Оглавление

[**Введение** 2](#_Toc122641711)

[**Первая глава** 3](#_Toc122641712)

[**Описание предметной области** 3](#_Toc122641713)

[**Глава 2. Результаты проектирования системы** 6](#_Toc122641714)

[**Диаграмма вариантов использования** 14](#_Toc122641715)

[**Диаграмма последовательностей** 19](#_Toc122641716)

[**Диаграмма развертывания** 20](#_Toc122641717)

[**Глава 3. Реализация системы** 22](#_Toc122641718)

[**Глава 4. Руководство пользователя и результат работы** 29](#_Toc122641719)

[**Заключение** 47](#_Toc122641720)

[**Список литературы** 48](#_Toc122641721)

[**Приложение** 49](#_Toc122641722)

# **Введение**

Темой данной курсовой работы является разработка приложения для сервиса технического обслуживания, в котором ведется учет клиентов (автомобилей), проводятся техническое обслуживание автомобиля, ремонт.

В современном мире каждая коммерческая организация стремится к увеличению КПД труда рабочих и увеличению прибыли. Данное приложение упростит работу по учёту клиентов, улучшит взаимодействие между различными отделами организации. И, что важнее всего, сможет исключить ненужные действия, что снизит число необходимых рабочих, но увеличит их награду. Именно поэтому выбранную задачу можно считать актуальной и необходимой.

В первой главе анализируется предметная область. Во второй главе обозреваются диаграммы вариантов использования, ер-диаграммы, диаграммы классов, диаграмма последовательностей и диаграмма состояний. В третьей главе описываются классы программы. В четвертой главе рассматривается инструкция пользователю.

# **Первая глава**

## **Описание предметной области**

В качестве предметной области выбрано СТО «Руки-крюки», который занимается обслуживанием автомобилей клиентов. СТО способно принимать клиентов (автомобили), хранить информацию о пройденных автомобилем ТО, отправлять автомобиль на техническое обслуживание или ремонт. К каждому ТО создаются необходимые услуги, которые содержат в себе запчасти. Программа «СТО «Руки-крюки». Работник» предназначена для добавления клиентов, проведения техосмотров и хранения записей о пройденных сервисах. Программа предоставляет возможность получения отчетов по ТО и используемых в них запчастей, по проведенным ТО за определенный период, с указанием запчастей и записей сервисов.

Данное программное решение было реализовано в версии для Веб-браузера для ПК.

## **Аналоги**

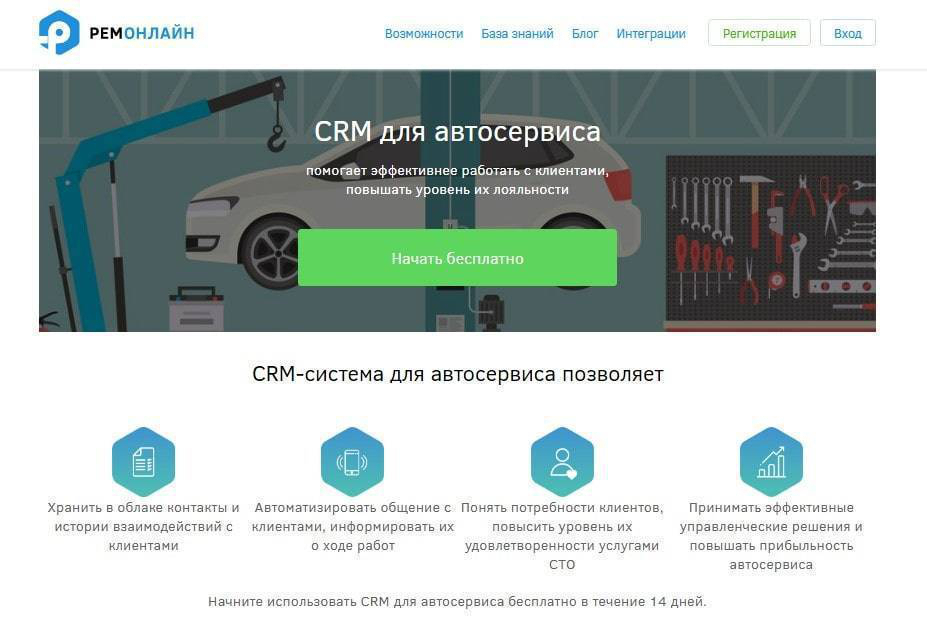
АвтоДилер

* Помогает автоматизировать работу автосервиса по расчету стоимости ремонта, планированию загрузки, общению с клиентами.
* Берет на себя работу с запчастями: подбор, заказ, складской учет, документы. Позволяет вести учёт материалов.
* Позволяет оценивать эффективность работы автосервиса, вести управленческий учет, увеличивать количество заездов клиентов, повышать средний чек и выручку. Есть наглядные отчёты и графики.



РемОнлайн

* Создание базы данных с возможностью группирования юридических, физических лиц, поставщиков и клиентов, удобный фильтр для быстрого поиска нужного клиента;
* Автоматизированное общение с клиентской базой, информирование их об этапах выполненных работ;
* Облако для хранения контактов и истории взаимодействий с клиентами и поставщиками;
* Быструю регистрацию новых клиентов и обработку заказов благодаря готовым шаблонам карточек;
* Создание предварительных записей клиентов на ТО и ремонт, фиксирование продолжительности выполнения работ, быстрый поиск свободного или необходимого клиенту мастера;
* Интеграция с сервисом IP-телефонии и контроль качества обслуживания клиентов по телефону;
* Оповещение сотрудников и клиентов в автоматическом режиме при помощи SMS или email-сообщений.



## **Техническое задание на разработку программного продукта «СТО «Руки-Крюки». Поставщик»**

Программа «СТО «Руки-крюки». Работник» предназначена для ведения учета обслуживания машин. Работники указывают доступные ТО, машины, которые их проходили, и записи сервисов машин. Программа предоставляет возможность получения отчётов по ТО.

**Задачи:**

**-**формирование записи (CRUD) по «ТО» с выбором из списка «работ» нескольких записей к ней;

-получение списка по «запчастям» на основе выбранных записей «ТО» в формате doc/xls;

-формирование записей (CRUD) «машин»;

-привязывание записей «машин» к записям «ТО»;

-формирование записей (CRUD) «сервисов» с привязкой записи «ТО»;

-получение отчёта за период по записям из «ТО» с расшифровкой по записям «запчастей», относящихся к записям «машин», относящихся к записям «ТО» в формате pdf на почту и вывод отчёта на форму.

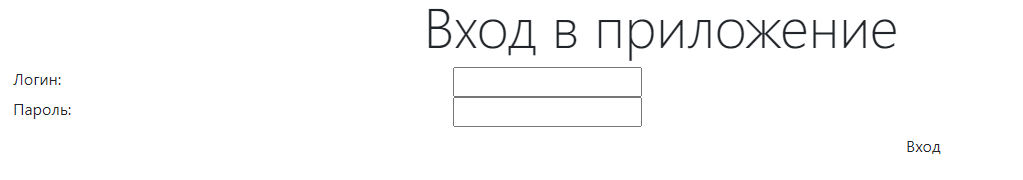
**Основной функционал программы:**

* Регистрация. Для регистрации пользователь должен заполнить следующие данные:

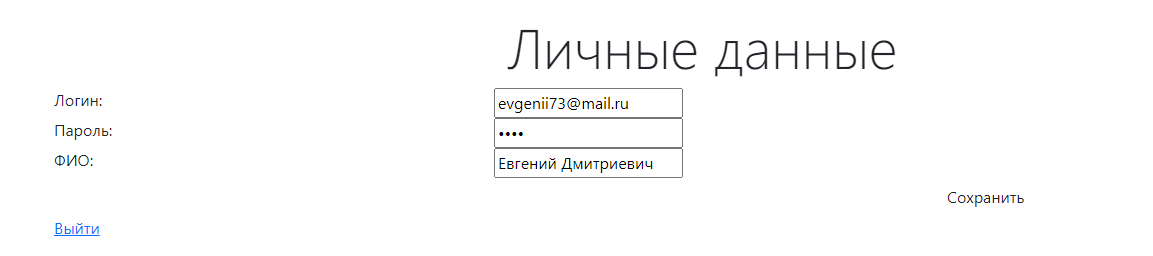
Логин, пароль, повтор пароля (пароль – не менее 8 символов, совпадение введённых паролей)



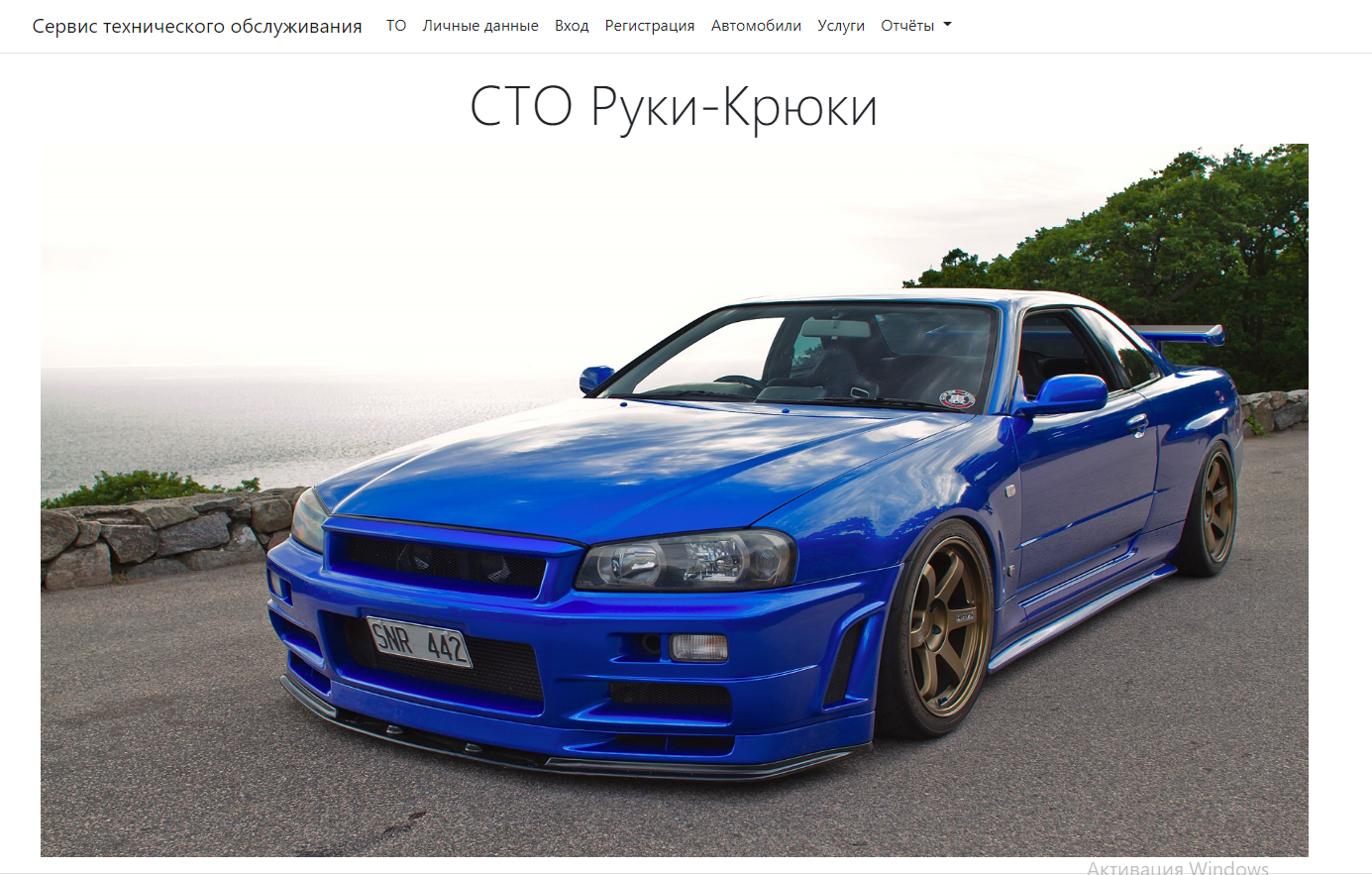
* Авторизация. Пользователь вводит данные. Если такой пользователь есть в системе, то происходит переход к основной форме. Если такого пользователя нет в системе, выводится сообщение об этом с просьбой ввести заново данные.



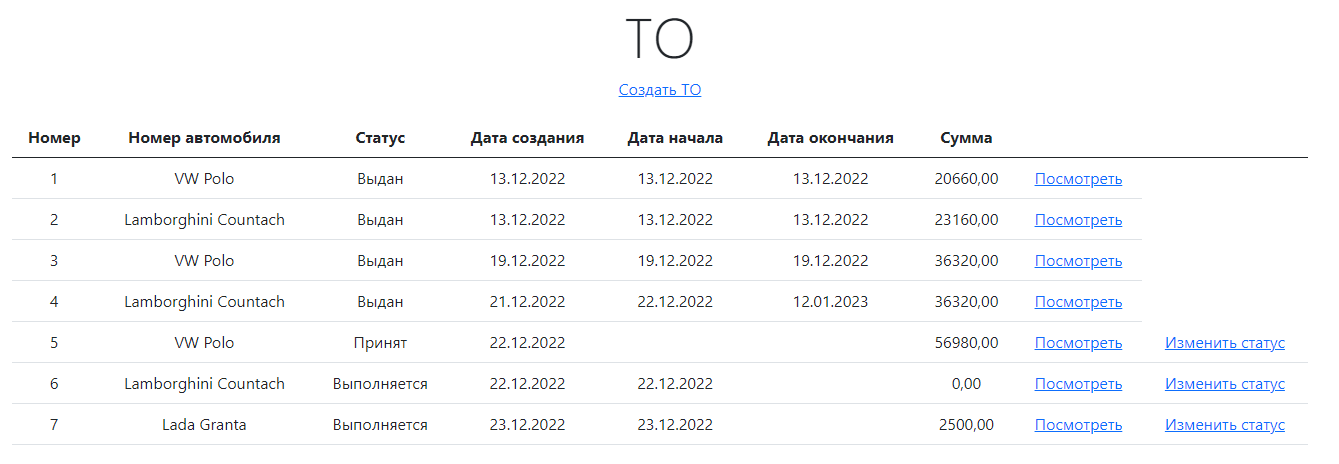
* Личные данные. Выводятся данные пользователя о его аккаунте (пароль, логин, ФИО). Есть возможность выйти из аккаунта или обновить данные.



* Основная форма. Состоит из пунктов меню и логотипа. Через пункты меню можно перейти на формы работы с ТО, машинами (в том числе и привязка машин), записями сервисов, на форму получения списка запчастей и на форму получения отчета.

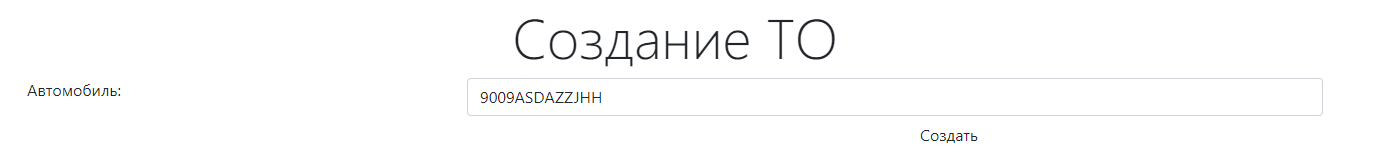


* Список всех ТО. Имеется форма, на которой расположены все существующие ТО, относящиеся к конкретному работнику. Можно создать новое ТО, изменить/посмотреть существующее, изменить статус ТО.

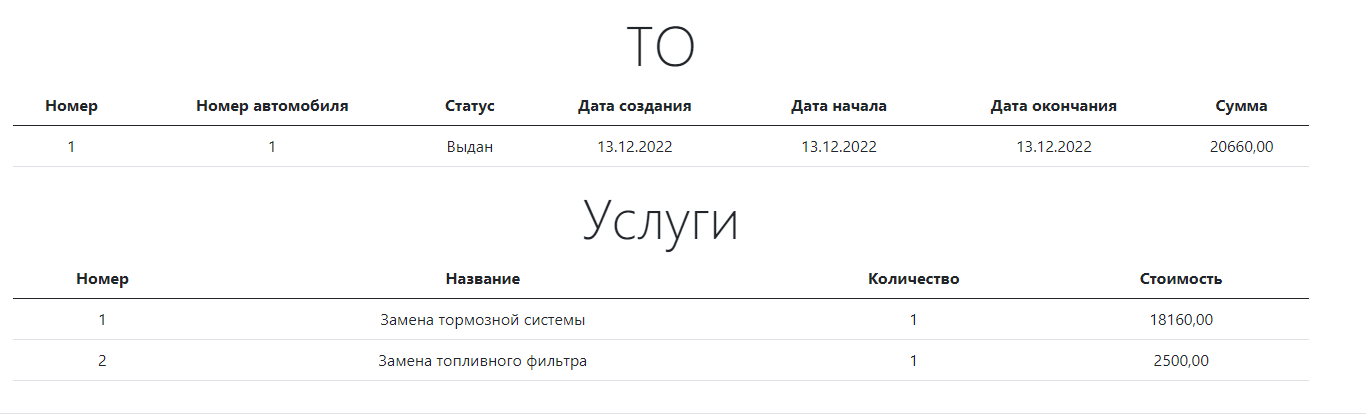


* Создание ТО (CRUD). Имеется форма. Пользователь может создать запись ТО, заполнив следующее поле:

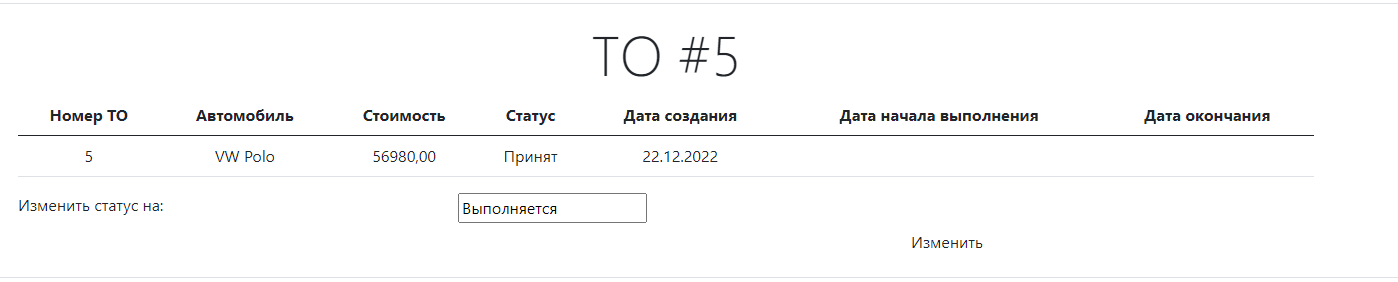
1. VIN-номер авто;



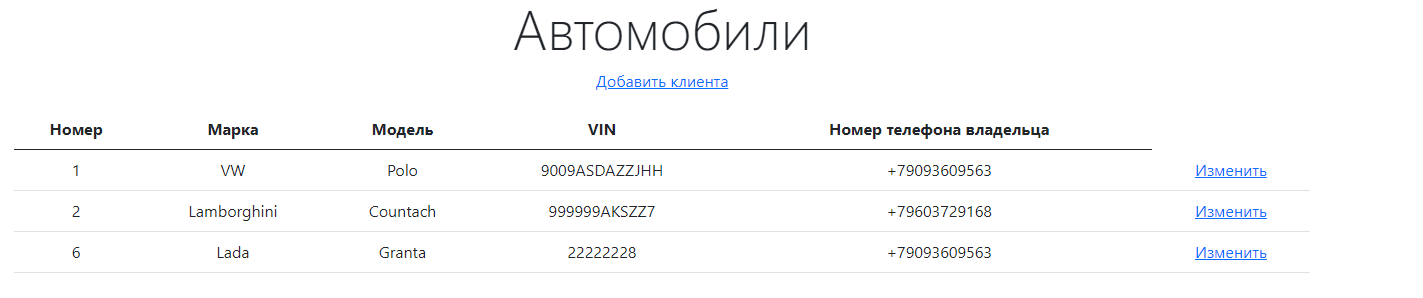
* Просмотр ТО. На форму выводятся данные о ТО и содержащиеся в нем услуги.



* Изменение статуса ТО. На форму выводятся данные о ТО. Есть возможность изменить статус ТО.

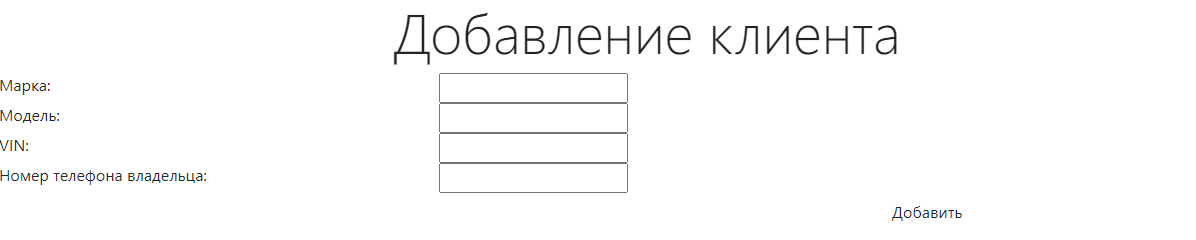


* Список всех автомобилей. На форму выводятся все автомобили, являющиеся клиентами СТО. Есть возможность добавить новый автомобиль, изменить запись.

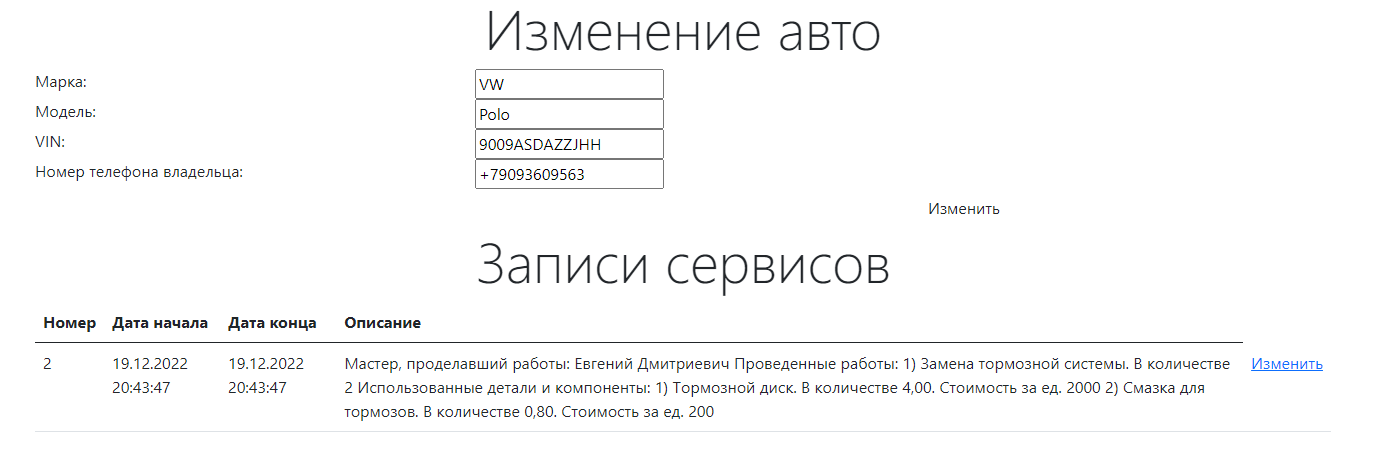


* Добавление автомобиля (CRUD). Выводится форма, в которой необходимо заполнить следующие поля:

1. Марка
2. Модель
3. VIN
4. Номер телефона владельца



* Изменение данных автомобиля. На форму выводятся данные об авто и связанные с ним записи сервисов. Есть возможность изменить данные автомобиля или изменить записи его сервисов.



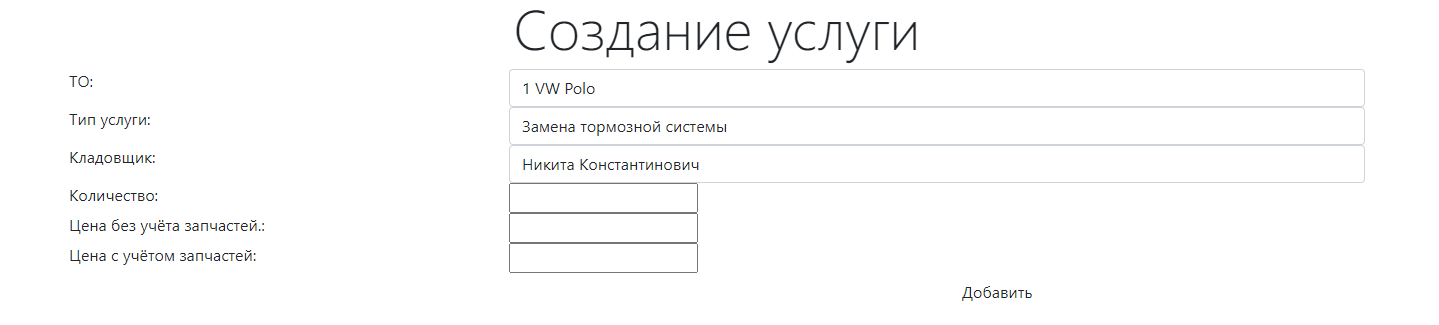
* Список всех услуг. На форму выводятся все услуги, связанные с ТО, которые проводит работник. Здесь можно следить за их прогрессом выполнения. А также просмотреть запчасти, используемые в той или иной услуге. Также есть возможность создать услугу.

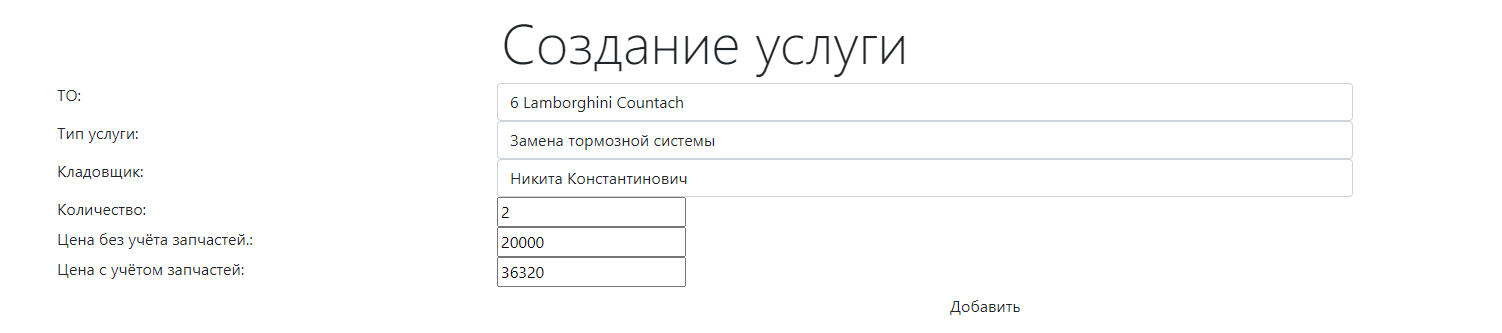


* Создание услуги (CRUD). Выводится форма, на которой необходимо заполнить следующие поля:

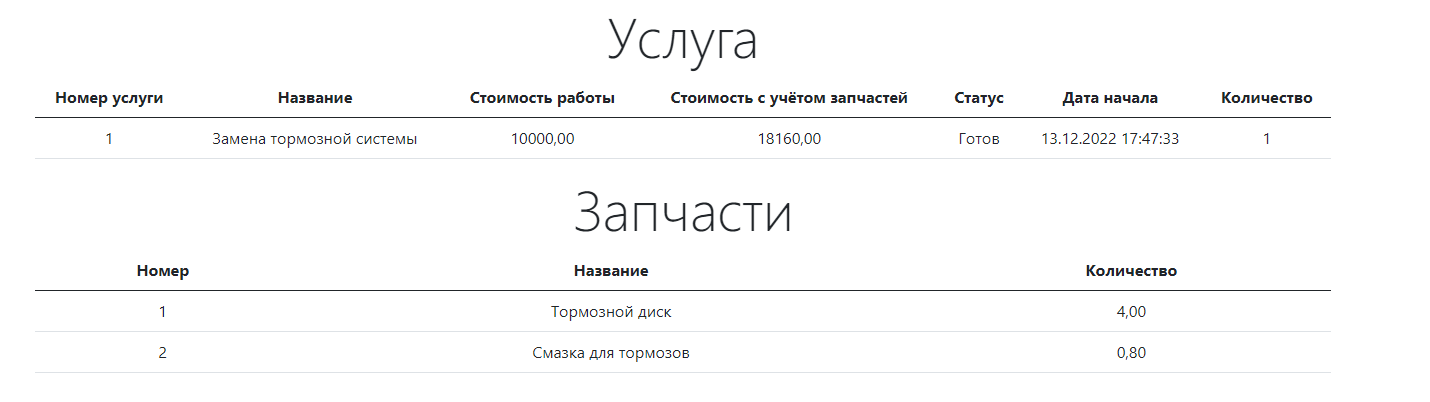
1. ТО
2. Тип услуги
3. Кладовщик
4. Количество

Поля цена без учёта запчастей и цена с учётом запчастей заполнятся автоматически.

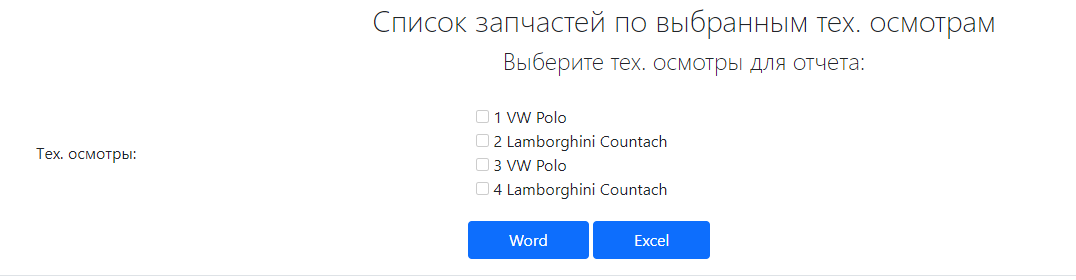




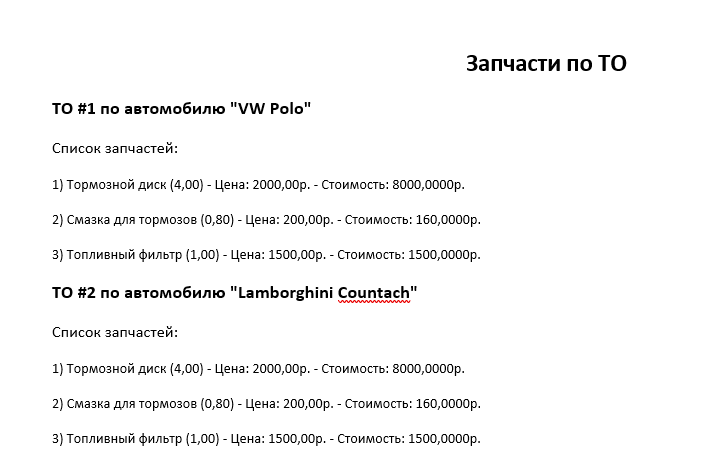
* Просмотр услуги. На форму выводятся данные об услуги и используемые в ней запчасти.



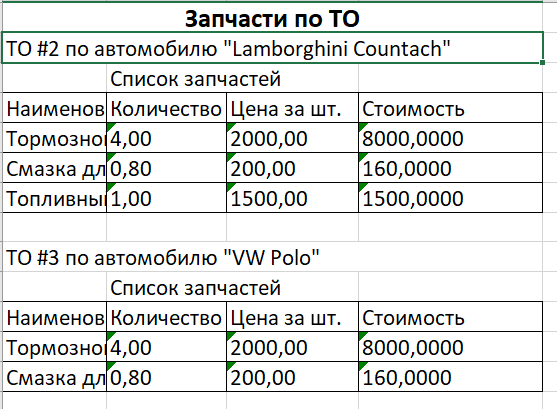
* Получение списка. Пользователь может получить список запчастей по выбранным ТО. Пользователь отмечает интересующие его ТО, указывает формат файла (docx или xlsx) и получает интересующий его список.



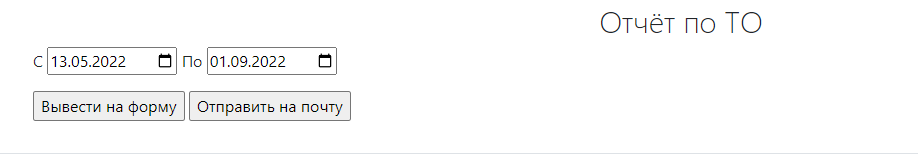
При выводе в docx-файл формат документа такой:

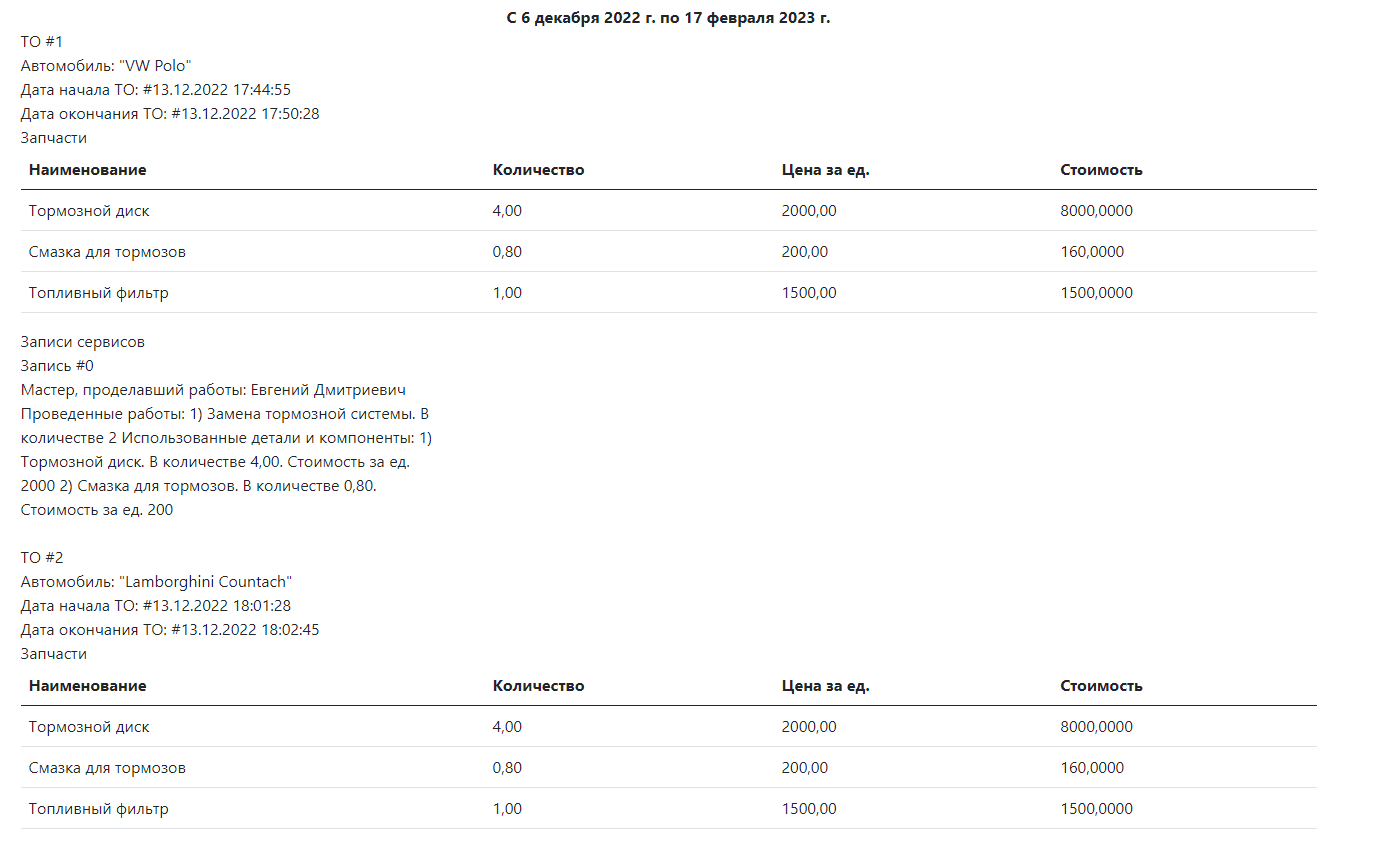
****

При выводе в xlsx-файл формат документа такой:



* Пользователь может получить сведения по полученным им запчастям и полученным записям сервисов за период по различным ТО. Для этого он должен указать интересующий его период, выбрать вариант получения сведений (отправка на почту или вывод на форму) и получить отчёт.





В качестве интерфейса пользователя будет выступать web-приложение, разработанное на ASP .NET Core.

# **Глава 2. Результаты проектирования системы**

## **Диаграмма вариантов использования**

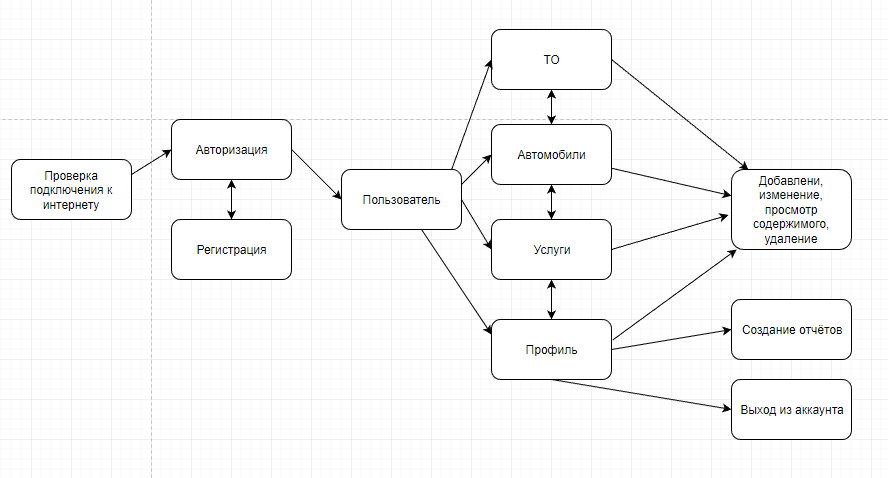


Рисунок 1. Диаграмма вариантов использования

Пользователем программы является работник сервиса. При входе в приложение необходимо зарегистрироваться или авторизоваться. В главном окне можно управлять текущими и завершенными ТО. Можно переключаться между экранами с помощью нижнего меню. При вводе неверных данных пользователю выведется ошибка. Также есть возможность получить отчеты. Также есть разграничение по ролям, если работник заходит под ролью “Старший работник”, то ему становятся доступны все текущие ТО, а не только проводимые им.

**ER-диаграмма**

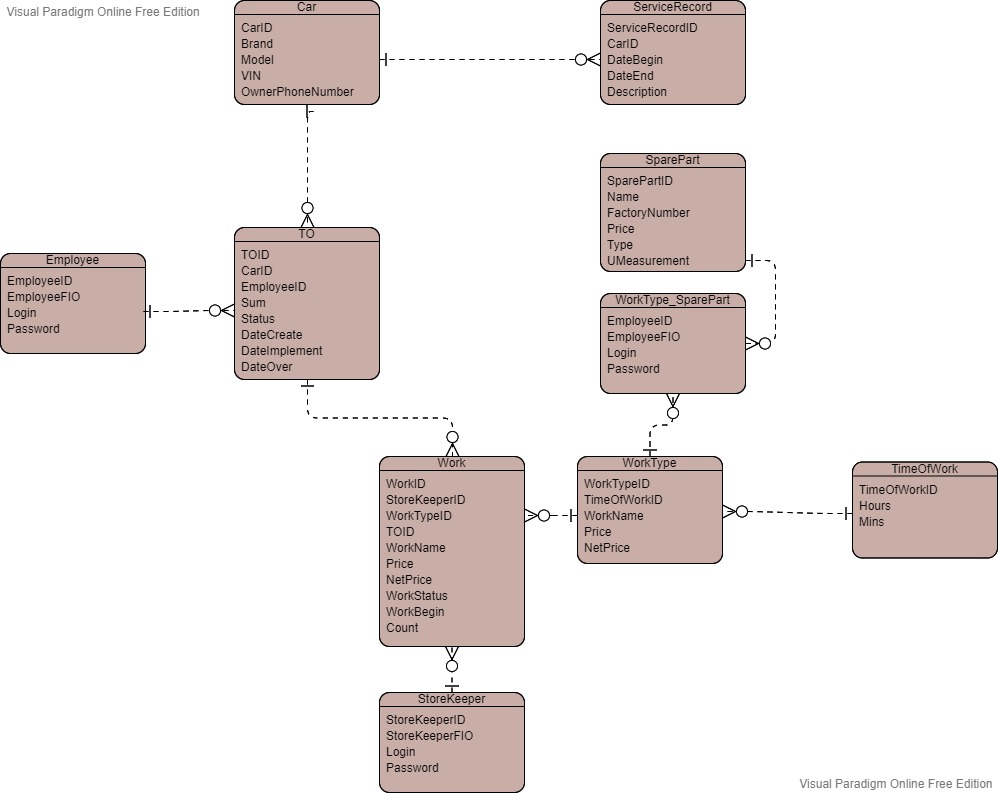


Рисунок 2. ER-диаграмма

В данной диаграмме представлена схема базы данных для всего проекта “СТО”Руки-крюки.””

Описание таблиц:

«Работник» (Employee) – сущность, содержащая информацию о работнике СТО.

Атрибуты:

* EmployeeID – идентификационный номер. Уникальный для каждой записи.
* Login – логин.
* Password – пароль.
* EmployeeFIO – фио.

«ТО» (TO) – сущность, содержащая информацию о ТО.

Атрибуты:

* TOID – идентификационный номер. Уникальный для каждой записи.
* CarID - идентификационный номер связанного авто.
* EmployeeID - идентификационный номер работника.
* Sum – стоимость.
* Status – статус.
* DateCreate – дата создания
* DateImplement – дата начала работ
* DateOver – дата окончания

«Автомобиль» (Car) - сущность, содержащая информацию об автомобиле

Атрибуты:

* CarID – идентификационный номер. Уникальный для каждой записи.
* Brand – марка авто.
* Model – модель авто.
* VIN – VIN-номер авто.
* OwnerPhoneNumber – номер телефона владельца.

«Запись сервиса» (ServiceRecord) - сущность, содержащая информацию о записи сервиса.

Атрибуты:

* ServiceRecordID – идентификационный номер. Уникальный для каждой записи.
* CarID – идентификационный номер автомобиля, связанного с записью.
* DateBegin – дата начала проведения ТО, связанного с записью.
* DateEnd – дата окончания проведения ТО, связанного с записью.
* Description – описание проведенного ТО.

«Услуга/Работа» (Work) - сущность, содержащая информацию об услуге.

Атрибуты:

* WorkID – идентификационный номер. Уникальный для каждой записи.
* StoreKeeperID – идентификационный номер кладовщика, связанного с записью.
* WorkTypeID – идентификационный номер типа услуги, связанного с услугой.
* TOID– идентификационный номер ТО, связанного с услугой.
* WorkName – название.
* Price – стоимость проведения услуги.
* NetPrice – стоимость услуги + запчастей.
* WorkStatus – статус услуги.
* WorkBegin – дата начала проведения услуги.
* Count – количество.

«Тип услуги» (WorkType) - сущность, содержащая информацию о типе услуги.

Атрибуты:

* WorkTypeID – идентификационный номер. Уникальный для каждой записи.
* TimeOfWorkID – идентификационный номер времени работы, связанного с типом услуги.
* WorkName – название
* Price – стоимость услуги.
* NetPrice – стоимость услуги + запчастей.

«Время работы» (TimeOfWork) - сущность, содержащая информацию о времени работы.

Атрибуты:

* TimeOfWorkID – идентификационный номер. Уникальный для каждой записи.
* Hours – часы.
* Mins – минуты.

«ТипРаботы\_Запчасть» (WorkType\_SparePart) - сущность, содержащая информацию о связанности типа услуги и запчастей.

Атрибуты:

* ID - идентификационный номер. Уникальный для каждой записи.
* WorkTypeID – идентификационный номер типа услуги.
* SparePartID – идентификационный номер запчасти.
* Count – количество запчастей.

«Запчасть» (SparePart) - сущность, содержащая информацию о запчасти.

Атрибуты:

* SparePartID - идентификационный номер. Уникальный для каждой записи.
* Name – название.
* FactoryNumber – заводской номер.
* Price – цена.
* Type – тип.
* UMeasurement – ед. измерения.

## **Диаграмма последовательностей**

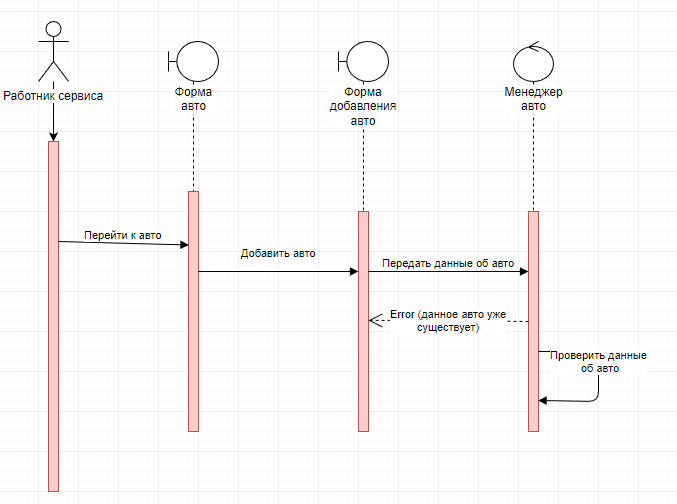


Рисунок 3. Диаграмма последовательности

На данной диаграмме предоставлена последовательность работы приложения на примере добавления автомобиля (клиента). В начале происходит ввод данных об авто. Далее данные передаются в БД и проверяются. Если нет ошибок, то создается новая запись. После проверки данных происходит связывание, и работник получает сообщение о том, что связывание прошло успешно.

## **Диаграмма развертывания**

Программа имеет следующую структуру:

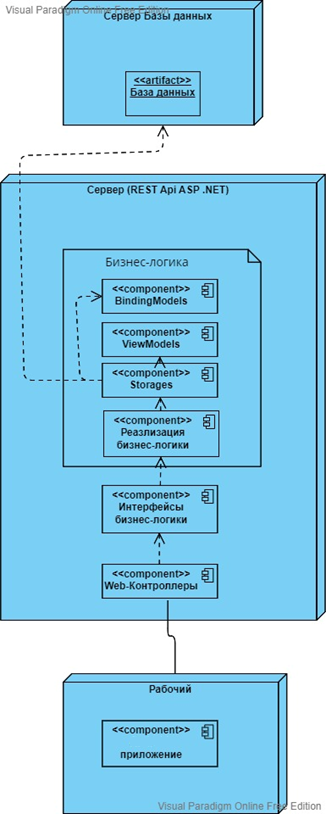


Рисунок . Диаграмма развертывания

Сервер БД и сервер REST Api, который обращается к бизнес-логике, которая обращается к логике хранения сущностей и взаимодействует с моделями (создаёт их, изменяет, удаляет). На диаграмме отображены:

* Классы бизнес-логики – описание логики работы, принципы CRUD
* CRUD – описание функций (запросов) для работы с БД (создание, чтение, изменение, удаление)
* BindingModels – классы для создания, изменения, удаления сущностей, которые сервер REST Api получает от клиентов.
* ViewModels – классы для представления сущностей пользователю. REST Api отправляет их клиенту.
* Интерфейсы для хранилищ (описание методов для получения списков, элементов, добавление/изменение записей). Взаимодействует непосредственно с базой данных.
* EntityFrameworkCore для работы с БД.

# **Глава 3. Реализация системы**

Системы была реализована с помощью паттерна MVC (Модель-Представление-Контроллер). MVC – схема разделения данных приложения и управляющий логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

* Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
* Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
* Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

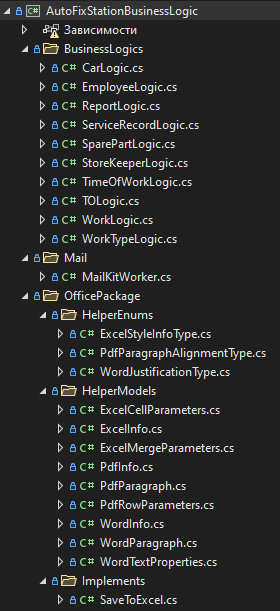
Controller – пользователь (приложение) обращается к контроллерам REST Api для изменения, удаления и добавления данных. Контроллер в свою очередь обращается к Model, которая содержит в себе всю логику и манипулирует базой данных (ASP.NET Core).

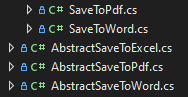
Model – бизнес-логика и логика управления БД. Манипулирует сущностями (Microsoft SQL Express, Microsoft.EntityFrameworkCore).

View – логика отправки данных пользователю. REST Api отправляет пользователю удобопредставленные данные, с которыми затем он может взаимодействовать и отправлять запрос на изменение или создание обратно на сервер. На устройстве пользователя всё предстаёт в человекопонятном виде.

Программа имеет следующую структуру:

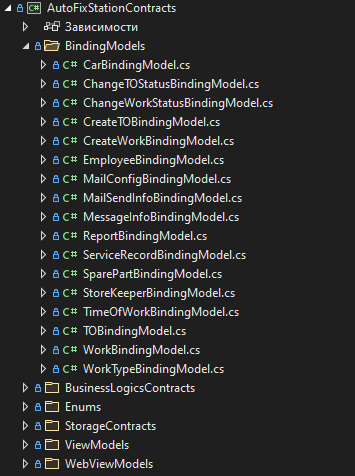
1. AutoFixStationBusinessLogic – проект, содержащий в себе классы, необходимые для функционирования логики.





* 1. BusinessLogics – папка, хранящая классы логики.
     1. CarLogic – класс, бизнес-логика для авто.
     2. EmployeeLogic – класс, бизнес-логика для работников.
     3. ReportLogic – класс, бизнес-логика для отчётов.
     4. ServiceRecordLogic – класс, бизнес-логика для записи сервисов.
     5. SparePartLogic – класс, бизнес-логика для запчастей.
     6. StoreKeeperLogic – класс, бизнес-логика для кладовщиков.
     7. TimeOfWorkLogic – класс, бизнес-логика для времени выполнения.
     8. TOLogic – класс, бизнес-логика для тех. осмотров.
     9. WorkLogic – класс, бизнес-логика для услуг.
     10. WorkTypeLogic – класс, бизнес-логика для типов услуг.
  2. Mail – папка, хранящая класс для отправки отчёта на почту.
     1. MailKitWorker – класс, бизнес-логика для отправки писем с отчётами на почту.
  3. OfficePackage – папка, хранящая классы для создания отчётов.
     1. HelperEnums – папка, хранящая перечисления для создания отчётов.
        1. ExcelStyleInfoType – перечисление для текста Excel.
        2. PdfParagraphAlignmentType – перечисление для текста PDF.
        3. WordJustificationType – перечисление для текста Word.
     2. HelperModels – папка, хранящая классы, необходимые для создания отчётов.
        1. ExcelCellParameters – класс, хранящий данные для клетки Excel.
        2. ExcelInfo – класс, хранящий данные, необходимые для создания отчётов Excel.
        3. ExcelMergeParameters – класс, хранящий данные для объединения клеток Excel.
        4. PdfInfo – класс, хранящий данные, необходимые для создания отчётов PDF.
        5. PdfParagraph – класс, хранящий данные для абзаца Pdf.
        6. PdfRowParameters – класс, хранящий данные для строки Pdf.
        7. WordInfo – класс, хранящий данные, необходимые для создания отчётов Word.
        8. WordParagraph – класс, хранящий данные для абзаца Word.
        9. WordTextProperties – класс, хранящий данные для текста Word.
     3. Implements
        1. SaveToExcel – класс для создания отчёта Excel.
        2. SaveToWord – класс для создания отчёта Word.
        3. SaveToPdf – класс для создания отчёта Pdf.
     4. AbstractSaveToExcel – абстрактный класс для создания отчёта Excel.
     5. AbstractSaveToWord – абстрактный класс для создания отчёта Word.
     6. AbstractSaveToPdf – абстрактный класс для создания отчёта Pdf.

1. AutoFixStationContracts – проект, содержащий в себе классы, необходимые для передачи данных между частями программы.



* 1. BindingModels – папка, содержащая классы binding моделей.
     1. CarBindingModel – класс для авто.
     2. ChangeTOStatusBindingModel – класс для изменения статуса ТО.
     3. ChangeWorkStatusBindingModel – класс для изменения статуса услуги.
     4. CreateTOBindingModel – класс для создания ТО.
     5. CreateWorkBindingModel – класс для создания услуги.
     6. EmployeeBindingModel – класс для работника.
     7. MailConfigBindingModel – класс для конфига данных для отправки письма.
     8. MailSendBindingModel – класс для конфига письма.
     9. MessageInfoBindingModel – класс для сообщения в письме.
     10. ReportBindingModel – класс для отчета.
     11. ServiceRecordBindingModel – класс для записи сервиса.
     12. SparePartBindingModel – класс для запчасти.
     13. StoreKeeperBindingModel – класс для кладовщика.
     14. TimeOfWorkBindingModel – класс для времени выполнения.
     15. TOBindingModel – класс для ТО.
     16. WorkBindingModel – класс для услуги.
     17. WorkTypeBindingModel - класс для типа услуги.
  2. BusinessLogicsContracts – папка, содержащая интерфейсы для бизнес-логики.
     1. ICarLogic – интерфейс для логики авто.
     2. IEmployeeLogic – интерфейс для логики работника.
     3. IReportLogic – интерфейс для логики отчета.
     4. IServiceRecordLogic – интерфейс для логики записи сервиса.
     5. ISparePartLogic - интерфейс для логики запчастей.
     6. IStoreKeeperLogic - интерфейс для логики кладовщика.
     7. ITimeOfWorkLogic - интерфейс для логики времени выполнения.
     8. ITOLogic - интерфейс для логики ТО.
     9. IWorkLogic - интерфейс для логики услуги.
     10. IWorkTypeLogic - интерфейс для логики типа услуги.
  3. Enums – папка, содержащая перечисления.
     1. SparePartStatus – перечисление для типа запчасти.
     2. TOStatus – перечисление для статуса ТО.
     3. UnitMeasurement – перечисление для ед. измерения.
     4. WorkStatus – перечисления для статуса услуги.
  4. StorageContracts – папка, содержащая интерфейсы для хранилищ.
     1. ICarStorage – интерфейс для хранилища авто.
     2. IEmployeeStorage – интерфейс для хранилища работников.
     3. IServiceRecordStorage – интерфейс для хранилища записи сервисов.
     4. ISparePartStorage – интерфейс для хранилища запчастей.
     5. IStoreKeeperStorage – интерфейс для хранилища кладовщиков.
     6. ITimeOfWorkStorage – интерфейс для хранилища времени выполнения.
     7. ITOStorage – интерфейс для хранилища ТО.
     8. IWorkStorage – интерфейс для хранилища услуг.
     9. IWorkTypeStorage - интерфейс для хранилища типа услуг.
  5. ViewModels – папка, содержащая классы для View моделей.
     1. CarViewModel – класс для авто.
     2. EmployeeViewModel - класс для работника.
     3. ReportTOSparePartViewModel - класс для создания отчета ТО с запчастями.
     4. ReportTOsViewModel - класс для создания отчета с ТО за период.
     5. ServiceRecordViewModel - класс для записи сервисов.
     6. SparePartViewModel - класс для запчастей.
     7. StoreKeeperViewModel - класс для кладовщика.
     8. TimeOfWorkViewModel - класс для времени выполнения.
     9. TOViewModel - класс для ТО.
     10. WorkTypeViewModel - класс для типа услуги.
     11. WorkViewModel - класс для услуги.
  6. WebViewModels – папка, содержащая класс для веб-представления модели.
     1. TimeOfWorkWebViewModel – класс для времени выполнения.

1. AutoFixStationDatabaseImplement – проект, содержащий в себе классы, необходимые для функционирования логики хранения данных.
   1. Implements – папка, содержащая классы-реализации хранилищ.
      1. CarStorage – класс-хранилище для авто.
      2. EmployeeStorage – класс-хранилище для работника.
      3. ServireRecordStorage – класс-хранилище для записи сервиса.
      4. SparePartStorage – класс-хранилище для запчасти.
      5. StoreKeeperStorage – класс-хранилище для кладовщика.
      6. TimeOfWorkStorage – класс-хранилище для времени выполнения.
      7. TOStorage – класс-хранилище для ТО.
      8. WorkStorage – класс-хранилище для услуги.
      9. WorkTypeStorage – класс-хранилище для типа услуги.
   2. Migrations – папка, содержащая миграции для БД, а также её снимок.
   3. Models – папка, содержащая классы для БД.
      1. Car – класс для авто.
      2. Employee – класс для работника.
      3. ServiceRecord – класс для записи сервиса.
      4. SparePart – класс для запчасти.
      5. StoreKeeper – класс для кладовщика.
      6. TimeOfWork – класс для времени выполнения.
      7. TO – класс для ТО.
      8. Work – класс для услуги.
      9. WorkType – класс для типа услуги.
      10. WorkType\_SparePart – класс для типа услуги/запчасти.
   4. AutoFixStationDatabase – класс для взаимодействия с БД.
2. AutoFixStationEmployeeWebApp – проект, представляющий собой веб-приложение.
   1. Controllers – папка, содержащая классы для контроллеров.
      1. HomeController – класс, представляющий собой контроллер для взаимодействия со всей логиков веб-приложения.
      2. ReportController – класс, представляющий собой контроллер для взаимодействия с логикой отчётов.
   2. Models – папка, содержащая модели.
      1. ErrorViewModel – класс, содержащий модель для ошибок.
   3. Views – папка для представлений страниц. Здесь содержатся страницы веб-сайта (их разметка).
   4. APIEmployee – класс, API для работника.
   5. Program – главный класс для запуска проекта.
   6. Startup – класс с настройками проекта.
3. AutoFixStationRestAPI – проект, содержащий логику для API (взаимодействие между веб-приложением и сервером).
   1. Controllers – папка, содержащая контроллеры.
      1. CarController – класс-контроллер для авто.
      2. EmployeeController – класс-контроллер для работника.
      3. ReportController – класс-контроллер для отчётов.
      4. SparePartController – класс-контроллер для запчастей.
      5. StoreKeeperController – класс-контроллер для кладовщика.
      6. TOController – класс-контроллер для ТО.
      7. WorkController – класс-контроллер для услуги.
   2. Program – главный класс для запуска API.
   3. Startup – класс с настройками проекта.

# **Глава 4. Руководство пользователя и результат работы**

Авторизация

Для авторизации пользователь должен заполнить логин и пароль. Данные отправляются на сервер. Там они проверяются на корректность и на наличие такого пользователя. Если такой пользователь существует, то приходит положительный ответ от сервера и происходит переход в главное меню.

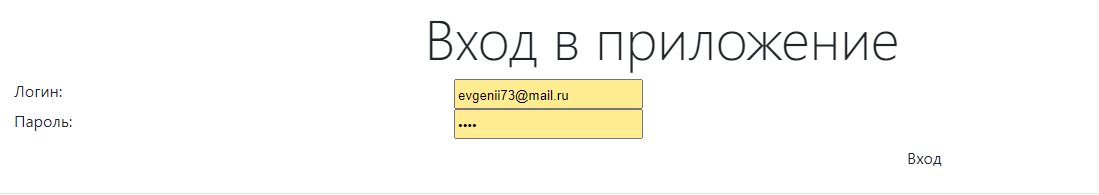


Рисунок 18. Авторизация

Начальный экран. После входа в аккаунт появляется главная страница.

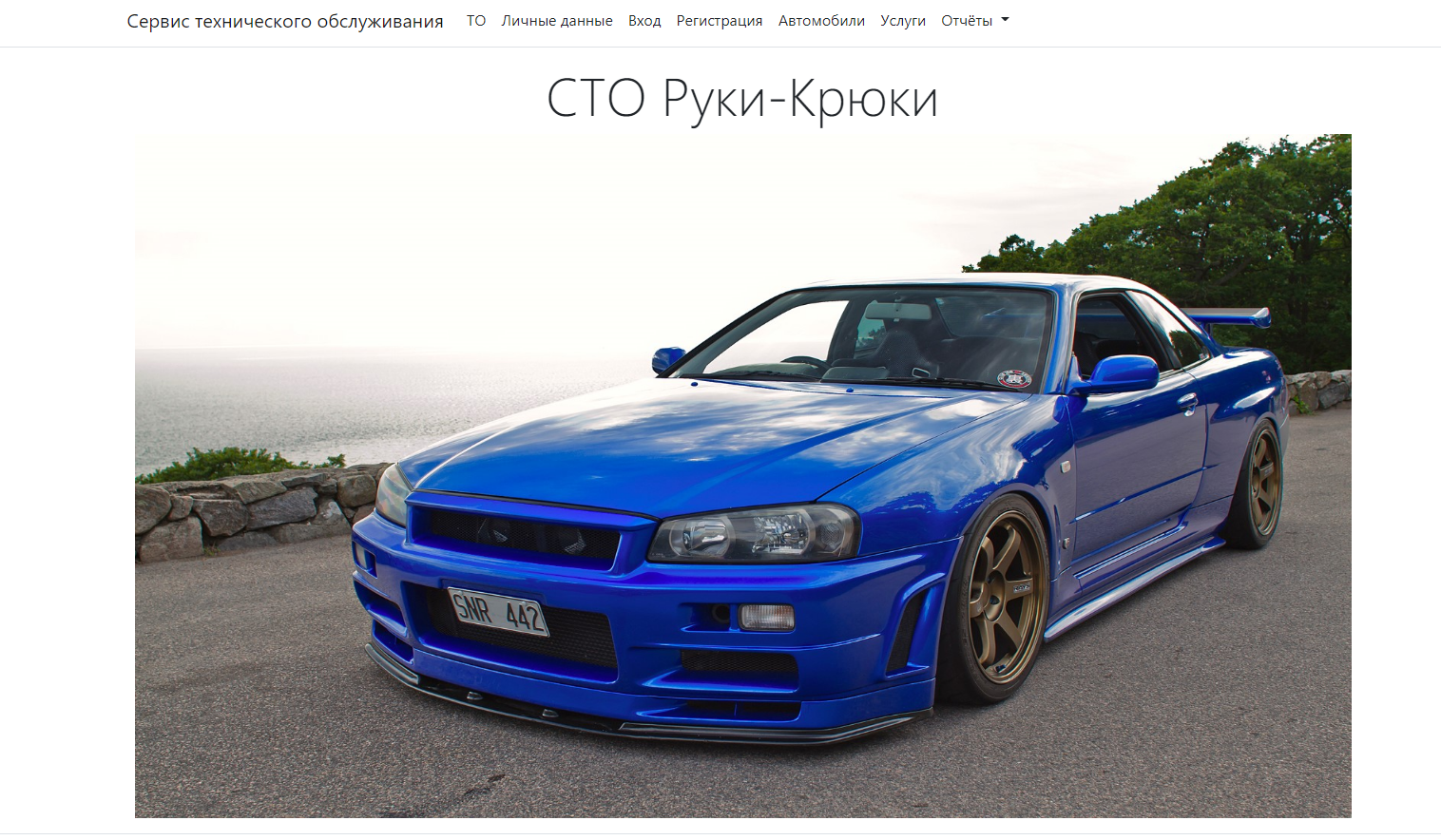


Рисунок 16. Главный экран

Регистрация

Для регистрации пользователь должен заполнить:

1. Логин
2. ФИО
3. Пароль

Затем эти данные отправляются на сервер. При корректных данных приходит положительный ответ, что пользователь зарегистрировался.

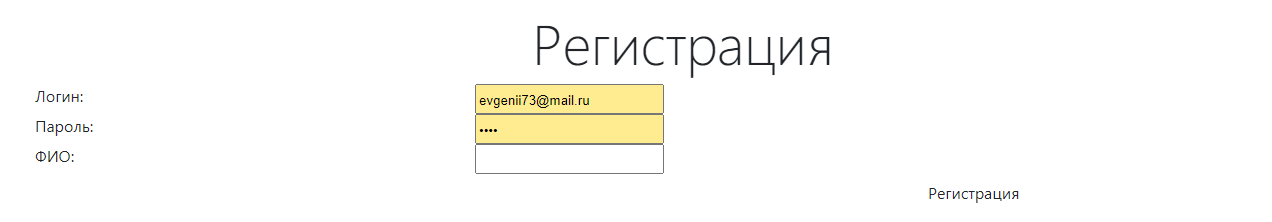


Рисунок 17. Фрагмент регистрации

Главное меню

Показывает список ТО, созданных работником. Они подгружаются с сервера. Содержит кнопку для создания нового ТО и перехода на такой экран. Содержит кнопку для просмотра информации о ТО и кнопку для изменения его статуса.

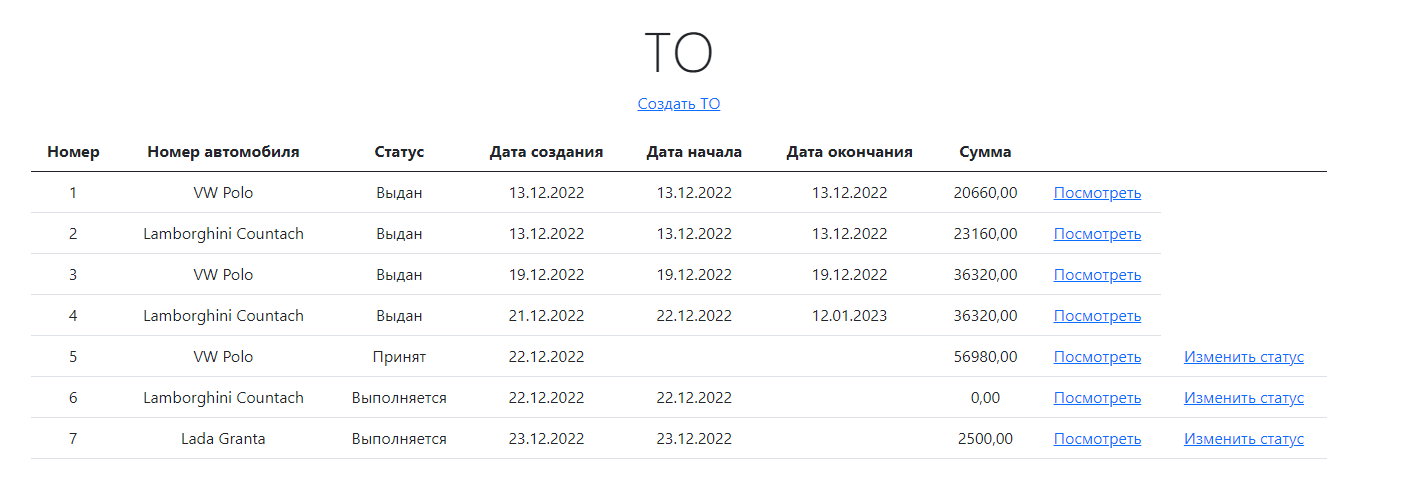


Рисунок 19. Тех. осмотры

Создание ТО

Подгружаются данные о существующих автомобилях. Необходимо выбрать автомобиль. При нажатии на кнопку “Создать” происходит сохранение ТО. Посылается запрос на создание. Если данные корректны, то приходит положительный ответ и данные записываются на сервер в хранилище.

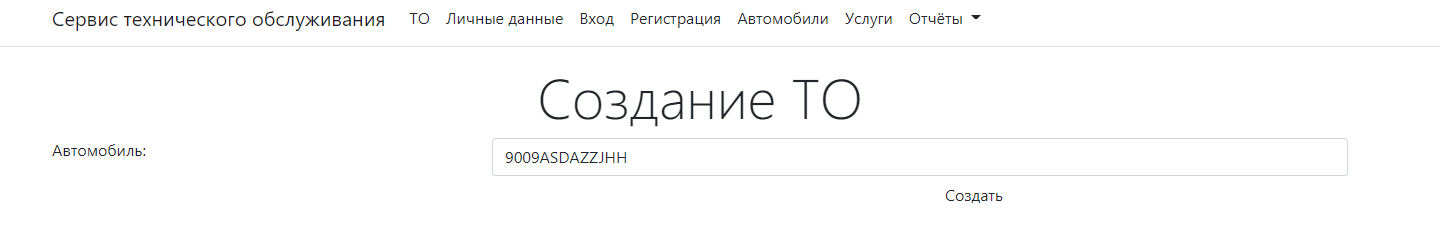


Рисунок 20. Создание ТО.

Просмотр ТО

При нажатии на соответствующую кнопку, происходит переход в этот экран. Подгружаются данные с сервера об этом ТО. Видно услуги, которые включены в ТО.

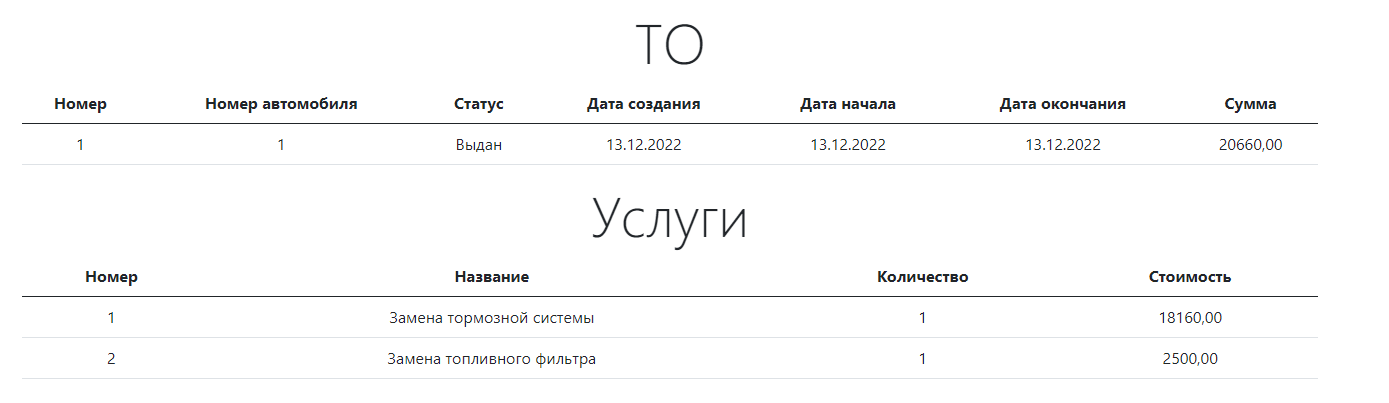


Рисунок 21. Просмотр ТО.

Изменение статуса ТО

Можно изменить статус ТО в соответствии с его текущим статусом. Если такое сделать, то произойдет запрос на сервер, который, в случае корректных данных изменит статус ТО, иначе придет отрицательный ответ и ничего с данными не произойдёт.

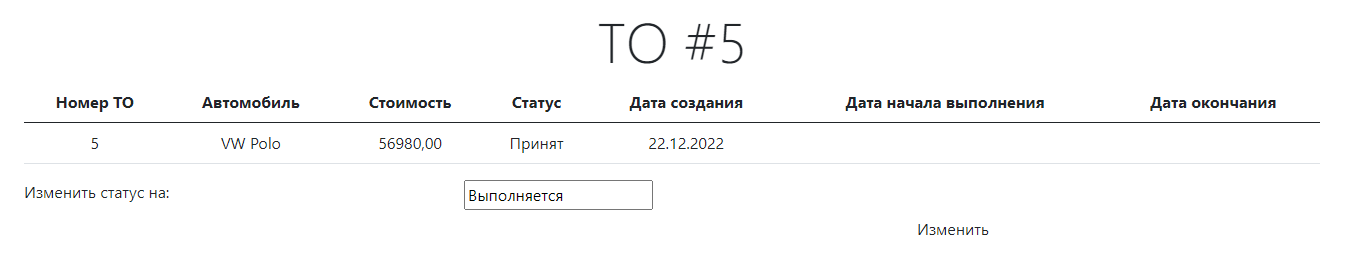


Рисунок 21. Просмотр ТО.

Экран просмотра автомобилей

При нажатии на пункт меню “Автомобили” происходит переход на данный экран. Подгружаются данные с сервера о существующих в системе авто. Здесь можно увидеть текущих клиентов СТО. При нажатии на кнопку добавления, происходит переход в фрагмент добавления нового клиента. При нажатии на кнопку “Изменить” происходит переход на экран просмотра авто и его записей.

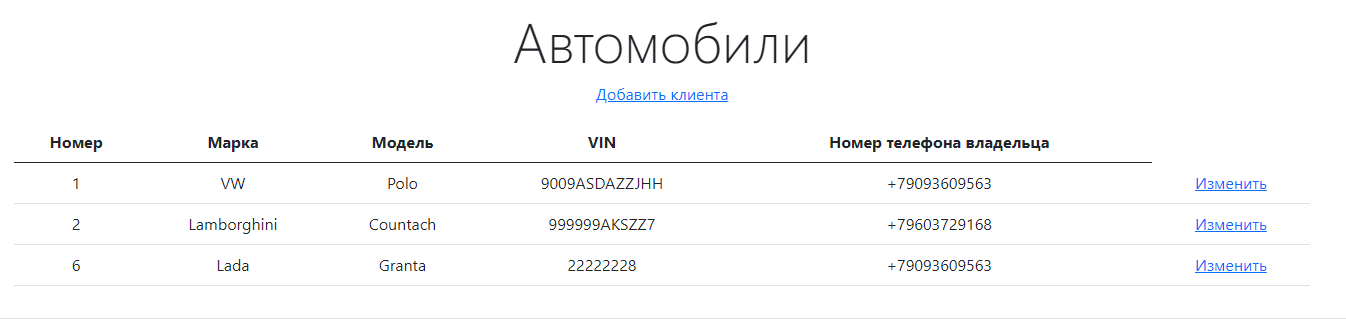


Рисунок 22. Просмотр авто

Экран добавления авто

При нажатии на кнопку происходит переход в этом фрагмент. При нажатии на кнопку “Добавить” происходит отправка запроса о создании новой записи на сервер, если данные корректны, то придёт положительный ответ и запись создастся, иначе ничего не произойдет. Произойдет переход на предыдущий экран. Необходимо заполнить:

1. Марку
2. Модель
3. VIN
4. Номер телефона владельца

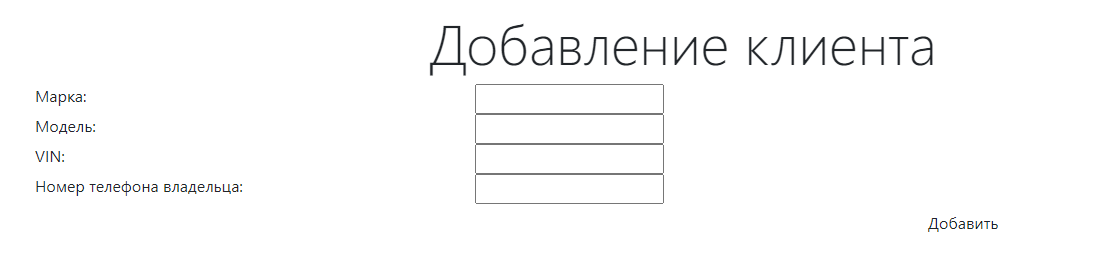


Рисунок 23. Добавление авто

Экран просмотра/изменения информации об авто

При нажатии на кнопку “Изменить” происходит переход на этот экран. Подгружаются данные об этом автомобиле. Поля заполняются данными. Показываются записи сервисов данного авто. При нажатии на кнопку “Изменить” пункта записи сервисов происходит переход на экран редактирования записи сервисов. При нажатии на кнопку “Изменить” произойдет запрос на сервер об изменении существующей сущности, если данные корректны, то поступит положительный ответ и сущности обновятся, иначе ничего не произойдёт.

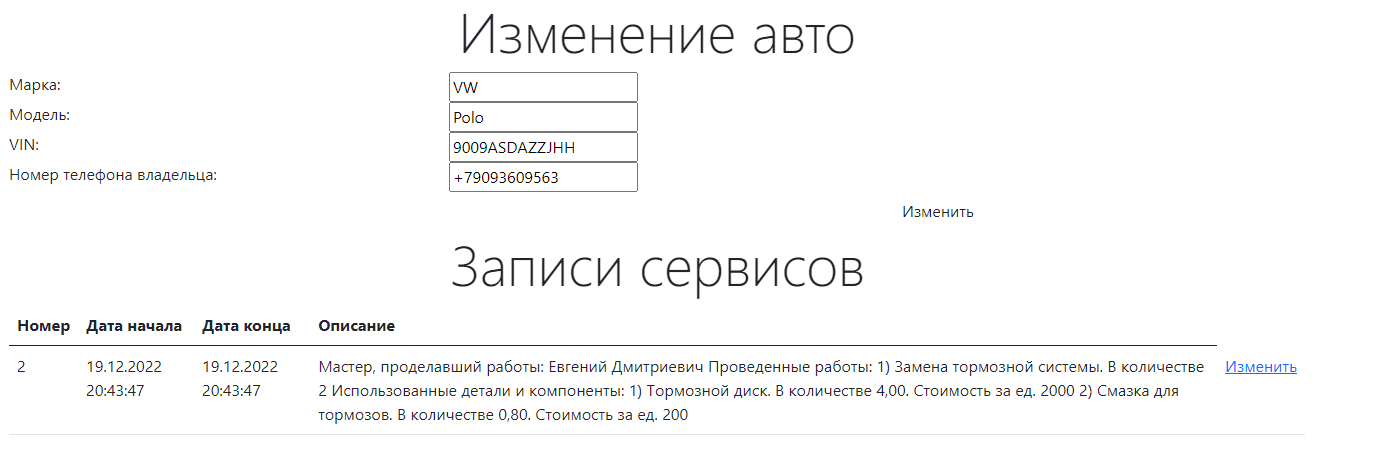


Рисунок 24. Редактирование авто

Экран редактирования записи сервисов

Происходит переход на данный экран, показывающий информацию о выбранном пункте записи сервиса. Данные о записях подгружаются с сервера. Поля заполняются данными. Можно изменять описание. При нажатии на кнопку “Изменить” происходит запрос на сохранения внесенных изменений, если данные корректны, то придёт положительный ответ и запись обновится, иначе ничего не произойдет. Далее будет переход на предыдущий экран.

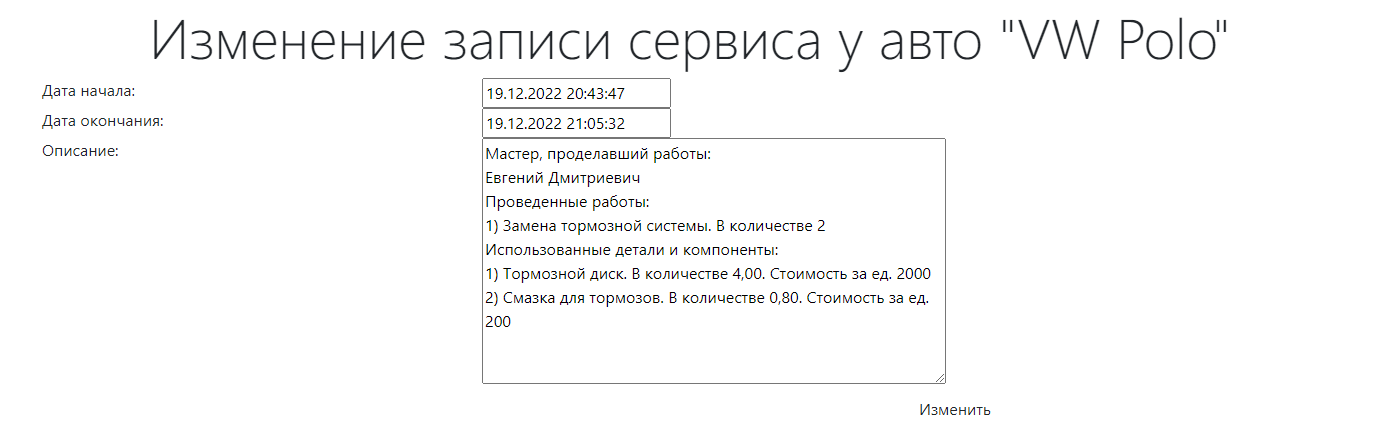


Рисунок 25. Изменение записи сервиса

Экран просмотра услуг

При нажатии на соответствующий пункт меню происходит переход на этот экран. Подгружаются данные с сервера об услугах. При нажатии на кнопку “Создать услугу” происходит переход на экран создания услуги. При нажатии на кнопку “Посмотреть” пункта таблицы происходит переход на экран просмотра информации о ней.



Рисунок 26. Услуги

Экран создания услуги

Необходимо заполнить требуемые поля и выбрать ТО, тип услуги и кладовщика. При вводе количества будет подсчитана стоимость работы. При нажатии на кнопку добавить произойдет запрос на сервер на создание услуги, если данные корректны, то придёт положительный ответ и данные сохранятся, иначе ничего не произойдет. Произойдет переход на предыдущий экран.

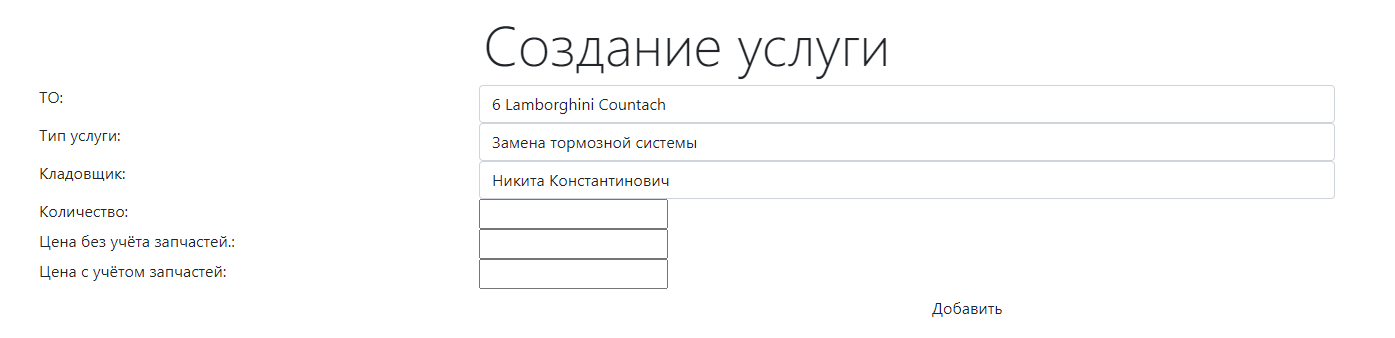


Рисунок 27. Создание услуги

Экран просмотра информации об услуге

При нажатии на кнопку “Посмотреть” происходит переход на данный экран. Подгружаются данные об услуге с сервера. Показывается информация об услуге, требуемые для нее запчасти и её статус.

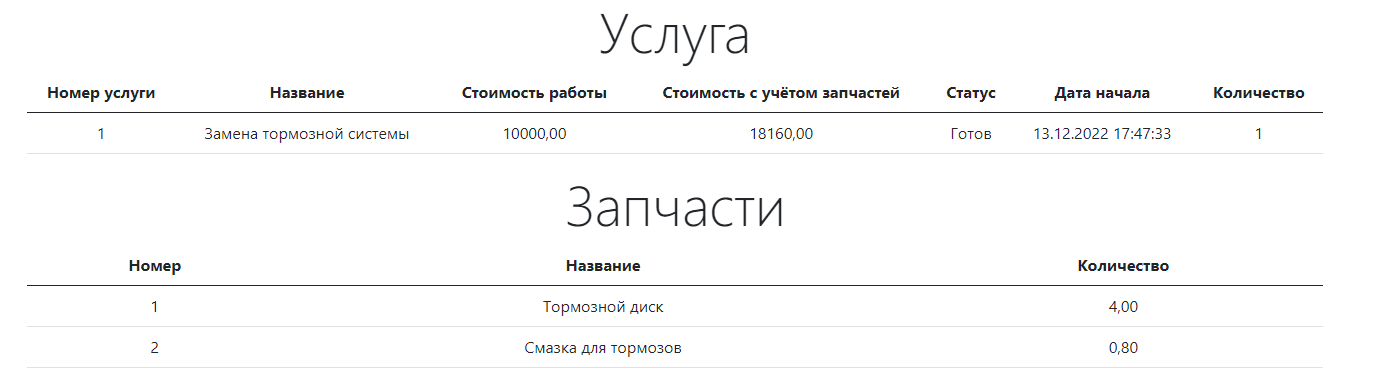


Рисунок 28. Просмотр услуги

Экран просмотра информации об аккаунте

При нажатии на соответствующий пункт меню происходит переход на экран данных о профиле. Подгружаются данные об аккаунте с сервера. Здесь показаны данные вашего аккаунта. При нажатии на кнопку “Сохранить” произойдет запрос на изменение данных об аккаунте, если данные корректны, то придёт положительный ответ и данные изменятся, иначе ничего не произойдет. При нажатии на кнопку “Выйти” произойдет выход из аккаунта и переход на экран авторизации.

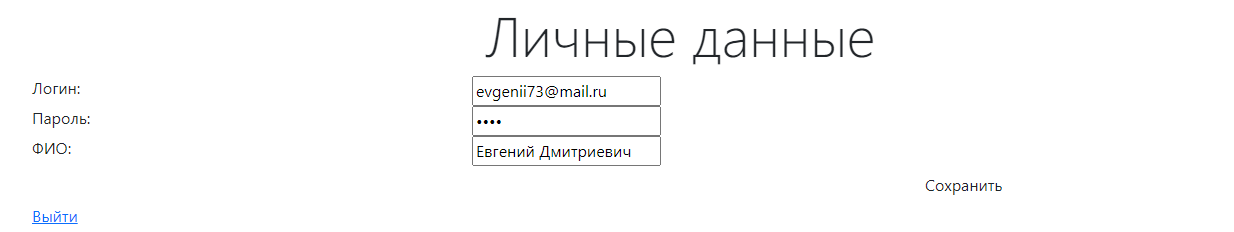


Рисунок 29. Личные данные

Экран создания отчёта Word/Excel

При выборе соответствующего пункта меню произойдет переход на экран выбора тех. осмотров, по которым требуется отчёт.

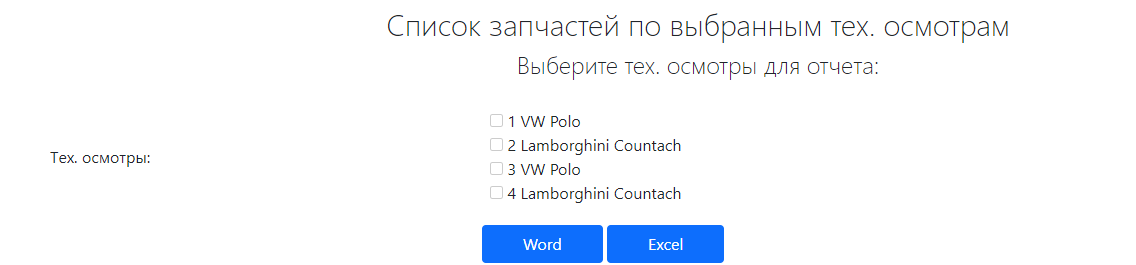


Рисунок 30. Создание отчёта Word/Excel

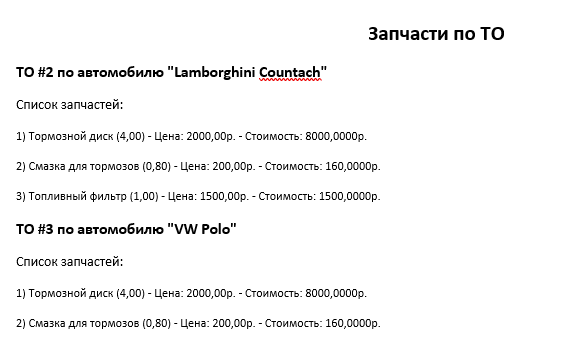


Рисунок 31. Word-отчёт

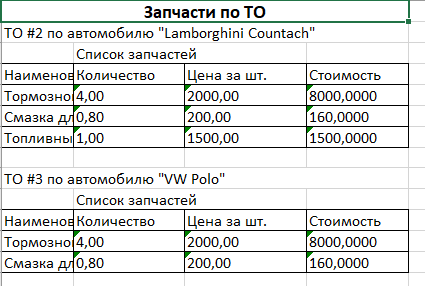


Рисунок 31. Excel-отчёт

Экран создания отчёта PDF

При выборе соответствующего пункта меню произойдет переход на экран выбора тех. осмотров, по которым требуется отчёт. Требуется указать период и выбрать кнопку или “Вывести на форму”, или “Отправить на почту”. Произойдет соответствующее действие.

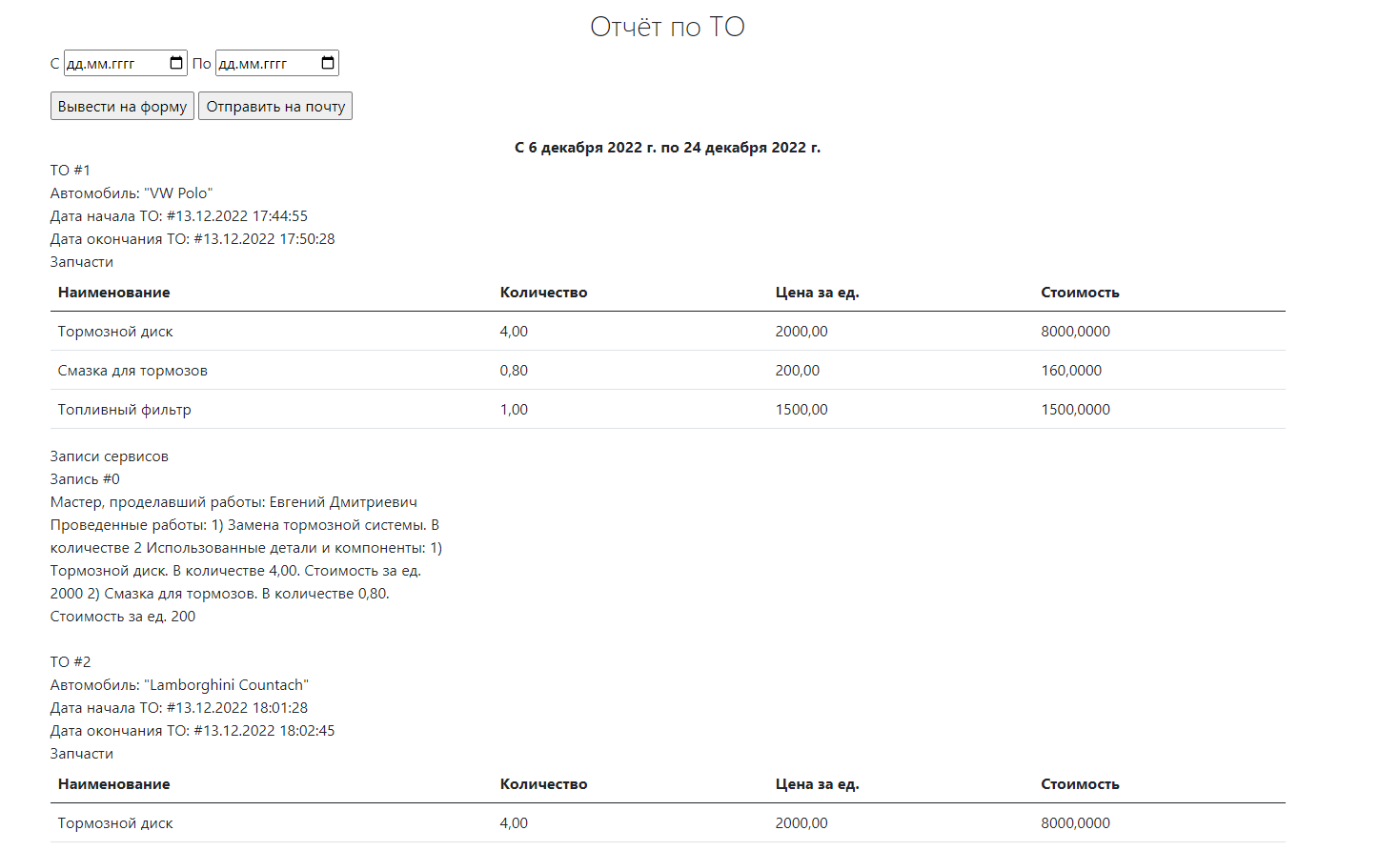


Рисунок 32. Вывод отчёта на экран

# **Заключение**

В процессе создания курсовой работы была проанализирована предметная область, были использованы возможности среды разработки Visual Studio, создан дизайн интерфейса, продуманы возможные действия пользователя, созданы диаграммы, созданы шаблоны возможных отчетов, разработан программный продукт.

Также были улучшено знание языка C#, улучшено понимание фреймворков ASP .NET Core и EntityFrameWork. Было улучшено знание о сериализации и десериализации с помощью библиотеки Newtonsoft.Json. Получено знание о взаимодействии с сервером через клиент на другом устройстве, произведена работа по создания файлов формата PDF, Word и Excel.

# **Список литературы**

1. Общие сведения об ASP.NET Core [Электронный ресурс] / Режим доступа: https://learn.microsoft.com/ru-ru/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0
2. Е.Н. ЭГОВ - РАЗРАБОТКА ПРОГРАММЫ УЧЕТА ЛИТЕРАТУРЫ В АБСТРАКТНОМ УНИВЕРСИТЕТЕ практикум по дисциплине «Технологии программирования» - Ульяновск, 2021
3. Электронный ресурс – Дистанционное образование УлГТУ Режим доступа: <https://lms.ulstu.ru/course/view.php?id=364> Дата обращения: 27.05.2022
4. Электронный ресурс – METANIT.COM

Режим доступа: <https://metanit.com/sharp/efcore/> Дата обращения: 13.05.2022

1. Электронный ресурс – LiveCharts

Режим доступа: <https://lvcharts.net/> Дата обращения: 18.05.2022

1. Наместников А.М., Филиппов А.А.: «Базы данных. Практический курс. Объектно-реляционные базы данных на примере PostgreSQL 9.5»: Учеб. пособие для вузов. — Ульяновск: УлГТУ, 2016.
2. [Курс лекций](https://www.w3resource.com/sql-exercises/employee-database-exercise/index.php) по дисциплине «Базы данных»
3. Электронный ресурс – metanit.com

Режим доступа: <https://metanit.com/>

# **Приложение**

Листинг кода

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.BusinessLogics

{

public class CarLogic : ICarLogic

{

private readonly ICarStorage \_carStorage;

public CarLogic(ICarStorage carStorage)

{

\_carStorage = carStorage;

}

public void CreateOrUpdate(CarBindingModel model)

{

var element = \_carStorage.GetElement(new CarBindingModel

{

VIN = model.VIN

});

if (element != null && element.Id != model.Id)

{

throw new Exception("Уже есть автомобиль с таким VIN-номером");

}

if (model.Id.HasValue)

{

\_carStorage.Update(model);

}

else

{

\_carStorage.Insert(model);

}

}

public void Delete(CarBindingModel model)

{

var element = \_carStorage.GetElement(new CarBindingModel

{

Id = model.Id

});

if (element == null)

{

throw new Exception("Элемент не найден");

}

\_carStorage.Delete(model);

}

public List<CarViewModel> Read(CarBindingModel model)

{

if (model == null)

{

return \_carStorage.GetFullList();

}

if (model.Id.HasValue)

{

return new List<CarViewModel>()

{

\_carStorage.GetElement(model)

};

}

return \_carStorage.GetFilteredList(model);

}

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.BusinessLogics

{

public class EmployeeLogic : IEmployeeLogic

{

private readonly IEmployeeStorage \_employeeStorage;

public EmployeeLogic(IEmployeeStorage employeeStorage)

{

\_employeeStorage = employeeStorage;

}

public void CreateOrUpdate(EmployeeBindingModel model)

{

var element = \_employeeStorage.GetElement(new EmployeeBindingModel

{

Login = model.Login,

});

if (element != null && element.Id != model.Id)

{

throw new Exception("Уже есть работник с таким логином");

}

if (model.Id.HasValue)

{

\_employeeStorage.Update(model);

}

else

{

\_employeeStorage.Insert(model);

}

}

public void Delete(EmployeeBindingModel model)

{

var element = \_employeeStorage.GetElement(new EmployeeBindingModel

{

Id = model.Id

});

if (element == null)

{

throw new Exception("Элемент не найден");

}

\_employeeStorage.Delete(model);

}

public List<EmployeeViewModel> Read(EmployeeBindingModel model)

{

if (model == null)

{

return \_employeeStorage.GetFullList();

}

if (model.Id.HasValue)

{

return new List<EmployeeViewModel>()

{

\_employeeStorage.GetElement(model)

};

}

return \_employeeStorage.GetFilteredList(model);

}

}

}

using AutoFixStationBusinessLogic.OfficePackage;

using AutoFixStationBusinessLogic.OfficePackage.HelperModels;

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.BusinessLogics

{

public class ReportLogic : IReportLogic

{

private readonly AbstractSaveToWord \_saveToWord;

private readonly AbstractSaveToExcel \_saveToExcel;

private readonly AbstractSaveToPdf \_saveToPdf;

private readonly ISparePartStorage \_sparePartStorage;

private readonly IServiceRecordStorage \_serviceRecordStorage;

private readonly ITOStorage \_tOStorage;

private readonly IWorkStorage \_workStorage;

private readonly IWorkTypeStorage \_workTypeStorage;

private readonly ICarStorage \_carStorage;

public ReportLogic(AbstractSaveToWord saveToWord,

AbstractSaveToExcel saveToExcel,

AbstractSaveToPdf saveToPdf,

ISparePartStorage sparePartStorage,

IServiceRecordStorage serviceRecordStorage,

ITOStorage tOStorage,

IWorkStorage workStorage,

IWorkTypeStorage workTypeStorage,

ICarStorage carStorage)

{

\_saveToWord = saveToWord;

\_saveToExcel = saveToExcel;

\_saveToPdf = saveToPdf;

\_sparePartStorage = sparePartStorage;

\_serviceRecordStorage = serviceRecordStorage;

\_tOStorage = tOStorage;

\_workStorage = workStorage;

\_workTypeStorage = workTypeStorage;

\_carStorage = carStorage;

}

public List<ReportTOsViewModel> GetTOs(ReportBindingModel model)

{

var list = new List<ReportTOsViewModel>();

var tos = \_tOStorage.GetFilteredList(new TOBindingModel

{

DateCreate = model.DateFrom.Value,

DateOver = model.DateTo

}).Where(x => x.Status.Equals("Выдан"));

foreach (var to in tos)

{

var record = new ReportTOsViewModel

{

DateBegin = to.DateCreate,

DateEnd = to.DateOver.Value,

CarName = to.CarName,

TOId = to.Id

};

//Получаем записи

var car = \_carStorage.GetElement(new CarBindingModel { Id = to.CarId });

var list\_records = new List<string>();

foreach(var sr in car.Records)

{

list\_records.Add(sr.Value.Item2);

}

record.ServiceRecords = list\_records;

//Получаем запчасти

var list\_parts = new Dictionary<int, (string, decimal, decimal)>();

foreach (var workId in to.Works.Keys)

{

var work = \_workStorage.GetElement(new WorkBindingModel

{

Id = workId

});

var worktype = \_workTypeStorage.GetElement(new WorkTypeBindingModel

{

Id = work.WorkTypeId

});

foreach (var part in worktype.WorkSpareParts)

{

if (list\_parts.ContainsKey(part.Key))

{

list\_parts[part.Key] = (list\_parts[part.Key].Item1, list\_parts[part.Key].Item2 + part.Value.Item2, list\_parts[part.Key].Item3);

}

else

{

list\_parts.Add(part.Key, part.Value);

}

}

}

record.SpareParts = list\_parts;

list.Add(record);

}

return list;

}

public List<ReportTOSparePartViewModel> GetTOSparePart(ReportBindingModel model)

{

var tos = model.TOs;

var list = new List<ReportTOSparePartViewModel>();

foreach (var tO in tos)

{

var record = new ReportTOSparePartViewModel

{

CarName = tO.CarName,

TOId = tO.Id

};

var list\_parts = new Dictionary<int, (string, decimal, decimal)>();

foreach(var workId in tO.Works.Keys)

{

var work = \_workStorage.GetElement(new WorkBindingModel

{

Id = workId

});

var worktype = \_workTypeStorage.GetElement(new WorkTypeBindingModel

{

Id = work.WorkTypeId

});

foreach(var part in worktype.WorkSpareParts)

{

if (list\_parts.ContainsKey(part.Key))

{

list\_parts[part.Key] = (list\_parts[part.Key].Item1, list\_parts[part.Key].Item2 + part.Value.Item2, list\_parts[part.Key].Item3);

}

else

{

list\_parts.Add(part.Key, part.Value);

}

}

}

record.SpareParts = list\_parts;

list.Add(record);

}

return list;

}

public void SaveTOsByDateToPdfFile(ReportBindingModel model)

{

\_saveToPdf.CreateReportTOsByDate(new PdfInfo

{

FileName = model.FileName,

Title = $"Сведения по ТО за период с {model.DateFrom} по {model.DateTo}",

DateFrom = model.DateFrom.Value,

DateTo = model.DateTo.Value,

TOs = GetTOs(model)

});

}

public void SaveTOSparePartToExcelFile(ReportBindingModel model)

{

\_saveToExcel.CreateReportTOSpareParts(new ExcelInfo

{

FileName = model.FileName,

Title = "Запчасти по ТО",

TOSpareParts = GetTOSparePart(model)

});

}

public void SaveTOSparePartToWordFile(ReportBindingModel model)

{

\_saveToWord.CreateReportTOSpareParts(new WordInfo

{

FileName = model.FileName,

Title = "Запчасти по ТО",

TOSpareParts = GetTOSparePart(model)

});

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

namespace AutoFixStationBusinessLogic.BusinessLogics

{

public class ServiceRecordLogic : IServiceRecordLogic

{

private readonly IServiceRecordStorage \_storage;

private readonly IEmployeeStorage \_employeeStorage;

private readonly IWorkTypeStorage \_workTypeStorage;

public ServiceRecordLogic(IServiceRecordStorage storage,

IEmployeeStorage employeeStorage,

IWorkTypeStorage workTypeStorage)

{

\_storage = storage;

\_employeeStorage = employeeStorage;

\_workTypeStorage = workTypeStorage;

}

public void Create(ServiceRecordBindingModel record,

TOBindingModel tO)

{

var element = \_storage.GetElement(new ServiceRecordBindingModel

{

CarId = record.CarId

});

if (element != null && element.Id != record.Id)

{

throw new Exception("Уже есть такая запись");

}

record.DateBegin = tO.DateCreate;

record.DateEnd = tO.DateOver.Value;

string empName = \_employeeStorage

.GetElement(new EmployeeBindingModel { Id = tO.EmployeeId }).FIO;

string worksStr = "Мастер, проделавший работы: \n"

+ empName + "\nПроведенные работы:\n";

int i = 1;

foreach(var work in tO.Works)

{

worksStr += i + ") " + work.Value.Item1 + ". В количестве " + work.Value.Item2.Item1 + "\n";

i++;

}

var neededParts = new Dictionary<string, (decimal, decimal)>();

foreach (var work in tO.Works)

{

var worktype = \_workTypeStorage.GetElement(new WorkTypeBindingModel { WorkName = work.Value.Item1 });

foreach (var parts in worktype.WorkSpareParts)

{

if (!neededParts.ContainsKey(parts.Value.Item1))

neededParts.Add(parts.Value.Item1, (parts.Value.Item2, parts.Value.Item3));

else

{

neededParts[parts.Value.Item1] = (neededParts[parts.Value.Item1].Item1 + parts.Value.Item2, parts.Value.Item3);

}

}

}

string sparePartsStr = "Использованные детали и компоненты: \n";

i = 1;

foreach(var part in neededParts)

{

sparePartsStr += i + ") " + part.Key + ". В количестве " + part.Value.Item1 + ". Стоимость за ед. " + (int)part.Value.Item2 + "\n";

i++;

}

record.Description = worksStr + sparePartsStr;

if (record.Id.HasValue)

{

\_storage.Update(record);

}

else

{

\_storage.Insert(record);

}

}

public void Delete(ServiceRecordBindingModel model)

{

var element = \_storage.GetElement(new ServiceRecordBindingModel

{

Id = model.Id

});

if (element == null)

{

throw new Exception("Элемент не найден");

}

\_storage.Delete(model);

}

public List<ServiceRecordViewModel> Read(ServiceRecordBindingModel model)

{

if (model == null)

{

return \_storage.GetFullList();

}

if (model.Id.HasValue)

{

return new List<ServiceRecordViewModel>()

{

\_storage.GetElement(model)

};

}

return \_storage.GetFilteredList(model);

}

public void Update(ServiceRecordBindingModel model)

{

var element = \_storage.GetElement(new ServiceRecordBindingModel

{

CarId = model.CarId

});

if (element != null && element.Id != model.Id)

{

throw new Exception("Уже есть такая запись");

}

\_storage.Update(model);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

namespace AutoFixStationBusinessLogic.BusinessLogics

{

public class SparePartLogic : ISparePartLogic

{

private readonly ISparePartStorage \_storage;

public SparePartLogic(ISparePartStorage storage)

{

\_storage = storage;

}

public void CreateOrUpdate(SparePartBindingModel model)

{

var element = \_storage.GetElement(new SparePartBindingModel

{

Name = model.Name

});

if (element != null && element.Id != model.Id)

{

throw new Exception("Уже есть деталь с таким названием");

}

if (model.Id.HasValue)

{

\_storage.Update(model);

}

else

{

\_storage.Insert(model);

}

}

public void Delete(SparePartBindingModel model)

{

var element = \_storage.GetElement(new SparePartBindingModel

{

Id = model.Id

});

if (element == null)

{

throw new Exception("Элемент не найден");

}

\_storage.Delete(model);

}

public List<SparePartViewModel> Read(SparePartBindingModel model)

{

if (model == null)

{

return \_storage.GetFullList();

}

if (model.Id.HasValue)

{

return new List<SparePartViewModel>

{

\_storage.GetElement(model)

};

}

return \_storage.GetFilteredList(model);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

namespace AutoFixStationBusinessLogic.BusinessLogics

{

public class StoreKeeperLogic : IStoreKeeperLogic

{

private readonly IStoreKeeperStorage \_storeKeeperStorage;

public StoreKeeperLogic(IStoreKeeperStorage storeKeeperStorage)

{

\_storeKeeperStorage = storeKeeperStorage;

}

public void CreateOrUpdate(StoreKeeperBindingModel model)

{

var element = \_storeKeeperStorage.GetElement(new StoreKeeperBindingModel

{

Login = model.Login,

});

if (element != null && element.Id != model.Id)

{

throw new Exception("Уже есть кладовщик с таким логином");

}

if (model.Id.HasValue)

{

\_storeKeeperStorage.Update(model);

}

else

{

\_storeKeeperStorage.Insert(model);

}

}

public void Delete(StoreKeeperBindingModel model)

{

var element = \_storeKeeperStorage.GetElement(new StoreKeeperBindingModel

{

Id = model.Id

});

if (element == null)

{

throw new Exception("Элемент не найден");

}

\_storeKeeperStorage.Delete(model);

}

public List<StoreKeeperViewModel> Read(StoreKeeperBindingModel model)

{

if (model == null)

{

return \_storeKeeperStorage.GetFullList();

}

if (model.Id.HasValue)

{

return new List<StoreKeeperViewModel>()

{

\_storeKeeperStorage.GetElement(model)

};

}

return \_storeKeeperStorage.GetFilteredList(model);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

namespace AutoFixStationBusinessLogic.BusinessLogics

{

public class TimeOfWorkLogic : ITimeOfWorkLogic

{

private readonly ITimeOfWorkStorage \_storage;

public TimeOfWorkLogic(ITimeOfWorkStorage storage)

{

\_storage = storage;

}

public void CreateOrUpdate(TimeOfWorkBindingModel model)

{

var element = \_storage.GetElement(new TimeOfWorkBindingModel

{

Hours = model.Hours,

Mins = model.Mins

});

if (element != null && element.Id != model.Id)

{

throw new Exception("Уже есть время выполнения с такими параметрами");

}

if (model.Id.HasValue)

{

\_storage.Update(model);

}

else

{

\_storage.Insert(model);

}

}

public void Delete(TimeOfWorkBindingModel model)

{

var element = \_storage.GetElement(new TimeOfWorkBindingModel

{

Id = model.Id

});

if (element == null)

{

throw new Exception("Элемент не найден");

}

\_storage.Delete(model);

}

public List<TimeOfWorkViewModel> Read(TimeOfWorkBindingModel model)

{

if (model == null)

{

return \_storage.GetFullList();

}

if (model.Id.HasValue)

{

return new List<TimeOfWorkViewModel>()

{

\_storage.GetElement(model)

};

}

return \_storage.GetFilteredList(model);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.Enums;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

namespace AutoFixStationBusinessLogic.BusinessLogics

{

public class TOLogic : ITOLogic

{

private readonly ITOStorage \_tOStorage;

private readonly IWorkStorage \_workStorage;

private readonly IServiceRecordLogic \_serviceRecordLogic;

public TOLogic(ITOStorage tOStorage,

IWorkStorage workStorage,

IServiceRecordLogic serviceRecordLogic)

{

\_tOStorage = tOStorage;

\_workStorage = workStorage;

\_serviceRecordLogic = serviceRecordLogic;

}

public void CreateTO(CreateTOBindingModel model)

{

\_tOStorage.Insert(new TOBindingModel

{

CarId = model.CarId,

EmployeeId = model.EmployeeId,

Sum = model.Sum.HasValue ? model.Sum.Value : 0,

Status = TOStatus.Принят,

DateCreate = DateTime.Now,

Works = model.Works

});

}

public void FinishTO(ChangeTOStatusBindingModel model)

{

var tO = \_tOStorage.GetElement(new TOBindingModel

{

Id = model.TOId

});

if (tO == null)

{

throw new Exception("ТО не найдено");

}

if (tO.Status != Enum.GetName(typeof(TOStatus), 1))

{

throw new Exception("ТО не в статусе \"Выполняется\"");

}

bool flag = true;

foreach (var stat in \_workStorage.GetFilteredList(new WorkBindingModel

{

TOId = tO.Id

}))

{

if (stat.WorkStatus != Enum.GetName(typeof(WorkStatus), 2))

{

flag = false;

}

}

if (!flag)

{

throw new Exception("Не все работы в статусе \"Готов\".");

}

\_tOStorage.Update(new TOBindingModel

{

Id = tO.Id,

CarId = tO.CarId,

EmployeeId = tO.EmployeeId,

Sum = tO.Sum.HasValue ? tO.Sum.Value : 0,

Status = TOStatus.Готов,

DateCreate = tO.DateCreate,

DateImplement = tO.DateImplement,

DateOver = DateTime.Now,

Works = tO.Works

});

}

public void IssueTO(ChangeTOStatusBindingModel model)

{

var tO = \_tOStorage.GetElement(new TOBindingModel

{

Id = model.TOId

});

if (tO == null)

{

throw new Exception("ТО не найдено");

}

if (tO.Status != Enum.GetName(typeof(TOStatus), 2))

{

throw new Exception("ТО не в статусе \"Готов\"");

}

\_serviceRecordLogic.Create(new ServiceRecordBindingModel

{

CarId = tO.CarId

}, new TOBindingModel

{

Id = tO.Id,

CarId = tO.CarId,

DateCreate = tO.DateCreate,

DateImplement=tO.DateImplement,

DateOver = tO.DateOver,

EmployeeId = tO.EmployeeId,

Status = (TOStatus) Enum.Parse(typeof(TOStatus), tO.Status),

Sum = tO.Sum.Value,

Works = tO.Works

});

\_tOStorage.Update(new TOBindingModel

{

Id = tO.Id,

CarId = tO.CarId,

EmployeeId = tO.EmployeeId,

Sum = tO.Sum.HasValue ? tO.Sum.Value : 0,

Status = TOStatus.Выдан,

DateCreate = tO.DateCreate,

DateImplement = tO.DateImplement,

DateOver = tO.DateOver,

Works = tO.Works

});

}

public List<TOViewModel> Read(TOBindingModel model)

{

if (model == null)

{

return \_tOStorage.GetFullList();

}

if (model.Id.HasValue)

{

return new List<TOViewModel>

{

\_tOStorage.GetElement(model)

};

}

return \_tOStorage.GetFilteredList(model);

}

public void TakeTOInWork(ChangeTOStatusBindingModel model)

{

var tO = \_tOStorage.GetElement(new TOBindingModel

{

Id = model.TOId

});

if (tO == null)

{

throw new Exception("ТО не найдено");

}

if (tO.Status != Enum.GetName(typeof(TOStatus), 0))

{

throw new Exception("ТО не в статусе \"Принят\"");

}

\_tOStorage.Update(new TOBindingModel

{

Id = tO.Id,

CarId = tO.CarId,

EmployeeId = tO.EmployeeId,

Sum = tO.Sum.HasValue ? tO.Sum.Value : 0,

Status = TOStatus.Выполняется,

DateCreate = tO.DateCreate,

DateImplement = DateTime.Now,

Works = tO.Works

});

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.Enums;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

namespace AutoFixStationBusinessLogic.BusinessLogics

{

public class WorkLogic : IWorkLogic

{

private readonly IWorkStorage \_workStorage;

private readonly IWorkTypeStorage \_workTypeStorage;

private readonly ITimeOfWorkStorage \_timeOfWorkStorage;

private readonly ITOStorage \_tOStorage;

public WorkLogic(IWorkStorage workStorage,

IWorkTypeStorage workTypeStorage,

ITimeOfWorkStorage timeOfWorkStorage,

ITOStorage tOStorage)

{

\_workStorage = workStorage;

\_workTypeStorage = workTypeStorage;

\_timeOfWorkStorage = timeOfWorkStorage;

\_tOStorage = tOStorage;

}

public void CreateWork(CreateWorkBindingModel model)

{

var tO = \_tOStorage.GetElement(new TOBindingModel { Id = model.TOId });

\_tOStorage.Update(new TOBindingModel

{

Id = tO.Id,

CarId = tO.CarId,

DateCreate = tO.DateCreate,

DateImplement = tO.DateImplement,

DateOver = tO.DateOver,

EmployeeId = tO.EmployeeId,

Status = (TOStatus)Enum.Parse(typeof(TOStatus), tO.Status),

Sum = (decimal)(tO.Sum + model.NetPrice)

});

var typeName = \_workTypeStorage.GetElement(new WorkTypeBindingModel { Id = model.WorkTypeId });

\_workStorage.Insert(new WorkBindingModel

{

StoreKeeperId = model.StoreKeeperId,

WorkTypeId = model.WorkTypeId,

TOId = model.TOId,

WorkName = typeName.WorkName,

Price = model.Price,

NetPrice = model.NetPrice,

WorkStatus = WorkStatus.Принят,

Count = model.Count

});

}

public void FinishWork(ChangeWorkStatusBindingModel model)

{

var work = \_workStorage

.GetElement(new WorkBindingModel { Id = model.WorkId });

if (work == null)

{

throw new Exception("Услуга не найдена");

}

if (work.WorkStatus

!= Enum.GetName(typeof(WorkStatus), 1))

{

throw new Exception("Услуга не в статусе \"Выполняется\"");

}

var time = \_timeOfWorkStorage.GetElement(new TimeOfWorkBindingModel

{

Id = \_workTypeStorage

.GetElement(new WorkTypeBindingModel

{

Id = work.WorkTypeId

}).TimeOfWorkId

});

DateTime dateTime = new DateTime(work.WorkBegin.Value.Year, work.WorkBegin.Value.Month, work.WorkBegin.Value.Day, time.Hours, time.Mins, 0);

int time1 = (int) TimeSpan.FromTicks(dateTime.Ticks).TotalMinutes;

int time2 = (int) TimeSpan.FromTicks(DateTime.Now.Ticks).TotalMinutes;

if (time2 < time1)

{

throw new Exception("Услуга ещё не выполнена");

}

\_workStorage.Update(new WorkBindingModel

{

Id = work.Id,

StoreKeeperId = work.StoreKeeperId,

WorkTypeId = work.WorkTypeId,

TOId = work.TOId,

WorkName = work.WorkName,

Price = work.Price,

NetPrice = work.NetPrice,

WorkStatus = WorkStatus.Готов,

WorkBegin = work.WorkBegin,

Count = work.Count

});

}

public List<WorkViewModel> Read(WorkBindingModel model)

{

if (model == null)

{

return \_workStorage.GetFullList();

}

if (model.Id.HasValue)

{

return new List<WorkViewModel>

{

\_workStorage.GetElement(model)

};

}

return \_workStorage.GetFilteredList(model);

}

public void TakeWorkInWork(ChangeWorkStatusBindingModel model)

{

var work = \_workStorage

.GetElement(new WorkBindingModel { Id = model.WorkId });

if (work == null)

{

throw new Exception("Услуга не найдена");

}

if (work.WorkStatus

!= Enum.GetName(typeof(WorkStatus), 0))

{

throw new Exception("Услуга не в статусе \"Принят\"");

}

if (\_tOStorage.GetElement(new TOBindingModel

{

Id = work.TOId

}).Status != Enum.GetName(typeof(TOStatus), 1))

{

throw new Exception("ТО #" + work.TOId + " не в статусе \"Выполняется\"");

}

\_workStorage.Update(new WorkBindingModel

{

Id = work.Id,

StoreKeeperId = work.StoreKeeperId,

WorkTypeId = work.WorkTypeId,

TOId = work.TOId,

WorkName = work.WorkName,

Price = work.Price,

NetPrice= work.NetPrice,

WorkStatus = WorkStatus.Выполняется,

WorkBegin = DateTime.Now,

Count = work.Count

});

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

namespace AutoFixStationBusinessLogic.BusinessLogics

{

public class WorkTypeLogic : IWorkTypeLogic

{

private readonly IWorkTypeStorage \_typeStorage;

public WorkTypeLogic(IWorkTypeStorage typeStorage)

{

\_typeStorage = typeStorage;

}

public void CreateOrUpdate(WorkTypeBindingModel model)

{

var element = \_typeStorage.GetElement(new WorkTypeBindingModel

{

WorkName = model.WorkName

});

if (element != null && element.Id != model.Id)

{

throw new Exception("Уже есть тип работ с таким названием");

}

if (model.Id.HasValue)

{

\_typeStorage.Update(model);

}

else

{

\_typeStorage.Insert(model);

}

}

public void Delete(WorkTypeBindingModel model)

{

var element = \_typeStorage.GetElement(new WorkTypeBindingModel

{

Id = model.Id

});

if (element == null)

{

throw new Exception("Элемент не найден");

}

\_typeStorage.Delete(model);

}

public List<WorkTypeViewModel> Read(WorkTypeBindingModel model)

{

if (model == null)

{

return \_typeStorage.GetFullList();

}

if (model.Id.HasValue)

{

return new List<WorkTypeViewModel>

{

\_typeStorage.GetElement(model)

};

}

return \_typeStorage.GetFilteredList(model);

}

}

}

using AutoFixStationContracts.BindingModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Mail;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.Mail

{

public class MailKitWorker

{

protected static string mailLogin;

protected static string mailPassword;

protected static string smtpClientHost;

protected static int smtpClientPort;

protected static string popHost;

protected static int popPort;

public void MailConfig(MailConfigBindingModel config)

{

mailLogin = config.MailLogin;

mailPassword = config.MailPassword;

smtpClientHost = config.SmtpClientHost;

smtpClientPort = config.SmtpClientPort;

popHost = config.PopHost;

popPort = config.PopPort;

}

public async void MailSendAsync(MailSendInfoBindingModel info)

{

if (string.IsNullOrEmpty(mailLogin) || string.IsNullOrEmpty(mailPassword))

{

return;

}

if (string.IsNullOrEmpty(smtpClientHost) || smtpClientPort == 0)

{

return;

}

if (string.IsNullOrEmpty(info.MailAddress) || string.IsNullOrEmpty(info.Subject) || string.IsNullOrEmpty(info.Text))

{

return;

}

await SendMailAsync(info);

}

protected async Task SendMailAsync(MailSendInfoBindingModel info)

{

using var objMailMessage = new MailMessage();

using var objSmtpClient = new SmtpClient(smtpClientHost, smtpClientPort);

try

{

objMailMessage.From = new MailAddress(mailLogin);

objMailMessage.To.Add(new MailAddress(info.MailAddress));

objMailMessage.Subject = info.Subject;

objMailMessage.Body = info.Text;

if (!string.IsNullOrEmpty(info.FileName) && File.Exists(info.FileName))

{

Attachment attachment = new Attachment(info.FileName);

objMailMessage.Attachments.Add(attachment);

}

objMailMessage.SubjectEncoding = Encoding.UTF8;

objMailMessage.BodyEncoding = Encoding.UTF8;

objSmtpClient.UseDefaultCredentials = false;

objSmtpClient.EnableSsl = true;

objSmtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;

objSmtpClient.Credentials = new NetworkCredential(mailLogin, mailPassword);

await Task.Run(() => objSmtpClient.Send(objMailMessage));

}

catch (Exception)

{

throw;

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage.HelperEnums

{

public enum ExcelStyleInfoType

{

Title,

Text,

TextWithBorder

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage.HelperEnums

{

public enum PdfParagraphAlignmentType

{

Center,

Left

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage.HelperEnums

{

public enum WordJustificationType

{

Center,

Both

}

}

using AutoFixStationBusinessLogic.OfficePackage.HelperEnums;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage.HelperModels

{

public class ExcelCellParameters

{

public string ColumnName { get; set; }

public uint RowIndex { get; set; }

public string Text { get; set; }

public string CellReference => $"{ColumnName}{RowIndex}";

public ExcelStyleInfoType StyleInfo { get; set; }

}

}

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage.HelperModels

{

public class ExcelInfo

{

public string FileName { get; set; }

public string Title { get; set; }

public List<ReportTOSparePartViewModel> TOSpareParts { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage.HelperModels

{

public class ExcelMergeParameters

{

public string CellFromName { get; set; }

public string CellToName { get; set; }

public string Merge => $"{CellFromName}:{CellToName}";

}

}

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage.HelperModels

{

public class PdfInfo

{

public string FileName { get; set; }

public string Title { get; set; }

public DateTime DateFrom { get; set; }

public DateTime DateTo { get; set; }

public List<ReportTOsViewModel> TOs { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage.HelperModels

{

public class PdfParagraph

{

public string Text { get; set; }

public string Style { get; set; }

}

}

using AutoFixStationBusinessLogic.OfficePackage.HelperEnums;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage.HelperModels

{

public class PdfRowParameters

{

public List<string> Texts { get; set; }

public string Style { get; set; }

public PdfParagraphAlignmentType ParagraphAlignment { get; set; }

}

}

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage.HelperModels

{

public class WordInfo

{

public string FileName { get; set; }

public string Title { get; set; }

public List<ReportTOSparePartViewModel> TOSpareParts { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage.HelperModels

{

public class WordParagraph

{

public List<(string, WordTextProperties)> Texts { get; set; }

public WordTextProperties TextProperties { get; set; }

}

}

using AutoFixStationBusinessLogic.OfficePackage.HelperEnums;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage.HelperModels

{

public class WordTextProperties

{

public string Size { get; set; }

public bool Bold { get; set; }

public WordJustificationType JustificationType { get; set; }

}

}

using AutoFixStationBusinessLogic.OfficePackage.HelperEnums;

using AutoFixStationBusinessLogic.OfficePackage.HelperModels;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Office2010.Excel;

using DocumentFormat.OpenXml.Office2013.Excel;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Spreadsheet;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage.Implements

{

public class SaveToExcel : AbstractSaveToExcel

{

private SpreadsheetDocument \_spreadsheetDocument;

private SharedStringTablePart \_shareStringPart;

private Worksheet \_worksheet;

/// <summary>

/// Настройка стилей для файла

/// </summary>

/// <param name="workbookpart"></param>

private static void CreateStyles(WorkbookPart workbookpart)

{

var sp = workbookpart.AddNewPart<WorkbookStylesPart>();

sp.Stylesheet = new Stylesheet();

var fonts = new Fonts() { Count = 2U, KnownFonts = true };

var fontUsual = new Font();

fontUsual.Append(new FontSize() { Val = 12D });

fontUsual.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color()

{

Theme = 1U

});

fontUsual.Append(new FontName() { Val = "Times New Roman" });

fontUsual.Append(new FontFamilyNumbering() { Val = 2 });

fontUsual.Append(new FontScheme() { Val = FontSchemeValues.Minor });

var fontTitle = new Font();

fontTitle.Append(new Bold());

fontTitle.Append(new FontSize() { Val = 14D });

fontTitle.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color()

{

Theme = 1U

});

fontTitle.Append(new FontName() { Val = "Times New Roman" });

fontTitle.Append(new FontFamilyNumbering() { Val = 2 });

fontTitle.Append(new FontScheme() { Val = FontSchemeValues.Minor });

fonts.Append(fontUsual);

fonts.Append(fontTitle);

var fills = new Fills() { Count = 2U };

var fill1 = new Fill();

fill1.Append(new PatternFill() { PatternType = PatternValues.None });

var fill2 = new Fill();

fill2.Append(new PatternFill() { PatternType = PatternValues.Gray125 });

fills.Append(fill1);

fills.Append(fill2);

var borders = new Borders() { Count = 2U };

var borderNoBorder = new Border();

borderNoBorder.Append(new LeftBorder());

borderNoBorder.Append(new RightBorder());

borderNoBorder.Append(new TopBorder());

borderNoBorder.Append(new BottomBorder());

borderNoBorder.Append(new DiagonalBorder());

var borderThin = new Border();

var leftBorder = new LeftBorder() { Style = BorderStyleValues.Thin };

leftBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color()

{

Indexed = 64U

});

var rightBorder = new RightBorder() { Style = BorderStyleValues.Thin };

rightBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color()

{

Indexed = 64U

});

var topBorder = new TopBorder() { Style = BorderStyleValues.Thin };

topBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color()

{

Indexed = 64U

});

var bottomBorder = new BottomBorder() { Style = BorderStyleValues.Thin };

bottomBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color()

{

Indexed = 64U

});

borderThin.Append(leftBorder);

borderThin.Append(rightBorder);

borderThin.Append(topBorder);

borderThin.Append(bottomBorder);

borderThin.Append(new DiagonalBorder());

borders.Append(borderNoBorder);

borders.Append(borderThin);

var cellStyleFormats = new CellStyleFormats() { Count = 1U };

var cellFormatStyle = new CellFormat()

{

NumberFormatId = 0U,

FontId = 0U,

FillId = 0U,

BorderId = 0U

};

cellStyleFormats.Append(cellFormatStyle);

var cellFormats = new CellFormats() { Count = 3U };

var cellFormatFont = new CellFormat()

{

NumberFormatId = 0U,

FontId = 0U,

FillId = 0U,

BorderId = 0U,

FormatId = 0U,

ApplyFont = true

};

var cellFormatFontAndBorder = new CellFormat()

{

NumberFormatId = 0U,

FontId = 0U,

FillId = 0U,

BorderId = 1U,

FormatId = 0U,

ApplyFont = true,

ApplyBorder = true

};

var cellFormatTitle = new CellFormat()

{

NumberFormatId = 0U,

FontId = 1U,

FillId = 0U,

BorderId = 0U,

FormatId = 0U,

Alignment = new Alignment()

{

Vertical = VerticalAlignmentValues.Center,

WrapText = true,

Horizontal = HorizontalAlignmentValues.Center

},

ApplyFont = true

};

cellFormats.Append(cellFormatFont);

cellFormats.Append(cellFormatFontAndBorder);

cellFormats.Append(cellFormatTitle);

var cellStyles = new CellStyles() { Count = 1U };

cellStyles.Append(new CellStyle()

{

Name = "Normal",

FormatId = 0U,

BuiltinId = 0U

});

var differentialFormats = new

DocumentFormat.OpenXml.Office2013.Excel.DifferentialFormats()

{ Count = 0U };

var tableStyles = new TableStyles()

{

Count = 0U,

DefaultTableStyle = "TableStyleMedium2",

DefaultPivotStyle = "PivotStyleLight16"

};

var stylesheetExtensionList = new StylesheetExtensionList();

var stylesheetExtension1 = new StylesheetExtension()

{

Uri = "{EB79DEF2-80B8-43e5 - 95BD - 54CBDDF9020C}"

};

stylesheetExtension1.AddNamespaceDeclaration("x14",

"http://schemas.microsoft.com/office/spreadsheetml/2009/9/main");

stylesheetExtension1.Append(new SlicerStyles()

{

DefaultSlicerStyle = "SlicerStyleLight1"

});

var stylesheetExtension2 = new StylesheetExtension()

{

Uri = "{9260A510-F301-46a8 - 8635 - F512D64BE5F5}"

};

stylesheetExtension2.AddNamespaceDeclaration("x15",

"http://schemas.microsoft.com/office/spreadsheetml/2010/11/main");

stylesheetExtension2.Append(new TimelineStyles()

{

DefaultTimelineStyle = "TimeSlicerStyleLight1"

});

stylesheetExtensionList.Append(stylesheetExtension1);

stylesheetExtensionList.Append(stylesheetExtension2);

sp.Stylesheet.Append(fonts);

sp.Stylesheet.Append(fills);

sp.Stylesheet.Append(borders);

sp.Stylesheet.Append(cellStyleFormats);

sp.Stylesheet.Append(cellFormats);

sp.Stylesheet.Append(cellStyles);

sp.Stylesheet.Append(differentialFormats);

sp.Stylesheet.Append(tableStyles);

sp.Stylesheet.Append(stylesheetExtensionList);

}

/// <summary>

/// Получение номера стиля из типа

/// </summary>

/// <param name="styleInfo"></param>

/// <returns></returns>

private static uint GetStyleValue(ExcelStyleInfoType styleInfo)

{

return styleInfo switch

{

ExcelStyleInfoType.Title => 2U,

ExcelStyleInfoType.TextWithBorder => 1U,

ExcelStyleInfoType.Text => 0U,

\_ => 0U,

};

}

protected override void CreateExcel(ExcelInfo info)

{

\_spreadsheetDocument = SpreadsheetDocument.Create(info.FileName,

SpreadsheetDocumentType.Workbook);

// Создаем книгу (в ней хранятся листы)

var workbookpart = \_spreadsheetDocument.AddWorkbookPart();

workbookpart.Workbook = new Workbook();

CreateStyles(workbookpart);

// Получаем/создаем хранилище текстов для книги

\_shareStringPart =

\_spreadsheetDocument.WorkbookPart

.GetPartsOfType<SharedStringTablePart>().Any() ?

\_spreadsheetDocument.WorkbookPart.GetPartsOfType<SharedStringTablePart>().First()

: \_spreadsheetDocument.WorkbookPart.AddNewPart<SharedStringTablePart>();

// Создаем SharedStringTable, если его нет

if (\_shareStringPart.SharedStringTable == null)

{

\_shareStringPart.SharedStringTable = new SharedStringTable();

}

// Создаем лист в книгу

var worksheetPart = workbookpart.AddNewPart<WorksheetPart>();

worksheetPart.Worksheet = new Worksheet(new SheetData());

// Добавляем лист в книгу

var sheets = \_spreadsheetDocument.WorkbookPart.Workbook.AppendChild(new

Sheets());

var sheet = new Sheet()

{

Id = \_spreadsheetDocument.WorkbookPart.GetIdOfPart(worksheetPart),

SheetId = 1,

Name = "Лист"

};

sheets.Append(sheet);

\_worksheet = worksheetPart.Worksheet;

}

protected override void InsertCellInWorksheet(ExcelCellParameters excelParams)

{

var sheetData = \_worksheet.GetFirstChild<SheetData>();

// Ищем строку, либо добавляем ее

Row row;

if (sheetData.Elements<Row>().Where(r => r.RowIndex ==

excelParams.RowIndex).Any())

{

row = sheetData.Elements<Row>().Where(r => r.RowIndex ==

excelParams.RowIndex).First();

}

else

{

row = new Row() { RowIndex = excelParams.RowIndex };

sheetData.Append(row);

}

// Ищем нужную ячейку

Cell cell;

if (row.Elements<Cell>().Where(c => c.CellReference.Value ==

excelParams.CellReference).Any())

{

cell = row.Elements<Cell>().Where(c => c.CellReference.Value ==

excelParams.CellReference).First();

}

else

{

// Все ячейки должны быть последовательно друг за другом расположены

// нужно определить, после какой вставлять

Cell refCell = null;

foreach (Cell rowCell in row.Elements<Cell>())

{

if (string.Compare(rowCell.CellReference.Value,

excelParams.CellReference, true) > 0)

{

refCell = rowCell;

break;

}

}

var newCell = new Cell() { CellReference = excelParams.CellReference };

row.InsertBefore(newCell, refCell);

cell = newCell;

}

// вставляем новый текст

\_shareStringPart.SharedStringTable.AppendChild(new SharedStringItem(new

Text(excelParams.Text)));

\_shareStringPart.SharedStringTable.Save();

cell.CellValue = new

CellValue((\_shareStringPart.SharedStringTable.Elements<SharedStringItem>().Count() -

1).ToString());

cell.DataType = new EnumValue<CellValues>(CellValues.SharedString);

cell.StyleIndex = GetStyleValue(excelParams.StyleInfo);

}

protected override void MergeCells(ExcelMergeParameters excelParams)

{

MergeCells mergeCells;

if (\_worksheet.Elements<MergeCells>().Any())

{

mergeCells = \_worksheet.Elements<MergeCells>().First();

}

else

{

mergeCells = new MergeCells();

if (\_worksheet.Elements<CustomSheetView>().Any())

{

\_worksheet.InsertAfter(mergeCells,

\_worksheet.Elements<CustomSheetView>().First());

}

else

{

\_worksheet.InsertAfter(mergeCells,

\_worksheet.Elements<SheetData>().First());

}

}

var mergeCell = new MergeCell()

{

Reference = new StringValue(excelParams.Merge)

};

mergeCells.Append(mergeCell);

}

protected override void SaveExcel(ExcelInfo info)

{

\_spreadsheetDocument.WorkbookPart.Workbook.Save();

\_spreadsheetDocument.Close();

}

}

}

using AutoFixStationBusinessLogic.OfficePackage.HelperEnums;

using AutoFixStationBusinessLogic.OfficePackage.HelperModels;

using MigraDoc.DocumentObjectModel;

using MigraDoc.DocumentObjectModel.Tables;

using MigraDoc.Rendering;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage.Implements

{

public class SaveToPdf : AbstractSaveToPdf

{

private Document \_document;

private Section \_section;

private Table \_table;

private static ParagraphAlignment GetParagraphAlignment(PdfParagraphAlignmentType

type)

{

return type switch

{

PdfParagraphAlignmentType.Center => ParagraphAlignment.Center,

PdfParagraphAlignmentType.Left => ParagraphAlignment.Left,

\_ => ParagraphAlignment.Justify,

};

}

/// <summary>

/// Создание стилей для документа

/// </summary>

/// <param name="document"></param>

private static void DefineStyles(Document document)

{

var style = document.Styles["Normal"];

style.Font.Name = "Times New Roman";

style.Font.Size = 14;

style = document.Styles.AddStyle("NormalTitle", "Normal");

style.Font.Bold = true;

}

/// <summary>

/// Создание файла-пдф

/// </summary>

/// <param name="info"></param>

protected override void CreatePdf(PdfInfo info)

{

\_document = new Document();

DefineStyles(\_document);

\_section = \_document.AddSection();

}

/// <summary>

/// Создание абзаца

/// </summary>

/// <param name="pdfParagraph"></param>

protected override void CreateParagraph(PdfParagraph pdfParagraph)

{

var paragraph = \_section.AddParagraph(pdfParagraph.Text);

paragraph.Format.SpaceAfter = "1cm";

paragraph.Format.Alignment = ParagraphAlignment.Center;

paragraph.Style = pdfParagraph.Style;

paragraph.Format.SpaceBefore = "0.5cm";

}

/// <summary>

/// Создание столбца

/// </summary>

/// <param name="columns"></param>

protected override void CreateTable(List<string> columns)

{

\_table = \_document.LastSection.AddTable();

foreach (var elem in columns)

{

\_table.AddColumn(elem);

}

}

/// <summary>

/// Создание строки

/// </summary>

/// <param name="rowParameters"></param>

protected override void CreateRow(PdfRowParameters rowParameters)

{

var row = \_table.AddRow();

for (int i = 0; i < rowParameters.Texts.Count; ++i)

{

row.Cells[i].AddParagraph(rowParameters.Texts[i]);

if (!string.IsNullOrEmpty(rowParameters.Style))

{

row.Cells[i].Style = rowParameters.Style;

}

Unit borderWidth = 0.5;

row.Cells[i].Borders.Left.Width = borderWidth;

row.Cells[i].Borders.Right.Width = borderWidth;

row.Cells[i].Borders.Top.Width = borderWidth;

row.Cells[i].Borders.Bottom.Width = borderWidth;

row.Cells[i].Format.Alignment =

GetParagraphAlignment(rowParameters.ParagraphAlignment);

row.Cells[i].VerticalAlignment = VerticalAlignment.Center;

}

}

/// <summary>

/// Сохранение файла

/// </summary>

/// <param name="info"></param>

protected override void SavePdf(PdfInfo info)

{

var renderer = new PdfDocumentRenderer(true)

{

Document = \_document

};

renderer.RenderDocument();

renderer.PdfDocument.Save(info.FileName);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using AutoFixStationBusinessLogic.OfficePackage.HelperEnums;

using AutoFixStationBusinessLogic.OfficePackage.HelperModels;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

namespace AutoFixStationBusinessLogic.OfficePackage.Implements

{

public class SaveToWord : AbstractSaveToWord

{

private WordprocessingDocument \_wordDocument;

private Body \_docBody;

private Table \_table;

/// <summary>

/// Получение типа выравнивания

/// </summary>

/// <param name="type"></param>

/// <returns></returns>

private static JustificationValues GetJustificationValues(WordJustificationType type)

{

return type switch

{

WordJustificationType.Both => JustificationValues.Both,

WordJustificationType.Center => JustificationValues.Center,

\_ => JustificationValues.Left,

};

}

/// <summary>

/// Настройки страницы

/// </summary>

/// <returns></returns>

private static SectionProperties CreateSectionProperties()

{

var properties = new SectionProperties();

var pageSize = new PageSize

{

Orient = PageOrientationValues.Portrait

};

properties.AppendChild(pageSize);

return properties;

}

/// <summary>

/// Задание форматирования для абзаца

/// </summary>

/// <param name="paragraphProperties"></param>

/// <returns></returns>

private static ParagraphProperties CreateParagraphProperties(WordTextProperties paragraphProperties)

{

if (paragraphProperties != null)

{

var properties = new ParagraphProperties();

properties.AppendChild(new Justification()

{

Val = GetJustificationValues(paragraphProperties.JustificationType)

});

properties.AppendChild(new SpacingBetweenLines

{

LineRule = LineSpacingRuleValues.Auto

});

properties.AppendChild(new Indentation());

var paragraphMarkRunProperties = new ParagraphMarkRunProperties();

if (!string.IsNullOrEmpty(paragraphProperties.Size))

{

paragraphMarkRunProperties.AppendChild(new FontSize

{

Val = paragraphProperties.Size

});

}

properties.AppendChild(paragraphMarkRunProperties);

return properties;

}

return null;

}

protected override void CreateParagraph(WordParagraph paragraph)

{

if (paragraph != null)

{

var docParagraph = new Paragraph();

docParagraph.AppendChild(CreateParagraphProperties(paragraph.TextProperties));

foreach (var run in paragraph.Texts)

{

var docRun = new Run();

var properties = new RunProperties();

properties.AppendChild(new FontSize { Val = run.Item2.Size });

if (run.Item2.Bold)

{

properties.AppendChild(new Bold());

}

docRun.AppendChild(properties);

docRun.AppendChild(new Text

{

Text = run.Item1,

Space = SpaceProcessingModeValues.Preserve

});

docParagraph.AppendChild(docRun);

}

\_docBody.AppendChild(docParagraph);

}

}

protected override void CreateWord(WordInfo info)

{

\_wordDocument = WordprocessingDocument.Create(info.FileName,

WordprocessingDocumentType.Document);

MainDocumentPart mainPart = \_wordDocument.AddMainDocumentPart();

mainPart.Document = new Document();

\_docBody = mainPart.Document.AppendChild(new Body());

}

protected override void SaveWord(WordInfo info)

{

\_docBody.AppendChild(CreateSectionProperties());

\_wordDocument.MainDocumentPart.Document.Save();

\_wordDocument.Close();

}

protected override void CreateTable(WordParagraph paragraph)

{

if (paragraph != null)

{

\_table = new Table();

var tableProp = new TableProperties();

tableProp.AppendChild(new TableLayout { Type = TableLayoutValues.Fixed });

tableProp.AppendChild(new TableBorders(

new TopBorder() { Val = new EnumValue<BorderValues>(BorderValues.Single), Size = 4 },

new LeftBorder() { Val = new EnumValue<BorderValues>(BorderValues.Single), Size = 4 },

new RightBorder() { Val = new EnumValue<BorderValues>(BorderValues.Single), Size = 4 },

new BottomBorder() { Val = new EnumValue<BorderValues>(BorderValues.Single), Size = 4 },

new InsideHorizontalBorder { Val = new EnumValue<BorderValues>(BorderValues.Single), Size = 4 },

new InsideVerticalBorder { Val = new EnumValue<BorderValues>(BorderValues.Single), Size = 4 }

));

\_table.AppendChild(tableProp);

CreateRowInTable(paragraph);

\_docBody.AppendChild(\_table);

}

}

protected override void CreateRowInTable(WordParagraph paragraph)

{

TableRow tableRow = new TableRow();

foreach (var elem in paragraph.Texts)

{

TableCell tableCell = new TableCell();

TableCellProperties tableCellProperties = new TableCellProperties();

tableCellProperties.AppendChild(new TableCellWidth() { Width = "3333", Type = TableWidthUnitValues.Dxa });

tableCell.AppendChild(tableCellProperties);

Paragraph paragraph1 = new Paragraph(new ParagraphProperties(new Justification() { Val = GetJustificationValues(elem.Item2.JustificationType) }));

Run run = new Run();

RunProperties runProperties = new RunProperties();

runProperties.AppendChild(new FontSize { Val = elem.Item2.Size });

if (elem.Item2.Bold)

{

runProperties.AppendChild(new Bold());

}

run.AppendChild(runProperties);

Text text = new Text(elem.Item1) { Space = SpaceProcessingModeValues.Preserve };

run.AppendChild(text);

paragraph1.AppendChild(run);

tableCell.AppendChild(paragraph1);

tableRow.AppendChild(tableCell);

}

\_table.AppendChild(tableRow);

}

}

}

using AutoFixStationBusinessLogic.OfficePackage.HelperEnums;

using AutoFixStationBusinessLogic.OfficePackage.HelperModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage

{

public abstract class AbstractSaveToExcel

{

/// <summary>

/// Создание отчeта

/// </summary>

/// <param name="info"></param>

public void CreateReportTOSpareParts(ExcelInfo info)

{

CreateExcel(info);

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = 1,

Text = info.Title,

StyleInfo = ExcelStyleInfoType.Title

});

MergeCells(new ExcelMergeParameters

{

CellFromName = "A1",

CellToName = "D1"

});

uint rowIndex = 2;

foreach(var element in info.TOSpareParts)

{

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = rowIndex,

Text = $"ТО #{element.TOId} по автомобилю \"{element.CarName}\"",

StyleInfo = ExcelStyleInfoType.Text

});

MergeCells(new ExcelMergeParameters

{

CellFromName = $"A{rowIndex}",

CellToName = $"D{rowIndex}"

});

rowIndex = InsertTOInfo(element.SpareParts, rowIndex) + 1;

}

SaveExcel(info);

}

private uint InsertTOInfo(Dictionary<int, (string, decimal, decimal)> spareParts, uint rowIndex)

{

rowIndex++;

//Заголовок

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "B",

RowIndex = rowIndex,

Text = "Список запчастей",

StyleInfo = ExcelStyleInfoType.Text

});

MergeCells(new ExcelMergeParameters

{

CellFromName = $"B{rowIndex}",

CellToName = $"C{rowIndex}"

});

rowIndex++;

//Вставка мини-заголовков

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = rowIndex,

Text = "Наименование",

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "B",

RowIndex = rowIndex,

Text = "Количество",

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = "Цена за шт.",

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "D",

RowIndex = rowIndex,

Text = "Стоимость",

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

//Вставка запчастей

rowIndex++;

foreach (var sp in spareParts)

{

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "A",

RowIndex = rowIndex,

Text = sp.Value.Item1,

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "B",

RowIndex = rowIndex,

Text = sp.Value.Item2.ToString(),

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "C",

RowIndex = rowIndex,

Text = sp.Value.Item3.ToString(),

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

InsertCellInWorksheet(new ExcelCellParameters

{

ColumnName = "D",

RowIndex = rowIndex,

Text = (sp.Value.Item2 \* sp.Value.Item3).ToString(),

StyleInfo = ExcelStyleInfoType.TextWithBorder

});

rowIndex++;

}

return rowIndex;

}

/// <summary>

/// Создание excel-файла

/// </summary>

/// <param name="info"></param>

protected abstract void CreateExcel(ExcelInfo info);

/// <summary>

/// Добавляем новую ячейку в лист

/// </summary>

/// <param name="excelCellParams"></param>

protected abstract void InsertCellInWorksheet(ExcelCellParameters cellParams);

/// <summary>

/// Объединение ячеек

/// </summary>

/// <param name="mergeParams"></param>

protected abstract void MergeCells(ExcelMergeParameters mergeParams);

/// <summary>

/// Сохранение файла

/// </summary>

/// <param name="info"></param>

protected abstract void SaveExcel(ExcelInfo info);

}

}

using AutoFixStationBusinessLogic.OfficePackage.HelperEnums;

using AutoFixStationBusinessLogic.OfficePackage.HelperModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage

{

public abstract class AbstractSaveToPdf

{

public void CreateReportTOsByDate(PdfInfo info)

{

CreatePdf(info);

CreateParagraph(new PdfParagraph

{

Text = info.Title,

Style = "NormalTitle"

});

CreateParagraph(new PdfParagraph

{

Text = $"Период с {info.DateFrom.ToShortDateString()} по {info.DateTo.ToShortDateString()}",

Style = "Normal"

});

foreach(var to in info.TOs)

{

CreateParagraph(new PdfParagraph

{

Text = $"ТО #{to.TOId}",

Style = "Normal"

});

CreateParagraph(new PdfParagraph

{

Text = $"Автомобиль: \"{to.CarName}\"",

Style = "Normal"

});

CreateParagraph(new PdfParagraph

{

Text = $"Дата начала ТО: {to.DateBegin}",

Style = "Normal"

});

CreateParagraph(new PdfParagraph

{

Text = $"Дата окончания ТО: {to.DateEnd}",

Style = "Normal"

});

InsertTOInfo(to.SpareParts, to.ServiceRecords);

}

SavePdf(info);

}

private void InsertTOInfo(Dictionary<int, (string, decimal, decimal)> spareParts, List<string> serviceRecords)

{

CreateParagraph(new PdfParagraph

{

Text = "Запчасти",

Style = "NormalTitle"

});

CreateTable(new List<string> { "5cm", "5cm", "3cm", "3cm" });

CreateRow(new PdfRowParameters

{

Texts = new List<string>

{

"Наименование", "Количество",

"Цена за ед.", "Стоимость"

},

Style = "NormalTitle",

ParagraphAlignment = PdfParagraphAlignmentType.Center

});

foreach (var part in spareParts)

{

CreateRow(new PdfRowParameters

{

Texts = new List<string>

{

part.Value.Item1,

part.Value.Item2.ToString(),

part.Value.Item3.ToString(),

(part.Value.Item2 \* part.Value.Item3).ToString()

},

Style = "Normal",

ParagraphAlignment = PdfParagraphAlignmentType.Left

});

}

CreateParagraph(new PdfParagraph

{

Text = "Записи сервисов",

Style = "NormalTitle"

});

uint num = 1;

foreach(var sr in serviceRecords)

{

CreateParagraph(new PdfParagraph

{

Text = $"Запись #{num}",

Style = "Normal"

});

CreateParagraph(new PdfParagraph

{

Text = sr,

Style = "Normal"

});

CreateParagraph(new PdfParagraph

{

Text = "",

Style = "Normal"

});

num++;

}

}

/// <summary>

/// Cоздание doc-файла

/// </summary>

/// <param name="info"></param>

protected abstract void CreatePdf(PdfInfo info);

/// <summary>

/// Создание параграфа с текстом

/// </summary>

/// <param name="title"></param>

/// <param name="style"></param>

protected abstract void CreateParagraph(PdfParagraph paragraph);

/// <summary>

/// Создание таблицы

/// </summary>

/// <param name="title"></param>

/// <param name="style"></param>

protected abstract void CreateTable(List<string> columns);

/// <summary>

/// Создание и заполнение строки

/// </summary>

/// <param name="rowParameters"></param>

protected abstract void CreateRow(PdfRowParameters rowParameters);

/// <summary>

/// Сохранение файла

/// </summary>

/// <param name="info"></param>

protected abstract void SavePdf(PdfInfo info);

}

}

using AutoFixStationBusinessLogic.OfficePackage.HelperEnums;

using AutoFixStationBusinessLogic.OfficePackage.HelperModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationBusinessLogic.OfficePackage

{

public abstract class AbstractSaveToWord

{

public void CreateReportTOSpareParts(WordInfo info)

{

CreateWord(info);

CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)>

{

(info.Title, new WordTextProperties

{

Bold = true, Size = "24",

})

},

TextProperties = new WordTextProperties

{

Size = "24",

JustificationType = WordJustificationType.Center

}

});

foreach (var element in info.TOSpareParts)

{

CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)>

{

($"ТО #{element.TOId} по автомобилю \"{element.CarName}\"", new WordTextProperties

{

Size = "18",

Bold = true

})

},

TextProperties = new WordTextProperties

{

Size = "18",

JustificationType = WordJustificationType.Both

}

});

InsertTOInfo(element.SpareParts);

}

SaveWord(info);

}

private void InsertTOInfo(Dictionary<int, (string, decimal, decimal)> spareParts)

{

CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)>

{

("Список запчастей:\n", new WordTextProperties

{

Size = "16",

Bold = false

})

},

TextProperties = new WordTextProperties

{

Size = "16",

JustificationType = WordJustificationType.Both

}

});

uint num = 1;

foreach(var part in spareParts)

{

CreateParagraph(new WordParagraph

{

Texts = new List<(string, WordTextProperties)>

{

($"{num}) {part.Value.Item1} ({part.Value.Item2}) - Цена: {part.Value.Item3}р. - Стоимость: {part.Value.Item2 \* part.Value.Item3}р.", new WordTextProperties

{

Size = "14",

Bold = false

})

},

TextProperties = new WordTextProperties

{

Size = "14",

JustificationType = WordJustificationType.Both

}

});

num++;

}

}

/// <summary>

/// Создание doc-файла

/// </summary>

/// <param name="info"></param>

protected abstract void CreateWord(WordInfo info);

/// <summary>

/// Создание абзаца с текстом

/// </summary>

/// <param name="paragraph"></param>

/// <returns></returns>

protected abstract void CreateParagraph(WordParagraph paragraph);

/// <summary>

/// Сохранение файла

/// </summary>

/// <param name="info"></param>

protected abstract void SaveWord(WordInfo info);

/// <summary>

/// Создание таблицы с текстом

/// </summary>

/// <param name="paragraph"></param>

protected abstract void CreateTable(WordParagraph paragraph);

/// <summary>

/// Создание строки в таблице

/// </summary>

/// <param name="paragraph"></param>

protected abstract void CreateRowInTable(WordParagraph paragraph);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

/// <summary>

/// Машина клиента сервиса

/// </summary>

public class CarBindingModel

{

public int? Id { get; set; }

/// <summary>

/// Бренд авто (VW, Ford)

/// </summary>

public string Brand { get; set; }

/// <summary>

/// Модель авто (Polo, Fusion)

/// </summary>

public string Model { get; set; }

/// <summary>

/// VIN-номер авто (длина 17 символов)

/// </summary>

public string VIN { get; set; }

/// <summary>

/// Номер телефона владельца

/// </summary>

public string OwnerPhoneNumber { get; set; }

/// <summary>

/// Записи сервисов

/// </summary>

/// <summary>

/// Записи сервисов

/// </summary>

public Dictionary<int, ((DateTime, DateTime), string)>? Records { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

public class ChangeTOStatusBindingModel

{

public int TOId { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

public class ChangeWorkStatusBindingModel

{

public int WorkId { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

public class CreateTOBindingModel

{

public int CarId { get; set; }

public int EmployeeId { get; set; }

public decimal? Sum { get; set; }

public Dictionary<int, (string, (int, decimal))>? Works { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

public class CreateWorkBindingModel

{

public int StoreKeeperId { get; set; }

public int WorkTypeId { get; set; }

public int TOId { get; set; }

public int Count { get; set; }

public decimal Price { get; set; }

public decimal NetPrice { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

public class EmployeeBindingModel

{

public int? Id { get; set; }

public string FIO { get; set; }

public string Login { get; set; }

public string Password { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

public class MailConfigBindingModel

{

public string MailLogin { get; set; }

public string MailPassword { get; set; }

public string SmtpClientHost { get; set; }

public int SmtpClientPort { get; set; }

public string PopHost { get; set; }

public int PopPort { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

public class MailSendInfoBindingModel

{

public string MailAddress { get; set; }

public string Subject { get; set; }

public string Text { get; set; }

public string FileName { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

/// <summary>

/// Сообщения, приходящие на почту

/// </summary>

public class MessageInfoBindingModel

{

public int? ClientId { get; set; }

public string MessageId { get; set; }

public string FromMailAddress { get; set; }

public string Subject { get; set; }

public string Body { get; set; }

public DateTime DateDelivery { get; set; }

public bool Checked { get; set; }

public string AnswerText { get; set; }

public int? ToSkip { get; set; }

public int? ToTake { get; set; }

}

}

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

public class ReportBindingModel

{

public string FileName { get; set; }

public DateTime? DateFrom { get; set; }

public DateTime? DateTo { get; set; }

public List<TOViewModel>? TOs { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

public class ServiceRecordBindingModel

{

public int? Id { get; set; }

public int CarId { get; set; }

[DataType(DataType.Date)]

public DateTime DateBegin { get; set; }

[DataType(DataType.Date)]

public DateTime DateEnd { get; set; }

public string Description { get; set; }

}

}

using AutoFixStationContracts.Enums;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

public class SparePartBindingModel

{

public int? Id { get; set; }

public string Name { get; set; }

public string FactoryNumber { get; set; }

public decimal Price { get; set; }

public SparePartStatus Type { get; set; }

public UnitMeasurement UMeasurement { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

public class StoreKeeperBindingModel

{

public int? Id { get; set; }

public string FIO { get; set; }

public string Login { get; set; }

public string Password { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

public class TimeOfWorkBindingModel

{

public int? Id { get; set; }

public int Hours { get; set; }

public int Mins { get; set; }

}

}

using AutoFixStationContracts.Enums;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

/// <summary>

/// Доступные ТО

/// </summary>

public class TOBindingModel

{

public int? Id { get; set; }

public int CarId { get; set; }

public int EmployeeId { get; set; }

public decimal Sum { get; set; }

public TOStatus Status { get; set; }

[DataType(DataType.Date)]

public DateTime DateCreate { get; set; }

[DataType(DataType.Date)]

public DateTime? DateImplement { get; set; }

[DataType(DataType.Date)]

public DateTime? DateOver { get; set; }

/// <summary>

/// Работы ТО (номер, (название, (кол-во, стоимость)))

/// </summary>

public Dictionary<int, (string, (int, decimal))>? Works { get; set; }

}

}

using AutoFixStationContracts.Enums;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

public class WorkBindingModel

{

public int? Id { get; set; }

public int StoreKeeperId { get; set; }

public int WorkTypeId { get; set; }

public int TOId { get; set; }

public string WorkName { get; set; }

public decimal Price { get; set; }

public decimal NetPrice { get; set; }

public WorkStatus WorkStatus { get; set; }

[DataType(DataType.Date)]

public DateTime? WorkBegin { get; set; }

public int Count { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BindingModels

{

public class WorkTypeBindingModel

{

public int? Id { get; set; }

public int TimeOfWorkId { get; set; }

public string WorkName { get; set; }

/// <summary>

/// Стоимость услуги

/// </summary>

public decimal Price { get; set; }

public decimal NetPrice { get; set; }//with spareparts

/// <summary>

/// Необходимые детали и расходники (int - id, string - название, decimal, потому что может быть не целое (например, 0.8 л масла))

/// </summary>

public Dictionary<int, (string, decimal, decimal)> WorkSpareParts { get; set; }

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BusinessLogicsContracts

{

public interface ICarLogic

{

List<CarViewModel> Read(CarBindingModel model);

void CreateOrUpdate(CarBindingModel model);

void Delete(CarBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BusinessLogicsContracts

{

public interface IEmployeeLogic

{

List<EmployeeViewModel> Read(EmployeeBindingModel model);

void CreateOrUpdate(EmployeeBindingModel model);

void Delete(EmployeeBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BusinessLogicsContracts

{

public interface IReportLogic

{

/// <summary>

/// Получение списка запчастей по выбранным ТО

/// </summary>

/// <returns></returns>

List<ReportTOSparePartViewModel> GetTOSparePart(ReportBindingModel model);

/// <summary>

/// Сохранение списка запчастей по выбранным ТО в файл-Word

/// </summary>

/// <param name="model"></param>

/// <returns></returns>

void SaveTOSparePartToWordFile(ReportBindingModel model);

/// <summary>

/// Сохранение списка запчастей по выбранным ТО в файл-Excel

/// </summary>

/// <param name="model"></param>

/// <returns></returns>

void SaveTOSparePartToExcelFile(ReportBindingModel model);

/// <summary>

/// Получение списка ТО за период

/// </summary>

/// <param name="model"></param>

/// <returns></returns>

List<ReportTOsViewModel> GetTOs(ReportBindingModel model);

/// <summary>

/// Сохранение ТО в файл-Pdf

/// </summary>

/// <param name="model"></param>

void SaveTOsByDateToPdfFile(ReportBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BusinessLogicsContracts

{

public interface IServiceRecordLogic

{

List<ServiceRecordViewModel> Read(ServiceRecordBindingModel model);

void Create(ServiceRecordBindingModel record, TOBindingModel tO);

void Update(ServiceRecordBindingModel record);

void Delete(ServiceRecordBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BusinessLogicsContracts

{

public interface ISparePartLogic

{

List<SparePartViewModel> Read(SparePartBindingModel model);

void CreateOrUpdate(SparePartBindingModel model);

void Delete(SparePartBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BusinessLogicsContracts

{

public interface IStoreKeeperLogic

{

List<StoreKeeperViewModel> Read(StoreKeeperBindingModel model);

void CreateOrUpdate(StoreKeeperBindingModel model);

void Delete(StoreKeeperBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BusinessLogicsContracts

{

public interface ITimeOfWorkLogic

{

List<TimeOfWorkViewModel> Read(TimeOfWorkBindingModel model);

void CreateOrUpdate(TimeOfWorkBindingModel model);

void Delete(TimeOfWorkBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BusinessLogicsContracts

{

public interface ITOLogic

{

List<TOViewModel> Read(TOBindingModel model);

void CreateTO(CreateTOBindingModel model);

void TakeTOInWork(ChangeTOStatusBindingModel model);

void FinishTO(ChangeTOStatusBindingModel model);

void IssueTO(ChangeTOStatusBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BusinessLogicsContracts

{

public interface IWorkLogic

{

List<WorkViewModel> Read(WorkBindingModel model);

void CreateWork(CreateWorkBindingModel model);

void TakeWorkInWork(ChangeWorkStatusBindingModel model);

void FinishWork(ChangeWorkStatusBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.BusinessLogicsContracts

{

public interface IWorkTypeLogic

{

List<WorkTypeViewModel> Read(WorkTypeBindingModel model);

void CreateOrUpdate(WorkTypeBindingModel model);

void Delete(WorkTypeBindingModel model);

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.Enums

{

public enum SparePartStatus

{

[Display(Name = "Б/У")]

БУ = 0,

[Display(Name = "Новая")]

Новая = 1

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.Enums

{

/// <summary>

/// Статус ТО

/// </summary>

public enum TOStatus

{

Принят = 0,

Выполняется = 1,

Готов = 2,

Выдан = 3

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.Enums

{

public enum UnitMeasurement

{

[Display(Name = "ШТ")]

шт = 0,

[Display(Name = "КГ")]

кг = 1,

[Display(Name = "Л")]

л = 2

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.Enums

{

/// <summary>

/// Статус работы

/// </summary>

public enum WorkStatus

{

Принят = 0,

Выполняется = 1,

Готов = 2,

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.StorageContracts

{

public interface ICarStorage

{

List<CarViewModel> GetFullList();

List<CarViewModel> GetFilteredList(CarBindingModel model);

CarViewModel GetElement(CarBindingModel model);

void Insert(CarBindingModel model);

void Update(CarBindingModel model);

void Delete(CarBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.StorageContracts

{

public interface IEmployeeStorage

{

List<EmployeeViewModel> GetFullList();

List<EmployeeViewModel> GetFilteredList(EmployeeBindingModel model);

EmployeeViewModel GetElement(EmployeeBindingModel model);

void Insert(EmployeeBindingModel model);

void Update(EmployeeBindingModel model);

void Delete(EmployeeBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.StorageContracts

{

public interface IServiceRecordStorage

{

List<ServiceRecordViewModel> GetFullList();

List<ServiceRecordViewModel> GetFilteredList(ServiceRecordBindingModel model);

ServiceRecordViewModel GetElement(ServiceRecordBindingModel model);

void Insert(ServiceRecordBindingModel model);

void Update(ServiceRecordBindingModel model);

void Delete(ServiceRecordBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.StorageContracts

{

public interface ISparePartStorage

{

List<SparePartViewModel> GetFullList();

List<SparePartViewModel> GetFilteredList(SparePartBindingModel model);

SparePartViewModel GetElement(SparePartBindingModel model);

void Insert(SparePartBindingModel model);

void Update(SparePartBindingModel model);

void Delete(SparePartBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.StorageContracts

{

public interface IStoreKeeperStorage

{

List<StoreKeeperViewModel> GetFullList();

List<StoreKeeperViewModel> GetFilteredList(StoreKeeperBindingModel model);

StoreKeeperViewModel GetElement(StoreKeeperBindingModel model);

void Insert(StoreKeeperBindingModel model);

void Update(StoreKeeperBindingModel model);

void Delete(StoreKeeperBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.StorageContracts

{

public interface ITimeOfWorkStorage

{

List<TimeOfWorkViewModel> GetFullList();

List<TimeOfWorkViewModel> GetFilteredList(TimeOfWorkBindingModel model);

TimeOfWorkViewModel GetElement(TimeOfWorkBindingModel model);

void Insert(TimeOfWorkBindingModel model);

void Update(TimeOfWorkBindingModel model);

void Delete(TimeOfWorkBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.StorageContracts

{

public interface ITOStorage

{

List<TOViewModel> GetFullList();

List<TOViewModel> GetFilteredList(TOBindingModel model);

TOViewModel GetElement(TOBindingModel model);

void Insert(TOBindingModel model);

void Update(TOBindingModel model);

void Delete(TOBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.StorageContracts

{

public interface IWorkStorage

{

List<WorkViewModel> GetFullList();

List<WorkViewModel> GetFilteredList(WorkBindingModel model);

WorkViewModel GetElement(WorkBindingModel model);

void Insert(WorkBindingModel model);

void Update(WorkBindingModel model);

void Delete(WorkBindingModel model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.StorageContracts

{

public interface IWorkTypeStorage

{

List<WorkTypeViewModel> GetFullList();

List<WorkTypeViewModel> GetFilteredList(WorkTypeBindingModel model);

WorkTypeViewModel GetElement(WorkTypeBindingModel model);

void Insert(WorkTypeBindingModel model);

void Update(WorkTypeBindingModel model);

void Delete(WorkTypeBindingModel model);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

namespace AutoFixStationContracts.ViewModels

{

public class CarViewModel

{

public int Id { get; set; }

[DisplayName("Марка")]

public string Brand { get; set; }

[DisplayName("Модель")]

public string Model { get; set; }

[DisplayName("VIN")]

public string VIN { get; set; }

[DisplayName("Номер телефона владельца")]

public string OwnerPhoneNumber { get; set; }

/// <summary>

/// Записи сервисов (номер, ((Дата начала, Дата конца), описание)

/// </summary>

public Dictionary<int, ((DateTime, DateTime), string)>? Records { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

namespace AutoFixStationContracts.ViewModels

{

public class EmployeeViewModel

{

public int Id { get; set; }

[DisplayName("ФИО")]

public string FIO { get; set; }

[DisplayName("Логин")]

public string Login { get; set; }

[DisplayName("Пароль")]

public string Password { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.ViewModels

{

public class ReportTOSparePartViewModel

{

public int TOId { get; set; }

public string CarName { get; set; }

public Dictionary<int, (string, decimal, decimal)> SpareParts { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.ViewModels

{

public class ReportTOsViewModel

{

public int TOId { get; set; }

public string CarName { get; set; }

public DateTime DateBegin { get; set; }

public DateTime DateEnd { get; set; }

public Dictionary<int, (string, decimal, decimal)> SpareParts { get; set; }

public List<string> ServiceRecords { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

namespace AutoFixStationContracts.ViewModels

{

public class ServiceRecordViewModel

{

public int Id { get; set; }

public int CarId { get; set; }

public string CarBrandAndName { get; set; }

[DisplayName("Дата начала")]

[DataType(DataType.Date)]

public DateTime DateBegin { get; set; }

[DisplayName("Дата окончания")]

[DataType(DataType.Date)]

public DateTime DateEnd { get; set; }

[DisplayName("Описание")]

public string Description { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

namespace AutoFixStationContracts.ViewModels

{

public class SparePartViewModel

{

public int Id { get; set; }

[DisplayName("Название")]

public string Name { get; set; }

[DisplayName("Заводской номер")]

public string FactoryNumber { get; set; }

[DisplayName("Стоимость ед.")]

public decimal Price { get; set; }

[DisplayName("Тип детали")]

public string Type { get; set; }

[DisplayName("Ед. измерения")]

public string UMeasurement { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.ViewModels

{

public class StoreKeeperViewModel

{

public int Id { get; set; }

[DisplayName("ФИО")]

public string FIO { get; set; }

[DisplayName("Логин")]

public string Login { get; set; }

[DisplayName("Пароль")]

public string Password { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

namespace AutoFixStationContracts.ViewModels

{

public class TimeOfWorkViewModel

{

public int Id { get; set; }

[DisplayName("Часы")]

public int Hours { get; set; }

[DisplayName("Минуты")]

public int Mins { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

namespace AutoFixStationContracts.ViewModels

{

public class TOViewModel

{

public int Id { get; set; }

public int CarId { get; set; }

public string CarName { get; set; }

public int EmployeeId { get; set; }

public string EmployeeName { get; set; }

public string TOAndCarName { get; set; }

[DisplayName("Стоимость")]

public decimal? Sum { get; set; }

[DisplayName("Статус")]

public string Status { get; set; }

[DisplayName("Дата создания")]

[DataType(DataType.Date)]

public DateTime DateCreate { get; set; }

[DataType(DataType.Date)]

[DisplayName("Дата начала")]

public DateTime? DateImplement { get; set; }

[DataType(DataType.Date)]

[DisplayName("Дата окончания")]

public DateTime? DateOver { get; set; }

/// <summary>

/// Работы ТО (номер, (название, (кол-во, стоимость

/// </summary>

public Dictionary<int, (string, (int, decimal))>? Works { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

namespace AutoFixStationContracts.ViewModels

{

public class WorkTypeViewModel

{

public int Id { get; set; }

public int TimeOfWorkId { get; set; }

public string ExecutionTime { get; set; }

[DisplayName("Название работы")]

public string WorkName { get; set; }

[DisplayName("Стоимость работы")]

public decimal Price { get; set; }

[DisplayName("Стоимость с учётом деталей")]

public decimal NetPrice { get; set; }//with spareparts

/// <summary>

/// Необходимые детали и расходники (int - id, string - название, decimal, потому что может быть не целое (например, 0.8 л масла))

/// </summary>

public Dictionary<int, (string, decimal, decimal)> WorkSpareParts { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

namespace AutoFixStationContracts.ViewModels

{

public class WorkViewModel

{

public int Id { get; set; }

public int StoreKeeperId { get; set; }

[DisplayName("Кладовщик")]

public string StoreKeeperFIO { get; set; }

public int WorkTypeId { get; set; }

public int TOId { get; set; }

[DisplayName("Название")]

public string WorkName { get; set; }

[DisplayName("Стоимость работы")]

public decimal Price { get; set; }

[DisplayName("Стоимость с учётом запчастей")]

public decimal NetPrice { get; set; }

[DisplayName("Статус")]

public string WorkStatus { get; set; }

[DisplayName("Дата начала")]

[DataType(DataType.Date)]

public DateTime? WorkBegin { get; set; }

[DisplayName("Кол-во")]

public int Count { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationContracts.WebViewModels

{

public class TimeOfWorkWebViewModel

{

public int Id { get; set; }

[DisplayName("Время")]

public string Time { get; set; }

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

using AutoFixStationDatabaseImplement.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationDatabaseImplement.Implements

{

public class CarStorage : ICarStorage

{

public void Delete(CarBindingModel model)

{

using var context = new AutoFixStationDatabase();

Car element = context.Cars

.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.Cars.Remove(element);

context.SaveChanges();

}

else

{

throw new Exception("Элемент не найден");

}

}

public CarViewModel GetElement(CarBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

var car = context.Cars

.Include(rec => rec.ServiceRecords)

.Include(rec => rec.TOs)

.FirstOrDefault(rec => rec.VIN == model.VIN

|| rec.Id == model.Id);

return car != null ? CreateModel(car) : null;

}

public List<CarViewModel> GetFilteredList(CarBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

return context.Cars

.Include(rec => rec.ServiceRecords)

.Include(rec => rec.TOs)

.Where(rec => rec.VIN.Equals(model.VIN))

.ToList()

.Select(CreateModel)

.ToList();

}

public List<CarViewModel> GetFullList()

{

using var context = new AutoFixStationDatabase();

return context.Cars

.Include(rec => rec.TOs)

.Include(rec => rec.ServiceRecords)

.ToList()

.Select(CreateModel)

.ToList();

}

public void Insert(CarBindingModel model)

{

using var context = new AutoFixStationDatabase();

using var transaction = context.Database.BeginTransaction();

try

{

context.Cars.Add(CreateModel(model, new Car()));

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

public void Update(CarBindingModel model)

{

using var context = new AutoFixStationDatabase();

using var transaction = context.Database.BeginTransaction();

try

{

var element = context.Cars

.FirstOrDefault(rec => rec.Id == model.Id);

if (element == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, element);

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

private static Car CreateModel(CarBindingModel model, Car car)

{

car.Brand = model.Brand;

car.Model = model.Model;

car.OwnerPhoneNumber = model.OwnerPhoneNumber;

car.VIN = model.VIN;

return car;

}

private static CarViewModel CreateModel(Car car)

{

return new CarViewModel

{

Id = car.Id,

Brand = car.Brand,

Model = car.Model,

OwnerPhoneNumber = car.OwnerPhoneNumber,

VIN = car.VIN,

Records = car.ServiceRecords?

.ToDictionary(

rec => rec.Id, rec =>

((rec.DateBegin, rec.DateEnd), rec.Description))

};

}

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

using AutoFixStationDatabaseImplement.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationDatabaseImplement.Implements

{

public class EmployeeStorage : IEmployeeStorage

{

public void Delete(EmployeeBindingModel model)

{

using var context = new AutoFixStationDatabase();

Employee employee = context.Employees

.FirstOrDefault(rec => rec.Id == model.Id);

if (employee != null)

{

context.Employees.Remove(employee);

context.SaveChanges();

}

else

{

throw new Exception("Элемент не найден");

}

}

public EmployeeViewModel GetElement(EmployeeBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

var employee = context.Employees

.Include(rec => rec.TOs)

.FirstOrDefault(rec => rec.Login == model.Login

|| rec.Id == model.Id);

return employee != null ? CreateModel(employee) : null;

}

public List<EmployeeViewModel> GetFilteredList(EmployeeBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

return context.Employees

.Include(rec => rec.TOs)

.Where(rec => rec.Login.Equals(model.Login))

.ToList()

.Select(CreateModel)

.ToList();

}

public List<EmployeeViewModel> GetFullList()

{

using var context = new AutoFixStationDatabase();

return context.Employees

.ToList()

.Select(CreateModel)

.ToList();

}

public void Insert(EmployeeBindingModel model)

{

using var context = new AutoFixStationDatabase();

using var transaction = context.Database.BeginTransaction();

try

{

context.Employees.Add(CreateModel(model, new Employee()));

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

public void Update(EmployeeBindingModel model)

{

using var context = new AutoFixStationDatabase();

using var transaction = context.Database.BeginTransaction();

try

{

var element = context.Employees

.FirstOrDefault(rec => rec.Id == model.Id);

if (element == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, element);

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

private static Employee CreateModel(EmployeeBindingModel model, Employee employee)

{

employee.FIO = model.FIO;

employee.Login = model.Login;

employee.Password = model.Password;

return employee;

}

private static EmployeeViewModel CreateModel(Employee employee)

{

return new EmployeeViewModel

{

Id = employee.Id,

FIO = employee.FIO,

Login = employee.Login,

Password = employee.Password

};

}

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

using AutoFixStationDatabaseImplement.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationDatabaseImplement.Implements

{

public class ServiceRecordStorage : IServiceRecordStorage

{

public void Delete(ServiceRecordBindingModel model)

{

using var context = new AutoFixStationDatabase();

ServiceRecord element = context.ServiceRecords

.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.ServiceRecords.Remove(element);

context.SaveChanges();

}

else

{

throw new Exception("Элемент не найден");

}

}

public ServiceRecordViewModel GetElement(ServiceRecordBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

var element = context.ServiceRecords

.Include(rec => rec.Car)

.FirstOrDefault(rec => (rec.CarId == model.CarId

&& rec.DateBegin.Date == model.DateBegin.Date)

|| rec.Id == model.Id);

return element != null ? CreateModel(element) : null;

}

public List<ServiceRecordViewModel> GetFilteredList(ServiceRecordBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

return context.ServiceRecords

.Include(rec => rec.Car)

.Where(rec => rec.Id.Equals(model.Id)

|| (rec.DateBegin.Date >= model.DateBegin.Date && rec.DateEnd.Date <= model.DateEnd.Date)

|| (rec.CarId.Equals(model.CarId)))

.ToList()

.Select(CreateModel)

.ToList();

}

public List<ServiceRecordViewModel> GetFullList()

{

using var context = new AutoFixStationDatabase();

return context.ServiceRecords

.Include(rec => rec.Car)

.ToList()

.Select(CreateModel)

.ToList();

}

public void Insert(ServiceRecordBindingModel model)

{

using var context = new AutoFixStationDatabase();

using var transaction = context.Database.BeginTransaction();

try

{

context.ServiceRecords.Add(CreateModel(model, new ServiceRecord()));

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

public void Update(ServiceRecordBindingModel model)

{

using var context = new AutoFixStationDatabase();

using var transaction = context.Database.BeginTransaction();

try

{

var element = context.ServiceRecords

.FirstOrDefault(rec => rec.Id == model.Id);

if (element == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, element);

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

private static ServiceRecord CreateModel(ServiceRecordBindingModel model, ServiceRecord serviceRecord)

{

serviceRecord.DateBegin = model.DateBegin;

serviceRecord.DateEnd = model.DateEnd;

serviceRecord.Description = model.Description;

serviceRecord.CarId = model.CarId;

return serviceRecord;

}

private static ServiceRecordViewModel CreateModel(ServiceRecord serviceRecord)

{

return new ServiceRecordViewModel

{

Id = serviceRecord.Id,

CarId = serviceRecord.CarId,

CarBrandAndName = serviceRecord.Car.Brand + " " + serviceRecord.Car.Model,

DateBegin = serviceRecord.DateBegin,

DateEnd = serviceRecord.DateEnd,

Description = serviceRecord.Description

};

}

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

using AutoFixStationDatabaseImplement.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationDatabaseImplement.Implements

{

public class SparePartStorage : ISparePartStorage

{

public void Delete(SparePartBindingModel model)

{

using var context = new AutoFixStationDatabase();

SparePart element = context.SpareParts

.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.SpareParts.Remove(element);

context.SaveChanges();

}

else

{

throw new Exception("Элемент не найден");

}

}

public SparePartViewModel GetElement(SparePartBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

var element = context.SpareParts

.FirstOrDefault(rec => rec.Name == model.Name

|| rec.Id == model.Id);

return element != null ? CreateModel(element) : null;

}

public List<SparePartViewModel> GetFilteredList(SparePartBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

return context.SpareParts

.Where(rec => rec.Name.Contains(model.Name))

.ToList()

.Select(CreateModel)

.ToList();

}

public List<SparePartViewModel> GetFullList()

{

using var context = new AutoFixStationDatabase();

return context.SpareParts

.ToList()

.Select(CreateModel)

.ToList();

}

public void Insert(SparePartBindingModel model)

{

using var context = new AutoFixStationDatabase();

context.SpareParts.Add(CreateModel(model, new SparePart()));

context.SaveChanges();

}

public void Update(SparePartBindingModel model)

{

using var context = new AutoFixStationDatabase();

var element = context.SpareParts

.FirstOrDefault(rec => rec.Id == model.Id);

if (element == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, element);

context.SaveChanges();

}

private static SparePart CreateModel(SparePartBindingModel model, SparePart

sparePart)

{

sparePart.Name = model.Name;

sparePart.FactoryNumber = model.FactoryNumber;

sparePart.Price = model.Price;

sparePart.Type = model.Type;

sparePart.UMeasurement = model.UMeasurement;

return sparePart;

}

private static SparePartViewModel CreateModel(SparePart sparePart)

{

return new SparePartViewModel

{

Id = sparePart.Id,

Name = sparePart.Name,

FactoryNumber = sparePart.FactoryNumber,

Price = sparePart.Price,

Type = sparePart.Type.ToString(),

UMeasurement = sparePart.UMeasurement.ToString()

};

}

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

using AutoFixStationDatabaseImplement.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationDatabaseImplement.Implements

{

public class SparePartStorage : ISparePartStorage

{

public void Delete(SparePartBindingModel model)

{

using var context = new AutoFixStationDatabase();

SparePart element = context.SpareParts

.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.SpareParts.Remove(element);

context.SaveChanges();

}

else

{

throw new Exception("Элемент не найден");

}

}

public SparePartViewModel GetElement(SparePartBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

var element = context.SpareParts

.FirstOrDefault(rec => rec.Name == model.Name

|| rec.Id == model.Id);

return element != null ? CreateModel(element) : null;

}

public List<SparePartViewModel> GetFilteredList(SparePartBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

return context.SpareParts

.Where(rec => rec.Name.Contains(model.Name))

.ToList()

.Select(CreateModel)

.ToList();

}

public List<SparePartViewModel> GetFullList()

{

using var context = new AutoFixStationDatabase();

return context.SpareParts

.ToList()

.Select(CreateModel)

.ToList();

}

public void Insert(SparePartBindingModel model)

{

using var context = new AutoFixStationDatabase();

context.SpareParts.Add(CreateModel(model, new SparePart()));

context.SaveChanges();

}

public void Update(SparePartBindingModel model)

{

using var context = new AutoFixStationDatabase();

var element = context.SpareParts

.FirstOrDefault(rec => rec.Id == model.Id);

if (element == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, element);

context.SaveChanges();

}

private static SparePart CreateModel(SparePartBindingModel model, SparePart

sparePart)

{

sparePart.Name = model.Name;

sparePart.FactoryNumber = model.FactoryNumber;

sparePart.Price = model.Price;

sparePart.Type = model.Type;

sparePart.UMeasurement = model.UMeasurement;

return sparePart;

}

private static SparePartViewModel CreateModel(SparePart sparePart)

{

return new SparePartViewModel

{

Id = sparePart.Id,

Name = sparePart.Name,

FactoryNumber = sparePart.FactoryNumber,

Price = sparePart.Price,

Type = sparePart.Type.ToString(),

UMeasurement = sparePart.UMeasurement.ToString()

};

}

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

using AutoFixStationDatabaseImplement.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationDatabaseImplement.Implements

{

public class TimeOfWorkStorage : ITimeOfWorkStorage

{

public void Delete(TimeOfWorkBindingModel model)

{

using var context = new AutoFixStationDatabase();

TimeOfWork element = context.TimeOfWorks

.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.TimeOfWorks.Remove(element);

context.SaveChanges();

}

else

{

throw new Exception("Элемент не найден");

}

}

public TimeOfWorkViewModel GetElement(TimeOfWorkBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

var element = context.TimeOfWorks

.Include(rec => rec.WorkTypes)

.FirstOrDefault(rec => (rec.Hours == model.Hours

&& rec.Mins == model.Mins) || rec.Id == model.Id);

return element != null ? CreateModel(element) : null;

}

public List<TimeOfWorkViewModel> GetFilteredList(TimeOfWorkBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

return context.TimeOfWorks

.Include(rec => rec.WorkTypes)

.Where(rec => (rec.Hours == model.Hours

&& rec.Mins == model.Mins))

.ToList()

.Select(CreateModel)

.ToList();

}

public List<TimeOfWorkViewModel> GetFullList()

{

using var context = new AutoFixStationDatabase();

return context.TimeOfWorks

.ToList()

.Select(CreateModel)

.ToList();

}

public void Insert(TimeOfWorkBindingModel model)

{

using var context = new AutoFixStationDatabase();

using var transaction = context.Database.BeginTransaction();

try

{

context.TimeOfWorks.Add(CreateModel(model, new TimeOfWork()));

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

public void Update(TimeOfWorkBindingModel model)

{

using var context = new AutoFixStationDatabase();

using var transaction = context.Database.BeginTransaction();

try

{

var element = context.TimeOfWorks

.FirstOrDefault(rec => rec.Id == model.Id);

if (element == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, element);

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

private static TimeOfWork CreateModel(TimeOfWorkBindingModel model, TimeOfWork timeOfWork)

{

timeOfWork.Hours = model.Hours;

timeOfWork.Mins = model.Mins;

return timeOfWork;

}

private static TimeOfWorkViewModel CreateModel(TimeOfWork timeOfWork)

{

return new TimeOfWorkViewModel

{

Id = timeOfWork.Id,

Hours = timeOfWork.Hours,

Mins = timeOfWork.Mins

};

}

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

using AutoFixStationDatabaseImplement.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationDatabaseImplement.Implements

{

public class TOStorage : ITOStorage

{

public void Delete(TOBindingModel model)

{

using var context = new AutoFixStationDatabase();

TO element = context.TOs

.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.TOs.Remove(element);

context.SaveChanges();

}

else

{

throw new Exception("Элемент не найден");

}

}

public TOViewModel GetElement(TOBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

var to = context.TOs

.Include(rec => rec.Works)

.Include(rec => rec.Car)

.Include(rec => rec.Employee)

.FirstOrDefault(rec => rec.Id == model.Id);

return to != null ? CreateModel(to) : null;

}

public List<TOViewModel> GetFilteredList(TOBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

return context.TOs

.Include(rec => rec.Works)

.Include(rec => rec.Car)

.Include(rec => rec.Employee)

.Where(rec => rec.Id.Equals(model.Id)

|| rec.CarId.Equals(model.CarId)

|| rec.Status.Equals(model.Status)

|| rec.EmployeeId.Equals(model.EmployeeId)

|| (rec.DateCreate.Date >= model.DateCreate

&& rec.DateOver <= model.DateOver))

.ToList()

.Select(CreateModel)

.ToList();

}

public List<TOViewModel> GetFullList()

{

using var context = new AutoFixStationDatabase();

return context.TOs

.Include(rec => rec.Car)

.Include(rec => rec.Employee)

.ToList()

.Select(CreateModel)

.ToList();

}

public void Insert(TOBindingModel model)

{

using var context = new AutoFixStationDatabase();

using var transaction = context.Database.BeginTransaction();

try

{

context.TOs.Add(CreateModel(model, new TO()));

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

public void Update(TOBindingModel model)

{

using var context = new AutoFixStationDatabase();

using var transaction = context.Database.BeginTransaction();

try

{

var element = context.TOs

.FirstOrDefault(rec => rec.Id == model.Id);

if (element == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, element);

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

private static TO CreateModel(TOBindingModel model, TO tO)

{

tO.CarId = model.CarId;

tO.EmployeeId = model.EmployeeId;

tO.Sum = model.Sum;

tO.Status = model.Status;

tO.DateCreate = model.DateCreate;

tO.DateImplement = model.DateImplement;

tO.DateOver = model.DateOver;

return tO;

}

private static TOViewModel CreateModel(TO tO)

{

return new TOViewModel

{

Id = tO.Id,

CarId = tO.CarId,

CarName = $"{tO.Car.Brand} {tO.Car.Model}",

TOAndCarName = $"{tO.Id} {tO.Car.Brand} {tO.Car.Model}",

EmployeeId = tO.EmployeeId,

EmployeeName = tO.Employee.FIO,

Sum = tO.Sum,

Status = tO.Status.ToString(),

DateCreate = tO.DateCreate,

DateImplement = tO.DateImplement,

DateOver = tO.DateOver,

Works = tO.Works

.ToDictionary(recPC => recPC.Id,

recPC => (recPC.WorkName, (recPC.Count, recPC.NetPrice)))

};

}

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

using AutoFixStationDatabaseImplement.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationDatabaseImplement.Implements

{

public class WorkStorage : IWorkStorage

{

public void Delete(WorkBindingModel model)

{

using var context = new AutoFixStationDatabase();

Work element = context.Works

.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.Works.Remove(element);

context.SaveChanges();

}

else

{

throw new Exception("Элемент не найден");

}

}

public WorkViewModel GetElement(WorkBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

var element = context.Works

.Include(rec => rec.WorkType)

.Include(rec => rec.StoreKeeper)

.FirstOrDefault(rec => rec.Id == model.Id);

return element != null ? CreateModel(element) : null;

}

public List<WorkViewModel> GetFilteredList(WorkBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

return context.Works

.Include(rec => rec.WorkType)

.Include(rec => rec.StoreKeeper)

.Where(rec => rec.WorkName.Contains(model.WorkName)

|| (rec.StoreKeeperId == model.StoreKeeperId)

|| (rec.WorkTypeId == model.WorkTypeId)

|| (rec.TOId == model.TOId))

.ToList()

.Select(CreateModel)

.ToList();

}

public List<WorkViewModel> GetFullList()

{

using var context = new AutoFixStationDatabase();

return context.Works

.Include(rec => rec.WorkType)

.Include(rec => rec.StoreKeeper)

.ToList()

.Select(CreateModel)

.ToList();

}

public void Insert(WorkBindingModel model)

{

using var context = new AutoFixStationDatabase();

using var transaction = context.Database.BeginTransaction();

try

{

context.Works.Add(CreateModel(model, new Work()));

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

public void Update(WorkBindingModel model)

{

using var context = new AutoFixStationDatabase();

using var transaction = context.Database.BeginTransaction();

try

{

var element = context.Works

.FirstOrDefault(rec => rec.Id == model.Id);

if (element == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, element);

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

private static Work CreateModel(WorkBindingModel model, Work work)

{

work.WorkTypeId = model.WorkTypeId;

work.StoreKeeperId = model.StoreKeeperId;

work.TOId = model.TOId;

work.WorkName = model.WorkName;

work.WorkStatus = model.WorkStatus;

work.Price = model.Price;

work.NetPrice = model.NetPrice;

work.WorkBegin = model.WorkBegin;

work.Count = model.Count;

return work;

}

private static WorkViewModel CreateModel(Work work)

{

return new WorkViewModel

{

Id = work.Id,

WorkTypeId = work.WorkTypeId,

StoreKeeperId = work.StoreKeeperId,

StoreKeeperFIO = work.StoreKeeper.FIO,

TOId = work.TOId,

WorkName = work.WorkName,

WorkStatus = work.WorkStatus.ToString(),

Price = work.Price,

NetPrice = work.NetPrice,

WorkBegin = work.WorkBegin,

Count = work.Count,

};

}

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationContracts.ViewModels;

using AutoFixStationDatabaseImplement.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoFixStationDatabaseImplement.Implements

{

public class WorkTypeStorage : IWorkTypeStorage

{

public void Delete(WorkTypeBindingModel model)

{

using var context = new AutoFixStationDatabase();

WorkType element = context.WorkTypes

.FirstOrDefault(rec => rec.Id == model.Id);

if (element != null)

{

context.WorkTypes.Remove(element);

context.SaveChanges();

}

else

{

throw new Exception("Элемент не найден");

}

}

public WorkTypeViewModel GetElement(WorkTypeBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

var element = context.WorkTypes

.Include(rec => rec.Work\_SpareParts)

.ThenInclude(rec => rec.SparePart)

.Include(rec => rec.TimeOfWork)

.FirstOrDefault(rec => rec.WorkName == model.WorkName

|| rec.Id == model.Id);

return element != null ? CreateModel(element) : null;

}

public List<WorkTypeViewModel> GetFilteredList(WorkTypeBindingModel model)

{

if (model == null)

{

return null;

}

using var context = new AutoFixStationDatabase();

return context.WorkTypes

.Include(rec => rec.Work\_SpareParts)

.ThenInclude(rec => rec.SparePart)

.Include(rec => rec.TimeOfWork)

.Where(rec => (rec.Id.Equals(model.Id))

|| (rec.WorkName.Equals(model.WorkName)))

.ToList()

.Select(CreateModel)

.ToList();

}

public List<WorkTypeViewModel> GetFullList()

{

using var context = new AutoFixStationDatabase();

return context.WorkTypes

.Include(rec => rec.Work\_SpareParts)

.ThenInclude(rec => rec.SparePart)

.Include(rec => rec.TimeOfWork)

.ToList()

.Select(CreateModel)

.ToList();

}

public void Insert(WorkTypeBindingModel model)

{

using var context = new AutoFixStationDatabase();

using var transaction = context.Database.BeginTransaction();

try

{

WorkType workType = new WorkType()

{

TimeOfWorkId = model.TimeOfWorkId,

WorkName = model.WorkName,

Price = model.Price,

NetPrice = model.NetPrice

};

context.WorkTypes.Add(workType);

context.SaveChanges();

CreateModel(model, workType, context);

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

public void Update(WorkTypeBindingModel model)

{

using var context = new AutoFixStationDatabase();

using var transaction = context.Database.BeginTransaction();

try

{

var element = context.WorkTypes

.FirstOrDefault(rec => rec.Id == model.Id);

if (element == null)

{

throw new Exception("Элемент не найден");

}

CreateModel(model, element, context);

context.SaveChanges();

transaction.Commit();

}

catch

{

transaction.Rollback();

throw;

}

}

private static WorkType CreateModel(WorkTypeBindingModel model, WorkType workType,

AutoFixStationDatabase context)

{

workType.TimeOfWorkId = model.TimeOfWorkId;

workType.WorkName = model.WorkName;

workType.Price = model.Price;

workType.NetPrice = model.NetPrice;

if (model.Id.HasValue)

{

var work\_parts = context.WorkType\_SpareParts

.Where(rec => rec.WorkTypeId == model.Id.Value)

.ToList();

// удалили те, которых нет в модели

context.WorkType\_SpareParts

.RemoveRange(work\_parts

.Where(rec => !model.WorkSpareParts

.ContainsKey(rec.SparePartId))

.ToList());

context.SaveChanges();

// обновили количество у существующих записей

foreach (var updateSparePart in work\_parts)

{

updateSparePart.Count =

model.WorkSpareParts[updateSparePart.SparePartId].Item2;

model.WorkSpareParts.Remove(updateSparePart.SparePartId);

}

context.SaveChanges();

}

// добавили новые значения в таблицу

foreach (var element in model.WorkSpareParts)

{

context.WorkType\_SpareParts.Add(new WorkType\_SparePart

{

WorkTypeId = workType.Id,

SparePartId = element.Key,

Count = element.Value.Item2

});

context.SaveChanges();

}

return workType;

}

private static WorkTypeViewModel CreateModel(WorkType workType)

{

return new WorkTypeViewModel

{

Id = workType.Id,

TimeOfWorkId = workType.TimeOfWorkId,

WorkName = workType.WorkName,

Price = workType.Price,

NetPrice = workType.NetPrice,

ExecutionTime = "Часов: " + workType.TimeOfWork.Hours + ". Минут: "+ workType.TimeOfWork.Mins,

WorkSpareParts = workType.Work\_SpareParts

.ToDictionary(recPC => recPC.SparePartId,

recPC => (recPC.SparePart.Name, recPC.Count, recPC.SparePart.Price))

};

}

}

}

using System;

using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace AutoFixStationDatabaseImplement.Migrations

{

public partial class InitialCreate : Migration

{

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.CreateTable(

name: "Cars",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

Brand = table.Column<string>(type: "nvarchar(max)", nullable: false),

Model = table.Column<string>(type: "nvarchar(max)", nullable: false),

VIN = table.Column<string>(type: "nvarchar(max)", nullable: false),

OwnerPhoneNumber = table.Column<string>(type: "nvarchar(max)", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Cars", x => x.Id);

});

migrationBuilder.CreateTable(

name: "Employees",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

FIO = table.Column<string>(type: "nvarchar(max)", nullable: false),

Login = table.Column<string>(type: "nvarchar(max)", nullable: false),

Password = table.Column<string>(type: "nvarchar(max)", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Employees", x => x.Id);

});

migrationBuilder.CreateTable(

name: "SpareParts",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

Name = table.Column<string>(type: "nvarchar(max)", nullable: false),

FactoryNumber = table.Column<string>(type: "nvarchar(max)", nullable: false),

Price = table.Column<decimal>(type: "decimal(18,2)", nullable: false),

Type = table.Column<int>(type: "int", nullable: false),

UMeasurement = table.Column<int>(type: "int", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_SpareParts", x => x.Id);

});

migrationBuilder.CreateTable(

name: "StoreKeepers",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

FIO = table.Column<string>(type: "nvarchar(max)", nullable: false),

Login = table.Column<string>(type: "nvarchar(max)", nullable: false),

Password = table.Column<string>(type: "nvarchar(max)", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_StoreKeepers", x => x.Id);

});

migrationBuilder.CreateTable(

name: "TimeOfWorks",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

Hours = table.Column<int>(type: "int", nullable: false),

Mins = table.Column<int>(type: "int", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_TimeOfWorks", x => x.Id);

});

migrationBuilder.CreateTable(

name: "ServiceRecords",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

CarId = table.Column<int>(type: "int", nullable: false),

DateBegin = table.Column<DateTime>(type: "datetime2", nullable: false),

DateEnd = table.Column<DateTime>(type: "datetime2", nullable: false),

Description = table.Column<string>(type: "nvarchar(max)", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_ServiceRecords", x => x.Id);

table.ForeignKey(

name: "FK\_ServiceRecords\_Cars\_CarId",

column: x => x.CarId,

principalTable: "Cars",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "TOs",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

CarId = table.Column<int>(type: "int", nullable: false),

EmployeeId = table.Column<int>(type: "int", nullable: false),

Sum = table.Column<decimal>(type: "decimal(18,2)", nullable: false),

Status = table.Column<int>(type: "int", nullable: false),

DateCreate = table.Column<DateTime>(type: "datetime2", nullable: false),

DateImplement = table.Column<DateTime>(type: "datetime2", nullable: true),

DateOver = table.Column<DateTime>(type: "datetime2", nullable: true)

},

constraints: table =>

{

table.PrimaryKey("PK\_TOs", x => x.Id);

table.ForeignKey(

name: "FK\_TOs\_Cars\_CarId",

column: x => x.CarId,

principalTable: "Cars",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_TOs\_Employees\_EmployeeId",

column: x => x.EmployeeId,

principalTable: "Employees",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "WorkTypes",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

TimeOfWorkId = table.Column<int>(type: "int", nullable: false),

WorkName = table.Column<string>(type: "nvarchar(max)", nullable: false),

Price = table.Column<decimal>(type: "decimal(18,2)", nullable: false),

NetPrice = table.Column<decimal>(type: "decimal(18,2)", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_WorkTypes", x => x.Id);

table.ForeignKey(

name: "FK\_WorkTypes\_TimeOfWorks\_TimeOfWorkId",

column: x => x.TimeOfWorkId,

principalTable: "TimeOfWorks",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "Works",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

StoreKeeperId = table.Column<int>(type: "int", nullable: false),

WorkTypeId = table.Column<int>(type: "int", nullable: false),

WorkName = table.Column<string>(type: "nvarchar(max)", nullable: false),

Price = table.Column<decimal>(type: "decimal(18,2)", nullable: false),

NetPrice = table.Column<decimal>(type: "decimal(18,2)", nullable: false),

WorkStatus = table.Column<int>(type: "int", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Works", x => x.Id);

table.ForeignKey(

name: "FK\_Works\_StoreKeepers\_StoreKeeperId",

column: x => x.StoreKeeperId,

principalTable: "StoreKeepers",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_Works\_WorkTypes\_WorkTypeId",

column: x => x.WorkTypeId,

principalTable: "WorkTypes",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "WorkType\_SpareParts",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

SparePartId = table.Column<int>(type: "int", nullable: false),

WorkTypeId = table.Column<int>(type: "int", nullable: false),

Count = table.Column<decimal>(type: "decimal(18,2)", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_WorkType\_SpareParts", x => x.Id);

table.ForeignKey(

name: "FK\_WorkType\_SpareParts\_SpareParts\_SparePartId",

column: x => x.SparePartId,

principalTable: "SpareParts",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_WorkType\_SpareParts\_WorkTypes\_WorkTypeId",

column: x => x.WorkTypeId,

principalTable: "WorkTypes",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "TO\_Works",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

TOId = table.Column<int>(type: "int", nullable: false),

WorkId = table.Column<int>(type: "int", nullable: false),

Count = table.Column<int>(type: "int", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_TO\_Works", x => x.Id);

table.ForeignKey(

name: "FK\_TO\_Works\_TOs\_TOId",

column: x => x.TOId,

principalTable: "TOs",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_TO\_Works\_Works\_WorkId",

column: x => x.WorkId,

principalTable: "Works",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateIndex(

name: "IX\_ServiceRecords\_CarId",

table: "ServiceRecords",

column: "CarId");

migrationBuilder.CreateIndex(

name: "IX\_TO\_Works\_TOId",

table: "TO\_Works",

column: "TOId");

migrationBuilder.CreateIndex(

name: "IX\_TO\_Works\_WorkId",

table: "TO\_Works",

column: "WorkId");

migrationBuilder.CreateIndex(

name: "IX\_TOs\_CarId",

table: "TOs",

column: "CarId");

migrationBuilder.CreateIndex(

name: "IX\_TOs\_EmployeeId",

table: "TOs",

column: "EmployeeId");

migrationBuilder.CreateIndex(

name: "IX\_Works\_StoreKeeperId",

table: "Works",

column: "StoreKeeperId");

migrationBuilder.CreateIndex(

name: "IX\_Works\_WorkTypeId",

table: "Works",

column: "WorkTypeId");

migrationBuilder.CreateIndex(

name: "IX\_WorkType\_SpareParts\_SparePartId",

table: "WorkType\_SpareParts",

column: "SparePartId");

migrationBuilder.CreateIndex(

name: "IX\_WorkType\_SpareParts\_WorkTypeId",

table: "WorkType\_SpareParts",

column: "WorkTypeId");

migrationBuilder.CreateIndex(

name: "IX\_WorkTypes\_TimeOfWorkId",

table: "WorkTypes",

column: "TimeOfWorkId");

}

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropTable(

name: "ServiceRecords");

migrationBuilder.DropTable(

name: "TO\_Works");

migrationBuilder.DropTable(

name: "WorkType\_SpareParts");

migrationBuilder.DropTable(

name: "TOs");

migrationBuilder.DropTable(

name: "Works");

migrationBuilder.DropTable(

name: "SpareParts");

migrationBuilder.DropTable(

name: "Cars");

migrationBuilder.DropTable(

name: "Employees");

migrationBuilder.DropTable(

name: "StoreKeepers");

migrationBuilder.DropTable(

name: "WorkTypes");

migrationBuilder.DropTable(

name: "TimeOfWorks");

}

}

}

using System;

using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace AutoFixStationDatabaseImplement.Migrations

{

public partial class Init2 : Migration

{

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropTable(

name: "TO\_Works");

migrationBuilder.AddColumn<int>(

name: "Count",

table: "Works",

type: "int",

nullable: false,

defaultValue: 0);

migrationBuilder.AddColumn<int>(

name: "TOId",

table: "Works",

type: "int",

nullable: false,

defaultValue: 0);

migrationBuilder.AddColumn<DateTime>(

name: "WorkBegin",

table: "Works",

type: "datetime2",

nullable: true);

migrationBuilder.CreateIndex(

name: "IX\_Works\_TOId",

table: "Works",

column: "TOId");

migrationBuilder.AddForeignKey(

name: "FK\_Works\_TOs\_TOId",

table: "Works",

column: "TOId",

principalTable: "TOs",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

}

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropForeignKey(

name: "FK\_Works\_TOs\_TOId",

table: "Works");

migrationBuilder.DropIndex(

name: "IX\_Works\_TOId",

table: "Works");

migrationBuilder.DropColumn(

name: "Count",

table: "Works");

migrationBuilder.DropColumn(

name: "TOId",

table: "Works");

migrationBuilder.DropColumn(

name: "WorkBegin",

table: "Works");

migrationBuilder.CreateTable(

name: "TO\_Works",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

TOId = table.Column<int>(type: "int", nullable: false),

WorkId = table.Column<int>(type: "int", nullable: false),

Count = table.Column<int>(type: "int", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_TO\_Works", x => x.Id);

table.ForeignKey(

name: "FK\_TO\_Works\_TOs\_TOId",

column: x => x.TOId,

principalTable: "TOs",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "FK\_TO\_Works\_Works\_WorkId",

column: x => x.WorkId,

principalTable: "Works",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateIndex(

name: "IX\_TO\_Works\_TOId",

table: "TO\_Works",

column: "TOId");

migrationBuilder.CreateIndex(

name: "IX\_TO\_Works\_WorkId",

table: "TO\_Works",

column: "WorkId");

}

}

}

// <auto-generated />

using System;

using AutoFixStationDatabaseImplement;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Infrastructure;

using Microsoft.EntityFrameworkCore.Metadata;

using Microsoft.EntityFrameworkCore.Storage.ValueConversion;

#nullable disable

namespace AutoFixStationDatabaseImplement.Migrations

{

[DbContext(typeof(AutoFixStationDatabase))]

partial class AutoFixStationDatabaseModelSnapshot : ModelSnapshot

{

protected override void BuildModel(ModelBuilder modelBuilder)

{

#pragma warning disable 612, 618

modelBuilder

.HasAnnotation("ProductVersion", "6.0.8")

.HasAnnotation("Relational:MaxIdentifierLength", 128);

SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder, 1L, 1);

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.Car", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"), 1L, 1);

b.Property<string>("Brand")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Model")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("OwnerPhoneNumber")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("VIN")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.HasKey("Id");

b.ToTable("Cars");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.Employee", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"), 1L, 1);

b.Property<string>("FIO")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Login")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Password")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.HasKey("Id");

b.ToTable("Employees");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.ServiceRecord", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"), 1L, 1);

b.Property<int>("CarId")

.HasColumnType("int");

b.Property<DateTime>("DateBegin")

.HasColumnType("datetime2");

b.Property<DateTime>("DateEnd")

.HasColumnType("datetime2");

b.Property<string>("Description")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.HasKey("Id");

b.HasIndex("CarId");

b.ToTable("ServiceRecords");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.SparePart", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"), 1L, 1);

b.Property<string>("FactoryNumber")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Name")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<decimal>("Price")

.HasColumnType("decimal(18,2)");

b.Property<int>("Type")

.HasColumnType("int");

b.Property<int>("UMeasurement")

.HasColumnType("int");

b.HasKey("Id");

b.ToTable("SpareParts");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.StoreKeeper", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"), 1L, 1);

b.Property<string>("FIO")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Login")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<string>("Password")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.HasKey("Id");

b.ToTable("StoreKeepers");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.TimeOfWork", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"), 1L, 1);

b.Property<int>("Hours")

.HasColumnType("int");

b.Property<int>("Mins")

.HasColumnType("int");

b.HasKey("Id");

b.ToTable("TimeOfWorks");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.TO", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"), 1L, 1);

b.Property<int>("CarId")

.HasColumnType("int");

b.Property<DateTime>("DateCreate")

.HasColumnType("datetime2");

b.Property<DateTime?>("DateImplement")

.HasColumnType("datetime2");

b.Property<DateTime?>("DateOver")

.HasColumnType("datetime2");

b.Property<int>("EmployeeId")

.HasColumnType("int");

b.Property<int>("Status")

.HasColumnType("int");

b.Property<decimal>("Sum")

.HasColumnType("decimal(18,2)");

b.HasKey("Id");

b.HasIndex("CarId");

b.HasIndex("EmployeeId");

b.ToTable("TOs");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.Work", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"), 1L, 1);

b.Property<int>("Count")

.HasColumnType("int");

b.Property<decimal>("NetPrice")

.HasColumnType("decimal(18,2)");

b.Property<decimal>("Price")

.HasColumnType("decimal(18,2)");

b.Property<int>("StoreKeeperId")

.HasColumnType("int");

b.Property<int>("TOId")

.HasColumnType("int");

b.Property<DateTime?>("WorkBegin")

.HasColumnType("datetime2");

b.Property<string>("WorkName")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.Property<int>("WorkStatus")

.HasColumnType("int");

b.Property<int>("WorkTypeId")

.HasColumnType("int");

b.HasKey("Id");

b.HasIndex("StoreKeeperId");

b.HasIndex("TOId");

b.HasIndex("WorkTypeId");

b.ToTable("Works");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.WorkType", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"), 1L, 1);

b.Property<decimal>("NetPrice")

.HasColumnType("decimal(18,2)");

b.Property<decimal>("Price")

.HasColumnType("decimal(18,2)");

b.Property<int>("TimeOfWorkId")

.HasColumnType("int");

b.Property<string>("WorkName")

.IsRequired()

.HasColumnType("nvarchar(max)");

b.HasKey("Id");

b.HasIndex("TimeOfWorkId");

b.ToTable("WorkTypes");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.WorkType\_SparePart", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"), 1L, 1);

b.Property<decimal>("Count")

.HasColumnType("decimal(18,2)");

b.Property<int>("SparePartId")

.HasColumnType("int");

b.Property<int>("WorkTypeId")

.HasColumnType("int");

b.HasKey("Id");

b.HasIndex("SparePartId");

b.HasIndex("WorkTypeId");

b.ToTable("WorkType\_SpareParts");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.ServiceRecord", b =>

{

b.HasOne("AutoFixStationDatabaseImplement.Models.Car", "Car")

.WithMany("ServiceRecords")

.HasForeignKey("CarId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.Navigation("Car");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.TO", b =>

{

b.HasOne("AutoFixStationDatabaseImplement.Models.Car", "Car")

.WithMany("TOs")

.HasForeignKey("CarId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.HasOne("AutoFixStationDatabaseImplement.Models.Employee", "Employee")

.WithMany("TOs")

.HasForeignKey("EmployeeId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.Navigation("Car");

b.Navigation("Employee");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.Work", b =>

{

b.HasOne("AutoFixStationDatabaseImplement.Models.StoreKeeper", "StoreKeeper")

.WithMany("Works")

.HasForeignKey("StoreKeeperId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.HasOne("AutoFixStationDatabaseImplement.Models.TO", "TO")

.WithMany("Works")

.HasForeignKey("TOId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.HasOne("AutoFixStationDatabaseImplement.Models.WorkType", "WorkType")

.WithMany("Works")

.HasForeignKey("WorkTypeId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.Navigation("StoreKeeper");

b.Navigation("TO");

b.Navigation("WorkType");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.WorkType", b =>

{

b.HasOne("AutoFixStationDatabaseImplement.Models.TimeOfWork", "TimeOfWork")

.WithMany("WorkTypes")

.HasForeignKey("TimeOfWorkId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.Navigation("TimeOfWork");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.WorkType\_SparePart", b =>

{

b.HasOne("AutoFixStationDatabaseImplement.Models.SparePart", "SparePart")

.WithMany("Work\_SpareParts")

.HasForeignKey("SparePartId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.HasOne("AutoFixStationDatabaseImplement.Models.WorkType", "WorkType")

.WithMany("Work\_SpareParts")

.HasForeignKey("WorkTypeId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.Navigation("SparePart");

b.Navigation("WorkType");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.Car", b =>

{

b.Navigation("ServiceRecords");

b.Navigation("TOs");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.Employee", b =>

{

b.Navigation("TOs");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.SparePart", b =>

{

b.Navigation("Work\_SpareParts");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.StoreKeeper", b =>

{

b.Navigation("Works");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.TimeOfWork", b =>

{

b.Navigation("WorkTypes");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.TO", b =>

{

b.Navigation("Works");

});

modelBuilder.Entity("AutoFixStationDatabaseImplement.Models.WorkType", b =>

{

b.Navigation("Work\_SpareParts");

b.Navigation("Works");

});

#pragma warning restore 612, 618

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace AutoFixStationDatabaseImplement.Models

{

public class Car

{

public int Id { get; set; }

[Required]

public string Brand { get; set; }

[Required]

public string Model { get; set; }

[Required]

public string VIN { get; set; }

[Required]

public string OwnerPhoneNumber { get; set; }

[ForeignKey("CarId")]

public virtual List<ServiceRecord> ServiceRecords { get; set; }

[ForeignKey("CarId")]

public virtual List<TO> TOs { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace AutoFixStationDatabaseImplement.Models

{

public class Employee

{

public int Id { get; set; }

[Required]

public string FIO { get; set; }

[Required]

public string Login { get; set; }

[Required]

public string Password { get; set; }

[ForeignKey("EmployeeId")]

public virtual List<TO> TOs { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

namespace AutoFixStationDatabaseImplement.Models

{

public class ServiceRecord

{

public int Id { get; set; }

public int CarId { get; set; }

[Required]

public DateTime DateBegin { get; set; }

[Required]

public DateTime DateEnd { get; set; }

[Required]

public string Description { get; set; }

public virtual Car Car { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

using AutoFixStationContracts.Enums;

using System.ComponentModel.DataAnnotations.Schema;

namespace AutoFixStationDatabaseImplement.Models

{

public class SparePart

{

public int Id { get; set; }

[Required]

public string Name { get; set; }

[Required]

public string FactoryNumber { get; set; }

[Required]

public decimal Price { get; set; }

[Required]

public SparePartStatus Type { get; set; }

[Required]

public UnitMeasurement UMeasurement { get; set; }

[ForeignKey("SparePartId")]

public virtual List<WorkType\_SparePart> Work\_SpareParts { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace AutoFixStationDatabaseImplement.Models

{

public class StoreKeeper

{

public int Id { get; set; }

[Required]

public string FIO { get; set; }

[Required]

public string Login { get; set; }

[Required]

public string Password { get; set; }

[ForeignKey("StoreKeeperId")]

public virtual List<Work> Works { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace AutoFixStationDatabaseImplement.Models

{

public class TimeOfWork

{

public int Id { get; set; }

[Required]

public int Hours { get; set; }

[Required]

public int Mins { get; set; }

[ForeignKey("TimeOfWorkId")]

public virtual List<WorkType> WorkTypes { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

using AutoFixStationContracts.Enums;

using System.ComponentModel.DataAnnotations.Schema;

namespace AutoFixStationDatabaseImplement.Models

{

public class TO

{

public int Id { get; set; }

public int CarId { get; set; }

public int EmployeeId { get; set; }

[Required]

public decimal Sum { get; set; }

[Required]

public TOStatus Status { get; set; }

[Required]

public DateTime DateCreate { get; set; }

public DateTime? DateImplement { get; set; }

public DateTime? DateOver { get; set; }

public virtual Car Car { get; set; }

public virtual Employee Employee { get; set; }

[ForeignKey("TOId")]

public virtual List<Work> Works { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

using AutoFixStationContracts.Enums;

using System.ComponentModel.DataAnnotations.Schema;

namespace AutoFixStationDatabaseImplement.Models

{

public class Work

{

public int Id { get; set; }

public int StoreKeeperId { get; set; }

public int WorkTypeId { get; set; }

public int TOId { get; set; }

[Required]

public string WorkName { get; set; }

//[Required]

//public int Count { get; set; }

[Required]

public decimal Price { get; set; }

[Required]

public decimal NetPrice { get; set; }

[Required]

public WorkStatus WorkStatus { get; set; }

public DateTime? WorkBegin { get; set; }

[Required]

public int Count { get; set; }

public virtual StoreKeeper StoreKeeper { get; set; }

public virtual WorkType WorkType { get; set; }

public virtual TO TO { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace AutoFixStationDatabaseImplement.Models

{

public class WorkType

{

public int Id { get; set; }

public int TimeOfWorkId { get; set; }

[Required]

public string WorkName { get; set; }

[Required]

public decimal Price { get; set; }

[Required]

public decimal NetPrice { get; set; }//with spareparts

public virtual TimeOfWork TimeOfWork { get; set; }

[ForeignKey("WorkTypeId")]

public virtual List<WorkType\_SparePart> Work\_SpareParts { get; set; }

[ForeignKey("WorkTypeId")]

public virtual List<Work> Works { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel.DataAnnotations;

namespace AutoFixStationDatabaseImplement.Models

{

public class WorkType\_SparePart

{

public int Id { get; set; }

public int SparePartId { get; set; }

public int WorkTypeId { get; set; }

[Required]

public decimal Count { get; set; }

public virtual SparePart SparePart { get; set; }

public virtual WorkType WorkType { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using AutoFixStationDatabaseImplement.Models;

using Microsoft.EntityFrameworkCore;

namespace AutoFixStationDatabaseImplement

{

public class AutoFixStationDatabase : DbContext

{

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

if (optionsBuilder.IsConfigured == false)

{

optionsBuilder.UseSqlServer(@"Data Source=localhost\SQLEXPRESS;Initial Catalog=AutoFixStationDatabase;Integrated Security=True;MultipleActiveResultSets=True;");

}

base.OnConfiguring(optionsBuilder);

}

public virtual DbSet<Car> Cars { set; get; }

public virtual DbSet<Employee> Employees { set; get; }

public virtual DbSet<ServiceRecord> ServiceRecords { set; get; }

public virtual DbSet<SparePart> SpareParts { set; get; }

public virtual DbSet<StoreKeeper> StoreKeepers { set; get; }

public virtual DbSet<TimeOfWork> TimeOfWorks { set; get; }

public virtual DbSet<TO> TOs { set; get; }

public virtual DbSet<Work> Works { set; get; }

public virtual DbSet<WorkType> WorkTypes { set; get; }

public virtual DbSet<WorkType\_SparePart> WorkType\_SpareParts { set; get; }

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using AutoFixStationWebApp.Models;

using Microsoft.AspNetCore.Mvc;

using System.Diagnostics;

namespace AutoFixStationEmployeeWebApp.Controllers

{

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

//Главная страница веб-сайта

public IActionResult Index()

{

if (Program.Employee == null)

{

return Redirect("~/Home/Enter");

}

return View();

}

//Тех. осмотры

public IActionResult TOs()

{

if (Program.Employee == null)

{

return Redirect("~/Home/Enter");

}

return View(APIEmployee.GetRequest<List<TOViewModel>>($"api/to/gettolist?employeeId={Program.Employee.Id}"));

}

//Проверка на авторизацию

[HttpGet]

public IActionResult Privacy()

{

if (Program.Employee == null)

{

return Redirect("~/Home/Enter");

}

return View(Program.Employee);

}

//Проверка на авторизацию

[HttpPost]

public void Privacy(string login, string password, string fio)

{

if (!string.IsNullOrEmpty(login) && !string.IsNullOrEmpty(password)

&& !string.IsNullOrEmpty(fio))

{

APIEmployee.PostRequest("api/employee/updatedata",

new EmployeeBindingModel

{

Id = Program.Employee.Id,

FIO = fio,

Login = login,

Password = password

});

Program.Employee.FIO = fio;

Program.Employee.Login = login;

Program.Employee.Password = password;

Response.Redirect("Index");

return;

}

throw new Exception("Введите логин, пароль и ФИО");

}

//Ошибка

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel

{

RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier

});

}

[HttpGet]

public IActionResult Exit()

{

Program.Employee = null;

return Redirect("~/Home/Enter");

}

//Вход в веб-сайт

[HttpGet]

public IActionResult Enter()

{

return View();

}

//Вход в веб-сайт

[HttpPost]

public void Enter(string login, string password)

{

if (!string.IsNullOrEmpty(login) && !string.IsNullOrEmpty(password))

{

Program.Employee =

APIEmployee.GetRequest<EmployeeViewModel>($"api/employee/login?login={login}&password={password}");

if (Program.Employee == null)

{

throw new Exception("Неверный логин/пароль");

}

Response.Redirect("Index");

return;

}

throw new Exception("Введите логин, пароль");

}

//Регистрация

[HttpGet]

public IActionResult Register()

{

return View();

}

//Регистрация

[HttpPost]

public void Register(string login, string password, string fio)

{

if (!string.IsNullOrEmpty(login) && !string.IsNullOrEmpty(password)

&& !string.IsNullOrEmpty(fio))

{

APIEmployee.PostRequest("api/employee/register",

new EmployeeBindingModel

{

FIO = fio,

Login = login,

Password = password

});

Response.Redirect("Enter");

return;

}

throw new Exception("Введите логин, пароль и ФИО");

}

//Просмотр инфы о ТО (его статуса)

[HttpGet]

public IActionResult TOInfo(int toId)

{

TOViewModel tO = APIEmployee.GetRequest<TOViewModel>($"api/to/getto?toId={toId}");

return View(tO);

}

//Создание ТО

[HttpGet]

public IActionResult CreateTO()

{

ViewBag.Cars = APIEmployee.GetRequest<List<CarViewModel>>("api/car/carlist");

return View();

}

//Создание ТО

[HttpPost]

public void CreateTO(int carId)

{

APIEmployee.PostRequest("api/to/createto", new CreateTOBindingModel

{

CarId = carId,

EmployeeId = Program.Employee.Id,

Sum = null,

Works = null

});

Response.Redirect("Index");

}

//Просмотр инфо об услуге

[HttpGet]

public IActionResult WorkInfo(int workId)

{

WorkViewModel work = APIEmployee.GetRequest<WorkViewModel>($"api/work/getwork?workId={workId}");

WorkTypeViewModel workType = APIEmployee.GetRequest<WorkTypeViewModel>($"api/work/getworktype?worktypeId={work.WorkTypeId}");

ViewBag.Status = work.WorkStatus;

ViewBag.WorkBegin = work.WorkBegin;

ViewBag.Count = work.Count;

return View(workType);

}

//Просмотр автомобилей

[HttpGet]

public IActionResult Cars()

{

if (Program.Employee == null)

{

return Redirect("~/Home/Enter");

}

return View(APIEmployee.GetRequest<List<CarViewModel>>("api/car/carlist"));

}

//Добавление клиента/автомобиля

[HttpGet]

public IActionResult AddClient()

{

return View();

}

//Добавление клиента/автомобиля

[HttpPost]

public void AddClient(string brand, string model, string VIN, string number)

{

if (!string.IsNullOrEmpty(brand) && !string.IsNullOrEmpty(model) && !string.IsNullOrEmpty(VIN)

&& !string.IsNullOrEmpty(number))

{

APIEmployee.PostRequest("api/car/createorupdatecar",

new CarBindingModel

{

Brand = brand,

Model = model,

VIN = VIN,

OwnerPhoneNumber = number

});

Response.Redirect("Cars");

}

else

throw new Exception("Введите данные");

}

//Изменение клиента/автомобиля

[HttpGet]

public IActionResult EditCar(int carId)

{

return View(APIEmployee.GetRequest<CarViewModel>($"api/car/getcar?carId={carId}"));

}

//Добавление клиента/автомобиля

[HttpPost]

public void EditCar(int carId, string brand, string model, string VIN, string number)

{

if (!string.IsNullOrEmpty(brand) && !string.IsNullOrEmpty(model) && !string.IsNullOrEmpty(VIN)

&& !string.IsNullOrEmpty(number))

{

CarViewModel carViewModel = APIEmployee.GetRequest<CarViewModel>($"api/car/getcar?carId={carId}");

APIEmployee.PostRequest("api/car/createorupdatecar",

new CarBindingModel

{

Id = carId,

Brand = brand,

Model = model,

VIN = VIN,

OwnerPhoneNumber = number,

Records = carViewModel.Records

});

Response.Redirect("Cars");

}

else

throw new Exception("Введите данные");

}

//Подсчёт стоимости

[HttpPost]

public decimal[] Calc(decimal count, int type)

{

WorkTypeViewModel workType =

APIEmployee.GetRequest<WorkTypeViewModel>($"api/work/getworktype?worktypeId={type}");

decimal[] t = new decimal[2];

t[0] = workType.Price \* count;

t[1] = workType.NetPrice \* count;

return t;

}

//Создание услуги

[HttpGet]

public IActionResult CreateWork()

{

List<WorkTypeViewModel> listtypes = APIEmployee

.GetRequest<List<WorkTypeViewModel>>("api/work/getworktypelist");

List<TOViewModel> listto = APIEmployee

.GetRequest<List<TOViewModel>>($"api/to/gettolistinstart?employeeId={Program.Employee.Id}");

List<StoreKeeperViewModel> listkeep = APIEmployee

.GetRequest<List<StoreKeeperViewModel>>($"api/storekeeper/getstorekeeperlist");

ViewBag.TOs = listto;

ViewBag.WorkTypes = listtypes;

ViewBag.StoreKeepers = listkeep;

return View();

}

//Создание услуги

[HttpPost]

public void CreateWork(int toId, int workTypeId, int storeKeeperId, int count, decimal price, decimal netprice)

{

if (count == 0 || price == 0 || netprice == 0)

{

return;

}

APIEmployee.PostRequest("api/work/creatework",

new CreateWorkBindingModel

{

TOId = toId,

WorkTypeId = workTypeId,

StoreKeeperId = storeKeeperId,

Count = count,

Price = price,

NetPrice = netprice

});

Response.Redirect("Index");

}

//Просмотр услуг

[HttpGet]

public IActionResult Works()

{

if (Program.Employee == null)

{

return Redirect("~/Home/Enter");

}

return View(APIEmployee.GetRequest<List<WorkViewModel>>($"api/work/getworklistbyemployee?employeeId={Program.Employee.Id}"));

}

//Изменение статуса ТО

[HttpGet]

public IActionResult ActionWithTO(int toId)

{

var tO = APIEmployee.GetRequest<TOViewModel>($"api/to/getto?toId={toId}");

if (tO.Status.Equals("Принят"))

{

ViewBag.Action = "Выполняется";

}

else if (tO.Status.Equals("Выполняется"))

{

ViewBag.Action = "Готов";

}

else if (tO.Status.Equals("Готов"))

{

ViewBag.Action = "Выдан";

}

return View(tO);

}

//Действие с услугой

[HttpPost]

public void ActionWithTO(int toId, string action)

{

if (action.Equals("Выполняется"))

{

APIEmployee.PostRequest($"api/to/taketoinwork", new ChangeTOStatusBindingModel

{

TOId = toId

});

}

else if (action.Equals("Готов"))

{

APIEmployee.PostRequest($"api/to/finishto", new ChangeTOStatusBindingModel

{

TOId = toId

});

}

else if (action.Equals("Выдан"))

{

APIEmployee.PostRequest($"api/to/issueto", new ChangeTOStatusBindingModel

{

TOId = toId

});

}

Response.Redirect("Index");

}

//Изменение записи

[HttpGet]

public IActionResult EditRecord(int recordId)

{

return View(APIEmployee.GetRequest<ServiceRecordViewModel>($"api/to/getservicerecord?servicerecordId={recordId}"));

}

//Изменение записи

[HttpPost]

public void EditRecord(int recordId, string description)

{

if (!string.IsNullOrEmpty(description))

{

var record = APIEmployee.GetRequest<ServiceRecordViewModel>($"api/to/getservicerecord?servicerecordId={recordId}");

APIEmployee.PostRequest("api/to/updateservicerecord", new ServiceRecordBindingModel

{

Id = recordId,

Description = description,

CarId = record.CarId,

DateBegin = record.DateBegin,

DateEnd = record.DateEnd

});

Response.Redirect("Cars");

}

}

}

}

using AutoFixStationBusinessLogic.Mail;

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.ViewModels;

using Microsoft.AspNetCore.Mvc;

namespace AutoFixStationEmployeeWebApp.Controllers

{

public class ReportController : Controller

{

private readonly ILogger<ReportController> \_logger;

private readonly IWebHostEnvironment \_environment;

private readonly MailKitWorker \_mailKitWorker;

public ReportController(ILogger<ReportController> logger, IWebHostEnvironment environment, MailKitWorker mailKitWorker)

{

\_logger = logger;

\_environment = environment;

\_mailKitWorker = mailKitWorker;

}

public IActionResult ReportWordExcel()

{

if (Program.Employee == null)

{

return Redirect("~/Home/Enter");

}

return View(APIEmployee.GetRequest<List<TOViewModel>>($"api/to/GetTOListAllDone?employeeId={Program.Employee.Id}"));

}

[HttpPost]

public IActionResult CreateReportTOSparePartsToWordFile(List<int> tosId)

{

if (tosId.Count != 0)

{

var model = new ReportBindingModel

{

FileName = @"..\AutoFixStationEmployeeWebApp\wwwroot\reports\ReportTOSparePart.docx"

};

var list = new List<TOViewModel>();

foreach (var id in tosId)

{

list.Add(APIEmployee.GetRequest<TOViewModel>($"api/to/getto?toId={id}"));

}

model.TOs = list;

APIEmployee.PostRequest("api/report/CreateReportTOSparePartsToWordFile", model);

var fileName = "ReportTOSparePart.docx";

var filePath = \_environment.WebRootPath + @"\reports\" + fileName;

return PhysicalFile(filePath, "application/docx", fileName);

}

else

throw new Exception("Выберите хотя бы одно ТО");

}

[HttpPost]

public IActionResult CreateReportTOSparePartsToExcelFile(List<int> tosId)

{

if (tosId.Count != 0)

{

var model = new ReportBindingModel

{

FileName = @"..\AutoFixStationEmployeeWebApp\wwwroot\reports\ReportTOSparePart.xlsx"

};

var list = new List<TOViewModel>();

foreach(var id in tosId)

{

list.Add(APIEmployee.GetRequest<TOViewModel>($"api/to/getto?toId={id}"));

}

model.TOs = list;

APIEmployee.PostRequest("api/report/CreateReportTOSparePartsToExcelFile", model);

var fileName = "ReportTOSparePart.xlsx";

var filePath = \_environment.WebRootPath + @"\reports\" + fileName;

return PhysicalFile(filePath, "application/xlsx", fileName);

}

else

throw new Exception("Выберите хотя бы одно ТО");

}

public IActionResult ReportPDF()

{

if (Program.Employee == null)

{

return Redirect("~/Home/Enter");

}

return View();

}

[HttpPost]

public IActionResult ReportGetTOsPDF(DateTime dateFrom, DateTime dateTo)

{

ViewBag.Period = "C " + dateFrom.ToLongDateString() + " по " + dateTo.ToLongDateString();

return View("ReportPdf", APIEmployee.GetRequest<List<ReportTOsViewModel>>($"api/report/GetTOsReport?dateFrom={dateFrom.ToLongDateString()}&dateTo={dateTo.ToLongDateString()}"));

}

[HttpPost]

public IActionResult SendReportOnMail(DateTime dateFrom, DateTime dateTo)

{

var model = new ReportBindingModel

{

DateFrom = dateFrom,

DateTo = dateTo,

FileName = @"..\AutoFixStationEmployeeWebApp\wwwroot\reports\ReportTOsPdf.pdf"

};

APIEmployee.PostRequest("api/report/CreateReportTOsToPdfFile", model);

\_mailKitWorker.MailSendAsync(new MailSendInfoBindingModel

{

MailAddress = Program.Employee.Login,

Subject = "Отчет по тех. осмотрам. СТО \"Руки-Крюки\"",

Text = "Отчет по тех. осмотрам с " + dateFrom.ToShortDateString() + " по " + dateTo.ToShortDateString() +

"\nРаботник - " + Program.Employee.FIO,

FileName = model.FileName

});

ViewBag.Mail = Program.Employee.Login;

return View();

}

}

}

namespace AutoFixStationWebApp.Models

{

public class ErrorViewModel

{

public string? RequestId { get; set; }

public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

}

}

using Newtonsoft.Json;

using System;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Text;

namespace AutoFixStationEmployeeWebApp

{

public class APIEmployee

{

private static readonly HttpClient client = new HttpClient();

public static void Connect(IConfiguration configuration)

{

client.BaseAddress = new Uri(configuration["IPAddress"]);

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(new

MediaTypeWithQualityHeaderValue("application/json"));

}

public static T GetRequest<T>(string requestUrl)

{

var response = client.GetAsync(requestUrl);

var result = response.Result.Content.ReadAsStringAsync().Result;

if (response.Result.IsSuccessStatusCode)

{

return JsonConvert.DeserializeObject<T>(result);

}

else

{

throw new Exception(result);

}

}

public static void PostRequest<T>(string requestUrl, T model)

{

var json = JsonConvert.SerializeObject(model);

var data = new StringContent(json, Encoding.UTF8, "application/json");

var response = client.PostAsync(requestUrl, data);

var result = response.Result.Content.ReadAsStringAsync().Result;

if (!response.Result.IsSuccessStatusCode)

{

throw new Exception(result);

}

}

}

}

using AutoFixStationContracts.ViewModels;

namespace AutoFixStationEmployeeWebApp

{

public class Program

{

public static EmployeeViewModel Employee { get; set; }

public static void Main(string[] args)

{

CreateHostBuilder(args).Build().Run();

}

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host

.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder =>

{

webBuilder.UseStartup<Startup>();

});

}

}

using AutoFixStationBusinessLogic.Mail;

using AutoFixStationContracts.BindingModels;

namespace AutoFixStationEmployeeWebApp

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

APIEmployee.Connect(configuration);

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services

//to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddControllersWithViews();

services.AddSingleton<MailKitWorker>();

}

// This method gets called by the runtime. Use this method to configure the

//HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

// The default HSTS value is 30 days. You may want to change

//this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

});

var mailSender = app.ApplicationServices.GetService<MailKitWorker>();

mailSender.MailConfig(new MailConfigBindingModel

{

MailLogin = Configuration?.GetSection("MailLogin")?.Value.ToString(),

MailPassword = Configuration?.GetSection("MailPassword")?.Value.ToString(),

SmtpClientHost = Configuration?.GetSection("SmtpClientHost")?.Value.ToString(),

SmtpClientPort = Convert.ToInt32(Configuration?.GetSection("SmtpClientPort")?.Value.ToString()),

PopHost = Configuration?.GetSection("PopHost")?.Value.ToString(),

PopPort = Convert.ToInt32(Configuration?.GetSection("PopPort")?.Value.ToString())

});

}

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.ViewModels;

using Microsoft.AspNetCore.Mvc;

namespace AutoFixStationRestAPI.Controllers

{

[Route("api/[controller]/[action]")]

[ApiController]

public class CarController : ControllerBase

{

private readonly ICarLogic \_logic;

public CarController(ICarLogic logic)

{

\_logic = logic;

}

[HttpGet]

public List<CarViewModel> CarList() => \_logic.Read(null)?.ToList();

[HttpGet]

public CarViewModel GetCar(int carId) => \_logic

.Read(new CarBindingModel { Id = carId })?[0];

[HttpPost]

public void CreateOrUpdateCar(CarBindingModel car) => \_logic.CreateOrUpdate(car);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.ViewModels;

using Microsoft.AspNetCore.Mvc;

namespace AutoFixStationRestAPI.Controllers

{

[Route("api/[controller]/[action]")]

[ApiController]

public class EmployeeController : ControllerBase

{

private readonly IEmployeeLogic \_logic;

public EmployeeController(IEmployeeLogic logic)

{

\_logic = logic;

}

[HttpGet]

public EmployeeViewModel Login(string login, string password)

{

var list = \_logic.Read(new EmployeeBindingModel

{

Login = login,

Password = password

});

return (list != null && list.Count > 0) ? list[0] : null;

}

[HttpPost]

public void Register(EmployeeBindingModel model) =>

\_logic.CreateOrUpdate(model);

[HttpPost]

public void UpdateData(EmployeeBindingModel model) =>

\_logic.CreateOrUpdate(model);

[HttpPost]

public void Delete(EmployeeBindingModel model) =>

\_logic.Delete(model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.ViewModels;

using Microsoft.AspNetCore.Mvc;

namespace AutoFixStationRestAPI.Controllers

{

[Route("api/[controller]/[action]")]

[ApiController]

public class ReportController : ControllerBase

{

private readonly IReportLogic \_reportLogic;

public ReportController(IReportLogic reportLogic)

{

\_reportLogic = reportLogic;

}

[HttpPost]

public void CreateReportTOSparePartsToWordFile(ReportBindingModel model) => \_reportLogic.SaveTOSparePartToWordFile(model);

[HttpPost]

public void CreateReportTOSparePartsToExcelFile(ReportBindingModel model) => \_reportLogic.SaveTOSparePartToExcelFile(model);

[HttpPost]

public void CreateReportTOsToPdfFile(ReportBindingModel model) => \_reportLogic.SaveTOsByDateToPdfFile(model);

[HttpGet]

public List<ReportTOsViewModel> GetTOsReport(string dateFrom, string dateTo) => \_reportLogic.GetTOs(new ReportBindingModel { DateFrom = Convert.ToDateTime(dateFrom), DateTo = Convert.ToDateTime(dateTo) });

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.ViewModels;

using Microsoft.AspNetCore.Mvc;

namespace AutoFixStationRestAPI.Controllers

{

[Route("api/[controller]/[action]")]

[ApiController]

public class SparePartController : ControllerBase

{

private readonly ISparePartLogic \_logic;

public SparePartController(ISparePartLogic logic)

{

\_logic = logic;

}

[HttpGet]

public List<SparePartViewModel> GetSparePartList() => \_logic.Read(null)?.ToList();

[HttpGet]

public SparePartViewModel GetSparePart(int sparepartId) => \_logic

.Read(new SparePartBindingModel { Id = sparepartId })?[0];

[HttpPost]

public void CreateOrUpdateSparePart(SparePartBindingModel model) => \_logic.CreateOrUpdate(model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.ViewModels;

using Microsoft.AspNetCore.Mvc;

namespace AutoFixStationRestAPI.Controllers

{

[Route("api/[controller]/[action]")]

[ApiController]

public class StoreKeeperController : ControllerBase

{

private readonly IStoreKeeperLogic \_logic;

public StoreKeeperController(IStoreKeeperLogic logic)

{

\_logic = logic;

}

[HttpGet]

public StoreKeeperViewModel Login(string login, string password)

{

var list = \_logic.Read(new StoreKeeperBindingModel

{

Login = login,

Password = password

});

return (list != null && list.Count > 0) ? list[0] : null;

}

[HttpGet]

public List<StoreKeeperViewModel> GetStoreKeeperList() =>

\_logic.Read(null);

[HttpPost]

public void Register(StoreKeeperBindingModel model) =>

\_logic.CreateOrUpdate(model);

[HttpPost]

public void UpdateData(StoreKeeperBindingModel model) =>

\_logic.CreateOrUpdate(model);

[HttpPost]

public void Delete(StoreKeeperBindingModel model) =>

\_logic.Delete(model);

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.ViewModels;

using Microsoft.AspNetCore.Mvc;

namespace AutoFixStationRestAPI.Controllers

{

[Route("api/[controller]/[action]")]

[ApiController]

public class TOController : ControllerBase

{

private readonly ITOLogic \_tOLogic;

private readonly IServiceRecordLogic \_serviceRecordLogic;

public TOController(ITOLogic tOLogic,

IServiceRecordLogic serviceRecordLogic)

{

\_tOLogic = tOLogic;

\_serviceRecordLogic = serviceRecordLogic;

}

/// <summary>

/// Получение списка ТО для работника по его номеру (Id)

/// </summary>

/// <param name="employeeId"></param>

/// <returns></returns>

[HttpGet]

public List<TOViewModel> GetTOList(int employeeId)

=> \_tOLogic

.Read(new TOBindingModel { EmployeeId = employeeId });

/// <summary>

/// Получение списка ТО для работника по его номеру (Id) для добавления услуги

/// </summary>

/// <param name="employeeId"></param>

/// <returns></returns>

[HttpGet]

public List<TOViewModel> GetTOListInStart(int employeeId)

{

return \_tOLogic

.Read(new TOBindingModel { EmployeeId = employeeId })

.Where(x => x.Status.Equals("Выполняется")).ToList();

}

/// <summary>

/// Получение списка ТО для работника по его номеру (Id)

/// </summary>

/// <param name="employeeId"></param>

/// <returns></returns>

[HttpGet]

public List<TOViewModel> GetTOListAllDone(int employeeId)

{

return \_tOLogic

.Read(new TOBindingModel { EmployeeId = employeeId })

.Where(x => x.Status.Equals("Выдан"))

.ToList();

}

/// <summary>

/// Получение определенного ТО по его номеру

/// (передается с помощью выбора из компонента)

/// </summary>

/// <param name="tOId"></param>

/// <returns></returns>

[HttpGet]

public TOViewModel GetTO(int tOId)

=> \_tOLogic.Read(new TOBindingModel { Id = tOId })?[0];

/// <summary>

/// Получение списка ТО по номерам

/// </summary>

/// <param name="tosId"></param>

/// <returns></returns>

[HttpGet]

public List<TOViewModel> GetTOs(List<int> tosId)

{

var list = new List<TOViewModel>();

foreach (var tO in tosId)

{

list.Add(\_tOLogic.Read(new TOBindingModel { Id = tO })?[0]);

}

return list;

}

/// <summary>

/// Создание ТО

/// </summary>

/// <param name="model"></param>

[HttpPost]

public void CreateTO(CreateTOBindingModel model)

=> \_tOLogic.CreateTO(model);

/// <summary>

/// Изменение статуса ТО на "Выполняется"

/// </summary>

/// <param name="model"></param>

[HttpPost]

public void TakeTOInWork(ChangeTOStatusBindingModel model)

=> \_tOLogic.TakeTOInWork(model);

/// <summary>

/// Изменение статуса ТО на "Готово"

/// </summary>

/// <param name="model"></param>

[HttpPost]

public void FinishTO(ChangeTOStatusBindingModel model)

=> \_tOLogic.FinishTO(model);

/// <summary>

/// Изменение статуса ТО на "Выдано"

/// </summary>

/// <param name="model"></param>

[HttpPost]

public void IssueTO(ChangeTOStatusBindingModel model)

=> \_tOLogic.IssueTO(model);

[HttpGet]

//Получение записи

public ServiceRecordViewModel GetServiceRecord(int servicerecordId)

{

return \_serviceRecordLogic.Read(new ServiceRecordBindingModel

{

Id = servicerecordId

})?[0];

}

[HttpPost]

//Изменение записи

public void UpdateServiceRecord(ServiceRecordBindingModel model)

{

\_serviceRecordLogic.Update(model);

}

}

}

using AutoFixStationContracts.BindingModels;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.ViewModels;

using AutoFixStationContracts.WebViewModels;

using Microsoft.AspNetCore.Mvc;

namespace AutoFixStationRestAPI.Controllers

{

[Route("api/[controller]/[action]")]

[ApiController]

public class WorkController : ControllerBase

{

private readonly IWorkLogic \_workLogic;

private readonly IWorkTypeLogic \_workTypeLogic;

private readonly ITimeOfWorkLogic \_timeOfWorkLogic;

private readonly ITOLogic \_tOLogic;

public WorkController(IWorkLogic workLogic,

IWorkTypeLogic workTypeLogic,

ITimeOfWorkLogic timeOfWorkLogic,

ITOLogic tOLogic)

{

\_workLogic = workLogic;

\_workTypeLogic = workTypeLogic;

\_timeOfWorkLogic = timeOfWorkLogic;

\_tOLogic = tOLogic;

}

/// <summary>

/// Получение списка типов работ

/// </summary>

/// <returns></returns>

[HttpGet]

public List<WorkTypeViewModel> GetWorkTypeList()

=> \_workTypeLogic.Read(null)?.ToList();

/// <summary>

/// Получение определенного типа работы по Id

/// (передался из селектед лист с помощью выбора

/// из всех типов работ)

/// </summary>

/// <param name="workTypeId"></param>

/// <returns></returns>

[HttpGet]

public WorkTypeViewModel GetWorkType(int workTypeId)

=> \_workTypeLogic

.Read(new WorkTypeBindingModel { Id = workTypeId })?[0];

//Создание типа услуги

[HttpPost]

public void CreateOrUpdateWorkType(WorkTypeBindingModel model) =>

\_workTypeLogic.CreateOrUpdate(model);

//Удаление типа услуги

[HttpPost]

public void DeleteWorkType(WorkTypeBindingModel model) =>

\_workTypeLogic.Delete(model);

/// <summary>

/// Получение всех текущих работ кладовщика по его номеру (Id)

/// </summary>

/// <param name="storeKeeperId"></param>

/// <returns></returns>

[HttpGet]

public List<WorkViewModel> GetWorkList(int storeKeeperId)

=> \_workLogic

.Read(new WorkBindingModel { StoreKeeperId = storeKeeperId });

/// <summary>

/// Получение всех текущих работ кладовщика по его номеру (Id)

/// </summary>

/// <param name="storeKeeperId"></param>

/// <returns></returns>

[HttpGet]

public List<WorkViewModel> GetWorkListByEmployee(int employeeId)

{

var tolist = \_tOLogic.Read(new TOBindingModel { EmployeeId = employeeId })?.ToList();

var listworks = new List<WorkViewModel>();

foreach (var i in tolist)

{

var elem = \_workLogic.Read(new WorkBindingModel { TOId = i.Id });

foreach(var item in elem)

{

listworks.Add(item);

}

}

return listworks;

}

/// <summary>

/// Создание работы

/// </summary>

/// <param name="model"></param>

[HttpPost]

public void CreateWork(CreateWorkBindingModel model)

=> \_workLogic.CreateWork(model);

/// <summary>

/// Изменение статуса работы на "Выполняется"

/// </summary>

/// <param name="model"></param>

[HttpPost]

public void TakeWorkInWork(ChangeWorkStatusBindingModel model)

=> \_workLogic.TakeWorkInWork(model);

/// <summary>

/// Изменение статуса работы на "Готово"

/// </summary>

/// <param name="model"></param>

[HttpPost]

public void FinishWork(ChangeWorkStatusBindingModel model)

=> \_workLogic.FinishWork(model);

/// <summary>

/// Получение работы по номеру

/// </summary>

/// <param name="workId"></param>

/// <returns></returns>

[HttpGet]

public WorkViewModel GetWork(int workId) => \_workLogic

.Read(new WorkBindingModel { Id = workId })?[0];

//Получение листа времени выполнений

[HttpGet]

public List<TimeOfWorkWebViewModel> GetTimeOfWorkList()

{

var list = \_timeOfWorkLogic.Read(null)?.ToList();

return list.Select(x => new TimeOfWorkWebViewModel

{

Id = x.Id,

Time = "Часов: " + x.Hours + ". Минут: " + x.Mins

}).ToList();

}

//Получение времени выполнения по номеру

[HttpGet]

public TimeOfWorkViewModel GetTimeOfWork(int timeofworkId) =>

\_timeOfWorkLogic.Read(new TimeOfWorkBindingModel

{

Id = timeofworkId

})?[0];

//Получение времени выполнения по номеру

[HttpPost]

public void CreateOrUpdateTimeOfWork(TimeOfWorkBindingModel model) =>

\_timeOfWorkLogic.CreateOrUpdate(model);

}

}

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace AutoFixStationRestAPI

{

public class Program

{

public static void Main(string[] args)

{

CreateHostBuilder(args).Build().Run();

}

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder =>

{

webBuilder.UseStartup<Startup>();

});

}

}

using AutoFixStationBusinessLogic.BusinessLogics;

using AutoFixStationBusinessLogic.OfficePackage;

using AutoFixStationBusinessLogic.OfficePackage.Implements;

using AutoFixStationContracts.BusinessLogicsContracts;

using AutoFixStationContracts.StorageContracts;

using AutoFixStationDatabaseImplement.Implements;

using Microsoft.OpenApi.Models;

namespace AutoFixStationRestAPI

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services

//to the container.

public void ConfigureServices(IServiceCollection services)

{

//Хранилища

services.AddTransient<IEmployeeStorage, EmployeeStorage>();

services.AddTransient<IStoreKeeperStorage, StoreKeeperStorage>();

services.AddTransient<ICarStorage, CarStorage>();

services.AddTransient<IServiceRecordStorage, ServiceRecordStorage>();

services.AddTransient<ISparePartStorage, SparePartStorage>();

services.AddTransient<ITimeOfWorkStorage, TimeOfWorkStorage>();

services.AddTransient<ITOStorage, TOStorage>();

services.AddTransient<IWorkStorage, WorkStorage>();

services.AddTransient<IWorkTypeStorage, WorkTypeStorage>();

//Логика

services.AddTransient<IEmployeeLogic, EmployeeLogic>();

services.AddTransient<IStoreKeeperLogic, StoreKeeperLogic>();

services.AddTransient<ICarLogic, CarLogic>();

services.AddTransient<IServiceRecordLogic, ServiceRecordLogic>();

services.AddTransient<ISparePartLogic, SparePartLogic>();

services.AddTransient<ITimeOfWorkLogic, TimeOfWorkLogic>();

services.AddTransient<ITOLogic, TOLogic>();

services.AddTransient<IWorkLogic, WorkLogic>();

services.AddTransient<IWorkTypeLogic, WorkTypeLogic>();

services.AddTransient<IReportLogic, ReportLogic>();

services.AddTransient<AbstractSaveToWord, SaveToWord>();

services.AddTransient<AbstractSaveToExcel, SaveToExcel>();

services.AddTransient<AbstractSaveToPdf, SaveToPdf>();

services.AddControllers().AddNewtonsoftJson();

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "AutoFixStationRestAPI",

Version = "v1"

});

});

}

// This method gets called by the runtime. Use this method to configure the

//HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

app.UseSwagger();

app.UseSwaggerUI(c =>

c.SwaggerEndpoint("/swagger/v1/swagger.json", "AutoFixStationRestApi v1"));

}

app.UseHttpsRedirection();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

}