и панели элементов на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

private void SubscribeMenuItems()

{

SubScribeChildMenuItems(menuBar.Items);

}

/// <summary>

/// Подписка дочерних элементов главного меню на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

/// <param name="col">Коллекция элементов главного меню</param>

private void SubScribeChildMenuItems(ToolStripItemCollection col)

{

for (int i = 0; i < col.Count; i++)

{

var item = col[i] as ToolStripMenuItem;

if (item != null)

{

item.MouseEnter += MenuItem\_MouseEnter;

item.MouseLeave += MenuItem\_MouseLeave;

if (item.DropDownItems.Count > 0)

SubScribeChildMenuItems(item.DropDownItems);

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который открывает диалоговое окно формы создания нового накопителя

/// </summary>

private void созданиеToolStripMenuItem\_Click(object sender, EventArgs e)

{

var editStorageForm = new EditStorageForm();

if (editStorageForm.ShowDialog() == DialogResult.OK)

InitBindings();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы редактирования выбранного накопителя

/// </summary>

private async void редактированиеToolStripMenuItem\_Click(object sender, EventArgs e)

{

SelectedStorage = await \_ctx.GetStorageByStorageInfo(CurrentStorage);

if (SelectedStorage != null)

{

var editStorageForm = new EditStorageForm(SelectedStorage);

if (editStorageForm.ShowDialog() == DialogResult.OK)

{

InitBindings();

if (CurrentStorage != null)

{

int pos = storageInfoBindingSource.IndexOf(CurrentStorage);

if (pos > -1)

storageInfoBindingSource.Position = pos;

}

}

}

else

MessageBox.Show("Выберите накопитель из списка для его редактирования!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который осуществляет удаление выбранного накопителя

/// </summary>

private async void удалениеToolStripMenuItem\_Click(object sender, EventArgs e)

{

SelectedStorage = await \_ctx.GetStorageByStorageInfo(CurrentStorage);

if (SelectedStorage != null)

{

var result = MessageBox.Show("Выбранныйнакопительбудетудален! Продолжить?",

"Удалениенакопителя", MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

try

{

await \_ctx.DeleteStrorage(SelectedStorage);

InitBindings();

}

catch

{

MessageBox.Show("Произошлаошибкаприудалениинакопителя!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

else

MessageBox.Show("Выберитенакопительизспискадляегоредактирования!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных накопителей

/// </summary>

private void storageInfoBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as StorageInfo;

if (selectedItem != null)

CurrentStorage = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит отмену выбора накопителя, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

\_canceled = true;

Close();

}

/// <summary>

/// Обработчиксобытиязакрытияформы

/// </summary>

private void SelectStorageForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_canceled)

{

var resut = MessageBox.Show("Накопительнебудетвыбран! Продолжить?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Hand);

if (resut == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит подтверждение выбранного накопителя

/// </summary>

private async void btn\_OK\_Click(object sender, EventArgs e)

{

SelectedStorage = await \_ctx.GetStorageByStorageInfo(CurrentStorage);

if (SelectedStorage != null)

{

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Выберите процессор из списка для его редактирования!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

}

}

Модуль «EditSANForm.cs»

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Dictionary;

using MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Parts;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.SANs

{

/// <summary>

/// Класс формы редактирования хранилища данных

/// </summary>

public partial class EditSANForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Режим работы формы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Редактируемоехранилищеданных.

/// </summary>

public SAN CurrentSAN { get; set; }

/// <summary>

/// Флаг активации привязки данных после ее приоставновке в источнике данных производителей

/// </summary>

private bool \_afterManufacturerSuspend;

/// <summary>

/// Списокподдерживаемыхинтерфейсов

/// </summary>

public BindingList<AvalibleInterface> AvalibleInterfaces { get; set; }

/// <summary>

/// Установленныенакопители.

/// </summary>

public BindingList<InstalledStorageOnSAN> InstalledStorageOnSAN { get; set; }

/// <summary>

/// Конструктор формы создания нового хранилища данных

/// </summary>

public EditSANForm()

{

InitializeComponent();

UpdateGridsFont(dg\_InstalledStorage);

\_ctx = new Context();

CurrentSAN = new SAN();

sANBindingSource.DataSource = CurrentSAN;

BindSANPartsInfo();

BindInstalledStorageOnSAN();

BindManufacturer();

BindStorageInterfaces();

BindAvalibleInterfacesNew();

BindStorages();

btn\_Save.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

\_formMode = FormMode.Add;

Text = "Cоздание нового хранилища данных";

comboBox1.SelectedIndex = -1;

}

/// <summary>

/// Конструктор формы редактирования выбранного хранилища данных.

/// </summary>

/// <param name="selectedSAN">Выбранноехранилищеданных</param>

public EditSANForm(SAN selectedSAN) : this()

{

\_formMode = FormMode.Edit;

Text = "Редактированиехранилищаданных";

CurrentSAN = selectedSAN;

sANBindingSource.DataSource = CurrentSAN;

sANBindingSource.ResetBindings(true);

PrepareBindingsToEditSAN();

}

/// <summary>

/// Инициализация привязок для редактирования выбранного хранилища данных

/// </summary>

private void PrepareBindingsToEditSAN()

{

BindAvalibleInterfacesForEdit();

BindStorages();

manufacturerBindingSource.Position =

((BindingList<Manufacturer>)manufacturerBindingSource.DataSource).IndexOf(CurrentSAN.Manufacturer);

comboBox1.SelectedIndex = manufacturerBindingSource.Position;

foreach (var s in CurrentSAN.SAN\_Storage)

{

if (s.Count != null)

InstalledStorageOnSAN.Add(new InstalledStorageOnSAN

{

Manufacturer = s.Strorage.Manufacturer.Name,

Model = s.Strorage.Model,

Interface = s.Strorage.StrorageInterface.Name,

Volume = s.Strorage.Volume,

Price = s.Strorage.Price,

Count = (byte)s.Count

});

}

var cfgPartInfo = sANPartsInfoBindingSource.DataSource as SANPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.Maintenance = CurrentSAN.AnnualMaintenance ?? 0;

var storageSumPrice = InstalledStorageOnSAN.Sum(iss => iss.Price \* iss.Count);

cfgPartInfo.InitialPrice = CurrentSAN.Price - storageSumPrice;

cfgPartInfo.StorageSumPrice += storageSumPrice;

cfgPartInfo.UpdateTotalPrice();

BindSANPartsInfo();

}

}

/// <summary>

/// Обновлениешрифтовтаблиц

/// </summary>

private void UpdateGridsFont(DataGridView dataGrid)

{

Font f = new Font("Segoe UI Semilight", 11F, GraphicsUnit.Pixel);

foreach (DataGridViewColumn column in dataGrid.Columns)

{

column.DefaultCellStyle.Font = f;

column.HeaderCell.Style.Font = f;

}

}

#region Bindings

/// <summary>

/// Привязка списка производителей

/// </summary>

private void BindManufacturer()

{

manufacturerBindingSource.DataSource = \_ctx.GetManufacturers();

comboBox1.DataSource = manufacturerBindingSource;

}

/// <summary>

/// Привязкаспискадоступныхинтерфейсовнакопителейдляновогохранилищаданных

/// </summary>

private void BindAvalibleInterfacesNew()

{

AvalibleInterfaces = new BindingList<AvalibleInterface>();

avalibleInterfaceBindingSource.DataSource = AvalibleInterfaces;

dg\_AvalibleInterfaces.DataSource = avalibleInterfaceBindingSource;

}

/// <summary>

/// Привязка списка доступных интерфейсов накопителей для редактируемого хранилища данных

/// </summary>

private void BindAvalibleInterfacesForEdit()

{

AvalibleInterfaces = \_ctx.GetAvalibleInterfacesOfSANBS(CurrentSAN.SAN\_StorageInt);

avalibleInterfaceBindingSource.DataSource = AvalibleInterfaces;

dg\_AvalibleInterfaces.DataSource = avalibleInterfaceBindingSource;

}

/// <summary>

/// Привязка списка существующих интерфейсов

/// </summary>

private void BindStorageInterfaces()

{

strorageInterfaceBindingSource.DataSource = \_ctx.GetStorageInterfaces();

lst\_Interfaces.DataSource = strorageInterfaceBindingSource;

}

/// <summary>

/// Привязка списка установленных накопителей на хранилище данных

/// </summary>

private void BindInstalledStorageOnSAN()

{

InstalledStorageOnSAN = new BindingList<InstalledStorageOnSAN>();

installedStorageOnSANBindingSource.DataSource = InstalledStorageOnSAN;

dg\_InstalledStorage.DataSource = installedStorageOnSANBindingSource;

}

/// <summary>

/// Привязка объекта, который отображает цены компонентов

/// </summary>

private void BindSANPartsInfo()

{

var sanPartInfo = sANPartsInfoBindingSource.DataSource as SANPartsInfo;

sANPartsInfoBindingSource.DataSource = sanPartInfo ?? new SANPartsInfo();

sANPartsInfoBindingSource.ResetBindings(true);

}

/// <summary>

/// Привязкаспискадоступныхнакопителей

/// </summary>

private void BindStorages()

{

if (AvalibleInterfaces == null || AvalibleInterfaces.Count == 0)

{

InstalledStorageOnSAN.Clear();

ManageStorageControls(SANPartSelectionMode.None);

var sanPartInfo = sANPartsInfoBindingSource.DataSource as SANPartsInfo;

if (sanPartInfo != null)

{

sanPartInfo.SelectedStoragePrice = 0;

sanPartInfo.StorageSumPrice = 0;

sanPartInfo.UpdateTotalPrice();

BindSANPartsInfo();

}

}

else

{

var storagesByAvalibleInterfaces = \_ctx.GetStoragesByAvalibleInterfaces(AvalibleInterfaces);

strorageBindingSource.DataSource = storagesByAvalibleInterfaces;

CheckInstalledStorage(storagesByAvalibleInterfaces);

ManageStorageControls(strorageBindingSource.Count > 0 ? SANPartSelectionMode.OK : SANPartSelectionMode.UnSupported);

}

}

#endregion

/// <summary>

/// Проверка установленных накопителей со списком доступных

/// </summary>

/// <param name="strorageList">Списокдоступныхнакопителей</param>

private void CheckInstalledStorage(BindingList<Strorage> strorageList)

{

var itemsToRemove = new Stack<InstalledStorageOnSAN>();

var sanPartInfo = sANPartsInfoBindingSource.DataSource as SANPartsInfo;

foreach (var storageOnServer in InstalledStorageOnSAN)

{

var existed = strorageList.SingleOrDefault(rl =>

rl.Manufacturer.Name.Equals(storageOnServer.Manufacturer) && rl.Model.Equals(storageOnServer.Model));

if (existed == null)

itemsToRemove.Push(storageOnServer);

}

while (itemsToRemove.Count > 0)

{

var itemToRemove = itemsToRemove.Pop();

sanPartInfo.StorageSumPrice -= (itemToRemove.Price \* itemToRemove.Count);

InstalledStorageOnSAN.Remove(itemToRemove);

}

sanPartInfo.UpdateTotalPrice();

BindSANPartsInfo();

}

/// <summary>

/// Управление элементами управления области накопителей

/// </summary>

/// <param name="selMode">Выбранныйрежим</param>

private void ManageStorageControls(SANPartSelectionMode selMode)

{

foreach (Control control in groupBox3.Controls)

control.Visible = selMode == SANPartSelectionMode.OK;

lbl\_NoInterfacesToStorage.Visible = selMode == SANPartSelectionMode.None;

lbl\_NoInterfacesToStorage.Dock = selMode == SANPartSelectionMode.None ? DockStyle.Fill : DockStyle.None;

lbl\_NoStorages.Visible = selMode == SANPartSelectionMode.UnSupported;

lbl\_NoStorages.Dock = selMode == SANPartSelectionMode.UnSupported ? DockStyle.Fill : DockStyle.None;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаграфическийобъект,

/// который осуществляет добавление выбранного интерфейса к доступным

/// </summary>

private void pictureBox7\_Click(object sender, EventArgs e)

{

var selectedInterface = strorageInterfaceBindingSource.Current as StrorageInterface;

if (selectedInterface == null)

{

MessageBox.Show("Выберитеинтерфейсизсписка!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

try

{

var existedInt = AvalibleInterfaces.SingleOrDefault(ai => ai.Name.Equals(selectedInterface.Name));

if (existedInt != null)

{

existedInt.Slot\_Count++;

avalibleInterfaceBindingSource.DataSource = AvalibleInterfaces;

dg\_AvalibleInterfaces.DataSource = avalibleInterfaceBindingSource;

dg\_AvalibleInterfaces.Refresh();

}

else

AvalibleInterfaces.Add(new AvalibleInterface

{

Name = selectedInterface.Name,

Slot\_Count = 1

});

BindStorages();

}

catch

{

MessageBox.Show("Произошлаошибкапридобавленииинтерфейса!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинатекстовуюметку,

/// который открывает диалоговое окно добавления нового накопителя, если он не выбран

/// </summary>

private void lbl\_NoStorages\_Click(object sender, EventArgs e)

{

var storageForm = new EditStorageForm();

if (storageForm.ShowDialog() == DialogResult.OK)

{

BindStorages();

strorageBindingSource.Position =

((BindingList<Strorage>)strorageBindingSource.DataSource).IndexOf(storageForm.CurrentStrorage);

comboBox\_Storage.SelectedIndex = strorageBindingSource.Position;

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных доступных интерфейсов накопителей

/// </summary>

private void strorageInterfaceBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var bs = sender as BindingSource;

var selStorage = bs.Current as Strorage;

if (comboBox\_Storage.SelectedIndex == -1)

{

bs.SuspendBinding();

selStorage = null;

}

else

bs.ResumeBinding();

var sanPartInfo = sANPartsInfoBindingSource.DataSource as SANPartsInfo;

if (sanPartInfo != null)

{

sanPartInfo.SelectedStoragePrice = selStorage != null ? selStorage.Price : 0;

BindSANPartsInfo();

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет удаление выбранного накопителя из установленных

/// </summary>

private void pictureBox14\_Click(object sender, EventArgs e)

{

var selectedStorage = strorageBindingSource.Current as Strorage;

if (selectedStorage == null)

{

MessageBox.Show("Накопительневыбран!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

var storageToDel = InstalledStorageOnSAN.SingleOrDefault(irs =>

irs.Model.Equals(selectedStorage.Model) && irs.Manufacturer.Equals(selectedStorage.Manufacturer.Name));

if (storageToDel == null)

{

MessageBox.Show("Выбранныйнакопительненайденвспискеустановленных!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

var result = MessageBox.Show("Выбранный накопитель будет удален из хранилища данных! Продолжить?",

"Удаление накопителя", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

if (result == DialogResult.Yes)

{

var sanPartInfo = sANPartsInfoBindingSource.DataSource as SANPartsInfo;

if (sanPartInfo != null)

{

sanPartInfo.StorageSumPrice -= (storageToDel.Price \* storageToDel.Count);

sanPartInfo.UpdateTotalPrice();

BindSANPartsInfo();

}

InstalledStorageOnSAN.Remove(storageToDel);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет добавление выбранного накопителя к установленным

/// </summary>

private void pictureBox13\_Click(object sender, EventArgs e)

{

var selectedStorage = strorageBindingSource.Current as Strorage;

if (selectedStorage == null)

{

MessageBox.Show("Выберитенакопительизсписка!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

var avalibleInterface =

AvalibleInterfaces.Single(av => av.Name.Equals(selectedStorage.StrorageInterface.Name));

var installedStoragesOfInterface =

InstalledStorageOnSAN.Where(iss => iss.Interface.Equals(avalibleInterface.Name)).ToList();

if (installedStoragesOfInterface.Count > 0)

{

int avalibleSlotCount =

avalibleInterface.Slot\_Count - installedStoragesOfInterface.Sum(isi => isi.Count);

if (avalibleSlotCount == 0)

{

MessageBox.Show($"Всеразъемыподинтерфейс {avalibleInterface.Name} заняты!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

}

try

{

var existedItem = InstalledStorageOnSAN.SingleOrDefault(iss =>

iss.Manufacturer.Equals(selectedStorage.Manufacturer.Name) && iss.Model.Equals(selectedStorage.Model));

if (existedItem != null)

{

existedItem.Count++;

installedStorageOnSANBindingSource.DataSource = InstalledStorageOnSAN;

dg\_InstalledStorage.DataSource = installedStorageOnSANBindingSource;

dg\_InstalledStorage.Refresh();

}

else

InstalledStorageOnSAN.Add(new InstalledStorageOnSAN

{

Model = selectedStorage.Model,

Manufacturer = selectedStorage.Manufacturer.Name,

Volume = selectedStorage.Volume,

Interface = selectedStorage.StrorageInterface.Name,

Count = 1,

Price = selectedStorage.Price

});

var sanPartInfo = sANPartsInfoBindingSource.DataSource as SANPartsInfo;

if (sanPartInfo != null)

{

sanPartInfo.StorageSumPrice += selectedStorage.Price;

sanPartInfo.UpdateTotalPrice();

BindSANPartsInfo();

}

}

catch

{

MessageBox.Show("Произошлаошибкапридобавлениинакопителя!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

#region Installed Storage operations

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет увеличение количества выбранного установленного накопителя

/// </summary>

private void picBtn\_IncreaseStorage\_Click(object sender, EventArgs e)

{

var selectedInstalledStorage = installedStorageOnSANBindingSource.Current as InstalledStorageOnSAN;

if (selectedInstalledStorage == null)

{

MessageBox.Show("Выберитеустановленныйнакопительизсписка!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

var avalibleInterface =

AvalibleInterfaces.Single(av => av.Name.Equals(selectedInstalledStorage.Interface));

var installedStoragesOfInterface =

InstalledStorageOnSAN.Where(iss => iss.Interface.Equals(avalibleInterface.Name)).ToList();

if (installedStoragesOfInterface.Count > 0)

{

int avalibleSlotCount =

avalibleInterface.Slot\_Count - installedStoragesOfInterface.Sum(isi => isi.Count);

if (avalibleSlotCount == 0)

{

MessageBox.Show($"Всеразъемыподинтерфейс {avalibleInterface.Name} заняты!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

}

try

{

selectedInstalledStorage.Count++;

installedStorageOnSANBindingSource.DataSource = InstalledStorageOnSAN;

dg\_InstalledStorage.DataSource = InstalledStorageOnSAN;

dg\_InstalledStorage.Refresh();

var sanPartInfo = sANPartsInfoBindingSource.DataSource as SANPartsInfo;

if (sanPartInfo != null)

{

sanPartInfo.StorageSumPrice += selectedInstalledStorage.Price;

sanPartInfo.UpdateTotalPrice();

BindSANPartsInfo();

}

}

catch

{

MessageBox.Show("Произошлаошибкапридобавлениинакопителя!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет уменьшение количества выбранного установленного накопителя

/// </summary>

private void picBtn\_DecreaseStorage\_Click(object sender, EventArgs e)

{

var selectedInstalledStorage = installedStorageOnSANBindingSource.Current as InstalledStorageOnSAN;

if (selectedInstalledStorage == null)

{

MessageBox.Show("Выберитеустановленныйнакопительизсписка!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

try

{

bool removed = false;

if (selectedInstalledStorage.Count > 1)

{

selectedInstalledStorage.Count--;

installedStorageOnSANBindingSource.DataSource = InstalledStorageOnSAN;

dg\_InstalledStorage.DataSource = InstalledStorageOnSAN;

dg\_InstalledStorage.Refresh();

removed = true;

}

else

{

var result = MessageBox.Show("Выбранный накопитель будет удален из хранилища данных! Продолжить?",

"Удаление накопителя", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

if (result == DialogResult.Yes)

{

InstalledStorageOnSAN.Remove(selectedInstalledStorage);

removed = true;

}

}

if (removed)

{

var sanPartInfo = sANPartsInfoBindingSource.DataSource as SANPartsInfo;

if (sanPartInfo != null)

{

sanPartInfo.StorageSumPrice -= selectedInstalledStorage.Price;

sanPartInfo.UpdateTotalPrice();

BindSANPartsInfo();

}

}

}

catch

{

MessageBox.Show("Произошлаошибкаприудалениинакопителя!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет удаление выбранного установленного накопителя

/// </summary>

private void picBtn\_DeleteStorage\_Click(object sender, EventArgs e)

{

var selectedInstalledStorage = installedStorageOnSANBindingSource.Current as InstalledStorageOnSAN;

if (selectedInstalledStorage == null)

{

MessageBox.Show("Выберитеустановленныйнакопительизсписка!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

var result = MessageBox.Show("Выбранныйнакопительбудетудаленизхранилищаданных! Продолжить?",

"Удалениенакопителя", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

if (result == DialogResult.Yes)

{

var sanPartInfo = sANPartsInfoBindingSource.DataSource as SANPartsInfo;

if (sanPartInfo != null)

{

sanPartInfo.StorageSumPrice -= (selectedInstalledStorage.Price \* selectedInstalledStorage.Count);

sanPartInfo.UpdateTotalPrice();

BindSANPartsInfo();

}

InstalledStorageOnSAN.Remove(selectedInstalledStorage);

}

}

#endregion

#region Avalible Interfaces operations

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет увеличение количества выбранного установленного интерфейса накопителя

/// </summary>

private void pictureBox21\_Click(object sender, EventArgs e)

{

var selectedInterface = avalibleInterfaceBindingSource.Current as AvalibleInterface;

if (selectedInterface != null)

{

selectedInterface.Slot\_Count++;

avalibleInterfaceBindingSource.DataSource = AvalibleInterfaces;

dg\_AvalibleInterfaces.DataSource = avalibleInterfaceBindingSource;

dg\_AvalibleInterfaces.Refresh();

}

else

MessageBox.Show("Выберитеинтерфейсизспискаилидобавьтеновый!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет уменьшение количества выбранного установленного интерфейса накопителя

/// </summary>

private void pictureBox20\_Click(object sender, EventArgs e)

{

var selectedInterface = avalibleInterfaceBindingSource.Current as AvalibleInterface;

if (selectedInterface == null)

{

MessageBox.Show("Выберитеинтерфейсизспискаилидобавьтеновый!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

try

{

var existedStorages = InstalledStorageOnSAN.Where(iss => iss.Interface.Equals(selectedInterface.Name))

.ToList();

if (existedStorages.Count == 0 || (existedStorages.Sum(es => es.Count) < selectedInterface.Slot\_Count))

{

if (selectedInterface.Slot\_Count > 1)

{

selectedInterface.Slot\_Count--;

avalibleInterfaceBindingSource.DataSource = AvalibleInterfaces;

dg\_AvalibleInterfaces.DataSource = avalibleInterfaceBindingSource;

dg\_AvalibleInterfaces.Refresh();

}

else

AvalibleInterfaces.Remove(selectedInterface);

}

else

MessageBox.Show(

"Заняты все слоты выбранного интерфейса! Для удаления интерфейса следует удалить 1 из накопителей!",

"Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

catch

{

MessageBox.Show("Произошла ошибка при удалении интерфейса!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет удаление выбранного установленного интерфейса накопителя

/// </summary>

private void pictureBox19\_Click(object sender, EventArgs e)

{

var selectedInterface = avalibleInterfaceBindingSource.Current as AvalibleInterface;

if (selectedInterface == null)

{

MessageBox.Show("Выберитеинтерфейсизспискаилидобавьтеновый!",

"Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

try

{

var existedStorages = InstalledStorageOnSAN.Where(iss => iss.Interface.Equals(selectedInterface.Name))

.ToList();

if (existedStorages.Count == 0)

AvalibleInterfaces.Remove(selectedInterface);

else

{

var result = MessageBox.Show(

"Вместе с интерфейсом будут удалены накопители, кототрые к нему принадлежат! Продолжить?",

"Предупреждение", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.Yes)

{

var sanPartInfo = sANPartsInfoBindingSource.DataSource as SANPartsInfo;

foreach (var es in existedStorages)

{

sanPartInfo.StorageSumPrice -= (es.Price \* es.Count);

InstalledStorageOnSAN.Remove(es);

}

AvalibleInterfaces.Remove(selectedInterface);

sanPartInfo.UpdateTotalPrice();

BindSANPartsInfo();

}

}

}

catch

{

MessageBox.Show("Произошлаошибкаприудаленииинтерфейса!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

#endregion

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит отмену последнего добавления/редактирования, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditSANForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.No)

{

e.Cancel = true;

return;

}

}

Owner?.Show();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который выполняет сохранение изменений

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (\_formMode == FormMode.None)

return;

var manufacturer = manufacturerBindingSource.Current as Manufacturer;

if (manufacturer == null || comboBox1.SelectedIndex < 0)

{

MessageBox.Show("Выберите производителя для сохранения хранилища данных!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (AvalibleInterfaces.Count == 0)

{

MessageBox.Show("Выберитеинтерфейсынакопителейдлясохраненияхранилищаданных!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (InstalledStorageOnSAN.Count == 0)

{

MessageBox.Show("Установитенакопителидлясохраненияхранилищаданных!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

var sanPartInfo = sANPartsInfoBindingSource.DataSource as SANPartsInfo;

if (sanPartInfo == null || sanPartInfo.TotalPrice == 0)

{

MessageBox.Show("Не удалось получить служебную информацию по текущему хранилищу данных!", "Ошибка",

MessageBoxButtons.OK);

return;

}

if (sanPartInfo.Maintenance <= 0)

{

MessageBox.Show("Введите стоимость обслуживания хранилища данных!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

CurrentSAN.Manufacturer = manufacturer;

CurrentSAN.AnnualMaintenance = sanPartInfo.Maintenance;

CurrentSAN.Price = sanPartInfo.TotalPrice;

CurrentSAN.SAN\_StorageInt.Clear();

foreach (var ai in AvalibleInterfaces)

{

CurrentSAN.SAN\_StorageInt.Add(new SAN\_StorageInt

{

StrorageInterface = \_ctx.GetStorageInterfaceByName(ai.Name),

SlotCount = ai.Slot\_Count

});

}

CurrentSAN.SAN\_Storage.Clear();

foreach (var storageOnSan in InstalledStorageOnSAN)

{

CurrentSAN.SAN\_Storage.Add(new SAN\_Storage

{

Strorage = \_ctx.GetStorageByModelManufacturer(storageOnSan.Model, storageOnSan.Manufacturer),

Count = storageOnSan.Count

});

}

switch (\_formMode)

{

case FormMode.Add:

await \_ctx.AddNewSAN(CurrentSAN);

MessageBox.Show("Новое хранилище данных успешно создано!", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

break;

case FormMode.Edit:

await \_ctx.EditSAN(CurrentSAN);

MessageBox.Show("Изменения в хранилище данных успешно сохранены!", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

break;

}

\_formMode = FormMode.None;

DialogResult = DialogResult.OK;

Close();

}

catch

{

switch (\_formMode)

{

case FormMode.Add:

MessageBox.Show("Не удалось создать новое хранилище данных!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

break;

case FormMode.Edit:

MessageBox.Show("Не удалось сохранить изменения в хранилище данных!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

break;

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет удаление выбранного производителя

/// </summary>

private void pictureBox1\_Click(object sender, EventArgs e)

{

comboBox1.SelectedIndex = -1;

manufacturerBindingSource.SuspendBinding();

if (CurrentSAN != null)

CurrentSAN.ManufacturerId = 0;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных производителей

/// </summary>

private void manufacturerBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (manufacturerBindingSource.IsBindingSuspended)

{

\_afterManufacturerSuspend = true;

manufacturerBindingSource.ResumeBinding();

}

else if (\_afterManufacturerSuspend)

{

\_afterManufacturerSuspend = false;

manufacturerBindingSource.Position = comboBox1.SelectedIndex;

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска производителя

/// </summary>

private void pictureBox17\_Click(object sender, EventArgs e)

{

var findManufacturer = new EditManufacturerForm(true);

if (findManufacturer.ShowDialog() == DialogResult.OK)

{

if (findManufacturer.Edited)

{

BindManufacturer();

}

manufacturerBindingSource\_CurrentChanged(manufacturerBindingSource, EventArgs.Empty);

manufacturerBindingSource.Position =

((BindingList<Manufacturer>)manufacturerBindingSource.DataSource).IndexOf(findManufacturer.CurrentManufacturer);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно редактирования производителей

/// </summary>

private void pictureBox4\_Click(object sender, EventArgs e)

{

var editManufacturer = new EditManufacturerForm();

if (editManufacturer.ShowDialog() == DialogResult.OK)

{

var selectedManufacturer = manufacturerBindingSource.Current as Manufacturer;

BindManufacturer();

if (selectedManufacturer != null)

{

int pos = manufacturerBindingSource.IndexOf(selectedManufacturer);

if (pos > -1)

manufacturerBindingSource.Position = pos;

}

}

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке производителей

/// </summary>

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox1.SelectedIndex > -1)

manufacturerBindingSource\_CurrentChanged(manufacturerBindingSource, EventArgs.Empty);

}

}

}

Модуль «EditServerConfigurationForm.cs»

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Parts;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Servers

{

/// <summary>

/// Класс формы редактирования конфигурации сервера.

/// </summary>

public partial class EditServerConfigurationForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Режим работы формы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Редактируемуя конфигурация сервера.

/// </summary>

public Server CurrentServer { get; set; }

/// <summary>

/// Выбраннаяплатформа

/// </summary>

public Platform CurrentPlatform { get; set; }

/// <summary>

/// Установленнаяпамять.

/// </summary>

public BindingList<InstalledRamOnServer> InstalledRamOnServer { get; set; }

/// <summary>

/// Установленныенакопители.

/// </summary>

public BindingList<InstalledStorageOnServer> InstalledStorageOnServer { get; set; }

/// <summary>

/// Доступные интерфейсы накопителей.

/// </summary>

public BindingList<AvalibleInterface> AvalibleInterfaces { get; set; }

/// <summary>

/// Конструктор формы создания новой конфигурации сервера.

/// </summary>

public EditServerConfigurationForm()

{

InitializeComponent();

UpdateGridsFont(dg\_InstalledRam);

UpdateGridsFont(dg\_InstalledStorage);

SubscribeMenuItems();

comboBox\_Platform.Format += ComboBox\_Platform\_Format;

comboBox\_CPU.Format += ComboBox\_CPU\_Format;

comboBox\_RAM.Format += ComboBox\_RAM\_Format;

comboBox\_Storage.Format += ComboBox\_Storage\_Format;

tsmi\_IncreaseRamCount.Click += pictureBox8\_Click;

tsmi\_DecreaseRamCount.Click += pictureBox9\_Click;

tsmi\_DeleteRam.Click += pictureBox10\_Click;

tsmi\_IncreaseStorage.Click += pictureBox21\_Click;

tsmi\_DecreaseStorage.Click += pictureBox20\_Click;

tsmi\_DeleteStorage.Click += pictureBox19\_Click;

\_ctx = new Context();

CurrentServer = new Server();

BindConfigurationPartsInfo();

BindInstalledRamOnServer();

BindInstalledStorageOnServer();

BindPlatforms();

btn\_Save.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

\_formMode = FormMode.Add;

Text = "Создание новой конфигурации сервера";

}

/// <summary>

/// Конструктор формы редактирования выбранной конфигурации сервера.

/// </summary>

/// <param name="selectedServer">Выбраннаяконфигурациясервера</param>

public EditServerConfigurationForm(Server selectedServer) : this()

{

\_formMode = FormMode.Edit;

Text = "Редактирование конфигурации сервера";

CurrentServer = selectedServer;

PrepareBindingsToEditServer();

}

/// <summary>

/// Инициализация привязок для редактирования конфигурации сервера

/// </summary>

private void PrepareBindingsToEditServer()

{

platformBindingSource.Position =

((BindingList<Platform>)platformBindingSource.DataSource).IndexOf(CurrentServer.Platform);

comboBox\_Platform.SelectedIndex = platformBindingSource.Position;

platformBindingSource\_CurrentChanged(platformBindingSource, EventArgs.Empty);

cPUBindingSource.Position =

((BindingList<CPU>)cPUBindingSource.DataSource).IndexOf(CurrentServer.CPU);

comboBox\_CPU.SelectedIndex = cPUBindingSource.Position;

cPUBindingSource\_CurrentChanged(cPUBindingSource, EventArgs.Empty);

foreach (var s in CurrentServer.Server\_Storage)

{

InstalledStorageOnServer.Add(new InstalledStorageOnServer

{

Manufacturer = s.Strorage.Manufacturer.Name,

Model = s.Strorage.Model,

Interface = s.Strorage.StrorageInterface.Name,

Volume = s.Strorage.Volume,

Price = s.Strorage.Price,

Count = s.Count

});

}

foreach (var r in CurrentServer.Server\_RAM)

{

InstalledRamOnServer.Add(new InstalledRamOnServer

{

Manufacturer = r.RAM.Manufacturer.Name,

Model = r.RAM.Model,

SingleVolume = r.RAM.Volume,

Price = r.RAM.Price,

Count = r.Count

});

}

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.Maintenance = CurrentServer.AnnualMaintenance ?? 0;

cfgPartInfo.StorageSumPrice += InstalledStorageOnServer.Sum(iss => iss.Price \* iss.Count);

cfgPartInfo.RAMSumPrice += InstalledRamOnServer.Sum(irs => irs.Price \* irs.Count);

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

}

#region Форматы comboBox

/// <summary>

/// Обработчик события форматирования отображаемого значения в списке накопителей

/// </summary>

private void ComboBox\_Storage\_Format(object sender, ListControlConvertEventArgs e)

{

var storageToFormat = e.ListItem as Strorage;

if (storageToFormat != null)

e.Value = $"{storageToFormat.Manufacturer.Name} {storageToFormat.Model}";

}

/// <summary>

/// Обработчик события форматирования отображаемого значения в списке оперативной памяти

/// </summary>

private void ComboBox\_RAM\_Format(object sender, ListControlConvertEventArgs e)

{

var ramToFormat = e.ListItem as RAM;

if (ramToFormat != null)

e.Value = $"{ramToFormat.Manufacturer.Name} {ramToFormat.Model}";

}

/// <summary>

/// Обработчик события форматирования отображаемого значения в списке процессоров

/// </summary>

private void ComboBox\_CPU\_Format(object sender, ListControlConvertEventArgs e)

{

var cpuToFormat = e.ListItem as CPU;

if (cpuToFormat != null)

e.Value = $"{cpuToFormat.Manufacturer.Name} {cpuToFormat.Model}";

}

/// <summary>

/// Обработчик события форматирования отображаемого значения в списке платформ

/// </summary>

private void ComboBox\_Platform\_Format(object sender, ListControlConvertEventArgs e)

{

var platformToFormat = e.ListItem as Platform;

if (platformToFormat != null)

e.Value = $"{platformToFormat.Manufacturer.Name} {platformToFormat.Model}";

}

#endregion

#region UI

/// <summary>

/// Обновлениешрифтовтаблиц

/// </summary>

private void UpdateGridsFont(DataGridView dataGrid)

{

Font f = new Font("Segoe UI Semilight", 11F, GraphicsUnit.Pixel);

foreach (DataGridViewColumn column in dataGrid.Columns)

{

column.DefaultCellStyle.Font = f;

column.HeaderCell.Style.Font = f;

}

}

/// <summary>

/// Обработчик события выхода курсора из элемента главного меню,

/// который очищает область в строке состояния.

/// </summary>

private void MenuItem\_MouseLeave(object sender, EventArgs e)

{

tip\_Label.Text = string.Empty;

}

/// <summary>

/// Обработчик события навдения курсора на элемент главного меню,

/// который выводит подсказку элемента в строку состояния.

/// </summary>

private void MenuItem\_MouseEnter(object sender, EventArgs e)

{

tip\_Label.Text = ((ToolStripItem)sender).ToolTipText;

}

/// <summary>

/// Подписка элементов главного меню и панели элементов на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

private void SubscribeMenuItems()

{

SubScribeChildMenuItems(menuBar.Items);

foreach (ToolStripItem toolBarControl in toolBar.Items)

{

toolBarControl.MouseEnter += MenuItem\_MouseEnter;

toolBarControl.MouseLeave += MenuItem\_MouseLeave;

}

}

/// <summary>

/// Подписка дочерних элементов главного меню на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

/// <param name="col">Коллекция элементов главного меню</param>

private void SubScribeChildMenuItems(ToolStripItemCollection col)

{

for (int i = 0; i < col.Count; i++)

{

var item = col[i] as ToolStripMenuItem;

if (item != null)

{

item.MouseEnter += MenuItem\_MouseEnter;

item.MouseLeave += MenuItem\_MouseLeave;

if (item.DropDownItems.Count > 0)

SubScribeChildMenuItems(item.DropDownItems);

}

}

}

#endregion

#region Manage UI controls

/// <summary>

/// Управление элементами управления области платформы

/// </summary>

/// <param name="selMode">Выбранныйрежим</param>

private void ManagePlatformControls(ServerPartSelectionMode selMode)

{

foreach (Control control in groupBox1.Controls)

control.Visible = selMode == ServerPartSelectionMode.OK;

lbl\_NoPlatforms.Visible = selMode == ServerPartSelectionMode.None;

lbl\_NoPlatforms.Dock = selMode == ServerPartSelectionMode.None ? DockStyle.Fill : DockStyle.None;

lbl\_SelectPlatform.Visible = selMode == ServerPartSelectionMode.NotSelected;

lbl\_SelectPlatform.Dock = selMode == ServerPartSelectionMode.NotSelected ? DockStyle.Fill : DockStyle.None;

}

/// <summary>

/// Управлениеэлементамиуправленияобластипроцессоров

/// </summary>

/// <param name="selMode">Выбранныйрежим</param>

private void ManageCpuControls(ServerPartSelectionMode selMode)

{

foreach (Control control in groupBox2.Controls)

control.Visible = selMode == ServerPartSelectionMode.OK;

lbl\_NoCPUs.Visible = selMode == ServerPartSelectionMode.UnSupported;

lbl\_NoCPUs.Dock = selMode == ServerPartSelectionMode.UnSupported ? DockStyle.Fill : DockStyle.None;

lbl\_NoPlatfromToCPU.Visible = selMode == ServerPartSelectionMode.NotSelected;

lbl\_NoPlatfromToCPU.Dock = selMode == ServerPartSelectionMode.NotSelected ? DockStyle.Fill : DockStyle.None;

lbl\_SelectAvalibleCPU.Visible = selMode == ServerPartSelectionMode.CPUNotSelected;

lbl\_SelectAvalibleCPU.Dock = selMode == ServerPartSelectionMode.CPUNotSelected ? DockStyle.Fill : DockStyle.None;

}

/// <summary>

/// Управление элементами управления области оперативной памяти

/// </summary>

/// <param name="selMode">Выбранныйрежим</param>

private void ManageRamControls(ServerPartSelectionMode selMode)

{

foreach (Control control in groupBox3.Controls)

control.Visible = selMode == ServerPartSelectionMode.OK;

lbl\_NoRams.Visible = selMode == ServerPartSelectionMode.UnSupported;

lbl\_NoRams.Dock = selMode == ServerPartSelectionMode.UnSupported ? DockStyle.Fill : DockStyle.None;

lbl\_NoPlatfromToRAM.Visible = selMode == ServerPartSelectionMode.NotSelected;

lbl\_NoPlatfromToRAM.Dock = selMode == ServerPartSelectionMode.NotSelected ? DockStyle.Fill : DockStyle.None;

lbl\_SelectAvalibleRAM.Visible = selMode == ServerPartSelectionMode.RAMNotSelected;

lbl\_SelectAvalibleRAM.Dock = selMode == ServerPartSelectionMode.RAMNotSelected ? DockStyle.Fill : DockStyle.None;

}

/// <summary>

/// Управлениеэлементамиуправленияобластинакопителей

/// </summary>

/// <param name="selMode">Выбранныйрежим</param>

private void ManageStorageControls(ServerPartSelectionMode selMode)

{

foreach (Control control in groupBox5.Controls)

control.Visible = selMode == ServerPartSelectionMode.OK;

lbl\_NoStorages.Visible = selMode == ServerPartSelectionMode.UnSupported;

lbl\_NoStorages.Dock = selMode == ServerPartSelectionMode.UnSupported ? DockStyle.Fill : DockStyle.None;

lbl\_NoPlatfromToStorage.Visible = selMode == ServerPartSelectionMode.NotSelected;

lbl\_NoPlatfromToStorage.Dock = selMode == ServerPartSelectionMode.NotSelected ? DockStyle.Fill : DockStyle.None;

lbl\_SelectAvalibleStorages.Visible = selMode == ServerPartSelectionMode.StorageNotSelected;

lbl\_SelectAvalibleStorages.Dock = selMode == ServerPartSelectionMode.StorageNotSelected ? DockStyle.Fill : DockStyle.None;

}

#endregion

#region Bindings

/// <summary>

/// Инициализацияпривязкиплатформ

/// </summary>

private void BindPlatforms()

{

platformBindingSource.DataSource = \_ctx.GetPlatformsList();

comboBox\_Platform.DataSource = platformBindingSource;

}

/// <summary>

/// Инициализация привязки списка установленной памяти

/// </summary>

private void BindInstalledRamOnServer()

{

InstalledRamOnServer = new BindingList<InstalledRamOnServer>();

installedRamOnServerBindingSource.DataSource = InstalledRamOnServer;

dg\_InstalledRam.DataSource = installedRamOnServerBindingSource;

}

/// <summary>

/// Инициализация привязки списка установленных накопителей

/// </summary>

private void BindInstalledStorageOnServer()

{

InstalledStorageOnServer = new BindingList<InstalledStorageOnServer>();

installedStorageOnServerBindingSource.DataSource = InstalledStorageOnServer;

dg\_InstalledStorage.DataSource = installedStorageOnServerBindingSource;

}

/// <summary>

/// Получение списка доступных процессоров выбранной платформы

/// </summary>

private void BindCPUs()

{

if (CurrentPlatform != null)

{

cPUBindingSource.DataSource = \_ctx.GetCpusOfSocket(CurrentPlatform.CpuSocket.Name);

if (cPUBindingSource.Count == 0)

ManageCpuControls(ServerPartSelectionMode.UnSupported);

}

else

{

cPUBindingSource.DataSource = null;

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.OneCPUPrice = 0;

cfgPartInfo.CPUSumPrice = 0;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

ManageCpuControls(ServerPartSelectionMode.NotSelected);

}

comboBox\_CPU.DataSource = cPUBindingSource;

}

/// <summary>

/// Получение списка доступной оперативной памяти выбранной платформы

/// </summary>

private void BindRAMs()

{

if (CurrentPlatform != null)

{

var ramsOfRamType = \_ctx.GetRamsOfRamType(CurrentPlatform.RamType.Name);

rAMBindingSource.DataSource = ramsOfRamType;

CheckInstalledRam(ramsOfRamType);

if (ramsOfRamType.Count == 0)

ManageRamControls(ServerPartSelectionMode.UnSupported);

}

else

{

rAMBindingSource.DataSource = null;

InstalledRamOnServer.Clear();

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.SelectedRamPrice = 0;

cfgPartInfo.RAMSumPrice = 0;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

ManageRamControls(ServerPartSelectionMode.NotSelected);

}

comboBox\_RAM.DataSource = rAMBindingSource;

}

/// <summary>

/// Получение списка доступных накопителей выбранной платформы

/// </summary>

private void BindStorages()

{

if (CurrentPlatform != null)

{

var storagesByAvalibleInterfaces = \_ctx.GetStoragesByAvaliblePlatformInterfaces(CurrentPlatform.Platform\_StorageInt);

strorageBindingSource.DataSource = storagesByAvalibleInterfaces;

CheckInstalledStorage(storagesByAvalibleInterfaces);

if (storagesByAvalibleInterfaces.Count == 0)

ManageStorageControls(ServerPartSelectionMode.UnSupported);

}

else

{

strorageBindingSource.DataSource = null;

InstalledStorageOnServer.Clear();

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.SelectedStoragePrice = 0;

cfgPartInfo.StorageSumPrice = 0;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

ManageStorageControls(ServerPartSelectionMode.NotSelected);

}

comboBox\_Storage.DataSource = strorageBindingSource;

}

/// <summary>

/// Получение списка доступных интерфейсов накопителей выбранной платформы

/// </summary>

private void BindAvalibleInterfaces()

{

if (CurrentPlatform != null)

{

AvalibleInterfaces = \_ctx.GetAvalibleInterfacesOfPlarformBS(CurrentPlatform.Platform\_StorageInt);

avalibleInterfaceBindingSource.DataSource = AvalibleInterfaces;

}

else

avalibleInterfaceBindingSource.DataSource = null;

dg\_AvalibleInterfaces.DataSource = avalibleInterfaceBindingSource;

}

/// <summary>

/// Привязка объекта, который отображает цены компонентов

/// </summary>

private void BindConfigurationPartsInfo()

{

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

configurationPartsInfoBindingSource.DataSource = cfgPartInfo ?? new ConfigurationPartsInfo();

configurationPartsInfoBindingSource.ResetBindings(true);

}

/// <summary>

/// Получениеподробногоописанияплатформы

/// </summary>

private void BindPlatfromInfo()

{

if (CurrentPlatform != null)

platformInfoBindingSource.DataSource = \_ctx.GetPlatformInfoByPlatform(CurrentPlatform);

}

#endregion

/// <summary>

/// Проверка установленной памяти в конфигурации с доступным списком ОЗУ

/// </summary>

/// <param name="ramList">СписокдоступныхОЗУ</param>

private void CheckInstalledRam(BindingList<RAM> ramList)

{

var itemsToRemove = new Stack<InstalledRamOnServer>();

foreach (var ramOnServer in InstalledRamOnServer)

{

var existed = ramList.SingleOrDefault(rl =>

rl.Manufacturer.Name.Equals(ramOnServer.Manufacturer) && rl.Model.Equals(ramOnServer.Model));

if (existed == null)

itemsToRemove.Push(ramOnServer);

}

while (itemsToRemove.Count > 0)

{

var itemToRemove = itemsToRemove.Pop();

InstalledRamOnServer.Remove(itemToRemove);

}

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.RAMSumPrice = 0;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

}

/// <summary>

/// Проверкаустановленныхнакопителейсоспискомдоступных

/// </summary>

/// <param name="strorageList">Списокдоступныхнакопителей</param>

private void CheckInstalledStorage(BindingList<Strorage> strorageList)

{

var itemsToRemove = new Stack<InstalledStorageOnServer>();

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

foreach (var storageOnServer in InstalledStorageOnServer)

{

var existed = strorageList.SingleOrDefault(rl =>

rl.Manufacturer.Name.Equals(storageOnServer.Manufacturer) && rl.Model.Equals(storageOnServer.Model));

if (existed == null)

itemsToRemove.Push(storageOnServer);

}

while (itemsToRemove.Count > 0)

{

var itemToRemove = itemsToRemove.Pop();

cfgPartInfo.StorageSumPrice -= (itemToRemove.Price \* itemToRemove.Count);

InstalledStorageOnServer.Remove(itemToRemove);

}

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных платформ

/// </summary>

private void platformBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var bs = sender as BindingSource;

CurrentPlatform = bs.Current as Platform;

if (comboBox\_Platform.SelectedIndex == -1)

{

bs.SuspendBinding();

CurrentPlatform = null;

}

else

bs.ResumeBinding();

BindAvalibleInterfaces();

if (CurrentPlatform != null)

{

try

{

CurrentPlatform = \_ctx.GetPlatformInclude(CurrentPlatform);

ManagePlatformControls(ServerPartSelectionMode.OK);

CurrentServer.Platform = CurrentPlatform;

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.PlatfromPrice = CurrentPlatform.Price;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

BindPlatfromInfo();

BindCPUs();

BindRAMs();

BindStorages();

}

catch

{

MessageBox.Show("Произошла ошибка при получении доступных компонентов для выбранной платформы!",

"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else

{

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.PlatfromPrice = 0;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

if (bs.Count < 0)

ManagePlatformControls(ServerPartSelectionMode.None);

else

ManagePlatformControls(ServerPartSelectionMode.NotSelected);

BindCPUs();

BindRAMs();

BindStorages();

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных процессоров

/// </summary>

private void cPUBindingSource\_CurrentChanged(object sender, EventArgs e)

{

bool unSelected = false;

var bs = sender as BindingSource;

var selCPU = bs.Current as CPU;

if (comboBox\_CPU.SelectedIndex == -1)

{

bs.SuspendBinding();

selCPU = null;

unSelected = true;

}

else

bs.ResumeBinding();

if (selCPU != null)

{

ManageCpuControls(ServerPartSelectionMode.OK);

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

var plarformInfo = platformInfoBindingSource.DataSource as PlatformInfo;

if (cfgPartInfo != null && plarformInfo != null)

{

cfgPartInfo.OneCPUPrice = selCPU.Price;

cfgPartInfo.CPUSumPrice = plarformInfo.CPU\_Count \* selCPU.Price;

}

else if (cfgPartInfo != null)

{

cfgPartInfo.OneCPUPrice = 0;

cfgPartInfo.CPUSumPrice = 0;

}

if (cfgPartInfo != null)

{

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

}

else

{

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.OneCPUPrice = 0;

cfgPartInfo.CPUSumPrice = 0;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

if (!unSelected)

ManageCpuControls(ServerPartSelectionMode.UnSupported);

else

ManageCpuControls(ServerPartSelectionMode.CPUNotSelected);

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных оперативной памяти

/// </summary>

private void rAMBindingSource\_CurrentChanged(object sender, EventArgs e)

{

bool unSelected = false;

var bs = sender as BindingSource;

var selRAM = bs.Current as RAM;

if (comboBox\_RAM.SelectedIndex == -1)

{

bs.SuspendBinding();

selRAM = null;

unSelected = true;

}

else

bs.ResumeBinding();

if (selRAM != null)

{

ManageRamControls(ServerPartSelectionMode.OK);

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.SelectedRamPrice = selRAM.Price;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

}

else

{

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.SelectedRamPrice = 0;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

if (!unSelected)

ManageRamControls(ServerPartSelectionMode.UnSupported);

else

ManageRamControls(ServerPartSelectionMode.RAMNotSelected);

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных накопителей

/// </summary>

private void strorageBindingSource\_CurrentChanged(object sender, EventArgs e)

{

bool unSelected = false;

var bs = sender as BindingSource;

var selStrorage = bs.Current as Strorage;

if (comboBox\_Storage.SelectedIndex == -1)

{

bs.SuspendBinding();

selStrorage = null;

unSelected = true;

}

else

bs.ResumeBinding();

if (selStrorage != null)

{

ManageStorageControls(ServerPartSelectionMode.OK);

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.SelectedStoragePrice = selStrorage.Price;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

}

else

{

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.SelectedStoragePrice = 0;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

if (!unSelected)

ManageStorageControls(ServerPartSelectionMode.UnSupported);

else

ManageStorageControls(ServerPartSelectionMode.RAMNotSelected);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление текущей платформы из конфигурации

/// </summary>

private void picBtn\_DeletePlatform\_Click(object sender, EventArgs e)

{

if (CurrentPlatform == null)

{

MessageBox.Show("Платформаневыбрана!", "Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

var result = MessageBox.Show("Выбранная платформа будет удалена из конфигурации! Продолжить?",

"Удаление платформы", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

if (result == DialogResult.Yes)

{

comboBox\_Platform.SelectedIndex = -1;

platformBindingSource\_CurrentChanged(platformBindingSource, EventArgs.Empty);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска платформы

/// </summary>

private void picBtn\_SearchPlatform\_Click(object sender, EventArgs e)

{

var selPlatformForm = new SelectPlatformForm();

if (selPlatformForm.ShowDialog() == DialogResult.OK)

{

try

{

BindPlatforms();

Platform selPl = \_ctx.GetPlatformByPlatformInfo(selPlatformForm.SelectedPlatformInfo);

platformBindingSource.Position =

((BindingList<Platform>)platformBindingSource.DataSource).IndexOf(selPl);

comboBox\_Platform.SelectedIndex = platformBindingSource.Position;

platformBindingSource\_CurrentChanged(platformBindingSource, EventArgs.Empty);

}

catch

{

MessageBox.Show("Ошибкапривыбореплатформы!",

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление текущего процессора из конфигурации

/// </summary>

private void pictureBox3\_Click(object sender, EventArgs e)

{

var currentCPU = cPUBindingSource.Current as CPU;

if (currentCPU == null)

{

MessageBox.Show("Процессорневыбран!", "Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

var result = MessageBox.Show("Выбранный процессор будет удален из конфигурации! Продолжить?",

"Удаление платформы", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

if (result == DialogResult.Yes)

{

comboBox\_CPU.SelectedIndex = -1;

cPUBindingSource\_CurrentChanged(cPUBindingSource, EventArgs.Empty);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление установленной оперативной памяти

/// </summary>

private void pictureBox4\_Click(object sender, EventArgs e)

{

var selectedRam = rAMBindingSource.Current as RAM;

if (selectedRam == null)

{

MessageBox.Show("Оперативнаяпамятьневыбрана!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

var ramToDel = InstalledRamOnServer.SingleOrDefault(irs =>

irs.Model.Equals(selectedRam.Model) && irs.Manufacturer.Equals(selectedRam.Manufacturer.Name));

if (ramToDel == null)

{

MessageBox.Show("Выбраннаяоперативнаяпамятьненайденавспискеустановленных!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

var result = MessageBox.Show("Выбранная оперативная память будет удалена из конфигурации! Продолжить?",

"Удаление оперативной памяти", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

if (result == DialogResult.Yes)

{

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.RAMSumPrice -= (ramToDel.Price \* ramToDel.Count);

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

InstalledRamOnServer.Remove(ramToDel);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление установленного накопителя

/// </summary>

private void pictureBox14\_Click(object sender, EventArgs e)

{

var selectedStorage = strorageBindingSource.Current as Strorage;

if (selectedStorage == null)

{

MessageBox.Show("Накопительневыбран!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

var storageToDel = InstalledStorageOnServer.SingleOrDefault(irs =>

irs.Model.Equals(selectedStorage.Model) && irs.Manufacturer.Equals(selectedStorage.Manufacturer.Name));

if (storageToDel == null)

{

MessageBox.Show("Выбранныйнакопительненайденвспискеустановленных!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

var result = MessageBox.Show("Выбранный накопитель будет удален из конфигурации! Продолжить?",

"Удаление накопителя", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

if (result == DialogResult.Yes)

{

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.StorageSumPrice -= (storageToDel.Price \* storageToDel.Count);

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

InstalledStorageOnServer.Remove(storageToDel);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска процессора

/// </summary>

private void pictureBox1\_Click(object sender, EventArgs e)

{

var selectCpuForm = new SelectCPUForm(CurrentPlatform.CpuSocket);

if (selectCpuForm.ShowDialog() == DialogResult.OK)

{

try

{

BindCPUs();

cPUBindingSource.Position =

((BindingList<CPU>)cPUBindingSource.DataSource).IndexOf(selectCpuForm.SelectedCPU);

comboBox\_CPU.SelectedIndex = cPUBindingSource.Position;

cPUBindingSource\_CurrentChanged(cPUBindingSource, EventArgs.Empty);

}

catch

{

MessageBox.Show("Ошибка при выборе процессора!",

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска оперативной памяти

/// </summary>

private void pictureBox5\_Click(object sender, EventArgs e)

{

var selectRamForm = new SelectRAMForm(CurrentPlatform.RamType);

if (selectRamForm.ShowDialog() == DialogResult.OK)

{

try

{

BindRAMs();

rAMBindingSource.Position =

((BindingList<RAM>)rAMBindingSource.DataSource).IndexOf(selectRamForm.SelectedRAM);

comboBox\_RAM.SelectedIndex = rAMBindingSource.Position;

rAMBindingSource\_CurrentChanged(rAMBindingSource, EventArgs.Empty);

}

catch

{

MessageBox.Show("Ошибка при выборе оперативной памяти!",

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска накопителя

/// </summary>

private void pictureBox15\_Click(object sender, EventArgs e)

{

var selectStorageForm = new SelectStorageForm();

if (selectStorageForm.ShowDialog() == DialogResult.OK)

{

try

{

BindStorages();

strorageBindingSource.Position =

((BindingList<Strorage>)strorageBindingSource.DataSource).IndexOf(selectStorageForm.SelectedStorage);

comboBox\_Storage.SelectedIndex = strorageBindingSource.Position;

strorageBindingSource\_CurrentChanged(strorageBindingSource, EventArgs.Empty);

}

catch

{

MessageBox.Show("Ошибка при выборе накопителя!",

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке процессоров

/// </summary>

private void comboBox\_CPU\_SelectedIndexChanged(object sender, EventArgs e)

{

if (((ComboBox)sender).SelectedIndex > -1)

ManageCpuControls(ServerPartSelectionMode.OK);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет добавление выбранной памяти к установленной

/// </summary>

private void pictureBox7\_Click(object sender, EventArgs e)

{

var selectedRAM = rAMBindingSource.Current as RAM;

if (selectedRAM == null)

{

MessageBox.Show("Выберитеоперативнуюпамятьизсписка!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

int avalibleSlotCount = CurrentPlatform.RamSocketCount - InstalledRamOnServer.Sum(ir => ir.Count);

if (avalibleSlotCount == 0)

{

MessageBox.Show("Всеразъемыподоперативнуюпамятьзаняты!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

int avalibleRamVolume =

CurrentPlatform.RamVolumeMax - InstalledRamOnServer.Sum(ir => ir.SingleVolume \* ir.Count);

if (avalibleRamVolume == 0)

{

MessageBox.Show("Занятмаксимальныйобъемоперативнойпамяти!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

try

{

var existedItem = InstalledRamOnServer.SingleOrDefault(ir =>

ir.Manufacturer.Equals(selectedRAM.Manufacturer.Name) && ir.Model.Equals(selectedRAM.Model));

if (existedItem != null)

{

existedItem.Count++;

installedRamOnServerBindingSource.DataSource = InstalledRamOnServer;

dg\_InstalledRam.DataSource = installedRamOnServerBindingSource;

dg\_InstalledRam.Refresh();

}

else

InstalledRamOnServer.Add(new InstalledRamOnServer

{

Model = selectedRAM.Model,

Manufacturer = selectedRAM.Manufacturer.Name,

SingleVolume = selectedRAM.Volume,

Count = 1,

Price = selectedRAM.Price

});

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.RAMSumPrice += selectedRAM.Price;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

}

catch

{

MessageBox.Show("Произошлаошибкапридобавленииоперативнойпамяти!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет добавление выбранного накопителя к установленным

/// </summary>

private void pictureBox13\_Click(object sender, EventArgs e)

{

var selectedStorage = strorageBindingSource.Current as Strorage;

if (selectedStorage == null)

{

MessageBox.Show("Выберитенакопительизсписка!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

var avalibleInterface =

AvalibleInterfaces.Single(av => av.Name.Equals(selectedStorage.StrorageInterface.Name));

var installedStoragesOfInterface =

InstalledStorageOnServer.Where(iss => iss.Interface.Equals(avalibleInterface.Name)).ToList();

if (installedStoragesOfInterface.Count > 0)

{

int avalibleSlotCount =

avalibleInterface.Slot\_Count - installedStoragesOfInterface.Sum(isi => isi.Count);

if (avalibleSlotCount == 0)

{

MessageBox.Show($"Всеразъемыподинтерфейс {avalibleInterface.Name} заняты!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

}

try

{

var existedItem = InstalledStorageOnServer.SingleOrDefault(iss =>

iss.Manufacturer.Equals(selectedStorage.Manufacturer.Name) && iss.Model.Equals(selectedStorage.Model));

if (existedItem != null)

{

existedItem.Count++;

installedStorageOnServerBindingSource.DataSource = InstalledStorageOnServer;

dg\_InstalledStorage.DataSource = installedStorageOnServerBindingSource;

dg\_InstalledStorage.Refresh();

}

else

InstalledStorageOnServer.Add(new InstalledStorageOnServer

{

Model = selectedStorage.Model,

Manufacturer = selectedStorage.Manufacturer.Name,

Volume = selectedStorage.Volume,

Interface = selectedStorage.StrorageInterface.Name,

Count = 1,

Price = selectedStorage.Price

});

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.StorageSumPrice += selectedStorage.Price;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

}

catch

{

MessageBox.Show("Произошлаошибкапридобавлениинакопителя!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинатекстовуюметку,

/// который открывает диалоговое окно добавления нового накопителя, если он не выбран

/// </summary>

private void lbl\_NoStorages\_Click(object sender, EventArgs e)

{

var editStorageForm = new EditStorageForm();

if (editStorageForm.ShowDialog() == DialogResult.OK)

{

try

{

BindStorages();

strorageBindingSource.Position =

((BindingList<Strorage>)strorageBindingSource.DataSource).IndexOf(editStorageForm.CurrentStrorage);

comboBox\_Storage.SelectedIndex = strorageBindingSource.Position;

strorageBindingSource\_CurrentChanged(strorageBindingSource, EventArgs.Empty);

}

catch

{

MessageBox.Show("Ошибка при выборе накопителя!",

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на текстовую метку,

/// который открывает диалоговое окно добавления новой оперативной памяти, если она не выбрана

/// </summary>

private void lbl\_NoRams\_Click(object sender, EventArgs e)

{

var editRamForm = new EditRAMForm();

if (editRamForm.ShowDialog() == DialogResult.OK)

{

try

{

BindRAMs();

rAMBindingSource.Position =

((BindingList<RAM>)rAMBindingSource.DataSource).IndexOf(editRamForm.CurrentRAM);

comboBox\_RAM.SelectedIndex = rAMBindingSource.Position;

rAMBindingSource\_CurrentChanged(rAMBindingSource, EventArgs.Empty);

}

catch

{

MessageBox.Show("Ошибка при выборе оперативной памяти!",

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на текстовую метку,

/// который открывает диалоговое окно добавления нового процессора, если он не выбран

/// </summary>

private void lbl\_NoCPUs\_Click(object sender, EventArgs e)

{

var editCpuForm = new EditCPUForm();

if (editCpuForm.ShowDialog() == DialogResult.OK)

{

try

{

BindCPUs();

cPUBindingSource.Position =

((BindingList<CPU>)cPUBindingSource.DataSource).IndexOf(editCpuForm.CurrentCPU);

comboBox\_CPU.SelectedIndex = cPUBindingSource.Position;

cPUBindingSource\_CurrentChanged(cPUBindingSource, EventArgs.Empty);

}

catch

{

MessageBox.Show("Ошибка при выборе процессора!",

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

#region Installed RAM operations

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет увеличение количества выбранной установленной оперативной памяти

/// </summary>

private void pictureBox8\_Click(object sender, EventArgs e)

{

var selectedInstalledRAM = installedRamOnServerBindingSource.Current as InstalledRamOnServer;

if (selectedInstalledRAM == null)

{

MessageBox.Show("Выберитеустановленнуюоперативнуюпамятьизсписка!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

int avalibleSlotCount = CurrentPlatform.RamSocketCount - InstalledRamOnServer.Sum(ir => ir.Count);

if (avalibleSlotCount == 0)

{

MessageBox.Show("Всеразъемыподоперативнуюпамятьзаняты!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

int avalibleRamVolume =

CurrentPlatform.RamVolumeMax - InstalledRamOnServer.Sum(ir => ir.SingleVolume \* ir.Count);

if (avalibleRamVolume == 0)

{

MessageBox.Show("Занятмаксимальныйобъемоперативнойпамяти!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

try

{

selectedInstalledRAM.Count++;

installedRamOnServerBindingSource.DataSource = InstalledRamOnServer;

dg\_InstalledRam.DataSource = installedRamOnServerBindingSource;

dg\_InstalledRam.Refresh();

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.RAMSumPrice += selectedInstalledRAM.Price;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

}

catch

{

MessageBox.Show("Произошлаошибкапридобавленииоперативнойпамяти!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет уменьшение количества выбранной установленной оперативной памяти

/// </summary>

private void pictureBox9\_Click(object sender, EventArgs e)

{

var selectedInstalledRAM = installedRamOnServerBindingSource.Current as InstalledRamOnServer;

if (selectedInstalledRAM == null)

{

MessageBox.Show("Выберитеустановленнуюоперативнуюпамятьизсписка!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

try

{

bool removed = false;

if (selectedInstalledRAM.Count > 1)

{

selectedInstalledRAM.Count--;

installedRamOnServerBindingSource.DataSource = InstalledRamOnServer;

dg\_InstalledRam.DataSource = installedRamOnServerBindingSource;

dg\_InstalledRam.Refresh();

removed = true;

}

else

{

var result = MessageBox.Show("Выбранная оперативная память будет удалена из конфигурации! Продолжить?",

"Удаление оперативной памяти", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

if (result == DialogResult.Yes)

{

InstalledRamOnServer.Remove(selectedInstalledRAM);

removed = true;

}

}

if (removed)

{

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.RAMSumPrice -= selectedInstalledRAM.Price;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

}

}

catch

{

MessageBox.Show("Произошлаошибкаприудаленииоперативнойпамяти!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет удаление выбранной установленной оперативной памяти

/// </summary>

private void pictureBox10\_Click(object sender, EventArgs e)

{

var selectedInstalledRAM = installedRamOnServerBindingSource.Current as InstalledRamOnServer;

if (selectedInstalledRAM == null)

{

MessageBox.Show("Выберитеустановленнуюоперативнуюпамятьизсписка!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

var result = MessageBox.Show("Выбраннаяоперативнаяпамятьбудетудаленаизконфигурации! Продолжить?",

"Удалениеоперативнойпамяти", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

if (result == DialogResult.Yes)

{

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.RAMSumPrice -= (selectedInstalledRAM.Price \* selectedInstalledRAM.Count);

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

InstalledRamOnServer.Remove(selectedInstalledRAM);

}

}

#endregion

#region Installed Storage operations

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет увеличение количества выбранного установленного накопителя

/// </summary>

private void pictureBox21\_Click(object sender, EventArgs e)

{

var selectedInstalledStorage = installedStorageOnServerBindingSource.Current as InstalledStorageOnServer;

if (selectedInstalledStorage == null)

{

MessageBox.Show("Выберитеустановленныйнакопительизсписка!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

var avalibleInterface =

AvalibleInterfaces.Single(av => av.Name.Equals(selectedInstalledStorage.Interface));

var installedStoragesOfInterface =

InstalledStorageOnServer.Where(iss => iss.Interface.Equals(avalibleInterface.Name)).ToList();

if (installedStoragesOfInterface.Count > 0)

{

int avalibleSlotCount =

avalibleInterface.Slot\_Count - installedStoragesOfInterface.Sum(isi => isi.Count);

if (avalibleSlotCount == 0)

{

MessageBox.Show($"Всеразъемыподинтерфейс {avalibleInterface.Name} заняты!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

}

try

{

selectedInstalledStorage.Count++;

installedStorageOnServerBindingSource.DataSource = InstalledStorageOnServer;

dg\_InstalledStorage.DataSource = InstalledStorageOnServer;

dg\_InstalledStorage.Refresh();

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.StorageSumPrice += selectedInstalledStorage.Price;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

}

catch

{

MessageBox.Show("Произошлаошибкапридобавлениинакопителя!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет уменьшение количества выбранного установленного накопителя

/// </summary>

private void pictureBox20\_Click(object sender, EventArgs e)

{

var selectedInstalledStorage = installedStorageOnServerBindingSource.Current as InstalledStorageOnServer;

if (selectedInstalledStorage == null)

{

MessageBox.Show("Выберитеустановленныйнакопительизсписка!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

try

{

bool removed = false;

if (selectedInstalledStorage.Count > 1)

{

selectedInstalledStorage.Count--;

installedStorageOnServerBindingSource.DataSource = InstalledStorageOnServer;

dg\_InstalledStorage.DataSource = InstalledStorageOnServer;

dg\_InstalledStorage.Refresh();

removed = true;

}

else

{

var result = MessageBox.Show("Выбранный накопитель будет удален из конфигурации! Продолжить?",

"Удаление накопителя", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

if (result == DialogResult.Yes)

{

InstalledStorageOnServer.Remove(selectedInstalledStorage);

removed = true;

}

}

if (removed)

{

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.StorageSumPrice -= selectedInstalledStorage.Price;

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

}

}

catch

{

MessageBox.Show("Произошлаошибкаприудалениинакопителя!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет удаление выбранного установленного накопителя

/// </summary>

private void pictureBox19\_Click(object sender, EventArgs e)

{

var selectedInstalledStorage = installedStorageOnServerBindingSource.Current as InstalledStorageOnServer;

if (selectedInstalledStorage == null)

{

MessageBox.Show("Выберитеустановленныйнакопительизсписка!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

var result = MessageBox.Show("Выбранныйнакопительбудетудаленизконфигурации! Продолжить?",

"Удалениенакопителя", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);

if (result == DialogResult.Yes)

{

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo != null)

{

cfgPartInfo.StorageSumPrice -= (selectedInstalledStorage.Price \* selectedInstalledStorage.Count);

cfgPartInfo.UpdateTotalPrice();

BindConfigurationPartsInfo();

}

InstalledStorageOnServer.Remove(selectedInstalledStorage);

}

}

#endregion

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену последнего добавления/редактирования, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditServerConfigurationForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.No)

{

e.Cancel = true;

return;

}

}

Owner?.Show();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который выполняет сохранение изменений

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (\_formMode == FormMode.None)

return;

if (CurrentPlatform == null)

{

MessageBox.Show("Выберитеплатформудлясохраненияконфигурациисервера!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

var selectedCPU = cPUBindingSource.Current as CPU;

if (selectedCPU == null)

{

MessageBox.Show("Выберите процессор для сохранения конфигурации сервера!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (InstalledRamOnServer.Count == 0)

{

MessageBox.Show("Установитеоперативнуюпамятьдлясохраненияконфигурациисервера!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (InstalledRamOnServer.Count == 0)

{

MessageBox.Show("Установитеоперативнуюпамятьдлясохраненияконфигурациисервера!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

var cfgPartInfo = configurationPartsInfoBindingSource.DataSource as ConfigurationPartsInfo;

if (cfgPartInfo == null || cfgPartInfo.TotalPrice == 0)

{

MessageBox.Show("Не удалось получить служебную информацию по текущей конфигурации сервера!", "Ошибка",

MessageBoxButtons.OK);

return;

}

if (cfgPartInfo.Maintenance <= 0)

{

MessageBox.Show("Введите стоимость обслуживания сервера!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

CurrentServer.Platform = CurrentPlatform;

CurrentServer.CPU = selectedCPU;

CurrentServer.AnnualMaintenance = cfgPartInfo.Maintenance;

CurrentServer.Price = cfgPartInfo.TotalPrice;

CurrentServer.Server\_RAM.Clear();

foreach (var ramOnServer in InstalledRamOnServer)

{

CurrentServer.Server\_RAM.Add(new Server\_RAM

{

RAM = await \_ctx.GetRAMByModelManufacturer(ramOnServer.Model, ramOnServer.Manufacturer),

Count = ramOnServer.Count

});

}

CurrentServer.Server\_Storage.Clear();

foreach (var storageOnServer in InstalledStorageOnServer)

{

CurrentServer.Server\_Storage.Add(new Server\_Storage

{

Strorage = \_ctx.GetStorageByModelManufacturer(storageOnServer.Model, storageOnServer.Manufacturer),

Count = storageOnServer.Count

});

}

switch (\_formMode)

{

case FormMode.Add:

await \_ctx.AddNewServerConfiguration(CurrentServer);

MessageBox.Show("Новая конфигурация сервера успешно создана!", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

break;

case FormMode.Edit:

await \_ctx.EditServerConfiguration(CurrentServer);

MessageBox.Show("Изменения в конфигурации сервера успешно сохранены!", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

break;

}

\_formMode = FormMode.None;

DialogResult = DialogResult.OK;

Close();

}

catch

{

switch (\_formMode)

{

case FormMode.Add:

MessageBox.Show("Не удалось создать новую конфигурацию сервера!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

break;

case FormMode.Edit:

MessageBox.Show("Не удалось сохранить изменения в конфигурации сервера!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

break;

}

}

}

}

}

Модуль «EditInfrastructureFrom.cs»

using System;

using System.ComponentModel;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Common;

using MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.SANs;

using MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure.Servers;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure

{

/// <summary>

/// Класс формы редактирования вычислительной инфраструктуры.

/// </summary>

public partial class EditInfrastructureFrom : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Список конфигураций серверов

/// </summary>

public BindingList<ServerInfo> ServersInfoMain { get; set; }

/// <summary>

/// Отобранный список конфигураций серверов

/// </summary>

public BindingList<ServerInfo> ServersInfoToShow { get; set; }

/// <summary>

/// Список конфигураций хранилищ данных

/// </summary>

public BindingList<SAN\_Info> SansInfoMain { get; set; }

/// <summary>

/// Отобранный список конфигураций хранилищ данных

/// </summary>

public BindingList<SAN\_Info> SansInfoToShow { get; set; }

/// <summary>

/// Флаг об выводе отобранного списка конфигураций серверов

/// </summary>

private bool \_serverSelected;

/// <summary>

/// Флаг об выводе отобранного списка хранилищ данных

/// </summary>

private bool \_sansSelected;

/// <summary>

/// Конструктор формы редактирования вычислительной инфраструктуры.

/// </summary>

public EditInfrastructureFrom()

{

InitializeComponent();

SubscribeMenuItems();

lbl\_NoServerConfigs.Click += созданиеНовойToolStripMenuItem\_Click;

lbl\_NoSANs.Click += созданиеНовогоToolStripMenuItem\_Click;

создатьНовыйToolStripMenuItem.Click += созданиеНовойToolStripMenuItem\_Click;

редактироватьToolStripMenuItem.Click += редактированиеВыбраннойToolStripMenuItem\_Click;

удалитьToolStripMenuItem.Click += удалениеВыбраннойToolStripMenuItem\_Click;

toolStripMenuItem1.Click += созданиеНовогоToolStripMenuItem\_Click;

toolStripMenuItem3.Click += редактированиеВыбранногоToolStripMenuItem\_Click;

toolStripMenuItem4.Click += удалениеВыбранногоToolStripMenuItem\_Click;

\_ctx = new Context();

timer.Enabled = true;

}

/// <summary>

/// Обновление привязок данных.

/// </summary>

public async void UpdateBindings()

{

try

{

sanStateBS.DataSource = await \_ctx.GetSanEquipmentState();

serverStateBS.DataSource = await \_ctx.GetServersEquipmentState();

}

catch

{

sanStateBS.DataSource = new EquipmentState();

serverStateBS.DataSource = new EquipmentState();

}

try

{

ServersInfoMain = \_ctx.GetServerInfo();

SansInfoMain = \_ctx.GetSanInfo();

serverInfoBindingSource.DataSource = ServersInfoMain;

sANInfoBindingSource.DataSource = SansInfoMain;

dataGridView1.DataSource = serverInfoBindingSource;

dataGridView2.DataSource = sANInfoBindingSource;

\_serverSelected = false;

\_sansSelected = false;

}

catch

{

MessageBox.Show("Ошибка при получении данных!");

}

finally

{

UpdateLabels();

UpdateStatBar();

}

}

/// <summary>

/// Скрытие таблицы с кратким описанием конфигураций серверов

/// </summary>

private void HideServerGrid()

{

dataGridView1.Visible = false;

dataGridView1.Dock = DockStyle.None;

lbl\_NoServerConfigs.Visible = true;

lbl\_NoServerConfigs.Dock = DockStyle.Fill;

}

/// <summary>

/// Скрытие таблицы с кратким описанием конфигураций систем хранения данных

/// </summary>

private void HideSanGrid()

{

dataGridView2.Visible = false;

dataGridView2.Dock = DockStyle.None;

lbl\_NoSANs.Visible = true;

lbl\_NoSANs.Dock = DockStyle.Fill;

}

/// <summary>

/// Обновление строки состояния

/// </summary>

private void UpdateStatBar()

{

switch (tabControl1.SelectedIndex)

{

case 0:

lbl\_RecordsTotal.Text = $"Общееколичествоконфигурацийсерверов: {ServersInfoMain?.Count ?? 0}";

lbl\_RecordsSelected.Text = $"Отобраноконфигураций: {dataGridView1.Rows.Count}";

break;

case 1:

lbl\_RecordsTotal.Text = $"Общееколичествоконфигурацийхранилищданных: {SansInfoMain?.Count ?? 0}";

lbl\_RecordsSelected.Text = $"Отобраноконфигураций: {dataGridView2.Rows.Count}";

break;

}

}

/// <summary>

/// Обновление таблиц и меток, которые сообщают об отсутствии информации.

/// </summary>

private void UpdateLabels()

{

try

{

if (serverInfoBindingSource.DataSource == null ||

(serverInfoBindingSource.DataSource as BindingList<ServerInfo>).Count == 0)

HideServerGrid();

else

{

dataGridView1.Visible = true;

dataGridView1.Dock = DockStyle.Fill;

lbl\_NoServerConfigs.Visible = false;

lbl\_NoServerConfigs.Dock = DockStyle.None;

}

}

catch

{

HideServerGrid();

}

try

{

if (sANInfoBindingSource.DataSource == null ||

(sANInfoBindingSource.DataSource as BindingList<SAN\_Info>).Count == 0)

HideSanGrid();

else

{

dataGridView2.Visible = true;

dataGridView2.Dock = DockStyle.Fill;

lbl\_NoSANs.Visible = false;

lbl\_NoSANs.Dock = DockStyle.None;

}

}

catch

{

HideSanGrid();

}

}

/// <summary>

/// Обработчик события выхода курсора из элемента главного меню,

/// который очищает область в строке состояния.

/// </summary>

private void MenuItem\_MouseLeave(object sender, EventArgs e)

{

tip\_Label.Text = string.Empty;

}

/// <summary>

/// Обработчик события навдения курсора на элемент главного меню,

/// который выводит подсказку элемента в строку состояния.

/// </summary>

private void MenuItem\_MouseEnter(object sender, EventArgs e)

{

tip\_Label.Text = ((ToolStripItem)sender).ToolTipText;

}

/// <summary>

/// Подписка элементов главного меню и панели элементов на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

private void SubscribeMenuItems()

{

SubScribeChildMenuItems(menuBar.Items);

foreach (ToolStripItem toolBarControl in toolBar.Items)

{

toolBarControl.MouseEnter += MenuItem\_MouseEnter;

toolBarControl.MouseLeave += MenuItem\_MouseLeave;

}

}

/// <summary>

/// Подписка дочерних элементов главного меню на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

/// <param name="col">Коллекция элементов главного меню</param>

private void SubScribeChildMenuItems(ToolStripItemCollection col)

{

for (int i = 0; i < col.Count; i++)

{

var item = col[i] as ToolStripMenuItem;

if (item != null)

{

item.MouseEnter += MenuItem\_MouseEnter;

item.MouseLeave += MenuItem\_MouseLeave;

if (item.DropDownItems.Count > 0)

SubScribeChildMenuItems(item.DropDownItems);

}

}

}

/// <summary>

/// Обработчиксобытиязакрытияформы

/// </summary>

private void EditInfrastructureFrom\_FormClosing(object sender, FormClosingEventArgs e)

{

Owner?.Show();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который выполняет переход на форму добавления новой конфигурации сервера

/// </summary>

private void созданиеНовойToolStripMenuItem\_Click(object sender, EventArgs e)

{

var serverConfigurationForm = new EditServerConfigurationForm { Owner = this };

serverConfigurationForm.Show();

Hide();

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который выполняет переход на форму редактирования выбранной конфигурации сервера

/// </summary>

private async void редактированиеВыбраннойToolStripMenuItem\_Click(object sender, EventArgs e)

{

var selectedServer = serverInfoBindingSource.Current as ServerInfo;

if (selectedServer == null)

{

MessageBox.Show("Выберитеконфигурациюсерверадляредактирования!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

var serverToEdit = await \_ctx.GetServerByServerInfo(selectedServer);

var serverConfigurationForm = new EditServerConfigurationForm(serverToEdit) { Owner = this };

serverConfigurationForm.Show();

Hide();

}

catch

{

MessageBox.Show("Не удалось получить информацию по выбранной конфигурации сервера!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который осуществляет удаление выбранной конфигурации сервера

/// </summary>

private async void удалениеВыбраннойToolStripMenuItem\_Click(object sender, EventArgs e)

{

var selectedServer = serverInfoBindingSource.Current as ServerInfo;

if (selectedServer == null)

{

MessageBox.Show("Выберите конфигурацию сервера для ее удаления!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

var serverToDel = await \_ctx.GetServerByServerInfo(selectedServer);

var result = MessageBox.Show("Выбранная конфигурация сервера будет удалена! Продолжить?",

"Удаление конфигурации сервера", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.Yes)

{

try

{

await \_ctx.DeleteServer(serverToDel);

UpdateBindings();

}

catch

{

MessageBox.Show("Произошлаошибкаприудаленииконфигурациисервера!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

}

}

catch

{

MessageBox.Show("Не удалось получить информацию по выбранной конфигурации сервера!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который выполняет переход на форму создания нового хранилища данных

/// </summary>

private void созданиеНовогоToolStripMenuItem\_Click(object sender, EventArgs e)

{

var editSanForm = new EditSANForm { Owner = this };

editSanForm.Show();

Hide();

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который выполняет переход на форму редактирования выбранного хранилища данных

/// </summary>

private async void редактированиеВыбранногоToolStripMenuItem\_Click(object sender, EventArgs e)

{

var selectedSAN = sANInfoBindingSource.Current as SAN\_Info;

if (selectedSAN == null)

{

MessageBox.Show("Выберитехранилищеданныхдляредактирования!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

var sanToEdit = await \_ctx.GetSANBySANInfo(selectedSAN);

var editSanForm = new EditSANForm(sanToEdit) { Owner = this };

editSanForm.Show();

Hide();

}

catch

{

MessageBox.Show("Не удалось получить информацию по выбранному хранилищу данных!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который осуществляет удаление выбранного хранилища данных

/// </summary>

private async void удалениеВыбранногоToolStripMenuItem\_Click(object sender, EventArgs e)

{

var selectedSAN = sANInfoBindingSource.Current as SAN\_Info;

if (selectedSAN == null)

{

MessageBox.Show("Выберитехранилищеданныхдляегоудаления!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

var sanToDel = await \_ctx.GetSANBySANInfo(selectedSAN);

var result = MessageBox.Show("Выбранное хранилище данных будет удалено! Продолжить?",

"Удаление конфигурации хранилища данных", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.Yes)

{

try

{

await \_ctx.DeleteSAN(sanToDel);

UpdateBindings();

}

catch

{

MessageBox.Show("Произошлаошибкаприудаленииконфигурациихранилищаданных!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

}

}

catch

{

MessageBox.Show("Не удалось получить информацию по выбранному хранилищу данных!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события активации формы

/// </summary>

private void EditInfrastructureFrom\_Activated(object sender, EventArgs e)

{

UpdateBindings();

}

/// <summary>

/// Обработчик события выполнения такта таймера

/// </summary>

private void timer\_Tick(object sender, EventArgs e)

{

if (\_ctx == null)

return;

try

{

\_ctx.CheckServerMaintenance();

\_ctx.CheckSANMaintenance();

UpdateBindings();

}

catch

{

}

}

/// <summary>

/// Обработчик нажатий клавиш на форме

/// </summary>

private void EditInfrastructureFrom\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.Escape:

Close();

break;

case Keys.Tab:

tabControl1.TabIndex = tabControl1.TabIndex > 0 ? 0 : 1;

break;

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно для задания условия поиска конфигурации сервера

/// </summary>

private void поискToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (ServersInfoMain != null && ServersInfoMain.Count > 0)

{

var serversCondition = new ServersConditionConstructorForm(ServersInfoMain);

if (serversCondition.ShowDialog() == DialogResult.OK)

{

int pos = 0;

if (!\_serverSelected)

pos = ServersInfoMain.IndexOf(serversCondition.ServersInfoToShow.First());

else

pos = ServersInfoToShow.IndexOf(serversCondition.ServersInfoToShow.First());

if (pos > -1)

serverInfoBindingSource.Position = pos;

}

}

else

MessageBox.Show("Отсутсвуют конфигурации серверов для совершения поиска!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно для задания условия отбора конфигураций серверов

/// </summary>

private void отборToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (ServersInfoMain != null && ServersInfoMain.Count > 0)

{

var serversCondition = new ServersConditionConstructorForm(ServersInfoMain);

if (serversCondition.ShowDialog() == DialogResult.OK)

{

\_serverSelected = true;

ServersInfoToShow = serversCondition.ServersInfoToShow;

serverInfoBindingSource.DataSource = ServersInfoToShow;

dataGridView1.DataSource = serverInfoBindingSource;

UpdateStatBar();

}

}

else

MessageBox.Show("Отсутсвуют конфигурации серверов для совершения отбора!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который совершает показ всех конфигураций серверов

/// </summary>

private void показатьВсеЗаписиToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (ServersInfoMain != null && ServersInfoMain.Count > 0)

{

\_serverSelected = false;

serverInfoBindingSource.DataSource = ServersInfoMain;

dataGridView1.DataSource = serverInfoBindingSource;

UpdateStatBar();

}

else

MessageBox.Show("Отсутсвуютконфигурациисерверов!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно для задания условия поиска конфигурации хранилища данных

/// </summary>

private void поискToolStripMenuItem1\_Click(object sender, EventArgs e)

{

if (SansInfoMain != null && SansInfoMain.Count > 0)

{

var sansCondition = new SAN\_ConditionConstructorForm(SansInfoMain);

if (sansCondition.ShowDialog() == DialogResult.OK)

{

int pos = 0;

if (!\_sansSelected)

pos = SansInfoMain.IndexOf(sansCondition.SansInfoToShow.First());

else

pos = SansInfoToShow.IndexOf(sansCondition.SansInfoToShow.First());

if (pos > -1)

sANInfoBindingSource.Position = pos;

}

}

else

MessageBox.Show("Отсутсвуют конфигурации хранилищ данных для совершения поиска!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно для задания условия отбора конфигураций хранилищ данных

/// </summary>

private void отборToolStripMenuItem1\_Click(object sender, EventArgs e)

{

if (SansInfoMain != null && SansInfoMain.Count > 0)

{

var sansCondition = new SAN\_ConditionConstructorForm(SansInfoMain);

if (sansCondition.ShowDialog() == DialogResult.OK)

{

\_sansSelected = true;

SansInfoToShow = sansCondition.SansInfoToShow;

sANInfoBindingSource.DataSource = SansInfoToShow;

dataGridView2.DataSource = sANInfoBindingSource;

UpdateStatBar();

}

}

else

MessageBox.Show("Отсутсвуют конфигурации хранилищ данных для совершения отбора!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который совершает показ всех конфигураций хранилищ данных

/// </summary>

private void показатьВсеЗаписиToolStripMenuItem1\_Click(object sender, EventArgs e)

{

if (SansInfoMain != null && SansInfoMain.Count > 0)

{

\_sansSelected = false;

sANInfoBindingSource.DataSource = SansInfoMain;

dataGridView2.DataSource = sANInfoBindingSource;

UpdateStatBar();

}

else

MessageBox.Show("Отсутсвуютконфигурациихранилищданных!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

/// <summary>

/// Обработчик события изменения индекса текущей вкладки

/// </summary>

private void tabControl1\_SelectedIndexChanged(object sender, EventArgs e)

{

UpdateStatBar();

}

}

}

Модуль «EditServiceTypeForm.cs»

using System;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Services.Dictionary

{

/// <summary>

/// Класс формы редактирования типов сервиса

/// </summary>

public partial class EditServiceTypeForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Список типов сервиса

/// </summary>

public BindingList<ServiceType> ServiceTypes { get; set; }

/// <summary>

/// Текущийтипсервиса

/// </summary>

public ServiceType CurrentServiceType { get; set; }

/// <summary>

/// Наименование типа сервиса до изменения

/// </summary>

private string \_serviceTypeNameBeforeEditing;

/// <summary>

/// Режимработыформы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Флаг редактирования типов сервиса

/// </summary>

public bool Edited { get; set; }

/// <summary>

/// Конструктор формы редактирования типов сервиса

/// </summary>

public EditServiceTypeForm()

{

InitializeComponent();

\_formMode = FormMode.None;

\_ctx = new Context();

EnDisFields(false);

BindCollection();

}

/// <summary>

/// Конструктор для выбора типа сервиса из списка

/// </summary>

/// <param name="findMode">Флагпоискатипасервиса</param>

public EditServiceTypeForm(bool findMode) : this()

{

if (!findMode)

throw new ArgumentException("Неверноезначениефлага!");

\_formMode = FormMode.None;

but\_Select.Visible = findMode;

but\_Select.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

}

/// <summary>

/// Привязкаколлекции

/// </summary>

private void BindCollection()

{

ServiceTypes = \_ctx.GetServiceTypesBS();

serviceTypeBindingSource.DataSource = ServiceTypes;

listBox1.DataSource = serviceTypeBindingSource;

}

/// <summary>

/// Активация/деактивацияполейдляредактирования

/// </summary>

/// <param name="enabled">Флагактивацииполейредактирования</param>

private void EnDisFields(bool enabled)

{

textBox2.Enabled = enabled;

btn\_Save.Enabled = enabled;

btn\_Cancel.Enabled = enabled;

textBox\_ConditionToFind.Enabled = !enabled;

listBox1.Enabled = !enabled;

picBtn\_FindByCondition.Enabled = !enabled;

picBtn\_ClearCondition.Enabled = !enabled;

button2.Enabled = !enabled;

button3.Enabled = !enabled;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных типов сервиса

/// </summary>

private void serviceTypeBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as ServiceType;

if (selectedItem != null)

CurrentServiceType = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который совершает добавление нового типа сервиса

/// </summary>

private void button2\_Click(object sender, EventArgs e)

{

ServiceTypes.AddNew();

serviceTypeBindingSource.MoveLast();

EnDisFields(true);

\_formMode = FormMode.Add;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который совершает редактирование выбранного типа сервиса

/// </summary>

private void button3\_Click(object sender, EventArgs e)

{

if (CurrentServiceType != null)

{

EnDisFields(true);

\_serviceTypeNameBeforeEditing = CurrentServiceType.Name;

\_formMode = FormMode.Edit;

}

else

MessageBox.Show("Выберите тип сервиса из списка или добавьте новый!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который выполняет сохранение изменений

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

try

{

if (string.IsNullOrWhiteSpace(CurrentServiceType.Name))

{

MessageBox.Show("Введите наименование типа сервиса!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

bool existed = false;

switch (\_formMode)

{

case FormMode.Add:

existed = await \_ctx.CheckServiceTypeForDublicate(CurrentServiceType.Name);

break;

case FormMode.Edit:

existed = await \_ctx.CheckServiceTypeForDublicate(CurrentServiceType.Name) && !CurrentServiceType.Name.Equals(\_serviceTypeNameBeforeEditing);

break;

}

if (existed)

{

MessageBox.Show("Тип сервиса с таким наименованием уже существует!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (\_formMode == FormMode.Add)

ServiceTypes.EndNew(ServiceTypes.IndexOf(CurrentServiceType));

await \_ctx.SaveChangesAsync();

EnDisFields(false);

Edited = true;

\_formMode = FormMode.None;

DialogResult = DialogResult.OK;

MessageBox.Show("Измененияуспешносохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

catch

{

MessageBox.Show("Изменениянеудалосьсохранить!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит отмену последнего добавления/редактирования

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

try

{

if (\_formMode == FormMode.Edit)

{

int indexOfElement = ServiceTypes.IndexOf(CurrentServiceType);

CurrentServiceType = \_ctx.CancelChanges(CurrentServiceType);

ServiceTypes[indexOfElement] = CurrentServiceType;

}

else

ServiceTypes.CancelNew(ServiceTypes.IndexOf(CurrentServiceType));

EnDisFields(false);

\_formMode = FormMode.None;

}

catch

{

MessageBox.Show("Произошлаошибкаприотменеизменений!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который совершает подтверждение выбранного типа сервиса

/// </summary>

private void but\_Select\_Click(object sender, EventArgs e)

{

if (CurrentServiceType != null)

{

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Выберите тип сервиса из списка или добавьте новый!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditServiceTypeForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Ограничение ввода в наименовании добавляемого/редактируемого типа сервиса

/// </summary>

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!Char.IsControl(e.KeyChar) && !Char.IsLetter(e.KeyChar) && e.KeyChar != (int) Keys.Space)

e.Handled = true;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаграфическийобъект,

/// который производит очистку строки для поиска

/// </summary>

private void picBtn\_ClearCondition\_Click(object sender, EventArgs e)

{

textBox\_ConditionToFind.Text = string.Empty;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит поиск типа сервиса по наименованию

/// </summary>

private void picBtn\_FindByCondition\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(textBox\_ConditionToFind.Text))

{

MessageBox.Show("Введите условие для поиска типа сервиса!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

var firstFoundedElement = ServiceTypes.FirstOrDefault(x => x.Name.ToUpper().Contains(textBox\_ConditionToFind.Text.ToUpper()));

if (firstFoundedElement != null)

serviceTypeBindingSource.Position = ServiceTypes.IndexOf(firstFoundedElement);

else

MessageBox.Show("Не удалось найти тип сервиса по указанному условию!", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события нажатий клавиш клавиатуры на форме

/// </summary>

private void EditServiceTypeForm\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.Escape:

btn\_Cancel\_Click(null, EventArgs.Empty);

break;

case Keys.Enter:

btn\_Save\_Click(null, EventArgs.Empty);

break;

}

}

}

}

Модуль «EditSoftwareForm.cs»

using System;

using System.ComponentModel;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Services.Dictionary

{

/// <summary>

/// Класс формы редактирования программного обеспечения

/// </summary>

public partial class EditSoftwareForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private Context \_ctx;

/// <summary>

/// Наименование программного обеспечения перед изменением

/// </summary>

private string \_softwareNameBeforeEdit;

/// <summary>

/// Наименование типа программного обеспечения перед изменением

/// </summary>

private string \_softawreTypeBeforeEdit;

/// <summary>

/// Списокпрограммногообеспечения

/// </summary>

public BindingList<SoftwareInfo> Softwares { get; set; }

/// <summary>

/// Текущее программное обеспечение

/// </summary>

public Software CurrentSoftware { get; set; }

/// <summary>

/// Режим работы формы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Флаг редактирования программного обеспечения

/// </summary>

public bool Edited { get; set; }

/// <summary>

/// Конструктор формы редактирования программного обеспечения

/// </summary>

public EditSoftwareForm()

{

InitializeComponent();

\_ctx = new Context();

EnDisFields(false);

BindCollection();

}

/// <summary>

/// Конструктор формы редактирования выбранного программного обеспечения

/// </summary>

public EditSoftwareForm(Software selectedSoftware) : this()

{

\_formMode = FormMode.Edit;

Text = "Редактирование оперативной памяти";

InitEditSoftware(selectedSoftware);

EnDisFields(true);

}

/// <summary>

/// Обновление привязки для редактирования выбранного программного обеспечения

/// </summary>

private void InitEditSoftware(Software selectedSoftware)

{

CurrentSoftware = selectedSoftware;

softwareBindingSource.DataSource = CurrentSoftware;

\_softwareNameBeforeEdit = CurrentSoftware.Name;

\_softawreTypeBeforeEdit = CurrentSoftware.SoftType.Name;

softwareBindingSource.ResumeBinding();

}

/// <summary>

/// Привязкаколлекции

/// </summary>

private void BindCollection()

{

softTypeBindingSource.DataSource = \_ctx.GetSoftwareTypes();

comboBox2.DataSource = softTypeBindingSource;

Softwares = \_ctx.GetSoftwaresInfo();

softwareInfoBindingSource.DataSource = Softwares;

dataGridView1.DataSource = softwareInfoBindingSource;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных программного обеспечения

/// </summary>

private void softwareInfoBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as SoftwareInfo;

if (selectedItem != null)

{

CurrentSoftware = \_ctx.GetSoftwareByInfo(selectedItem);

softwareBindingSource.DataSource = CurrentSoftware;

}

}

/// <summary>

/// Активация/деактивацияполейдляредактирования

/// </summary>

/// <param name="enabled">Флагактивацииполейредактирования</param>

private void EnDisFields(bool enabled)

{

textBox2.Enabled = enabled;

textBox3.Enabled = enabled;

comboBox2.Enabled = enabled;

btn\_Cancel.Enabled = enabled;

btn\_Save.Enabled = enabled;

textBox1.Enabled = !enabled;

comboBox1.Enabled = !enabled;

dataGridView1.Enabled = !enabled;

pictureBox1.Enabled = !enabled;

pictureBox2.Enabled = !enabled;

button2.Enabled = !enabled;

button3.Enabled = !enabled;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который совершает добавление нового программного обеспечения

/// </summary>

private void button2\_Click(object sender, EventArgs e)

{

CurrentSoftware = new Software();

softwareBindingSource.DataSource = CurrentSoftware;

EnDisFields(true);

\_formMode = FormMode.Add;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который совершает редактирование выбранного программного обеспечения

/// </summary>

private void button3\_Click(object sender, EventArgs e)

{

if (CurrentSoftware != null)

{

EnDisFields(true);

\_softawreTypeBeforeEdit = CurrentSoftware.SoftType.Name;

\_softwareNameBeforeEdit = CurrentSoftware.Name;

\_formMode = FormMode.Edit;

}

else

MessageBox.Show("Выберите программное обеспечение из списка или добавьте новое!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену последнего добавления/редактирования

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который выполняет сохранение изменений

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

try

{

var selectedSoftType = softTypeBindingSource.Current as SoftType;

if (selectedSoftType == null || string.IsNullOrWhiteSpace(CurrentSoftware.Name))

{

MessageBox.Show("Заполните все поля перед сохранением!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

bool existed = false;

switch (\_formMode)

{

case FormMode.Add:

existed = await \_ctx.CheckSoftwareForDublicate(CurrentSoftware.Name, selectedSoftType.Name);

break;

case FormMode.Edit:

existed = await \_ctx.CheckSoftwareForDublicate(CurrentSoftware.Name, selectedSoftType.Name)

&& (!CurrentSoftware.Name.Equals(\_softwareNameBeforeEdit) || !selectedSoftType.Name.Equals(\_softawreTypeBeforeEdit));

break;

}

if (existed)

{

MessageBox.Show("Программное обеспечение с таким типом и наименованием уже существует!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

switch (\_formMode)

{

case FormMode.Add:

await \_ctx.AddNewSoftware(CurrentSoftware);

break;

case FormMode.Edit:

await \_ctx.EditSoftware(CurrentSoftware);

break;

}

EnDisFields(false);

if (\_formMode == FormMode.Add)

Softwares.Add(new SoftwareInfo

{

Software = CurrentSoftware.Name,

SoftType = selectedSoftType.Name,

Cost = CurrentSoftware.Cost

});

else

BindCollection();

\_formMode = FormMode.None;

Edited = true;

DialogResult = DialogResult.OK;

MessageBox.Show("Изменения успешно сохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

catch

{

MessageBox.Show("Изменения не удалось сохранить!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditSoftwareForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который совершает подтверждение выбранного программного обеспечения

/// </summary>

private void but\_Select\_Click(object sender, EventArgs e)

{

if (CurrentSoftware != null)

{

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Выберите программное обеспечение из списка или добавьте новое!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит отмену выбранного типа программного обеспечения для поиска

/// </summary>

private void pictureBox2\_Click(object sender, EventArgs e)

{

comboBox1.SelectedIndex = -1;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных типов программного обеспечения

/// </summary>

private void softTypeBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (comboBox2.SelectedIndex == -1)

((BindingSource)sender).SuspendBinding();

else

((BindingSource)sender).ResumeBinding();

}

/// <summary>

/// Обработчик события нажатий клавиш клавиатуры на форме

/// </summary>

private void EditSoftwareForm\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.Escape:

btn\_Cancel\_Click(null, EventArgs.Empty);

break;

case Keys.Enter:

btn\_Save\_Click(null, EventArgs.Empty);

break;

}

}

//todo: сделатьпоиск

private void pictureBox1\_Click(object sender, EventArgs e)

{

}

}

}

Модуль «EditSoftwareTypeForm.cs»

using System;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Services.Dictionary

{

/// <summary>

/// Класс формы редактирования типов программного обеспечения

/// </summary>

public partial class EditSoftwareTypeForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private Context \_ctx;

/// <summary>

/// Список типов программного обеспечения

/// </summary>

public BindingList<SoftType> SoftTypes { get; set; }

/// <summary>

/// Текущий тип программного обеспечения

/// </summary>

public SoftType CurrentSoftType { get; set; }

/// <summary>

/// Наименование типа программного обеспечения до изменения

/// </summary>

private string \_softwareTypeNameBeforeEditing;

/// <summary>

/// Режимработыформы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Флаг редактирования типов программного обеспечения

/// </summary>

public bool Edited { get; set; }

/// <summary>

/// Конструктор формы редактирования типов программного обеспечения

/// </summary>

public EditSoftwareTypeForm()

{

InitializeComponent();

\_formMode = FormMode.None;

\_ctx = new Context();

EnDisFields(false);

BindCollection();

}

/// <summary>

/// Конструктор для выбора типа программного обеспечения из списка

/// </summary>

/// <param name="findMode">Флаг поиска типа программного обеспечения</param>

public EditSoftwareTypeForm(bool findMode) : this()

{

if (!findMode)

throw new ArgumentException("Неверноезначениефлага!");

\_formMode = FormMode.None;

but\_Select.Visible = findMode;

but\_Select.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

}

/// <summary>

/// Привязкаколлекции

/// </summary>

private void BindCollection()

{

SoftTypes = \_ctx.GetSoftTypesBS();

softTypeBindingSource.DataSource = SoftTypes;

listBox1.DataSource = softTypeBindingSource;

}

/// <summary>

/// Активация/деактивацияполейдляредактирования

/// </summary>

/// <param name="enabled">Флагактивацииполейредактирования</param>

private void EnDisFields(bool enabled)

{

textBox2.Enabled = enabled;

btn\_Save.Enabled = enabled;

btn\_Cancel.Enabled = enabled;

textBox\_ConditionToFind.Enabled = !enabled;

listBox1.Enabled = !enabled;

picBtn\_FindByCondition.Enabled = !enabled;

picBtn\_ClearCondition.Enabled = !enabled;

button2.Enabled = !enabled;

button3.Enabled = !enabled;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных типов программного обеспечения

/// </summary>

private void softTypeBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as SoftType;

if (selectedItem != null)

CurrentSoftType = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который совершает добавление нового типа программного обеспечения

/// </summary>

private void button2\_Click(object sender, EventArgs e)

{

SoftTypes.AddNew();

softTypeBindingSource.MoveLast();

EnDisFields(true);

\_formMode = FormMode.Add;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который совершает редактирование выбранного типа программного обеспечения

/// </summary>

private void button3\_Click(object sender, EventArgs e)

{

if (CurrentSoftType != null)

{

EnDisFields(true);

\_softwareTypeNameBeforeEditing = CurrentSoftType.Name;

\_formMode = FormMode.Edit;

}

else

MessageBox.Show("Выберите тип программного обеспечения из списка или добавьте новое!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который выполняет сохранение изменений

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

try

{

if (string.IsNullOrWhiteSpace(CurrentSoftType.Name))

{

MessageBox.Show("Введите наименование типа программного обеспечения!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

bool existed = false;

switch (\_formMode)

{

case FormMode.Add:

existed = await \_ctx.CheckSoftTypeForDublicate(CurrentSoftType.Name);

break;

case FormMode.Edit:

existed = await \_ctx.CheckSoftTypeForDublicate(CurrentSoftType.Name) && !CurrentSoftType.Name.Equals(\_softwareTypeNameBeforeEditing);

break;

}

if (existed)

{

MessageBox.Show("Тип программного обеспечения с таким наименованием уже существует!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (\_formMode == FormMode.Add)

SoftTypes.EndNew(SoftTypes.IndexOf(CurrentSoftType));

\_ctx.SaveChanges();

EnDisFields(false);

Edited = true;

\_formMode = FormMode.None;

DialogResult = DialogResult.OK;

MessageBox.Show("Изменения успешно сохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

catch

{

MessageBox.Show("Изменения не удалось сохранить!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену последнего добавления/редактирования

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

try

{

if (\_formMode == FormMode.Edit)

{

int indexOfElement = SoftTypes.IndexOf(CurrentSoftType);

CurrentSoftType = \_ctx.CancelChanges(CurrentSoftType);

SoftTypes[indexOfElement] = CurrentSoftType;

}

else

SoftTypes.CancelNew(SoftTypes.IndexOf(CurrentSoftType));

EnDisFields(false);

\_formMode = FormMode.None;

}

catch

{

MessageBox.Show("Произошлаошибкаприотменеизменений!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который совершает подтверждение выбранного программного обеспечения

/// </summary>

private void but\_Select\_Click(object sender, EventArgs e)

{

if (CurrentSoftType != null)

{

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Выберите тип программного обеспечения из списка или добавьте новый!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditSoftwareTypeForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Ограничение ввода в наименовании добавляемого/редактируемого программного обеспечения

/// </summary>

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!Char.IsControl(e.KeyChar) && !Char.IsLetter(e.KeyChar) && e.KeyChar != (int)Keys.Space && !Char.IsDigit(e.KeyChar))

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит очистку строки для поиска

/// </summary>

private void picBtn\_ClearCondition\_Click(object sender, EventArgs e)

{

textBox\_ConditionToFind.Text = string.Empty;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит поиск программного обеспечения по наименованию

/// </summary>

private void picBtn\_FindByCondition\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(textBox\_ConditionToFind.Text))

{

MessageBox.Show("Введите условие для поиска программного обеспечения!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

var firstFoundedElement =SoftTypes.FirstOrDefault(x => x.Name.ToUpper().Contains(textBox\_ConditionToFind.Text.ToUpper()));

if (firstFoundedElement != null)

softTypeBindingSource.Position = SoftTypes.IndexOf(firstFoundedElement);

else

MessageBox.Show("Не удалось найти программное обеспечение по указанному условию!", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события нажатий клавиш клавиатуры на форме

/// </summary>

private void EditSoftwareTypeForm\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.Escape:

btn\_Cancel\_Click(null,EventArgs.Empty);

break;

case Keys.Enter:

btn\_Save\_Click(null, EventArgs.Empty);

break;

}

}

}

}

Модуль «SelectSANForm.cs»

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Services.Parts

{

/// <summary>

/// Класс формы выбора хранилища данных

/// </summary>

public partial class SelectSANForm : Form

{

private readonly List<int> \_selectedIDs;

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Выбранное хранилище данных

/// </summary>

public SANPaasInfo SelectedSANPaasInfo { get; set; }

/// <summary>

/// Коллекция хранилищ данных

/// </summary>

public BindingList<SANPaasInfo> SANPaasInfo { get; set; }

/// <summary>

/// Конструктор формы выбора хранилища данных

/// </summary>

public SelectSANForm(List<int> selectedIDs)

{

\_selectedIDs = selectedIDs;

InitializeComponent();

btn\_OK.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

\_ctx = new Context();

InitCollection();

}

/// <summary>

/// Инициализация привязки коллекции хранилищ данных.

/// </summary>

private async void InitCollection()

{

SANPaasInfo = new BindingList<SANPaasInfo>();

var sansList = await \_ctx.GetSANsList();

sansList.ForEach(sl => SANPaasInfo.Add(\_ctx.GetSANPaasInfoBySAN(sl)));

sANPaasInfoBindingSource.DataSource = SANPaasInfo;

dataGridView1.DataSource = sANPaasInfoBindingSource;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену выбора хранилища данных, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит подтверждение выбранного хранилища данных

/// </summary>

private void btn\_OK\_Click(object sender, EventArgs e)

{

if (SelectedSANPaasInfo != null)

{

if (\_selectedIDs.Contains(SelectedSANPaasInfo.Id))

{

MessageBox.Show("Выбранное файловое хранилище уже присутсвует в списке выбранных!",

"Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Выберите хранилище данных из списка!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных конфигураций хранилищ данных

/// </summary>

private void sANPaasInfoBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as SANPaasInfo;

if (selectedItem != null)

SelectedSANPaasInfo = selectedItem;

}

/// <summary>

/// Обработчик события нажатий клавиш клавиатуры на форме

/// </summary>

private void SelectSANForm\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.Escape:

Close();

break;

case Keys.Enter:

btn\_OK\_Click(null, EventArgs.Empty);

break;

}

}

}

}

Модуль «SelectServerForm.cs»

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Services.Parts

{

/// <summary>

/// Класс формы выбора конфигурации сервера

/// </summary>

public partial class SelectServerForm : Form

{

private readonly List<int> \_selectedIDs;

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Выбранныйсервер

/// </summary>

public ServerPaasInfo SelectedServerPaasInfo { get; set; }

/// <summary>

/// Коллекция конфигураций серверов

/// </summary>

public BindingList<ServerPaasInfo> ServersPaasInfo { get; set; }

/// <summary>

/// Конструктор формы выбора конфигурации сервера

/// </summary>

public SelectServerForm(List<int> selectedIDs)

{

\_selectedIDs = selectedIDs;

InitializeComponent();

btn\_OK.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

\_ctx = new Context();

InitCollection();

}

/// <summary>

/// Инициализация привязки коллекции конфигураций серверов.

/// </summary>

private async void InitCollection()

{

ServersPaasInfo = new BindingList<ServerPaasInfo>();

var serverList = await \_ctx.GetServerConfigs();

serverList.ForEach(sl => ServersPaasInfo.Add(\_ctx.GetServerPaasInfoByServer(sl)));

serverPaasInfoBindingSource.DataSource = ServersPaasInfo;

dataGridView1.DataSource = serverPaasInfoBindingSource;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену выбора конфигурации сервера, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит подтверждение выбранной конфигурации сервера

/// </summary>

private void btn\_OK\_Click(object sender, EventArgs e)

{

if (SelectedServerPaasInfo != null)

{

if (\_selectedIDs.Contains(SelectedServerPaasInfo.Id))

{

MessageBox.Show("Выбранная конфигурация сервера уже присутсвует в списке выбранных!",

"Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Выберитесерверизсписка!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Warning);

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных конфигураций серверов

/// </summary>

private void serverPaasInfoBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as ServerPaasInfo;

if (selectedItem != null)

SelectedServerPaasInfo = selectedItem;

}

/// <summary>

/// Обработчик события нажатий клавиш клавиатуры на форме

/// </summary>

private void SelectServerForm\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.Escape:

Close();

break;

case Keys.Enter:

btn\_OK\_Click(null, EventArgs.Empty);

break;

}

}

}

}

Модуль «SelectSoftwareForm.cs»

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Forms.SysAdmin.Services.Dictionary;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Services.Parts

{

/// <summary>

/// Класс формы выбора программного обеспечения

/// </summary>

public partial class SelectSoftwareForm : Form

{

private readonly List<int> \_selectedIDs;

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Текущee программное обеспечение

/// </summary>

public SoftwareInfo CurrentSoftware { get; set; }

/// <summary>

/// Выбранное программное обеспечение

/// </summary>

public Software SelectedSoftware { get; set; }

/// <summary>

/// Флаг отмены выбора программного обеспечения

/// </summary>

private bool \_canceled;

/// <summary>

/// Конструктор формы выбора программного обеспечения

/// </summary>

public SelectSoftwareForm(List<int> selectedIDs)

{

\_selectedIDs = selectedIDs;

InitializeComponent();

SubscribeMenuItems();

btn\_OK.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

созданиеToolStripMenuItem1.Click += созданиеToolStripMenuItem\_Click;

редактированиеToolStripMenuItem1.Click += редактированиеToolStripMenuItem\_Click;

удалениеToolStripMenuItem1.Click += удалениеToolStripMenuItem\_Click;

\_ctx = new Context();

InitBindings();

}

/// <summary>

/// Инициализацияпривязокданных

/// </summary>

private void InitBindings()

{

softwareInfoBindingSource.DataSource = \_ctx.GetSoftwaresInfo();

dataGridView1.DataSource = softwareInfoBindingSource;

txt\_RecCount.Text = $"Общееколичество: {softwareInfoBindingSource.Count}";

}

/// <summary>

/// Обработчик события выхода курсора из элемента главного меню,

/// который очищает область в строке состояния.

/// </summary>

private void MenuItem\_MouseLeave(object sender, EventArgs e)

{

tip\_Label.Text = string.Empty;

}

/// <summary>

/// Обработчик события навдения курсора на элемент главного меню,

/// который выводит подсказку элемента в строку состояния.

/// </summary>

private void MenuItem\_MouseEnter(object sender, EventArgs e)

{

tip\_Label.Text = ((ToolStripItem)sender).ToolTipText;

}

/// <summary>

/// Подписка элементов главного меню и панели элементов на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

private void SubscribeMenuItems()

{

SubScribeChildMenuItems(menuBar.Items);

}

/// <summary>

/// Подписка дочерних элементов главного меню на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

/// <param name="col">Коллекция элементов главного меню</param>

private void SubScribeChildMenuItems(ToolStripItemCollection col)

{

for (int i = 0; i < col.Count; i++)

{

var item = col[i] as ToolStripMenuItem;

if (item != null)

{

item.MouseEnter += MenuItem\_MouseEnter;

item.MouseLeave += MenuItem\_MouseLeave;

if (item.DropDownItems.Count > 0)

SubScribeChildMenuItems(item.DropDownItems);

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который открывает диалоговое окно формы создания нового программного обеспечения

/// </summary>

private void созданиеToolStripMenuItem\_Click(object sender, EventArgs e)

{

var editSoftwareForm = new EditSoftwareForm();

if (editSoftwareForm.ShowDialog() == DialogResult.OK)

InitBindings();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы редактирования выбранного программного обеспечения

/// </summary>

private void редактированиеToolStripMenuItem\_Click(object sender, EventArgs e)

{

SelectedSoftware = \_ctx.GetSoftwareByInfo(CurrentSoftware);

if (SelectedSoftware != null)

{

var editRamForm = new EditSoftwareForm(SelectedSoftware);

if (editRamForm.ShowDialog() == DialogResult.OK)

{

InitBindings();

if (CurrentSoftware != null)

{

int pos = softwareInfoBindingSource.IndexOf(CurrentSoftware);

if (pos > -1)

softwareInfoBindingSource.Position = pos;

}

}

}

else

MessageBox.Show("Выберите программное обеспечение из списка для его редактирования!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который выполняет удаление выбранного программного обеспечения

/// </summary>

private async void удалениеToolStripMenuItem\_Click(object sender, EventArgs e)

{

SelectedSoftware = \_ctx.GetSoftwareByInfo(CurrentSoftware);

if (SelectedSoftware != null)

{

var result = MessageBox.Show("Выбранное программное обеспечение будет удалено! Продолжить?",

"Удаление программного обеспечения", MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

try

{

await \_ctx.DeleteSoftware(SelectedSoftware);

InitBindings();

}

catch

{

MessageBox.Show("Произошла ошибка при удалении программного обеспечения!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

else

MessageBox.Show("Выберите программное обеспечение из списка для его удаления!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит подтверждение выбранного программного обеспечения

/// </summary>

private void btn\_OK\_Click(object sender, EventArgs e)

{

if (CurrentSoftware != null)

{

if (\_selectedIDs.Contains(CurrentSoftware.Id))

{

MessageBox.Show("Выбранное программное обеспечение уже присутсвует в списке выбранных!",

"Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Выберите программное обеспечение из списка!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену выбора программного обеспечения, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

\_canceled = true;

Close();

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных программного обеспечения

/// </summary>

private void softwareInfoBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as SoftwareInfo;

if (selectedItem != null)

CurrentSoftware = selectedItem;

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void SelectSoftwareForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_canceled)

{

var resut = MessageBox.Show("Программное обеспечение не будет выбрано! Продолжить?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Hand);

if (resut == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Обработчик события нажатий клавиш клавиатуры на форме

/// </summary>

private void SelectSoftwareForm\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.Escape:

Close();

break;

case Keys.Enter:

btn\_OK\_Click(null, EventArgs.Empty);

break;

}

}

}

}

Модуль «EditPlatformForm.cs»

using System;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Forms.SysAdmin.Services.Parts;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Services

{

/// <summary>

/// Класс формы редактирования платформы

/// </summary>

public partial class EditPlatformForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Редактируемуя оперативная память

/// </summary>

public PaasType CurrentPaas { get; set; }

/// <summary>

/// Выбранныесервера

/// </summary>

public BindingList<ServerPaasInfo> SelectedServers { get; set; }

/// <summary>

/// Выбранные хранилища данных

/// </summary>

public BindingList<SANPaasInfo> SelectedSANs { get; set; }

/// <summary>

/// Наименование модели оперативной памяти перед редактированием

/// </summary>

private string \_paasTypeNameBeforeEdit;

/// <summary>

/// Режимработыформы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Конструктор формы создания новой платформы

/// </summary>

public EditPlatformForm()

{

InitializeComponent();

btn\_Save.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

\_ctx = new Context();

CurrentPaas = new PaasType();

paasTypeBindingSource.DataSource = CurrentPaas;

BindSelectedServers();

BindSelectedSANs();

\_formMode = FormMode.Add;

Text = "Созданиеновойплатформы";

}

/// <summary>

/// Конструктор формы редактирования выбранной платформы

/// </summary>

/// <param name="selectedPaas">Выбраннаяплатформа</param>

public EditPlatformForm(PaasType selectedPaas) : this()

{

\_formMode = FormMode.Edit;

Text = "Редактированиеплатформы";

InitEditPaas(selectedPaas);

}

/// <summary>

/// Инициализация привязок выбранной платформы для редактирования

/// </summary>

/// <param name="selectedPaas">Выбраннаяплатформа</param>

private void InitEditPaas(PaasType selectedPaas)

{

CurrentPaas = selectedPaas;

paasTypeBindingSource.DataSource = CurrentPaas;

\_paasTypeNameBeforeEdit = CurrentPaas.Name;

paasTypeBindingSource.ResumeBinding();

var serverList = \_ctx.GetServerConfigsByPaas(CurrentPaas);

serverList.ForEach(sl => SelectedServers.Add(\_ctx.GetServerPaasInfoByServer(sl)));

serverPaasInfoBindingSource.DataSource = SelectedServers;

dataGridView1.DataSource = serverPaasInfoBindingSource;

var sansList = \_ctx.GetSANsListByPaas(CurrentPaas);

sansList.ForEach(sl => SelectedSANs.Add(\_ctx.GetSANPaasInfoBySAN(sl)));

sANPaasInfoBindingSource.DataSource = SelectedSANs;

dataGridView2.DataSource = sANPaasInfoBindingSource;

}

/// <summary>

/// Привязка списка выбранных конфигураций серверов

/// </summary>

private void BindSelectedServers()

{

SelectedServers = new BindingList<ServerPaasInfo>();

serverPaasInfoBindingSource.DataSource = SelectedServers;

dataGridView1.DataSource = serverPaasInfoBindingSource;

}

/// <summary>

/// Привязка списка выбранных хранилищ данных

/// </summary>

private void BindSelectedSANs()

{

SelectedSANs = new BindingList<SANPaasInfo>();

sANPaasInfoBindingSource.DataSource = SelectedSANs;

dataGridView2.DataSource = sANPaasInfoBindingSource;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно добавления сервера к платформе

/// </summary>

private void picBtn\_AddServer\_Click(object sender, EventArgs e)

{

var selectedIDs = SelectedServers.ToList().Select(s => s.Id).ToList();

var selectServerForm = new SelectServerForm(selectedIDs);

if (selectServerForm.ShowDialog() == DialogResult.OK)

{

try

{

var selServ = selectServerForm.SelectedServerPaasInfo;

selServ.UsedCoreCount = 0;

selServ.UsedRAMVolume = 0;

selServ.UsedStorageVolume = 0;

SelectedServers.Add(selServ);

}

catch

{

MessageBox.Show("Произошлаошибкапридобавлениивыбранногосервера!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет удаление выбранного сервера

/// </summary>

private void picBtn\_DeleteServer\_Click(object sender, EventArgs e)

{

var selectedServer = serverPaasInfoBindingSource.Current as ServerPaasInfo;

if (selectedServer == null)

{

MessageBox.Show("Конфигурациясервераневыбрана!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

SelectedServers.Remove(selectedServer);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно добавления хранилища данных к платформе

/// </summary>

private void picBtn\_AddSAN\_Click(object sender, EventArgs e)

{

var selectedIDs = SelectedSANs.ToList().Select(s => s.Id).ToList();

var selectSanForm = new SelectSANForm(selectedIDs);

if (selectSanForm.ShowDialog() == DialogResult.OK)

{

try

{

var selSan = selectSanForm.SelectedSANPaasInfo;

selSan.UsedVolume = 0;

SelectedSANs.Add(selSan);

}

catch

{

MessageBox.Show("Произошла ошибка при добавлении выбранного хранилища данных!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который осуществляет удаление выбранного хранилища данных

/// </summary>

private void picBtn\_DeleteSAN\_Click(object sender, EventArgs e)

{

var sanPaasInfo = sANPaasInfoBindingSource.Current as SANPaasInfo;

if (sanPaasInfo == null)

{

MessageBox.Show("Хранилищеданныхсервераневыбрано!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

SelectedSANs.Remove(sanPaasInfo);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который выполняет сохранение изменений

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (\_formMode == FormMode.None)

return;

if (string.IsNullOrWhiteSpace(CurrentPaas.Name))

{

MessageBox.Show("Введите наименование платформы!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

if (SelectedServers.Count == 0)

{

MessageBox.Show("Выберите конфигурации серверов для сохранения платформы!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (SelectedSANs.Count == 0)

{

MessageBox.Show("Выберитехранилищаданныхдлясохраненияплатформы!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

bool existed = false;

switch (\_formMode)

{

case FormMode.Add:

existed = await \_ctx.CheckPaasForDublicate(CurrentPaas.Name);

break;

case FormMode.Edit:

existed = await \_ctx.CheckPaasForDublicate(CurrentPaas.Name) && !CurrentPaas.Name.Equals(\_paasTypeNameBeforeEdit);

break;

}

if (existed)

{

MessageBox.Show("Платформа с таким наименованием уже существует!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

CurrentPaas.Server.Clear();

foreach (var selectedServer in SelectedServers)

CurrentPaas.Server.Add(await \_ctx.GetServerByID(selectedServer.Id));

CurrentPaas.SAN.Clear();

foreach (var selectedSaN in SelectedSANs)

CurrentPaas.SAN.Add(await \_ctx.GetSANByID(selectedSaN.Id));

switch (\_formMode)

{

case FormMode.Add:

await \_ctx.AddNewPaas(CurrentPaas);

MessageBox.Show("Новаяплатформауспешносоздана!", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

break;

case FormMode.Edit:

await \_ctx.EditPaas(CurrentPaas);

MessageBox.Show("Изменения в платформе успешно сохранены!", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

break;

}

\_formMode = FormMode.None;

DialogResult = DialogResult.OK;

Close();

}

catch

{

switch (\_formMode)

{

case FormMode.Add:

MessageBox.Show("Не удалось создать новую платформу!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

break;

case FormMode.Edit:

MessageBox.Show("Не удалось сохранить изменения в платформе!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

break;

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену добавления/редактирования, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditPlatformForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.No)

{

e.Cancel = true;

return;

}

}

Owner?.Show();

}

/// <summary>

/// Обработчик события нажатий клавиш клавиатуры на форме

/// </summary>

private void EditPlatformForm\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.Escape:

Close();

break;

case Keys.Enter:

btn\_Save\_Click(null, EventArgs.Empty);

break;

}

}

}

}

Модуль «EditServiceForm.cs»

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Forms.SysAdmin.Services.Dictionary;

using MTF\_Services.WinForms.Forms.SysAdmin.Services.Parts;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Services

{

/// <summary>

/// Класс формы редактирования сервиса

/// </summary>

public partial class EditServiceForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Режим работы формы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Редактируемыйсервис

/// </summary>

public Service CurrentService { get; set; }

/// <summary>

/// Информация о стоимости компонентов сервиса

/// </summary>

public ServicePartsInfo ServicePartsInfo { get; set; }

/// <summary>

/// Выбранное программное обеспечение

/// </summary>

public List<SoftwareInfo> SelectedSoftware { get; set; }

/// <summary>

/// Конструктор формы создания нового сервиса

/// </summary>

public EditServiceForm()

{

InitializeComponent();

btn\_Save.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

\_ctx = new Context();

CurrentService = new Service();

BindServerPartsInfo();

BindPlatforms();

BindServiceTypes();

BindSelectedSoftware();

\_formMode = FormMode.Add;

Text = "Создание нового сервиса";

}

/// <summary>

/// Конструктор формы создания нового сервиса с выбранной платформой

/// </summary>

/// <param name="selectedPlatform"></param>

public EditServiceForm(PaasType selectedPlatform) : this()

{

int selectedIndex = ((BindingList<PaasType>)paasTypeBindingSource.DataSource).IndexOf(selectedPlatform);

if (selectedIndex > -1)

{

paasTypeBindingSource.Position = selectedIndex;

comboBox\_PaasTypes.SelectedIndex = selectedIndex;

}

}

/// <summary>

/// Конструктор формы редактирования выбранного сервиса

/// </summary>

/// <param name="selectedService">Выбранныйсервис</param>

public EditServiceForm(Service selectedService)

{

InitializeComponent();

btn\_Save.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

btn\_Cancel.Image = new Bitmap(Resources.no, new Size(20, 20));

\_ctx = new Context();

CurrentService = selectedService;

PrepareBindingsToEditService();

BindServerPartsInfo();

BindPlatforms();

BindServiceTypes();

BindSelectedSoftware();

\_formMode = FormMode.Edit;

Text = "Редактированиесервиса";

}

/// <summary>

/// Инициализация привязок для редактирования конфигурации сервиса

/// </summary>

private void PrepareBindingsToEditService()

{

serviceBindingSource.DataSource = CurrentService;

serviceBindingSource.ResumeBinding();

paasTypeBindingSource\_CurrentChanged(paasTypeBindingSource, EventArgs.Empty);

foreach (var software in CurrentService.Software)

{

SelectedSoftware.Add(new SoftwareInfo

{

Id = software.Id,

Software = software.Name,

SoftType = software.SoftType.Name,

Cost = software.Cost

});

}

UpdateServerPartsInfo();

}

#region Bindings

/// <summary>

/// Инициализация привязки платформ

/// </summary>

private void BindPlatforms()

{

paasTypeBindingSource.DataSource = \_ctx.GetPaasTypes();

comboBox\_PaasTypes.DataSource = paasTypeBindingSource;

}

/// <summary>

/// Инициализация привязки типов сервиса

/// </summary>

private void BindServiceTypes()

{

serviceTypeBindingSource.DataSource = \_ctx.GetServiceTypes();

comboBox\_ServiceTypes.DataSource = serviceTypeBindingSource;

}

/// <summary>

/// Инициализация привязки информации о стоимости компонентов сервиса

/// </summary>

private void BindServerPartsInfo()

{

ServicePartsInfo = new ServicePartsInfo();

servicePartsInfoBindingSource.DataSource = ServicePartsInfo;

}

/// <summary>

/// Обновлениепривязкиинформацииостоимостикомпонентовсервиса

/// </summary>

private void UpdateServerPartsInfo()

{

servicePartsInfoBindingSource.DataSource = ServicePartsInfo;

servicePartsInfoBindingSource.ResetBindings(true);

CurrentService.CostPerHour = ServicePartsInfo.FinalPrice;

serviceBindingSource.DataSource = CurrentService;

serviceBindingSource.ResetBindings(false);

}

/// <summary>

/// Привязка списка выбранного программного обеспечения

/// </summary>

private void BindSelectedSoftware()

{

SelectedSoftware = new List<SoftwareInfo>();

softwareInfoBindingSource.DataSource = new BindingList<SoftwareInfo>(SelectedSoftware);

dg\_Software.DataSource = softwareInfoBindingSource;

dg\_Software.Refresh();

}

/// <summary>

/// Обновление привязки списка выбранного программного обеспечения

/// </summary>

private void UpdateSelectedSoftware()

{

softwareInfoBindingSource.DataSource = new BindingList<SoftwareInfo>(SelectedSoftware);

dg\_Software.DataSource = softwareInfoBindingSource;

dg\_Software.Refresh();

}

#endregion

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных типов платформ

/// </summary>

private void paasTypeBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var currentPaasType = paasTypeBindingSource.Current as PaasType;

if (currentPaasType != null)

{

platformServerItemBindingSource.DataSource = \_ctx.GetServersByPlatform(currentPaasType);

comboBox\_Servers.DataSource = platformServerItemBindingSource;

platfromSANItemBindingSource.DataSource = \_ctx.GetSANsByPlatform(currentPaasType);

comboBox\_sans.DataSource = platfromSANItemBindingSource;

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных серверов

/// </summary>

private void platformServerItemBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var currentServerItem = platformServerItemBindingSource.Current as PlatformServerItem;

if (currentServerItem != null)

{

var server = \_ctx.GetServerDetailsByID(currentServerItem.ID);

var serverPlarformInfo = \_ctx.GetServerPlatformInfoByServer(server);

serverPlarformInfoBindingSource.DataSource = serverPlarformInfo;

serverPlarformInfoBindingSource.ResetBindings(true);

ServicePartsInfo.Server = server;

UpdateServerPartsInfo();

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных хранилищ данных

/// </summary>

private void platfromSANItemBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var currentSANItem = platfromSANItemBindingSource.Current as PlatfromSANItem;

if (currentSANItem != null)

{

var san = \_ctx.GetSANDetailsByID(currentSANItem.ID);

var sanPlatformInfo = \_ctx.GetSanPlatformInfoBySAN(san);

sANPlatformInfoBindingSource.DataSource = sanPlatformInfo;

sANPlatformInfoBindingSource.ResetBindings(true);

ServicePartsInfo.San = san;

UpdateServerPartsInfo();

}

}

/// <summary>

/// Обработчик события изменения значения в numericUpDown,

/// который отвечает за кол-во используемых ядер

/// </summary>

private void numericUpDown1\_ValueChanged(object sender, EventArgs e)

{

double frequency = (double)numericUpDown1.Value \* (ServicePartsInfo.CPUFrequency / ServicePartsInfo.CPUCount);

ServicePartsInfo.Frequency = frequency;

textBox\_Frequency.Text = frequency.ToString("##.00");

UpdateServerPartsInfo();

}

/// <summary>

/// Обработчиксобытияизменениязначенияв numericUpDown,

/// который отвечает за кол-во используемой оперативной памяти

/// </summary>

private void numericUpDown2\_ValueChanged(object sender, EventArgs e)

{

ServicePartsInfo.RAMVolume = (double)numericUpDown2.Value;

UpdateServerPartsInfo();

}

/// <summary>

/// Обработчиксобытияизменениязначенияв numericUpDown,

/// который отвечает за кол-во используемый объем жесткого диска

/// </summary>

private void numericUpDown3\_ValueChanged(object sender, EventArgs e)

{

ServicePartsInfo.StorageVolume = (double)numericUpDown3.Value;

UpdateServerPartsInfo();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска типа сервиса

/// </summary>

private void pictureBox3\_Click(object sender, EventArgs e)

{

var findServiceType = new EditServiceTypeForm(true);

if (findServiceType.ShowDialog() == DialogResult.OK)

{

if (findServiceType.Edited)

BindServiceTypes();

serviceTypeBindingSource.Position =

((BindingList<ServiceType>)serviceTypeBindingSource.DataSource).IndexOf(findServiceType.CurrentServiceType);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно редактирования типов сервисов

/// </summary>

private void pictureBox5\_Click(object sender, EventArgs e)

{

var editServiceType = new EditServiceTypeForm();

if (editServiceType.ShowDialog() == DialogResult.OK)

{

var selectedServiceType = serviceTypeBindingSource.Current as ServiceType;

BindServiceTypes();

if (selectedServiceType != null)

{

int pos = serviceTypeBindingSource.IndexOf(selectedServiceType);

if (pos > -1)

serviceTypeBindingSource.Position = pos;

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит отмену выбранного сервиса

/// </summary>

private void pictureBox2\_Click(object sender, EventArgs e)

{

comboBox\_ServiceTypes.SelectedIndex = -1;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных типов сервиса

/// </summary>

private void serviceTypeBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (comboBox\_ServiceTypes.SelectedIndex == -1)

((BindingSource)sender).SuspendBinding();

else

((BindingSource)sender).ResumeBinding();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно добавления программного обеспечения

/// </summary>

private void picBtn\_AddSoftware\_Click(object sender, EventArgs e)

{

var selectedIDs = SelectedSoftware.Select(s => s.Id).ToList();

var selectSoftwareForm = new SelectSoftwareForm(selectedIDs);

if (selectSoftwareForm.ShowDialog() == DialogResult.OK)

{

try

{

SelectedSoftware.Add(selectSoftwareForm.CurrentSoftware);

UpdateSelectedSoftware();

ServicePartsInfo.SoftwareInfo = SelectedSoftware;

UpdateServerPartsInfo();

}

catch

{

MessageBox.Show("Произошла ошибка при добавлении выбранного программного обеспечения!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление выбранного программного обеспечения

/// </summary>

private void picBtn\_DeleteSoftware\_Click(object sender, EventArgs e)

{

var selectedSoftware = softwareInfoBindingSource.Current as SoftwareInfo;

if (selectedSoftware == null)

{

MessageBox.Show("Программноеобеспечениеневыбрано!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

SelectedSoftware.Remove(selectedSoftware);

UpdateSelectedSoftware();

UpdateServerPartsInfo();

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который выполняет сохранение изменений

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

var currentSANItem = platfromSANItemBindingSource.Current as PlatfromSANItem;

var currentServerItem = platformServerItemBindingSource.Current as PlatformServerItem;

var currentPaasType = paasTypeBindingSource.Current as PaasType;

if (currentPaasType == null || comboBox\_PaasTypes.SelectedIndex < 0)

{

MessageBox.Show("Выберите платформу для сохранения изменений!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

if (comboBox\_ServiceTypes.SelectedIndex < 0)

{

MessageBox.Show("Выберитетипсервисадлясохраненияизменений!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

if (currentServerItem == null)

{

MessageBox.Show("Выберите сервер для сохранения изменений!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

if (currentSANItem == null)

{

MessageBox.Show("Выберите хранилище данных для сохранения изменений!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

if (CurrentService.HDDVolume <= 0 || CurrentService.CoreCount <= 0 || CurrentService.RamCount <= 0)

{

MessageBox.Show("Введитекорректныезначенияхарактеристиксервиса!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

if (SelectedSoftware.Count == 0)

{

MessageBox.Show("Выберитепрограммноеобеспечение!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

if (ServicePartsInfo.CPUCostByHour <= 0 || ServicePartsInfo.RAMCostByHour <= 0 ||

ServicePartsInfo.SanStorageCostByHour <= 0 || ServicePartsInfo.FinalPrice <= 0)

{

MessageBox.Show("Невозможносохранитьизменения, таккакпроизошлаошибкаприрассчетестоимости!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

try

{

CurrentService.Software.Clear();

foreach (var softwareInfo in SelectedSoftware)

CurrentService.Software.Add(await \_ctx.GetSoftwareById(softwareInfo.Id));

switch (\_formMode)

{

case FormMode.Add:

await \_ctx.AddNewService(CurrentService);

MessageBox.Show("Новыйсервисуспешносоздан!", "Информация", MessageBoxButtons.OK, MessageBoxIcon.Information);

break;

case FormMode.Edit:

await \_ctx.EditService(CurrentService);

MessageBox.Show("Изменениявсервисеуспешносохранены!", "Информация", MessageBoxButtons.OK, MessageBoxIcon.Information);

break;

}

\_formMode = FormMode.None;

Close();

}

catch

{

MessageBox.Show("Неудалосьсохранитьизменения!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит отмену добавления/редактирования, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditServiceForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.No)

{

e.Cancel = true;

return;

}

}

Owner?.Show();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление выбранного типа платформы

/// </summary>

private void pictureBox1\_Click(object sender, EventArgs e)

{

comboBox\_PaasTypes.SelectedIndex = -1;

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных платформ сервера

/// </summary>

private void serverPlarformInfoBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var s = serverPlarformInfoBindingSource.Current as ServerPlarformInfo;

if (s != null)

{

numericUpDown1.Maximum = s.AvalibleCoreCount;

numericUpDown2.Maximum = (decimal)s.AvalibleRAMVolume;

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных хранилищ данных

/// </summary>

private void sANPlatformInfoBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var s = sANPlatformInfoBindingSource.Current as SANPlatformInfo;

if (s != null)

numericUpDown3.Maximum = (decimal)s.AvalibleVolume;

}

/// <summary>

/// Обработчик события нажатий клавиш клавиатуры на форме

/// </summary>

private void EditServiceForm\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.Escape:

Close();

break;

case Keys.Enter:

btn\_Save\_Click(null, EventArgs.Empty);

break;

}

}

}

}

Модуль «ListOfServicesForm.cs»

using System;

using System.ComponentModel;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Services

{

/// <summary>

/// Класс формы списка сервисов

/// </summary>

public partial class ListOfServicesForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Текущаяплатформа

/// </summary>

public PlatformServiceUser SelectedPlatform { get; set; }

/// <summary>

/// Текущийсервис

/// </summary>

public ServiceDetailInfo SelectedService { get; set; }

/// <summary>

/// Список детального описания сервисов

/// </summary>

private BindingList<ServiceDetailInfo> ServicesDetailInfoMain { get; set; }

/// <summary>

/// Отобранный список детального описания сервисов

/// </summary>

private BindingList<ServiceDetailInfo> ServicesDetailInfoToShow { get; set; }

/// <summary>

/// Конструктор формы списка сервисов

/// </summary>

public ListOfServicesForm()

{

InitializeComponent();

SubscribeMenuItems();

созданиеНовойПлатформыToolStripMenuItem.Click += picBtn\_AddNewPlatform\_Click;

редактированиеВыбраннойПлатформыToolStripMenuItem.Click += picBtn\_EditSelectedPlatform\_Click;

удалениеВыбраннойПлатформыToolStripMenuItem.Click += picBtn\_DeleteSelectedPlatform\_Click;

созданиеНовогоСервисаToolStripMenuItem.Click += picBtn\_AddNewService\_Click;

редактированиеВыбранногоСервисаToolStripMenuItem.Click += picBtn\_EditSelectedService\_Click;

удалениеВыбранногоСервисаToolStripMenuItem.Click += picBtn\_DeleteSelectedService\_Click;

tsmi\_AddNewService.Click += picBtn\_AddNewService\_Click;

tsmi\_EditSelectedService.Click += picBtn\_EditSelectedService\_Click;

tsmi\_DeleteSelectedService.Click += picBtn\_DeleteSelectedService\_Click;

tsmi\_ShowAllServices.Click += ttmi\_ShowAllServices\_Click;

tsmi\_AddNewPlatfrom.Click += picBtn\_AddNewPlatform\_Click;

tsmi\_EditSelectedPlatform.Click += picBtn\_EditSelectedPlatform\_Click;

tsmi\_DeleteSelectedPlatform.Click += picBtn\_DeleteSelectedPlatform\_Click;

tsmi\_ShowServicesBySelectedPlatform.Click += ttmi\_ShowServicesBySelectedPlatform\_Click;

\_ctx = new Context();

}

/// <summary>

/// Обработчик события выхода курсора из элемента главного меню,

/// который очищает область в строке состояния.

/// </summary>

private void MenuItem\_MouseLeave(object sender, EventArgs e)

{

tip\_Label.Text = string.Empty;

}

/// <summary>

/// Обработчик события навдения курсора на элемент главного меню,

/// который выводит подсказку элемента в строку состояния.

/// </summary>

private void MenuItem\_MouseEnter(object sender, EventArgs e)

{

tip\_Label.Text = ((ToolStripItem)sender).ToolTipText;

}

/// <summary>

/// Подписка элементов главного меню и панели элементов на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

private void SubscribeMenuItems()

{

SubScribeChildMenuItems(menuBar.Items);

}

/// <summary>

/// Подписка дочерних элементов главного меню на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

/// <param name="col">Коллекция элементов главного меню</param>

private void SubScribeChildMenuItems(ToolStripItemCollection col)

{

for (int i = 0; i < col.Count; i++)

{

var item = col[i] as ToolStripMenuItem;

if (item != null)

{

item.MouseEnter += MenuItem\_MouseEnter;

item.MouseLeave += MenuItem\_MouseLeave;

if (item.DropDownItems.Count > 0)

SubScribeChildMenuItems(item.DropDownItems);

}

}

}

/// <summary>

/// Инициализация привязки списка платформ с информацией о количестве пользователей и сервисов

/// </summary>

private void BindPlatformList()

{

platformServiceUserBindingSource.DataSource = \_ctx.GetPlatformServiceUsers();

dg\_Platforms.DataSource = platformServiceUserBindingSource;

}

/// <summary>

/// Инициализация привязки полного списка конфигураций сервисов с детальным описанием

/// </summary>

private void BindServicesList()

{

ServicesDetailInfoMain = \_ctx.GetServicesDetailInfo();

serviceDetailInfoBindingSource.DataSource = ServicesDetailInfoMain;

dg\_Services.DataSource = serviceDetailInfoBindingSource;

UpdateStatBar();

}

/// <summary>

/// Обновлениестрокисостояния

/// </summary>

private void UpdateStatBar()

{

lbl\_PlatformsCount.Text = $"Кол-воплатформ: {dg\_Platforms.Rows.Count}";

lbl\_ServiceTotalCount.Text = $"Общеекол-воконфигурацийсервисов: {ServicesDetailInfoMain?.Count ?? 0}";

lbl\_ServiceSelectedCount.Text = $"Отобраноконфигураций: {dg\_Services.Rows.Count}";

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных списка платформ

/// </summary>

private void platformServiceUserBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as PlatformServiceUser;

if (selectedItem != null)

{

SelectedPlatform = selectedItem;

if (ServicesDetailInfoMain != null)

{

ServicesDetailInfoToShow = new BindingList<ServiceDetailInfo>(ServicesDetailInfoMain

.Where(sdi => sdi.PaasType.Equals(SelectedPlatform.Name)).ToList());

serviceDetailInfoBindingSource.DataSource = ServicesDetailInfoToShow;

dg\_Services.DataSource = serviceDetailInfoBindingSource;

dg\_Services.Columns[1].Visible = false;

UpdateStatBar();

}

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных списка сервисов

/// </summary>

private void serviceDetailInfoBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as ServiceDetailInfo;

if (selectedItem != null)

SelectedService = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который совершает переход на форму создания новой платформы

/// </summary>

private void picBtn\_AddNewPlatform\_Click(object sender, EventArgs e)

{

if (!\_ctx.CheckInfrastructureToCreatePlatform())

{

MessageBox.Show("Имеетсянеполныеданныеобвычислительнойинфраструктруре!" +

"\nПеред тем как добавить платформу заполните сведения об конфигурациях серверов и хранилищ данных!",

"Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

var editPlatformForm = new EditPlatformForm { Owner = this };

editPlatformForm.Show();

Hide();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который совершает переход на форму редактирования выбранной платформы

/// </summary>

private async void picBtn\_EditSelectedPlatform\_Click(object sender, EventArgs e)

{

if (SelectedPlatform == null)

{

MessageBox.Show("Выберитеплатформудляредактирования!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

PaasType platformToEdit = await \_ctx.GetPlatformByPlatformServiceUser(SelectedPlatform);

var editPlatformForm = new EditPlatformForm(platformToEdit) { Owner = this };

editPlatformForm.Show();

Hide();

}

catch

{

MessageBox.Show("Неудалосьполучитьданныеповыбраннойплатформе!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление выбранной платформы

/// </summary>

private async void picBtn\_DeleteSelectedPlatform\_Click(object sender, EventArgs e)

{

if (SelectedPlatform == null)

{

MessageBox.Show("Выберитеплатформудляудаления!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

var result = MessageBox.Show("Выбранная платформа и все ее сервисы будут удалены! Продолжить?",

"Удаление платформы", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.Yes)

{

try

{

PaasType platformToDel = await \_ctx.GetPlatformByPlatformServiceUser(SelectedPlatform);

await \_ctx.DeletePaasType(platformToDel);

BindPlatformList();

BindServicesList();

MessageBox.Show("Платформаивсеееданныеуспешноудалены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

catch

{

MessageBox.Show("Неудалосьудалитьвыбраннуюплатформу!", "Ошибкаприудалении",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который совершает переход на форму создания нового сервиса

/// </summary>

private async void picBtn\_AddNewService\_Click(object sender, EventArgs e)

{

if (\_ctx.CheckActivePaas() == 0)

{

MessageBox.Show("Отсутсвуют платформы, на которые были распределены вычислительные ресурсы!", "Ошибка",

MessageBoxButtons.OK);

return;

}

EditServiceForm editServiceForm = null;

if (SelectedPlatform != null)

{

try

{

PaasType selPlatf = await \_ctx.GetPlatformByPlatformServiceUser(SelectedPlatform);

if (selPlatf.Server.Count > 0 && selPlatf.SAN.Count > 0)

editServiceForm = new EditServiceForm(selPlatf);

else

{

var result = MessageBox.Show(

"У выбранной платформы отсутствуют привязанные конфигурации серверов и/или хранилища данных! Открыть редактор сервиса?", "Предупреждение",

MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.Yes)

editServiceForm = new EditServiceForm();

else

return;

}

}

catch

{

var result = MessageBox.Show(

"Произошла ошибка при получении типа выбранной платформы! Открыть редактор сервиса?", "Ошибка",

MessageBoxButtons.YesNo, MessageBoxIcon.Error);

if (result == DialogResult.Yes)

editServiceForm = new EditServiceForm();

else

return;

}

}

else

editServiceForm = new EditServiceForm();

editServiceForm.Owner = this;

editServiceForm.Show();

Hide();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который совершает переход на форму редактирования выбранного сервиса

/// </summary>

private async void picBtn\_EditSelectedService\_Click(object sender, EventArgs e)

{

if (SelectedService == null)

{

MessageBox.Show("Выберитесервисдляредактирования!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

Service serviceToEdit = await \_ctx.GetServiceByServiceDetailInfo(SelectedService);

var editServiceForm = new EditServiceForm(serviceToEdit) { Owner = this };

editServiceForm.Show();

Hide();

}

catch

{

MessageBox.Show("Не удалось получить информацию по выбранному сервису!", "Ошибка при удалении",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит удаление выбранного сервиса

/// </summary>

private async void picBtn\_DeleteSelectedService\_Click(object sender, EventArgs e)

{

if (SelectedService == null)

{

MessageBox.Show("Выберитесервисдляудаления!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

var result = MessageBox.Show("Выбраннаясерсвисбудетудален! Продолжить?",

"Удалениесервиса", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (result == DialogResult.Yes)

{

try

{

Service serviceToDel = await \_ctx.GetServiceByServiceDetailInfo(SelectedService);

await \_ctx.DeleteService(serviceToDel);

BindPlatformList();

BindServicesList();

MessageBox.Show("Сервисуспешноудален!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

catch

{

MessageBox.Show("Неудалосьудалитьвыбранныйсервис!", "Ошибкаприудалении",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который производит просмотр всех конфигураций сервисов

/// </summary>

private void ttmi\_ShowAllServices\_Click(object sender, EventArgs e)

{

BindServicesList();

dg\_Services.Columns[1].Visible = true;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который производит отбор сервисов по выбранной платформе

/// </summary>

private void ttmi\_ShowServicesBySelectedPlatform\_Click(object sender, EventArgs e)

{

var selectedItem = platformServiceUserBindingSource.Current as PlatformServiceUser;

if (selectedItem != null)

{

SelectedPlatform = selectedItem;

if (ServicesDetailInfoMain != null)

{

ServicesDetailInfoToShow = new BindingList<ServiceDetailInfo>(ServicesDetailInfoMain

.Where(sdi => sdi.PaasType.Equals(SelectedPlatform.Name)).ToList());

serviceDetailInfoBindingSource.DataSource = ServicesDetailInfoToShow;

dg\_Services.DataSource = serviceDetailInfoBindingSource;

dg\_Services.Columns[1].Visible = false;

UpdateStatBar();

}

}

else

MessageBox.Show("Выберитеплатформуизсписка!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчиксобытияактивацииформы

/// </summary>

private void ListOfServicesForm\_Activated(object sender, EventArgs e)

{

BindPlatformList();

BindServicesList();

}

/// <summary>

/// Обработчиксобытиязакрытияформы

/// </summary>

private void ListOfServicesForm\_FormClosing(object sender, FormClosingEventArgs e)

{

Owner?.Show();

}

/// <summary>

/// Обработчик события нажатий клавиш клавиатуры на форме

/// </summary>

private void ListOfServicesForm\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Escape)

Close();

}

}

}

Модуль «EditUsersForm.cs»

using System;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Users

{

/// <summary>

/// Класс формы редактирования пользователей

/// </summary>

public partial class EditUsersForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Режим работы формы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Логин пользователя перед изменением

/// </summary>

private string \_loginBeforeEdit;

/// <summary>

/// Флаг активации привязки данных после ее приоставновке в источнике данных должностей

/// </summary>

private bool \_afterSuspend;

/// <summary>

/// Экземпляртекущегопользователя

/// </summary>

public User CurrentUser { get; set; }

/// <summary>

/// Конструктор формы регистрации нового польователя

/// </summary>

public EditUsersForm()

{

InitializeComponent();

btn\_Cancel.Image = new Bitmap(Resources.no, 20, 20);

btn\_Save.Image = new Bitmap(Resources.camera\_test, 20, 20);

\_ctx = new Context();

InitBindings();

CurrentUser = new User();

userBindingSource.DataSource = CurrentUser;

\_formMode = FormMode.Add;

pictureBox1\_Click(null, EventArgs.Empty);

}

/// <summary>

/// Конструктор формы редактирования учетных данных пользователя

/// </summary>

/// <param name="selectedUser">Выбранныйпользователь</param>

public EditUsersForm(User selectedUser) : this()

{

CurrentUser = selectedUser;

userBindingSource.DataSource = CurrentUser;

\_loginBeforeEdit = CurrentUser.Login;

userBindingSource.ResumeBinding();

Text = "Редактирование учетных данных пользователя";

\_formMode = FormMode.Edit;

}

/// <summary>

/// Инициализация привязок

/// </summary>

private void InitBindings()

{

rightsLevelBindingSource.DataSource = \_ctx.GetRightsLevels();

BindPositions();

}

/// <summary>

/// Привязкадолжностейсотрудника

/// </summary>

private void BindPositions()

{

positionBindingSource.DataSource = \_ctx.GetPositions();

comboBox2.DataSource = positionBindingSource;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаграфическийобъект,

/// который открывает диалоговое окно редактирования должностей сотрудников

/// </summary>

private void pictureBox3\_Click(object sender, EventArgs e)

{

var editUsrPositionForm = new EditUsrPositionForm();

if (editUsrPositionForm.ShowDialog() == DialogResult.OK)

{

var selectedPosition = positionBindingSource.Current as Position;

BindPositions();

if (selectedPosition != null)

{

int pos = positionBindingSource.IndexOf(selectedPosition);

if (pos > -1)

positionBindingSource.Position = pos;

}

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который открывает диалоговое окно поиска должности сотрудника

/// </summary>

private void pictureBox2\_Click(object sender, EventArgs e)

{

var editUsrPositionForm = new EditUsrPositionForm(true);

if (editUsrPositionForm.ShowDialog() == DialogResult.OK)

{

if (editUsrPositionForm.Edited)

BindPositions();

positionBindingSource\_CurrentChanged(positionBindingSource, EventArgs.Empty);

positionBindingSource.Position = positionBindingSource.IndexOf(editUsrPositionForm.CurrentPosition);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит отмену последнего добавления/редактирования, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который выполняет сохранение изменений

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

try

{

var selectedRightsLevel = rightsLevelBindingSource.Position > -1 ? rightsLevelBindingSource.Current as RightsLevel : null;

var selectedPosition = positionBindingSource.Position > -1 ? positionBindingSource.Current as Position : null;

if (string.IsNullOrWhiteSpace(CurrentUser.Login) || string.IsNullOrWhiteSpace(CurrentUser.Password)

|| string.IsNullOrWhiteSpace(txt\_ConfirmPass.Text) || selectedRightsLevel == null || selectedPosition == null)

{

MessageBox.Show("Введите все поля для сохранения изменений!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

if (!txt\_ConfirmPass.Text.Equals(CurrentUser.Password))

{

MessageBox.Show("Пароли не совпадают!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

bool existed = false;

switch (\_formMode)

{

case FormMode.Add:

existed = await \_ctx.CheckUserLoginForDublicate(CurrentUser.Login);

break;

case FormMode.Edit:

existed = await \_ctx.CheckUserLoginForDublicate(CurrentUser.Login) && !CurrentUser.Login.Equals(\_loginBeforeEdit);

break;

}

if (existed)

{

MessageBox.Show("Пользователь с таким логином уже существует!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

switch (\_formMode)

{

case FormMode.Add:

await \_ctx.AddNewUser(CurrentUser);

break;

case FormMode.Edit:

await \_ctx.EditUser(CurrentUser);

break;

}

\_formMode = FormMode.None;

DialogResult = DialogResult.OK;

MessageBox.Show("Изменения успешно сохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

Close();

}

catch

{

MessageBox.Show("Изменения не удалось сохранить!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditUsersForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных уровней привилегий

/// </summary>

private void rightsLevelBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (comboBox1.SelectedIndex == -1)

{

rightsLevelBindingSource.SuspendBinding();

if (CurrentUser != null)

CurrentUser.RightLevelId = 0;

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных должностей

/// </summary>

private void positionBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (positionBindingSource.IsBindingSuspended)

{

\_afterSuspend = true;

positionBindingSource.ResumeBinding();

}

else if (\_afterSuspend)

{

\_afterSuspend = false;

positionBindingSource.Position = comboBox2.SelectedIndex;

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит отмену выбранной должности

/// </summary>

private void pictureBox1\_Click(object sender, EventArgs e)

{

comboBox2.SelectedIndex = -1;

positionBindingSource.SuspendBinding();

if (CurrentUser != null)

CurrentUser.PositionId = 0;

textBox4.Text = string.Empty;

textBox5.Text = string.Empty;

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке должностей

/// </summary>

private void comboBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox2.SelectedIndex > -1)

positionBindingSource\_CurrentChanged(positionBindingSource, EventArgs.Empty);

}

/// <summary>

/// Обработчик события нажатия клавиши в текстовом поле с Логином,

/// который ограничивает ввод только латиницы, цифр, символа нижнего подчеркивания и управляющих символов

/// </summary>

private void textBox1\_KeyPress(object sender, KeyPressEventArgs e)

{

var l = e.KeyChar;

if (!

((l >= 'A' && l <= 'z')

|| (l >= '0' && l <= '9')

|| l == '\_'

|| l == '\b'

|| char.IsControl(l)))

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатия клавиши в текстовом поле с ФИО,

/// который ограничивает ввод только русских букв, пробела и управляющих символов

/// </summary>

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)

{

var l = e.KeyChar;

if (!

((l >= 'А' && l <= 'я')

|| l == (int)Keys.Space

|| l == '\b'

|| char.IsControl(l)))

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатий клавиш клавиатуры на форме

/// </summary>

private void EditUsersForm\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.Escape:

Close();

break;

case Keys.Enter:

btn\_Save\_Click(null, EventArgs.Empty);

break;

}

}

}

}

Модуль: «EditUsrPositionForm.cs»

using System;

using System.ComponentModel;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Users

{

/// <summary>

/// Класс формы редактирования должностей сотрудников

/// </summary>

public partial class EditUsrPositionForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Режим работы формы.

/// </summary>

private FormMode \_formMode;

/// <summary>

/// Списокдолжностей

/// </summary>

public BindingList<Position> Positions { get; set; }

/// <summary>

/// Текущаядолжность

/// </summary>

public Position CurrentPosition { get; set; }

/// <summary>

/// Наименование должности до изменения

/// </summary>

private string \_positionNameBeforeEditing;

/// <summary>

/// Флагредактированиядолжностей

/// </summary>

public bool Edited { get; set; }

/// <summary>

/// Конструктор формы редактирования должностей сотрудников

/// </summary>

public EditUsrPositionForm()

{

InitializeComponent();

EnDisFields(false);

\_ctx = new Context();

BindCollection();

\_formMode = FormMode.None;

}

/// <summary>

/// Конструктор формы для выбора должности из списка

/// </summary>

/// <param name="findMode"></param>

public EditUsrPositionForm(bool findMode) : this()

{

if (!findMode)

throw new ArgumentException("Неверноезначениефлага!");

but\_Select.Visible = findMode;

\_formMode = FormMode.None;

but\_Select.Image = new Bitmap(Resources.camera\_test, new Size(20, 20));

}

/// <summary>

/// Привязкаколлекции

/// </summary>

private void BindCollection()

{

Positions = \_ctx.GetPositions();

positionBindingSource.DataSource = Positions;

dataGridView1.DataSource = positionBindingSource;

}

/// <summary>

/// Активация/деактивацияполейдляредактирования

/// </summary>

/// <param name="enabled">Флагактивацииполейредактирования</param>

private void EnDisFields(bool enabled)

{

textBox1.Enabled = enabled;

textBox2.Enabled = enabled;

textBox3.Enabled = enabled;

btn\_Cancel.Enabled = enabled;

btn\_Save.Enabled = enabled;

dataGridView1.Enabled = !enabled;

button2.Enabled = !enabled;

button3.Enabled = !enabled;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит добавление новой должности

/// </summary>

private void button2\_Click(object sender, EventArgs e)

{

Positions.AddNew();

positionBindingSource.MoveLast();

EnDisFields(true);

\_formMode = FormMode.Add;

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинакнопку,

/// который производит редактирование выбранной должности

/// </summary>

private void button3\_Click(object sender, EventArgs e)

{

if (CurrentPosition != null)

{

EnDisFields(true);

\_positionNameBeforeEditing = CurrentPosition.Name;

\_formMode = FormMode.Edit;

}

else

MessageBox.Show("Выберите должность из списка или добавьте новую!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который выполняет сохранение изменений

/// </summary>

private async void btn\_Save\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

try

{

if (string.IsNullOrWhiteSpace(CurrentPosition.Name) || CurrentPosition.AvgSalary <= 0

|| CurrentPosition.WorkHours <= 0)

{

MessageBox.Show("Заполнитевсеполядлясохраненияизменений!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

bool existed = false;

switch (\_formMode)

{

case FormMode.Add:

existed = await \_ctx.CheckPositionForDublicate(CurrentPosition.Name);

break;

case FormMode.Edit:

existed = await \_ctx.CheckPositionForDublicate(CurrentPosition.Name) && !CurrentPosition.Name.Equals(\_positionNameBeforeEditing);

break;

}

if (existed)

{

MessageBox.Show("Должность с таким наименованием уже существует!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (\_formMode == FormMode.Add)

Positions.EndNew(Positions.IndexOf(CurrentPosition));

await \_ctx.SaveChangesAsync();

EnDisFields(false);

\_formMode = FormMode.None;

Edited = true;

MessageBox.Show("Изменения успешно сохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

catch

{

MessageBox.Show("Изменения не удалось сохранить!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

/// <summary>

/// Обработчик события закрытия формы

/// </summary>

private void EditUsrPositionForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.No)

e.Cancel = true;

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных должностей

/// </summary>

private void positionBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as Position;

if (selectedItem != null)

CurrentPosition = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который подтверждает выбранный элемент из списка должностей

/// </summary>

private void but\_Select\_Click(object sender, EventArgs e)

{

if (CurrentPosition != null)

{

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Выберитедолжностьизсписка!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события нажатия клавиши в текстовом поле с наименованием должности,

/// который ограничивает ввод только русских букв, дефиса, пробела и управляющих символов

/// </summary>

private void textBox1\_KeyPress(object sender, KeyPressEventArgs e)

{

var l = e.KeyChar;

if (!

((l >= 'А' && l <= 'я')

|| l == (int)Keys.Space

|| l == '-'

|| l == '\b'

|| char.IsControl(l)))

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатия клавиши в текстовом поле с кол-вом рабочих часов,

/// который ограничивает ввод только цифр и управляющих символов

/// </summary>

private void textBox3\_KeyPress(object sender, KeyPressEventArgs e)

{

var l = e.KeyChar;

if (!(char.IsDigit(l) || char.IsControl(l) || l == '\b'))

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатия клавиши в текстовом поле с размером средней зарплаты,

/// который ограничивает ввод только цифр, управляющих символов, и 1 запятой

/// </summary>

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)

{

var txtBox = (TextBox)sender;

if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))

{

if (txtBox.Text.Length == 10 && txtBox.Text.Length <= 10)

e.Handled = true;

return;

}

// Точкузаменимзапятой

if (e.KeyChar == '.')

e.KeyChar = ',';

if (e.KeyChar == ',')

{

// Не более одной запятой и запятая не может быть первым символом.

if ((txtBox.Text.IndexOf(',') != -1) || (txtBox.Text.Length == 0))

e.Handled = true;

return;

}

if (Char.IsControl(e.KeyChar))

if (e.KeyChar == (char) Keys.Back)

return;

e.Handled = true;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену последнего добавления/редактирования, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

if (\_formMode != FormMode.None)

{

var result = MessageBox.Show("Изменениянебудутсохранены! Продолжить?", "Отменаизменений",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

try

{

if (\_formMode == FormMode.Edit)

{

int indexOfElement = Positions.IndexOf(CurrentPosition);

CurrentPosition = \_ctx.CancelChanges(CurrentPosition);

Positions[indexOfElement] = CurrentPosition;

}

else

Positions.CancelNew(Positions.IndexOf(CurrentPosition));

EnDisFields(false);

\_formMode = FormMode.None;

}

catch

{

MessageBox.Show("Произошлаошибкаприотменеизменений!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

/// <summary>

/// Обработчиксобытиянажатийклавишклавиатурынаформе

/// </summary>

private void EditUsrPositionForm\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Escape)

Close();

}

}

}

Модуль «UserConditionForm.cs»

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Properties;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Users

{

/// <summary>

/// Класс формы отбора пользователей

/// </summary>

public partial class UserConditionForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Флаг активации привязки данных после ее приоставновки в источнике данных должностей

/// </summary>

private bool \_afterPositionSuspend;

/// <summary>

/// Флаг активации привязки данных после ее приоставновки в источнике данных уровней привилегий

/// </summary>

private bool \_afterRightsLevelSuspend;

/// <summary>

/// Выбраннаядолжность

/// </summary>

public Position SelectedPosition { get; set; }

/// <summary>

/// Выбранный уровень привелегий

/// </summary>

public RightsLevel SelectedRightsLevel { get; set; }

/// <summary>

/// Списокпользователей

/// </summary>

public BindingList<UserInfo> UsersInfoMain { get; set; }

/// <summary>

/// Отобранный список пользователей

/// </summary>

public BindingList<UserInfo> UsersInfoToShow { get; set; }

/// <summary>

/// Конструктор формы отбора пользователей

/// </summary>

public UserConditionForm(BindingList<UserInfo> usersInfoMain)

{

InitializeComponent();

btn\_Cancel.Image = new Bitmap(Resources.no, 20, 20);

btn\_OK.Image = new Bitmap(Resources.camera\_test, 20, 20);

\_ctx = new Context();

UsersInfoMain = usersInfoMain;

InitBindings();

pictureBox1\_Click(null,EventArgs.Empty);

pictureBox2\_Click(null,EventArgs.Empty);

}

/// <summary>

/// Инициализацияпривязок

/// </summary>

private void InitBindings()

{

positionBindingSource.DataSource = \_ctx.GetPositions();

rightsLevelBindingSource.DataSource = \_ctx.GetRightsLevels();

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных уровней привилегий

/// </summary>

private void rightsLevelBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (rightsLevelBindingSource.IsBindingSuspended)

{

\_afterRightsLevelSuspend = true;

rightsLevelBindingSource.ResumeBinding();

}

else if (\_afterRightsLevelSuspend)

{

\_afterRightsLevelSuspend = false;

rightsLevelBindingSource.Position = comboBox1.SelectedIndex;

}

if (rightsLevelBindingSource.Position > -1)

{

var rightsLevel = rightsLevelBindingSource.Current as RightsLevel;

if (rightsLevel != null)

{

SelectedRightsLevel = rightsLevel;

lbl\_SelectedUsersStatus.Text = $"Отобранопользователей: {RunSelection()}";

}

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных должностей

/// </summary>

private void positionBindingSource\_CurrentChanged(object sender, EventArgs e)

{

if (positionBindingSource.IsBindingSuspended)

{

\_afterPositionSuspend = true;

positionBindingSource.ResumeBinding();

}

else if (\_afterPositionSuspend)

{

\_afterPositionSuspend = false;

positionBindingSource.Position = comboBox2.SelectedIndex;

}

if (positionBindingSource.Position > -1)

{

var position = positionBindingSource.Current as Position;

if (position != null)

{

SelectedPosition = position;

lbl\_SelectedUsersStatus.Text = $"Отобранопользователей: {RunSelection()}";

}

}

}

/// <summary>

/// Выполнение отбора пользователей по условию

/// </summary>

private int RunSelection()

{

List<UserInfo> selectedUsers = null;

try

{

if (SelectedRightsLevel != null && SelectedPosition != null)

selectedUsers = UsersInfoMain.Where(ui =>

ui.Position.Equals(SelectedPosition.Name) && ui.RightsLevel.Equals(SelectedRightsLevel.Name))

.ToList();

else if (SelectedRightsLevel != null)

selectedUsers = UsersInfoMain.Where(ui => ui.RightsLevel.Equals(SelectedRightsLevel.Name)).ToList();

else if (SelectedPosition != null)

selectedUsers = UsersInfoMain.Where(ui => ui.Position.Equals(SelectedPosition.Name)).ToList();

else

selectedUsers = UsersInfoMain.ToList();

UsersInfoToShow = new BindingList<UserInfo>(selectedUsers);

return UsersInfoToShow.Count;

}

catch

{

UsersInfoToShow = null;

return 0;

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит отмену выбора в списке уровней привилегий

/// </summary>

private void pictureBox1\_Click(object sender, EventArgs e)

{

comboBox1.SelectedIndex = -1;

SelectedRightsLevel = null;

rightsLevelBindingSource.SuspendBinding();

lbl\_SelectedUsersStatus.Text = $"Отобранопользователей: {RunSelection()}";

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на графический объект,

/// который производит отмену выбора в списке должностей

/// </summary>

private void pictureBox2\_Click(object sender, EventArgs e)

{

comboBox2.SelectedIndex = -1;

SelectedPosition = null;

positionBindingSource.SuspendBinding();

lbl\_SelectedUsersStatus.Text = $"Отобранопользователей: {RunSelection()}";

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит отмену ввода условия отбора пользователей, закрывая форму

/// </summary>

private void btn\_Cancel\_Click(object sender, EventArgs e)

{

Close();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на кнопку,

/// который производит подтверждение совершенного отбора пользователей

/// </summary>

private void btn\_OK\_Click(object sender, EventArgs e)

{

if (UsersInfoToShow != null && UsersInfoToShow.Count > 0)

{

DialogResult = DialogResult.OK;

Close();

}

else

MessageBox.Show("Ненайденыпользователипоуказанномуусловию!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке уровней привbлегий

/// </summary>

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox1.SelectedIndex > -1)

rightsLevelBindingSource\_CurrentChanged(rightsLevelBindingSource, EventArgs.Empty);

}

/// <summary>

/// Обработчик события изменения текущего индекса в списке должностей

/// </summary>

private void comboBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox2.SelectedIndex > -1)

positionBindingSource\_CurrentChanged(positionBindingSource, EventArgs.Empty);

}

}

}

Модуль «UserListForm.cs»

using System;

using System.ComponentModel;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

namespace MTF\_Services.WinForms.Forms.SysAdmin.Users

{

/// <summary>

/// Класс формы редактирования пользователей

/// </summary>

public partial class UserListForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Текущий пользователь

/// </summary>

public UserInfo CurrentUser { get; set; }

/// <summary>

/// Список пользователей

/// </summary>

public BindingList<UserInfo> UsersInfoMain { get; set; }

/// <summary>

/// Отобранный список пользователей

/// </summary>

public BindingList<UserInfo> UsersInfoToShow { get; set; }

/// <summary>

/// Конструктор формы редактирования пользователей

/// </summary>

publicUserListForm()

{

InitializeComponent();

SubscribeMenuItems();

добавитьToolStripMenuItem.Click += новыйПользовательToolStripMenuItem\_Click;

редактироватьToolStripMenuItem1.Click += редактироватьToolStripMenuItem\_Click;

удалитьToolStripMenuItem1.Click += удалитьToolStripMenuItem\_Click;

\_ctx = new Context();

InitBindings();

}

/// <summary>

/// Инициализация привязок данных

/// </summary>

private void InitBindings()

{

UsersInfoMain = \_ctx.GetUsersInfo();

UsersInfoToShow = null;

userInfoBindingSource.DataSource = UsersInfoMain;

dataGridView1.DataSource = userInfoBindingSource;

dataGridView1.Refresh();

UpdateStatBar();

}

/// <summary>

/// Обновление строки состояния

/// </summary>

private void UpdateStatBar()

{

lbl\_UserCount.Text = $"Общее кол-во пользователей: {UsersInfoMain?.Count ?? 0}";

lbl\_SelectedCount.Text = $"Отобрано записей: {dataGridView1.Rows.Count}";

}

/// <summary>

/// Обработчик события выхода курсора из элемента главного меню,

/// который очищает область в строке состояния.

/// </summary>

private void MenuItem\_MouseLeave(object sender, EventArgs e)

{

tip\_Label.Text = string.Empty;

}

/// <summary>

/// Обработчик события навдения курсора на элемент главного меню,

/// который выводит подсказку элемента в строку состояния.

/// </summary>

private void MenuItem\_MouseEnter(object sender, EventArgs e)

{

tip\_Label.Text = ((ToolStripItem)sender).ToolTipText;

}

/// <summary>

/// Подписка элементов главного меню и панели элементов на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

private void SubscribeMenuItems()

{

SubScribeChildMenuItems(menuBar.Items);

foreach (ToolStripItem toolBarControl in toolBar.Items)

{

toolBarControl.MouseEnter += MenuItem\_MouseEnter;

toolBarControl.MouseLeave += MenuItem\_MouseLeave;

}

}

/// <summary>

/// Подписка дочерних элементов главного меню на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

/// <paramname="col">Коллекция элементов главного меню</param>

private void SubScribeChildMenuItems(ToolStripItemCollection col)

{

for (int i = 0; i < col.Count; i++)

{

var item = col[i] as ToolStripMenuItem;

if (item != null)

{

item.MouseEnter += MenuItem\_MouseEnter;

item.MouseLeave += MenuItem\_MouseLeave;

if (item.DropDownItems.Count > 0)

SubScribeChildMenuItems(item.DropDownItems);

}

}

}

/// <summary>

/// Обработчик события изменения текущего элемента в источнике данных пользователей

/// </summary>

private void userInfoBindingSource\_CurrentChanged(object sender, EventArgs e)

{

var selectedItem = ((BindingSource)sender).Current as UserInfo;

if (selectedItem != null)

CurrentUser = selectedItem;

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы регистрации нового пользователя

/// </summary>

private void новыйПользовательToolStripMenuItem\_Click(object sender, EventArgs e)

{

var editUsersForm = new EditUsersForm();

if (editUsersForm.ShowDialog() == DialogResult.OK)

InitBindings();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно формы редактирования выбранного пользователя

/// </summary>

private async void редактироватьToolStripMenuItem\_Click(object sender, EventArgs e)

{

var selUsr = await \_ctx.GetUserByUserInfo(CurrentUser);

if (selUsr != null)

{

var editUsersForm = new EditUsersForm(selUsr);

if (editUsersForm.ShowDialog() == DialogResult.OK)

{

InitBindings();

if (CurrentUser != null)

{

int pos = userInfoBindingSource.IndexOf(CurrentUser);

if (pos > -1)

userInfoBindingSource.Position = pos;

}

}

}

else

MessageBox.Show("Выберите пользователя из списка для его редактирования!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который выполняет удаление выбранного пользователя

/// </summary>

private async void удалитьToolStripMenuItem\_Click(object sender, EventArgs e)

{

var selUsr = await \_ctx.GetUserByUserInfo(CurrentUser);

if (selUsr != null)

{

varresult = MessageBox.Show("Выбранный пользователь и вся информация о нем будет удалена! Продолжить?",

"Удаление пользователя", MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.No)

return;

try

{

await \_ctx.DeleteUser(selUsr);

InitBindings();

}

catch

{

MessageBox.Show("Произошла ошибка при удалении учетной записи!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

else

MessageBox.Show("Выберите пользователя из списка для его редактирования!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который выполняет просмотр всех зарегистрированных пользователей

/// </summary>

private void всеЗаписиToolStripMenuItem\_Click(object sender, EventArgs e)

{

InitBindings();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который открывает диалоговое окно задания критериев для отбора зарегистрированных пользователей

/// </summary>

private void отборToolStripMenuItem\_Click(object sender, EventArgs e)

{

var userConditionForm = new UserConditionForm(UsersInfoMain);

if (userConditionForm.ShowDialog() == DialogResult.OK)

{

UsersInfoToShow = userConditionForm.UsersInfoToShow;

userInfoBindingSource.DataSource = UsersInfoToShow;

dataGridView1.DataSource = userInfoBindingSource;

UpdateStatBar();

}

}

/// <summary>

/// Обработчик события нажатия клавиш клавиатуры на форме

/// </summary>

private void UserListForm\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Escape)

Close();

}

}

}

Модуль «SysAdminForm.cs»

using System;

using System.ComponentModel;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Forms.SysAdmin.EquipmentIdle;

using MTF\_Services.WinForms.Forms.SysAdmin.Infrastructure;

using MTF\_Services.WinForms.Forms.SysAdmin.Services;

using MTF\_Services.WinForms.Forms.SysAdmin.Users;

namespace MTF\_Services.WinForms.Forms.SysAdmin

{

/// <summary>

/// Класс главной формы системного администратора.

/// </summary>

public partial class SysAdminForm : Form

{

#region StatBar\_Labels

private const string USED\_PLATFORMS\_COUNT\_LBL = "Кол-воиспользуемыхплатформ:";

private const string USED\_SERVICE\_COUNT\_LBL = "Кол-вопредоставляемыхсервисов:";

private const string SERVICE\_CONFIG\_COUNT\_LBL = "Кол-воконфигурацийсервисов:";

private const string ZERO\_LBL = "0";

#endregion

/// <summary>

/// Флаг совершенного выхода из учетной записи

/// </summary>

private bool \_logout;

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Конструктор главной формы системного администратора.

/// </summary>

public SysAdminForm()

{

InitializeComponent();

SubscribeMenuItems();

FormClosing += SysAdminForm\_FormClosing;

\_ctx = new Context();

timer.Enabled = true;

}

/// <summary>

/// Обновлениепривязокданных.

/// </summary>

private async void UpdateBindings()

{

try

{

sanStateBS.DataSource = await \_ctx.GetSanEquipmentState();

serverStateBS.DataSource = await \_ctx.GetServersEquipmentState();

}

catch

{

sanStateBS.DataSource = new EquipmentState();

serverStateBS.DataSource = new EquipmentState();

}

try

{

var paasInfos = await \_ctx.GetPaasInfo();

paasInfoBindingSource.DataSource = paasInfos;

lbl\_usedPlatformsCount.Text = $"{USED\_PLATFORMS\_COUNT\_LBL} {\_ctx.UsedPlatformsCount(paasInfos)}";

var serviceInfos = \_ctx.GetServiceInfo();

serviceInfoBindingSource.DataSource = serviceInfos;

lbl\_usedServiceCount.Text = $"{USED\_SERVICE\_COUNT\_LBL} {\_ctx.UsedServiceCount(serviceInfos)}";

lbl\_serviceConfigCount.Text = $"{SERVICE\_CONFIG\_COUNT\_LBL} {\_ctx.ServiceConfigsCount(serviceInfos)}";

}

catch

{

lbl\_usedPlatformsCount.Text = $"{USED\_PLATFORMS\_COUNT\_LBL} {ZERO\_LBL}";

lbl\_usedServiceCount.Text = $"{USED\_SERVICE\_COUNT\_LBL} {ZERO\_LBL}";

lbl\_serviceConfigCount.Text = $"{SERVICE\_CONFIG\_COUNT\_LBL} {ZERO\_LBL}";

MessageBox.Show("Ошибка при получении данных!");

}

finally

{

UpdateLabels();

}

}

/// <summary>

/// Скрытие таблицы с информацией о сервисах.

/// </summary>

public void HideServiceGrid()

{

dataGridView3.Visible = false;

dataGridView3.Dock = DockStyle.None;

label9.Visible = true;

label9.Dock = DockStyle.Fill;

}

/// <summary>

/// Скрытие таблицы с информацией о платформах.

/// </summary>

private void HidePaasGrid()

{

dataGridView2.Visible = false;

dataGridView2.Dock = DockStyle.None;

label10.Visible = true;

label10.Dock = DockStyle.Fill;

}

/// <summary>

/// Обновление таблиц и меток, которые сообщают об отсутствии информации.

/// </summary>

private void UpdateLabels()

{

try

{

if (serviceInfoBindingSource.DataSource == null ||

(serviceInfoBindingSource.DataSource as BindingList<ServiceInfo>).Count == 0)

HideServiceGrid();

else

{

dataGridView3.Visible = true;

dataGridView3.Dock = DockStyle.Fill;

label9.Visible = false;

label9.Dock = DockStyle.None;

}

}

catch

{

HideServiceGrid();

}

try

{

if (paasInfoBindingSource.DataSource == null ||

(paasInfoBindingSource.DataSource as BindingList<PaasInfo>).Count == 0)

HidePaasGrid();

else

{

dataGridView2.Visible = true;

dataGridView2.Dock = DockStyle.Fill;

label10.Visible = false;

label10.Dock = DockStyle.None;

}

}

catch

{

HidePaasGrid();

}

}

/// <summary>

/// Обработчик события закрытия формы.

/// </summary>

private void SysAdminForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (e.CloseReason == CloseReason.ApplicationExitCall || \_logout)

return;

var result = MessageBox.Show("Выберитедальнейшеедействие:" +

"\n\tДа - сменить пользователя;" +

"\n\tНет - закрыть приложение;" +

"\n\tОтмена - остаться в текущем окне",

"Выход из основного меню",

MessageBoxButtons.YesNoCancel,

MessageBoxIcon.Asterisk,

MessageBoxDefaultButton.Button3);

switch (result)

{

case DialogResult.Cancel:

e.Cancel = true;

break;

case DialogResult.Yes:

\_ctx.Logout();

Owner?.Show();

break;

case DialogResult.No:

Application.Exit();

break;

}

}

/// <summary>

/// Обработчик события выхода курсора из элемента главного меню,

/// который очищает область в строке состояния.

/// </summary>

private void MenuItem\_MouseLeave(object sender, EventArgs e)

{

tip\_Label.Text = string.Empty;

}

/// <summary>

/// Обработчик события наведения курсора на элемент главного меню,

/// который выводит подсказку элемента в строку состояния.

/// </summary>

private void MenuItem\_MouseEnter(object sender, EventArgs e)

{

tip\_Label.Text = ((ToolStripItem)sender).ToolTipText;

}

/// <summary>

/// Подписка элементов главного меню и панели элементов на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

private void SubscribeMenuItems()

{

SubScribeChildMenuItems(menuBar.Items);

foreach (ToolStripItem toolBarControl in toolBar.Items)

{

toolBarControl.MouseEnter += MenuItem\_MouseEnter;

toolBarControl.MouseLeave += MenuItem\_MouseLeave;

}

}

/// <summary>

/// Подписка дочерних элементов главного меню на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

/// <param name="col">Коллекция элементов главного меню</param>

private void SubScribeChildMenuItems(ToolStripItemCollection col)

{

for (int i = 0; i < col.Count; i++)

{

var item = col[i] as ToolStripMenuItem;

if (item != null)

{

item.MouseEnter += MenuItem\_MouseEnter;

item.MouseLeave += MenuItem\_MouseLeave;

if (item.DropDownItems.Count > 0)

SubScribeChildMenuItems(item.DropDownItems);

}

}

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который открывает диалоговое окно формы просмотра и редактирования текущей учетной записи

/// </summary>

private void просмотрПрофиляToolStripMenuItem\_Click(object sender, EventArgs e)

{

new ShowUserProfileForm().ShowDialog();

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который выполняет переход на форму редактирования инфраструктуры

/// </summary>

private void редактированиеИнфраструктурыToolStripMenuItem\_Click(object sender, EventArgs e)

{

new EditInfrastructureFrom { Owner = this }.Show();

Hide();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который выполняет переход в меню простоя инфраструктуры

/// </summary>

private void простойToolStripMenuItem\_Click(object sender, EventArgs e)

{

var equipmentIdleMenu = new EquipmentIdleMenu() { Owner = this};

equipmentIdleMenu.Show();

Hide();

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который открывает диалоговое окно формы редактирования учетных записей пользователей

/// </summary>

private void редактированиеToolStripMenuItem\_Click(object sender, EventArgs e)

{

new UserListForm().ShowDialog();

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который открывает диалоговое окно формы регистрации новой учетной записи

/// </summary>

private void регистрацияToolStripMenuItem\_Click(object sender, EventArgs e)

{

new EditUsersForm().ShowDialog();

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который совершает выход из текущей учетной записи

/// </summary>

private void сменитьПользователяToolStripMenuItem\_Click(object sender, EventArgs e)

{

var result = MessageBox.Show("Будет произведен выход из текущей учетной записи! Продолжить?",

"Смена пользователя",

MessageBoxButtons.YesNo,

MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

\_ctx.Logout();

Owner?.Show();

\_logout = true;

this.Close();

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который завершает работу приложения

/// </summary>

private void завершениеРаботыToolStripMenuItem\_Click(object sender, EventArgs e)

{

var result = MessageBox.Show("Приложениебудетзакрыто! Продолжить?",

"Закрытиеприложения",

MessageBoxButtons.YesNo,

MessageBoxIcon.Exclamation);

if (result == DialogResult.Yes)

Application.Exit();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который выполняет переход на форму создания новой платформы

/// </summary>

private void добавитьНовуюToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (!\_ctx.CheckInfrastructureToCreatePlatform())

{

MessageBox.Show("Имеетсянеполныеданныеобвычислительнойинфраструктруре!" +

"\nПеред тем как добавить платформу заполните сведения об конфигурациях серверов и хранилищ данных!",

"Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

var editPlatformForm = new EditPlatformForm() { Owner = this };

editPlatformForm.Show();

Hide();

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который выполняет переход на форму редактирования выбранной платформы

/// </summary>

private async void редактироватьВыбраннуюToolStripMenuItem\_Click(object sender, EventArgs e)

{

var selectedPlatform = paasInfoBindingSource.Current as PaasInfo;

if (selectedPlatform == null)

{

MessageBox.Show("Платформаневыбрана!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

PaasType paasTypeToEdit = null;

try

{

paasTypeToEdit = await \_ctx.GetPaasFromPaasInfo(selectedPlatform);

}

catch

{

MessageBox.Show("Неудалосьполучитьданныеповыбраннойплатформе!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

var editPlatformForm = new EditPlatformForm(paasTypeToEdit) { Owner = this };

editPlatformForm.Show();

Hide();

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который выполняет переход на форму создания нового сервиса

/// </summary>

private void добавитьНовыйToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (\_ctx.CheckActivePaas() == 0)

{

MessageBox.Show("Отсутсвуют платформы, на которые были распределены вычислительные ресурсы!", "Ошибка",

MessageBoxButtons.OK);

return;

}

var editServiceForm = new EditServiceForm() { Owner = this };

editServiceForm.Show();

Hide();

}

/// <summary>

/// Обработчиксобытиянажатияклавишимышинаэлементглавногоменю,

/// который выполняет удаление выбранной категории сервиса

/// </summary>

private async void удалитьВыбранныйToolStripMenuItem\_Click(object sender, EventArgs e)

{

var selectedServiceInfo = serviceInfoBindingSource.Current as ServiceInfo;

if (selectedServiceInfo == null)

{

MessageBox.Show("Выберитекатегориюсервиса, которуюследуетудалить!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

try

{

var serviceType = await \_ctx.GetPaasTypeByName(selectedServiceInfo.Name);

DialogResult result = DialogResult.No;

if (serviceType.Service.Count == 0)

result = MessageBox.Show("Выбранная категория сервисов будет удалена! Продолжить?",

"Удаление категории сервиса", MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

else

result = MessageBox.Show("Выбранная категория сервисов и все ее конфигурации будет удалены! Продолжить?",

"Удаление категории сервиса", MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

try

{

await \_ctx.DeleteServiceType(serviceType);

UpdateBindings();

}

catch

{

MessageBox.Show("Произошлаошибкаприудалениикатегориисервисов!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

catch

{

MessageBox.Show("Произошла ошибка при получении информации о категории сервисов и ее конфигурациях!",

"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события активации формы

/// </summary>

private void SysAdminForm\_Activated(object sender, EventArgs e)

{

UpdateBindings();

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который выполняет удаление выбранной платформы

/// </summary>

private async void удалитьВыбраннуюToolStripMenuItem\_Click(object sender, EventArgs e)

{

var selectedPlatform = paasInfoBindingSource.Current as PaasInfo;

if (selectedPlatform == null)

{

MessageBox.Show("Платформаневыбрана!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

var paasTypeToDel = await \_ctx.GetPaasFromPaasInfo(selectedPlatform);

DialogResult result = DialogResult.No;

if (paasTypeToDel.Service.Count == 0)

result = MessageBox.Show("Выбранная платформа будет удалена! Продолжить?",

"Удаление категории платформы", MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

else

result = MessageBox.Show("Выбранная платформа и все ее конфигурации сервисов будет удалены! Продолжить?",

"Удаление категории платформы", MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk);

if (result == DialogResult.Yes)

{

try

{

await \_ctx.DeletePaasType(paasTypeToDel);

UpdateBindings();

}

catch

{

MessageBox.Show("Произошлаошибкаприудаленииплатформы!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

catch

{

MessageBox.Show("Неудалосьполучитьданныеповыбраннойплатформе!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события нажатия клавиши мыши на элемент главного меню,

/// который выполняет переход на форму редактора платформ и сервисов

/// </summary>

private void редакторПлатформИСервисовToolStripMenuItem\_Click(object sender, EventArgs e)

{

var listOfServicesForm = new ListOfServicesForm() {Owner = this};

listOfServicesForm.Show();

Hide();

}

/// <summary>

/// Обработчиксобытиявыполнениятактатаймера

/// </summary>

private void timer\_Tick(object sender, EventArgs e)

{

if (\_ctx == null)

return;

try

{

\_ctx.CheckServerMaintenance();

\_ctx.CheckSANMaintenance();

UpdateBindings();

}

catch

{

}

}

}

}

Модуль «LoginForm.cs»

using System;

using System.Drawing;

using System.Windows.Forms;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Forms.Director;

using MTF\_Services.WinForms.Forms.Employee;

using MTF\_Services.WinForms.Forms.SysAdmin;

namespace MTF\_Services.WinForms.Forms

{

/// <summary>

/// Класс формы авторизации пользователя.

/// </summary>

public partial class LoginForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Символ для скрытия пароля.

/// </summary>

private char passwordChar;

/// <summary>

/// Конструктор формы авторизации пользователя.

/// </summary>

public LoginForm()

{

InitializeComponent();

textBox1.KeyPress += TextBox1\_KeyPress;

textBox2.KeyPress += TextBox1\_KeyPress;

FormClosing += LoginForm\_FormClosing;

label1.Font = new Font(Font.FontFamily.Name, 20);

passwordChar = textBox2.PasswordChar;

\_ctx = new Context();

}

/// <summary>

/// Обработчик события нажатия клавиш на текстовом поле,

/// который при нажатии "Enter" производит попытку авторизации.

/// </summary>

private void TextBox1\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar == (char)Keys.Enter)

button1\_Click(null,EventArgs.Empty);

}

/// <summary>

/// Обработчик события закрытия формы.

/// </summary>

private void LoginForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (e.CloseReason == CloseReason.ApplicationExitCall)

return;

var result = MessageBox.Show("Приложениебудетзакрыто! Продолжить?", "Закрытиеприложения",

MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk, MessageBoxDefaultButton.Button2);

if (result == DialogResult.No)

e.Cancel = true;

}

/// <summary>

/// Обработчик события нажатия на кнопку авторизации

/// </summary>

private async void button1\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(textBox1.Text) || string.IsNullOrWhiteSpace(textBox2.Text))

{

MessageBox.Show("Заполните все поля для прохождения авторизации!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

Form frm = null;

try

{

if (!await \_ctx.Login(textBox1.Text, textBox2.Text))

{

MessageBox.Show("Неверный логин и/или пароль", "Неверные данные", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

switch (Context.CurrentUser.RightsLevel.Name)

{

case "Директор":

frm = new DirectorForm();

break;

case "Системныйадминистратор":

frm = new SysAdminForm();

break;

case "Сотрудник":

frm = new EmployeeForm();

break;

default:

throw new Exception();

}

textBox1.Text = textBox2.Text = string.Empty;

frm.Owner = this;

frm.Show();

Hide();

}

catch

{

frm?.Dispose();

this.Show();

MessageBox.Show("Произошлаошибкавовремяавторизации!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик события перемещения курсора мыши на метке "Пароль",

/// которыйотображаетпароль

/// </summary>

private void label3\_MouseHover(object sender, EventArgs e)

{

textBox2.PasswordChar = '\0';

}

/// <summary>

/// Обработчик события выхода курсора мыши на метке "Пароль",

/// которыйскрываетпароль

/// </summary>

private void label3\_MouseLeave(object sender, EventArgs e)

{

textBox2.PasswordChar = passwordChar;

}

}

}

Модуль «ReportingForm.cs»

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Windows.Forms;

using Microsoft.Reporting.WinForms;

using MTF\_Services.Model.Views;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

namespace MTF\_Services.WinForms.Forms

{

/// <summary>

/// Класс формы предварительного просмотра отчетов

/// </summary>

public partial class ReportingForm : Form

{

private readonly ServiceIdleParams \_serviceIdleParams;

private readonly BindingList<ServiceDetailInfo> \_servicesDetailInfo;

private readonly string \_costColumnHeaderName;

private readonly ServiceRequestItem \_serviceRequestItem;

private readonly List<SoftwareInfo> \_softwaresInfo;

/// <summary>

/// Тип выбранного отчета

/// </summary>

private readonly ReportType \_reportType;

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Конструктор формы предварительного просмотра отчетов

/// </summary>

private ReportingForm(ReportType reportType)

{

\_reportType = reportType;

InitializeComponent();

SubscribeMenuItems();

\_ctx = new Context();

}

/// <summary>

/// Конструктор формы для предварительного просмотра заявки на предоставление сервиса

/// </summary>

/// <param name="serviceRequestItem">Выбраннаязаявка</param>

/// <param name="softwaresInfo">Программное обеспечение сервиса</param>

public ReportingForm(ServiceRequestItem serviceRequestItem, List<SoftwareInfo> softwaresInfo) : this(ReportType.ServiceRequest)

{

\_serviceRequestItem = serviceRequestItem;

\_softwaresInfo = softwaresInfo;

}

/// <summary>

/// Конструктор формы для предварительного просмотра стоимости предоставления сервисов

/// </summary>

/// <param name="servicesDetailInfo">Спискоиспользуемыхсервисов</param>

/// <param name="softwaresInfo"></param>

public ReportingForm(BindingList<ServiceDetailInfo> servicesDetailInfo, string costColumnHeaderName) : this(ReportType.ServiceCost)

{

\_servicesDetailInfo = servicesDetailInfo;

\_costColumnHeaderName = costColumnHeaderName;

}

/// <summary>

/// Конструктор формы для предварительного просмотра информации о простое сервиса

/// </summary>

/// <param name="serviceIdleParams"></param>

public ReportingForm(ServiceIdleParams serviceIdleParams) : this(ReportType.ServiceIdle)

{

\_serviceIdleParams = serviceIdleParams;

}

/// <summary>

/// Обработчиксобытиявыходакурсораизэлементаглавногоменю,

/// который очищает область в строке состояния.

/// </summary>

private void MenuItem\_MouseLeave(object sender, EventArgs e)

{

tip\_Label.Text = string.Empty;

}

/// <summary>

/// Обработчик события наведения курсора на элемент главного меню,

/// который выводит подсказку элемента в строку состояния.

/// </summary>

private void MenuItem\_MouseEnter(object sender, EventArgs e)

{

tip\_Label.Text = ((ToolStripItem)sender).ToolTipText;

}

/// <summary>

/// Подписка элементов главного меню и панели элементов на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

private void SubscribeMenuItems()

{

SubScribeChildMenuItems(menuBar.Items);

}

/// <summary>

/// Подписка дочерних элементов главного меню на события

/// наведения и выхода курсора для отображения подсказок в строке состояния.

/// </summary>

/// <param name="col">Коллекция элементов главного меню</param>

private void SubScribeChildMenuItems(ToolStripItemCollection col)

{

for (int i = 0; i < col.Count; i++)

{

var item = col[i] as ToolStripMenuItem;

if (item != null)

{

item.MouseEnter += MenuItem\_MouseEnter;

item.MouseLeave += MenuItem\_MouseLeave;

if (item.DropDownItems.Count > 0)

SubScribeChildMenuItems(item.DropDownItems);

}

}

}

/// <summary>

/// Обработчиксобытиязагрузкиформы

/// </summary>

private void ReportingForm\_Load(object sender, EventArgs e)

{

reportViewer1.LocalReport.DataSources.Clear();

switch (\_reportType)

{

case ReportType.ServiceRequest:

this.reportViewer1.LocalReport.ReportEmbeddedResource = "MTF\_Services.WinForms.Reports.ServiceRequestReport.rdlc";

ServiceRequestItemBindingSource.DataSource = \_serviceRequestItem;

SoftwareInfoBindingSource.DataSource = \_softwaresInfo;

ReportDataSource ds\_Request = new ReportDataSource("ds\_Request",ServiceRequestItemBindingSource);

ReportDataSource ds\_SoftwareInfo = new ReportDataSource("ds\_SoftwareInfo", SoftwareInfoBindingSource);

reportViewer1.LocalReport.DataSources.Add(ds\_Request);

reportViewer1.LocalReport.DataSources.Add(ds\_SoftwareInfo);

break;

case ReportType.ServiceCost:

this.reportViewer1.LocalReport.ReportEmbeddedResource = "MTF\_Services.WinForms.Reports.ServiceCostReport.rdlc";

ServiceDetailInfoBindingSource.DataSource = \_servicesDetailInfo;

ReportDataSource serviceCostDS = new ReportDataSource("ds\_ServiceDetail", ServiceDetailInfoBindingSource);

reportViewer1.LocalReport.DataSources.Add(serviceCostDS);

break;

case ReportType.ServiceIdle:

this.reportViewer1.LocalReport.ReportEmbeddedResource = "MTF\_Services.WinForms.Reports.ServiceIdleReport.rdlc";

BindingSource idleBs = new BindingSource() {DataSource = \_serviceIdleParams};

ReportDataSource serviceIdleDS = new ReportDataSource("ds\_ServiceIdleParams", idleBs);

reportViewer1.LocalReport.DataSources.Add(serviceIdleDS);

break;

}

this.reportViewer1.RefreshReport();

}

}

}

Модуль «ShowUserProfileForm.cs»

using System;

using System.Linq;

using System.Windows.Forms;

using MTF\_Services.Model;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Extentions;

namespace MTF\_Services.WinForms.Forms

{

/// <summary>

/// Класс формы просмотра данных текущей учетной записи.

/// </summary>

public partial class ShowUserProfileForm : Form

{

/// <summary>

/// Объект для доступа к контекту данных.

/// </summary>

private readonly Context \_ctx;

/// <summary>

/// Символ для скрытия пароля.

/// </summary>

private readonly char \_passwordCh;

private string \_loginBeforeChange;

/// <summary>

/// Конструктор формы просмотра данных текущей учетной записи.

/// </summary>

public ShowUserProfileForm()

{

InitializeComponent();

\_passwordCh = textBox2.PasswordChar;

\_ctx = new Context();

\_loginBeforeChange = Context.CurrentUser.Login;

user.DataSource = Context.CurrentUser;

user.ResumeBinding();

var bindingList = \_ctx.GetUsersInfo();

var x = bindingList.Single(s => s.TabNo == Context.CurrentUser.TabNo);

userInfo.DataSource = x;

}

/// <summary>

/// Обработчик события перемещения курсора мыши на метке "Пароль",

/// которыйотображаетпароль

/// </summary>

private void label2\_MouseHover(object sender, EventArgs e)

{

textBox2.PasswordChar = '\0';

}

/// <summary>

/// Обработчик события выхода курсора мыши на метке "Пароль",

/// которыйскрываетпароль

/// </summary>

private void label2\_MouseLeave(object sender, EventArgs e)

{

textBox2.PasswordChar = \_passwordCh;

}

private async void button1\_Click(object sender, EventArgs e)

{

var CurrentUser = user.DataSource as User;

try

{

if (string.IsNullOrWhiteSpace(CurrentUser.Login) || string.IsNullOrWhiteSpace(CurrentUser.Password))

{

MessageBox.Show("Введите все поля для сохранения изменений!", "Предупреждение",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

bool existed = false;

existed = await \_ctx.CheckUserLoginForDublicate(CurrentUser.Login) && !CurrentUser.Login.Equals(\_loginBeforeChange);

if (existed)

{

MessageBox.Show("Пользователь с таким логином уже существует!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

await \_ctx.SaveChangesAsync();

\_loginBeforeChange = CurrentUser.Login;

MessageBox.Show("Изменения успешно сохранены!", "Информация", MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

catch

{

MessageBox.Show("Произошла ошибка при сохранении изменений!", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

Модуль «Program.cs»

using System;

using System.Collections.Generic;

using System.Reflection;

using System.Runtime.InteropServices;

using System.Threading;

using System.Windows.Forms;

using MTF\_Services.WinForms.Data;

using MTF\_Services.WinForms.Forms;

using MTF\_Services.WinForms.Forms.SysAdmin.Services.Dictionary;

using MTF\_Services.WinForms.Forms.SysAdmin.Services.Parts;

namespace MTF\_Services.WinForms

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.ThreadException += Application\_ThreadException;

Application.SetUnhandledExceptionMode(UnhandledExceptionMode.CatchException);

AppDomain.CurrentDomain.UnhandledException += CurrentDomain\_UnhandledException;

bool existed;

// получаем GIUD приложения

string guid = Marshal.GetTypeLibGuidForAssembly(Assembly.GetExecutingAssembly()).ToString();

Mutex mutexObj = new Mutex(true, guid, out existed);

if (!existed)

{

MessageBox.Show("Экземплярприложенияужезапущен!", "Предупреждение", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

try

{

if (!new Context().TestConnection())

throw new Exception("Пустая ссылка на контекст данных.");

}

catch (Exception)

{

MessageBox.Show("Не удалось подключиться к базе данных! Приложение будет закрыто.",

"Критическая ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

Application.Exit();

}

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new EditSoftwareTypeForm());

}

#region Необработанныеисключения

/// <summary>

/// Обработчик события перехвата необработанного исключения в текущем домене приложений

/// </summary>

private static void CurrentDomain\_UnhandledException(object sender, UnhandledExceptionEventArgs e)

{

try

{

MessageBox.Show("Произошла ошибка во время выполения операции! Обратитесь к системному администратору!" +

$"Описание ошибки: {((Exception)e.ExceptionObject).Message}", "Необработанное испключение",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

catch

{

try

{

MessageBox.Show("Фатальнаяошибка! Приложение будет закрыто!", "Необработанное испключение",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

finally

{

Application.Exit();

}

}

}

/// <summary>

/// Обработчик события перехвата необработанного исключения в текущем потоке

/// </summary>

private static void Application\_ThreadException(object sender, System.Threading.ThreadExceptionEventArgs e)

{

DialogResult result = DialogResult.Cancel;

try

{

result = MessageBox.Show("Произошла ошибка во время выполения операции! Обратитесь к системному администратору!" +

$"Описание ошибки: {e.Exception.Message}", "Необработанное испключение",

MessageBoxButtons.AbortRetryIgnore, MessageBoxIcon.Error);

}

catch

{

try

{

MessageBox.Show("Фатальнаяошибка! Приложение будет закрыто!", "Необработанное испключение",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

finally

{

Application.Exit();

}

}

if (result == DialogResult.Abort)

Application.Exit();

}

#endregion

}

}

# **Приложение Ж****.**

**Инсталляционныйдиск**

Зятына Евгений Эдуардович, Расчет стоимости использования вычислительных сервисов, а также оценка стоимости их простоя в ПАО «Мурманский траловый флот»: Дипломный проект. – Мурманск, 2018. – 421с.

Научный руководитель:

Рецензент:

Дипломный проект выполнен мною совершенно самостоятельно. Все использованные в работе материалы из опубликованной литературы и других источников имеют ссылки на них.

«15» июня 2018 г.

Зятына Евгений Эдуардович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(ФИО выпускника, (подпись)

Выпускная квалификационная работа представлена к защите

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись начальника учебного отдела)

Защищена «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_ 2018 г.

Протокол № \_\_\_\_ от «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2018 г.

Оценка: «\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_»

1. Данные таблицы имеют предопределенные значения [↑](#footnote-ref-1)