

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГТУ»)

Факультет экономики, менеджмента и информационных технологий
(факультет)

Кафедра Систем управления информационных технологий в строительстве

КУРСОВОЙ ПРОЕКТ

по дисциплине «Объектно-ориентированное программирование»

тема «Разработка программного обеспечения с использованием объектно-ориентированного подхода»

Расчетно-пояснительная записка

Разработал студент

05.06.2022 Д. В. Тюленев
Подпись, дата Инициалы, фамилия

Руководитель

Е. Н. Королев
Подпись, дата Инициалы, фамилия

Члены комиссии

Подпись, дата Инициалы, фамилия

Подпись, дата Инициалы, фамилия

Нормоконтролер

Е. Н. Королев
Подпись, дата Инициалы, фамилия

Защищена

05.06.22

дата

Оценка

отлично

2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГТУ»)

Кафедра Систем управления информационных технологий в строительстве

ЗАДАНИЕ
на курсовой проект

по дисциплине «Объектно-ориентированное программирование»

тема «Разработка программного обеспечения с использованием объектно-ориентированного подхода»

Студент группы ИСТ-214

Тюленев Данил Вячеславович

Фамилия, имя, отчество

Вариант 4. Служба такси

Технические условия процессор Intel® Core™ i3-8145U CPU @ 2,10 ГГц,
операционная система Windows 10, ОЗУ 8192 МБ

Содержание и объем проекта (графические работы, расчеты и прочее):
анализ предметной области и требований к программному обеспечению
(7 страниц); моделирование и разработка системы на основе принципов
ООП (19 страниц); реализация системы на общесистемном языке
программирования (11 страниц); 25 рисунков, 1 приложение

Сроки выполнения этапов анализ предметной области и требований к
программному обеспечению (01.03 – 25.03); моделирование и разработка
системы на основе принципов ООП (26.03 – 19.04); реализация системы на
общесистемном языке программирования (20.04 – 11.05); описание диалога
с пользователем (12.05 – 21.05); оформление пояснительной записи
(22.05 – 04.06)

Срок защиты курсового проекта 05.06.2022

Руководитель

Подпись, дата

Е. Н. Королев

Инициалы, фамилия

Задание принял студент

Подпись, дата

Д. В. Тюленев

Инициалы, фамилия

Содержание

Введение.....	2
1. Анализ предметной области.....	4
1.1. Особенности предметной области.....	4
1.2. Описание бизнес процессов в рамках предметной области.....	5
1.3. Проблемы, возникающие в данной предметной области и перспективы их решения с использованием программных средств.....	6
1.4. Анализ существующих аналогов.....	7
1.5. Цель и задачи курсового проектирования.....	9
2. Моделирование и разработка системы на основе принципов ООП.....	11
2.1. Постановка задачи.....	11
2.2. Проектирование базы данных для системы.....	12
2.3. Объектно-ориентированные принципы и компоненты.....	15
2.4. Разработка классов.....	22
3. Реализация системы на общесистемном языке программирования.....	30
3.1. Выбор средств программной реализации.....	30
3.2. Алгоритм выполнения программы.....	31
3.3. Модульная структура программы.....	32
3.4. Описание диалога с пользователем.....	32
Заключение.....	41
Список использованной литературы.....	43
Приложение.....	44

Введение

В настоящее время в связи с развитием компьютерной техники появилась возможность автоматизировать многие процессы, увеличился объем обрабатываемой информации, вследствие чего возникла объективная необходимость автоматизировать большую часть сферы человеческой деятельности.

Для принятия обоснованных и эффективных решений в производственной деятельности появилась возможность при помощи компьютеров и средств связи получать, накапливать, хранить и обрабатывать данные.

Агрегаторы такси реализовали новые способы предоставления и потребления услуг такси, что в значительно повлияло на активное развитие сферы транспортных услуг. Агрегаторы могут конкурировать между собой, с классическими службами такси, с общественным и персональным транспортом, а также другими формами решения транспортных задач.

Агрегаторы работают на двустороннем рынке, где обе стороны, пассажир и перевозчик являются потребителем услуги агрегатора. Для пассажира агрегатор предлагает удобный способ запроса на транспортную услугу. Для водителя агрегатор является поставщиком информации о запросе на перевозку от пассажира. Специфика потребителей рынка агрегаторов в том, что обе группы заинтересованы в нарастании количества пользователей данной платформы. Чем больше водителей, тем удобнее платформа пассажирам. Чем больше пассажиров, тем востребованнее сервис агрегатора у водителей. Водители и пассажиры в пределах региона никак не ограничены в выборе того или иного агрегатора. Денежные потоки агрегатора могут формироваться только за счёт агентских комиссионных сборов от поездки.

Агрегаторы заявляют себя ИТ-компаниями, которые помогают найти друг друга водителю и пассажиру. Поэтому они не подпадают под действие действующей редакции закона «О такси». У агрегатора нет «диспетчерской службы такси» и «транспортного средства». Агрегаторы не несут

ответственности за риски ДТП, за несвоевременную доставку пассажира в точку назначения, не несут расходов на лицензирование, техническое обслуживание, медицинский контроль, что делает бизнес-модель чрезвычайно привлекательной. Онлайн-сервисы могут заключать партнёрства с целыми таксопарками, могут привлекать «независимых» самозанятых водителей.

1. Анализ предметной области

1.1. Особенности предметной области

Работа такси осуществляется следующим образом: каждый водитель, заступая на смену, связывается с диспетчером, и тот вносит его в карточку работающих в данный момент. Сведения карточки отображаются на экране. Водители держат обратную связь по радио.

Заказы поступают по телефону к диспетчеру, он записывает необходимые данные в базу заказов. Дата и время поступления заказа диспетчер вводит сам или вводиться автоматически. Клиент может сразу у диспетчера узнать стоимость заказа, и только потом заказывать.



Рисунок 1 — Иллюстрация процесса работы сервиса такси

Потом из списка свободных водителей выбирает того, кто будет выполнять заказ или водитель сам отвечает по обратной связи, что примет заказ. После выполнения заказа водитель по радио отчитывается перед диспетчером.

По завершению каждой смены диспетчер формирует отчёт, который показывает, сколько заказов поступило, сколько выполнено, сколько было отменено, и их общую стоимость.

Эти отчёты поступают к администратору, на основании этого всего администратор формирует общий отчёт (сводный отчёт) за определённую дату. В конце месяца администратор, пользуясь своими отчётами, формирует отчёт по итогам месяца и отправляет его высшему руководству такси.

Также администратор занимается кадрами: формирует дела новых сотрудников, и вносит необходимые поправки в дела уже работающих на этом предприятии. На администраторе лежит ответственность за правильное формирование дел и их сохранность.

1.2. Описание бизнес процессов в рамках предметной области

Сегодня очень часто слышны разговоры о том, что в недалёком будущем такси будут работать без водителя. Об этом рассуждают специалисты Uber и Яндекс-такси.

В принципе, обе эти компании уже идут по пути автоматизации и отказа от человеческого фактора везде, где удаётся. В результате можно прийти к следующей схеме:

- Заказ такси автоматически, через сайт или приложение без участия диспетчера.
- Доставка клиента до места назначения.
- Оплата автоматически, с банковской карты или интернет-денег после поездки на основе GPS-данных.

Моделирование и оптимизация бизнес процессов являются актуальными задачами современности. На основе изучения бизнес процессов можно предвидеть и избежать многих проблем в работе компаний, связанных прежде всего с повышенным уровнем затрат, низким качеством выполняемых работ и произведённых продуктов, излишней длительностью выполнения функций. Применение инструмента имитационного моделирования для бизнес процессов позволяет ещё на этапе планирования оценить различные показатели эффективности функционирования процессов, выявить наилучшую последовательность выполнения функций, определить затраты ресурсов,

вычислить загруженность персонала, что тем самым способствует своевременному принятию правильных вариантов управленческих решений. На основе методологии моделирования Business Model and Notation (BPMN) построена функциональная диаграмма бизнес процессов, заданы используемые ресурсы, затраты на их использование, построены календарные графики работ. Далее осуществлено имитационное моделирование функционирования процесса при различных входных данных: количества поступающих заявок, числа диспетчеров и бюджета на рекламу.



Рисунок 2 — Схема бизнес процессов внутри службы такси

1.3. Проблемы, возникающие в данной предметной области и перспективы их решения с использованием программных средств

Конечно, при этом в самой службе такси все равно работают люди (операторы техподдержки, специалисты по обслуживанию программного обеспечения и техники, модераторы отзывов и т.д.). Но в описанном выше бизнес-процессе в случае отказа от водителей они перестают принимать участие вообще.

С одной стороны, все получается удобно и выгодно. Нет людей – нет случайных ошибок, затрат на создание рабочих мест и заработную плату. С

другой, если из цепочки полностью исключаются люди, то возникает множество рисков.

Что будет, если программа водителя-автомата даст сбой, и человека отвезут не туда? А как робот будет реагировать в случае аварии, особенно, если по причине ДТП повредится какой-то аппаратный узел? А если преступники или террористы решат захватить и использовать робота-такси в своих целях? При всей внешней выгоде исключение из цепочки человека ведёт к непредсказуемым последствиям и требует внедрения каких-то защитных механизмов, в результате внедрения которых (или даже не внедрения) компания несёт дополнительные расходы, т.е. результат противоположен тому, который планировался.

1.4. Анализ существующих аналогов

Относительно молодые и перспективные игроки – агрегаторы. К ним относятся как всем известные компании вроде «Яндекс.Такси», Uber, Wheely, Gett, «Максим», таки ещё пока не слишком популярный, но занимающий лидирующие позиции Fasten.

Подобные сервисы выполняют роль связующего звена между пассажиром и водителем. Каждая из сторон устанавливает на свой гаджет специальное приложение (в разных версиях). Пассажир с его помощью делает заказ, указывая своё местонахождение и пункт назначения. Водитель видит этот заказ и откликается на него.

Таким образом, агрегатор просто предоставляет неограниченному кругу таксистов доступ к базе клиентов и заказов, которые формируются в режиме онлайн. Крупные агрегаторы дополнительно к этому заключают договора с таксопарками и ИП на сотрудничества только с ними. При этом за доступ к базе клиентуры агрегатора через приложение, выплачивается фиксированная комиссия(процент) от каждого заказа. Если брендировать свой автомобиль (раскрасить его в цвета агрегатора), величину комиссии можно уменьшить. Но

работать тогда придётся только с одним агрегатором. Последнему это нужно для повышения узнаваемости бренда, и увеличению конкурентоспособности.

В целом принцип работы всех компаний агрегаторов такси схож, однако каждый сервис имеет свои особенности. «Яндекс.Такси» вместе с Uber, например, задумались над идеей страхования жизни и здоровья пассажиров на время поездок. Gett предлагает пассажирам поездки от 50 рублей. Бизнес-модель Fasten ориентирована в первую очередь на водителя, а не на пассажира. Через «Максим» заказ можно сделать и по телефону, и через мобильное приложение.

Был выявлен ряд недостатков, которые существуют на данный момент в мобильных сервисах по заказу такси:

— Uber. Американская международная частная компания из Сан-Франциско, создавшая одноимённое мобильное приложение для поиска, вызова и оплаты такси или частных водителей. При первом запуске приложение вынуждает потенциального клиента выполнять множество раздражающих действий, которые можно было бы спрятать в дополнительные настройки. При регистрации обязательным пунктом является привязка кредитной карты (20 ± 10 секунд), кроме того, бывают ситуации когда кредитки у человека в данный момент просто нет. Подобные решения со стороны Uber на мой взгляд можно отнести к минусам, так как они увеличивают не дружелюбность интерфейса. Во многих ситуациях легче будет найти такси на дороге, пусть даже дороже.

— GetTaxi. Большое количество ненужных настроек, из-за чего приложение долго загружается. Часто выпадает реклама, примерно по 1-2 экрана для заполнения email адресов и уведомления о новых услугах. При первом запуске требует регистрацию, без регистрации заказ такси невозможен.

— Яндекс-Такси. Некоторые элементы приложения не интуитивно понятны для новых пользователей, к примеру, первый экран с картой не всегда показывает геолокацию с самого запуска. Какое-то время идёт поиск, и кнопка дальнейшего шага не появляется, в следствии чего появляется мысль о том, что приложение не работает, потому что в дизайне нет никаких обновляющихся

элементов, которые указывали бы на то, что приложение пытается найти Ваше текущее местоположение. Также приложение требует обязательной регистрации.

По результатам анализа существующих решений можно сделать вывод, что конкуренты предоставляют слишком много ненужных функций, помещая их на передний план, тем самым перегружая приложение, которое изначально должно уметь хорошо выполнять всего одну функцию.

1.5. Цель и задачи курсового проектирования

Целью работы является создание эффективной и удобную программу, позволяющую выполнять следующие функции: заказать транспорт клиентам службы такси с необходимыми характеристиками, предоставлять перевозчикам список доступных заказов с возможностью принять более подходящий, администрирование зарегистрированных учётных записей двух сторон, создание системы мониторинга для таксопарка. Для разработки программного обеспечения были поставлены следующие задачи:

- Предоставление в панели администратора возможностей редактирования наполнения таксопарка: добавление и удаление моделей машин.
- Реализация интуитивно понятного для пользователя графического интерфейса приложения.
- Обеспечение пользователя сервиса возможностью создания учётной записи под конкретный тип аккаунта (клиент, водитель, администратор).
- Приложение должно предоставлять возможность оформление заказов со стороны клиента сервиса: адрес точки отправки, модель машин и тип транспорта.
- Готовому программному решению необходимо обрабатывать запросы перевозчика на аренду необходимой модели машины из таксопарка.

- Программа должна обеспечивать всех пользователей с учётной записью типа «перевозчик» возможностью просматривать доступные, свободные клиентские заказы на транспортировку.
- Соблюдение основополагающих принципов объектно-ориентированного программирования при разработке программного решения.
- Проверка валидности совершения операций оформления заказа, создания аккаунта и разработки модели транспорта.
- Создание возможности для разработки микросервисных модулей с управляющей сущностью в роли менеджера служб, чтобы уменьшить связанность отдельный составляющих частей программы.

2. Моделирование и разработка системы на основе принципов ООП

2.1. Постановка задачи

Разработанное приложение должно выполнять ряд задач, связанных с процессом агрегации машин по вызову.

Программа должна предоставлять: удобный инструмент с возможностью администрирования состояний машин в сервисном гараже и зарегистрированных аккаунтов, профилей клиентов и водителей; возможность оформления заказов с клиенткой стороны (выбор типа машины, её класс, и определение адреса вызова); обработчик доступных заказов, с возможностью зарегистрировать заказ за определённым доступным водителем; возможность для выбора необходимой машины под определённого авторизованного водителя.

В начале проектирования программного решения выбранной темы необходимо разделить главную задачу на отдельные сегменты подзадачи различного характера:

- Поиск средств решения поставленной задачи (платформа для создания графического пользовательского интерфейса, основной язык программирования, система управления базами данных)
- Разработать модуль, который будет ответственен за контроль заказов такси (создать механизм для формирования заказов со стороны клиента и процесса оформления заказа на водителя).
- Создать интуитивный инструмент для процесса обработки данных со стороны администратора службы такси (мониторинг учётных записей пользователей, проектирование новых моделей машин в таксопарке).
- Проектирование базы данных для хранения данных, приходящих от сервисов (создание схем таблиц, разработка их связей между собой).

- Создание интерфейса для соединения клиентской части приложения с базой данных, понятный и простой инструмент для создания запросов к необходимым таблицам.
- Формирование представления удобного графического интерфейса (создание форм, разработка функций обработчиков).
- Провести тестирование работоспособности готового решения выбранной темы курсовой работы.

Приложение должно работать, с использованием системы средств для взаимодействия пользователя с экземпляром, основанной на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов на экран стационарной машины.

Готовое решение должно обладать интуитивно понятной структурной особенностью, которая будет работать с разными пользователями по-разному. Для каждого из типов аккаунтов будет доступен различный функционал, необходимый под каждого определённого пользователя.

При открытии процесса экземпляра программы необходимо загружать представление начального окна, которое будет предоставлять возможность авторизоваться в созданный ранее аккаунт, либо зарегистрировать новый.

2.2. Проектирование базы данных для системы

При проектировании таблиц базы данных необходимо учитывать специфику возможностей данного решения.

В проекте предусматривается создание профиля, для хранения данных, необходимых под каждый из типов аккаунта. Процесс регистрации и авторизации в аккаунт требует держать информацию о логинах и паролей, уникальных для каждой сущности таблицы. Каждая запись данной таблицы (`account_authentication`) хранит три поля: логин учётной записи (`account_login`), пароль (`account_password`), состояние (`account_state`), предложенный тип (`account_type`) и уникальный токен (`account_token`). Логин и пароль позволяет

определить имеется ли учётная запись, зарегистрированная на данном сервисе. При успешном процессе аутентификация пользователь сможет получить информацию о типе аккаунта, состоянии (имеется ли система, в которой вход в профиль уже выполнен). Тип аккаунта позволяет определить с какой из доступных таблиц необходимо далее работать, а токен (статический уникальный идентификатор GUID) позволяет из выбранной таблицы найти запись с данными.

Для хранения данных о пользователях с разными типами учётных записей в СУБД спроектированы три таблицы под каждый вариант аккаунта. Каждая из них имеет поле account_guid, чтобы была возможность для связи с другими таблицами, относящихся к данному разделу проектирования. Каждое отношение содержит данные о имени (username), возрасте (age), полу (gender) и отдельную информацию касательно вариации пользователя.

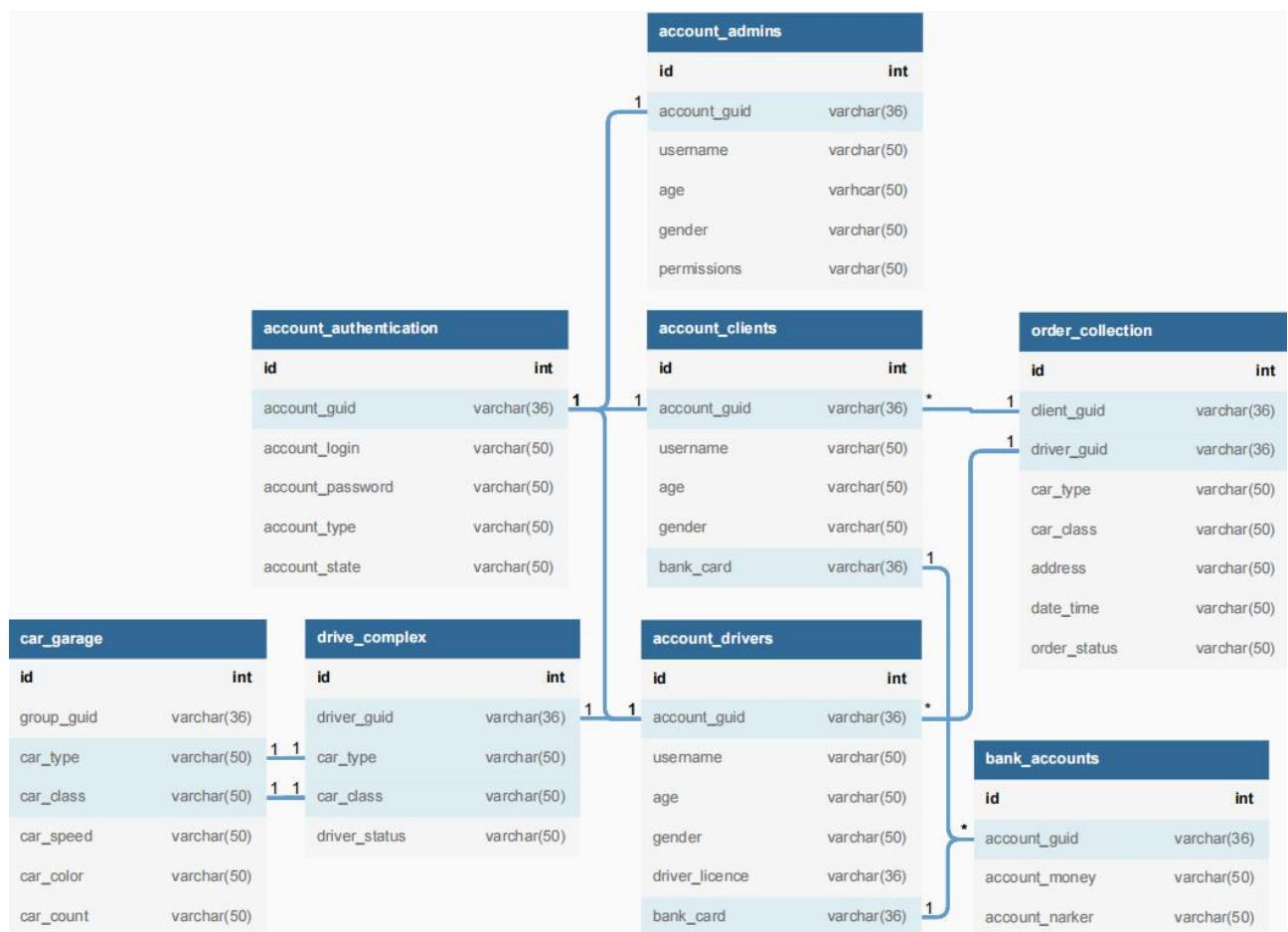


Рисунок 3 — Схема отношений таблиц базы данных

Таблица для пользователей, являющихся администраторами службы (account_admins) хранит запись о доступных полномочиях: доступ к другим аккаунтам водителей и клиентов, и разрешение контролировать поставку машин в гараж (добавление новых и удаление доступных моделей).

Отношения представляющие сущности клиента (account_clients) и перевозчика (account_drivers) содержат строки с данными банковских аккаунтов (bank_card) для возможности транзакций (оплаты и выдачи зарплаты). Записи таблицы account_drivers также включают поле с номером водительского удостоверения (driver_licence).

В drive_complex представлены установленные совокупности водителей (driver_guid) с привязанными к ним моделями машин, полученных от службы отвечающей за множество машин в гараже таксопарка. Таблица содержит поля: типа модели машины (car_type), класс транспорта (car_class), статус водителя (driver_status — устанавливается в перевозчиком приложении и виден всем авторизированным клиентам).

Отношение car_garage представляет собой сущность автогаража, содержит информацию о всех добавленных моделях автомобилей: идентификатор группы идентичных машин (group_guid), количество моделей в группе (car_count), типа модели машины (car_type — уровень эконом, премиум и детский), класс транспорта (car_class — легковой и тяжёлый), скорость (car_speed) и цвет (car_color) модели.

Для хранения позиций зарегистрированных клиентских заказов представлена таблица order_collection. Она содержит следующие поля: идентификатор пользователя (client_guid), идентификатор водителя (driver_guid — значение может быть пропущено, пока заказ не возьмёт один из перевозчиков), тип заказанной модели машины (car_type), класс транспорта (car_class), адрес вызова такси (address), время и дата оформления заказа

(date_time) и статус заказа (order_status — имеет два состояния; если заказ ожидает установления водителем, значение false, иначе true).

Отношение bank_account формирует концепцию банковского профиля, для хранение денежных средств, доступных для перевода между аккаунтами (оформление платы за проезд, списание с счёта при покупки услуг водителем). Поле account_guid содержит номер к которому прикрепляются денежные средства хранящиеся в account_money.

2.3. Объектно-ориентированные принципы и компоненты

Объектно-ориентированное программирование (ООП) представляет собой паттерн программирования, в которой программа состоит из объектов, которые взаимодействуют между собой. Объекты создаются на основании класса, определённого в коде и объединяют данные и действия, которые можно выполнять с данными, в одно целое. Основные принципы структурирования в случае ООП связаны с различными аспектами базового понимания предметной задачи, требующее нужного управления определённой моделью:

- Инкапсуляция — свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе.
- Наследование — свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствованной функциональностью.
- Полиморфизм — свойство системы, которое позволяет использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

В процессе разработки архитектуры программного решения необходимо разделить реализацию проекта на отдельные малосвязанные составляющие. Для работы с такими сервисными компонентами необходимо разработать удобный инструмент, который будет способен позволять размещать данные

компоненты в контролируемой коллекций, опрашивать их состояние и, при необходимости, предоставлять к ним доступ.

Для программной реализации данного механизма применяется паттерн проектирования под названием «Строитель» (Builder). «Строитель» является паттерном проектирования, который инкапсулирует создание объекта и позволяет разделить на различные этапы. В роли участников выступают: «Продукт» (Product) — объект, который необходимо создать, «Строитель» (Builder) — определяет интерфейс для создания различных частей объекта «Продукта», «Директор» (Director) — распорядитель, создаёт «Продукт», используя объекты Builder.

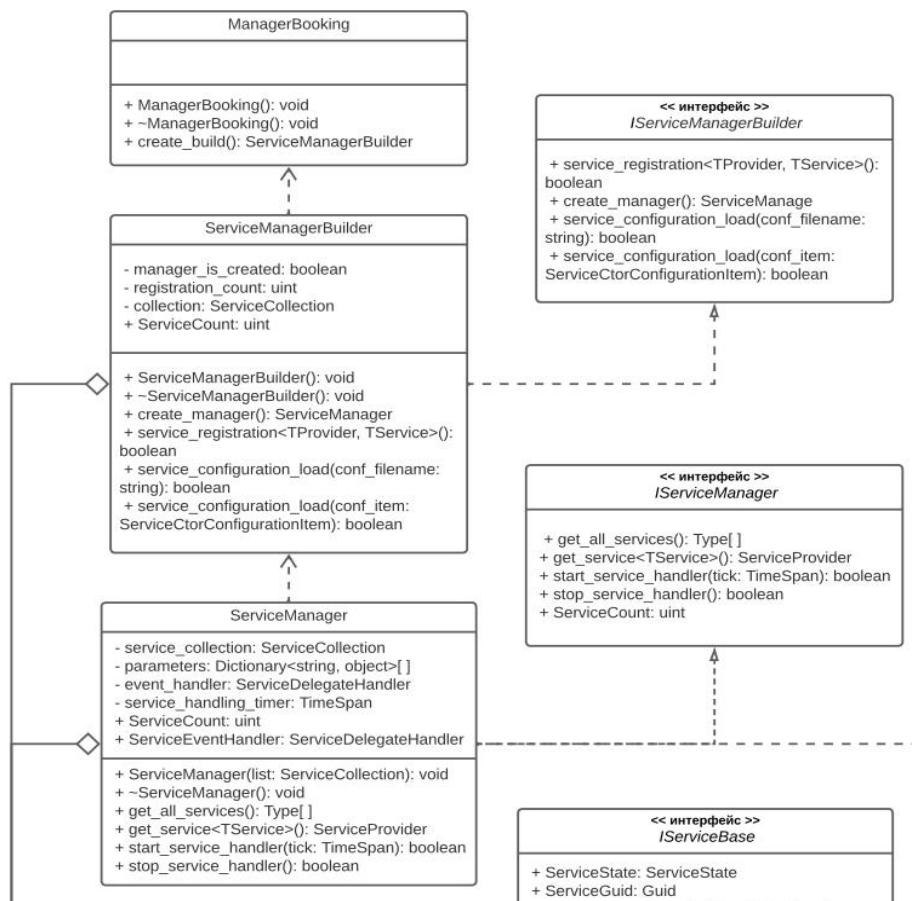


Рисунок 4 — Внутренняя архитектура отношений классов «ServiceManager» и «ServiceManagerBuilder» (Диаграмма UML)

В качестве продукта платформы будет выступать класс «Менеджер Сервисов» ([class] Service Manager). Данный пользовательский тип данных, предоставляет необходимый функционал для работы с компонентами. Класс инкапсулирует внутри себя коллекцию сервисов, которые представляют собой объекты класса «ServiceBase»; реализует интерфейс «IServiceBase», который определяет некоторое внутреннее поведение.

Пользовательский тип «ServiceManagerBuilder» проводит первоначальную настройку, после чего появляется возможность создания экземпляра объекта «ServiceManager».

При помощи обобщённого метода «service_registration» производиться регистрация сервисов (добавление в коллекцию сервисов), определяется тип сервиса, за которым закрепляется провайдер (с его помощью будет упакован сервис, и будет предоставляться доступ). Данный метод проводит настройку службы в соответствии с указанной в самом методе конфигурацией: внедряются зависимости (другие сервисы, которые ранее были уже зарегистрированы), подключается список с конфигурационными переменными. В качестве возвращаемого значения, используется значение, для определение состояния, регистрации пользовательского сервиса (true = сервис был успешно зарегистрирован, false = произошла ошибка при добавлении службы).

Для установки конфигурационного списка, используемого в регистрации сервисов, используется перегруженный метод «service_configuration_load», каждый из которых определяет значения для структуры, в которой, для каждой записи относящейся к отдельной секции (узлу) конфигурации присоединяется таблица с элементами пар ключ-значение. Одна из перегрузок позволяет указать объект типа «ServiceCtorConfigurationItem», который определяет конфигурацию в виде иерархии узлов, за которыми цепляются списки с конфигурационными значениями. Другая перегрузка позволяет установить конфигурационные узлы, используя специальный форматированный файл конфигураций XML, в котором указывается настройка для необходимых

сервисов. Метод возвращает значение состояния ошибки подключения настроек.

```
<?xml version="1.0" encoding="utf-8"?>
<service_parameters_list>
    <service_parameters service_name="database_provider">
        <sql_connection_server>localhost</sql_connection_server>
        <sql_connection_user>root</sql_connection_user>
        <sql_connection_database>test</sql_connection_database>
        <sql_connection_password>prologdy778</sql_connection_password>
    </service_parameters>
    <service_parameters service_name="depot_manager">
        <garage_collection_size>100</garage_collection_size>
        <car_child_type_price>300</car_child_type_price>
        <car_premium_type_price>500</car_premium_type_price>
        <car_econom_type_price>200</car_econom_type_price>
    </service_parameters>
    <service_parameters service_name="order_controller">
        <order_request_wait_second>20</order_request_wait_second>
    </service_parameters>
</service_parameters_list>
```

Рисунок 5 — Формат хранения конфигурации внутри XML файла

Делегат типа «ServiceDelegateHandler» инкапсулирует внутри себя поведение, которое должны совершать подписанные сервисные функции, отвечающие за создание сигнала, отклика в ответ на запросы со стороны обработчика сервисов.

Чтобы создать экземпляр объекта класса «ServiceManager» применяется метод «create_manager»; он внедряет собранную коллекцию подготовленных сервисов внутрь конструктора менеджера.

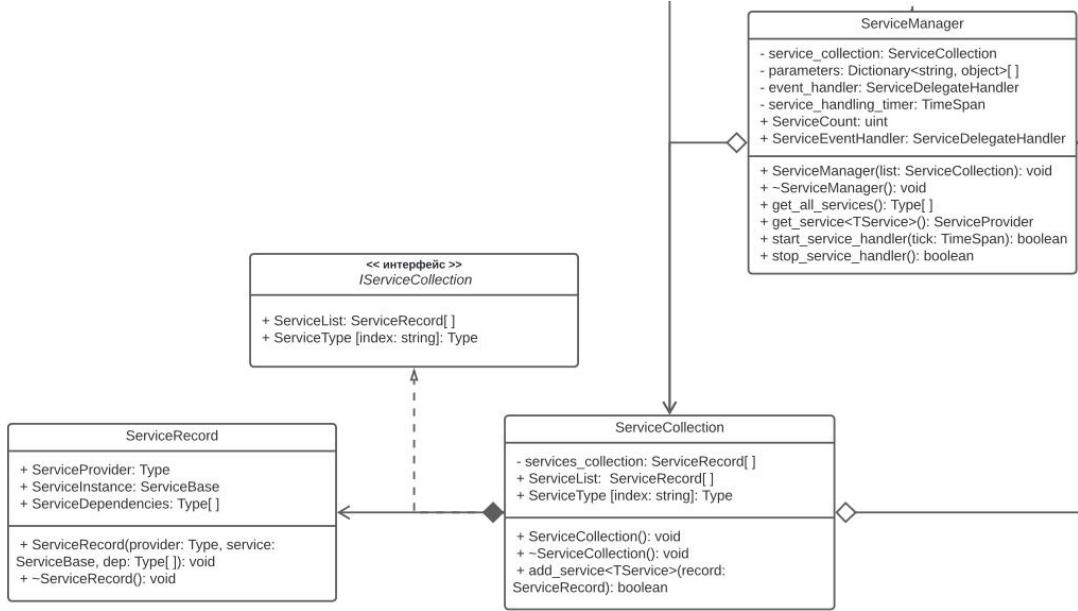


Рисунок 6 — Внутренняя архитектура отношений классов «ServiceManager» и «ServiceCollection» (Диаграмма UML)

Также «Builder» позволяет получить количество зарегистрированных сервисов, используя свойство «ServiceCount».

Класс «ServiceCollection» представляет сущность, которая удерживает в себе все необходимую информацию о сервисах, их зависимостях и провайдерах. Чтобы добавить службу в контейнер. Необходимо вызвать шаблонный метод «`add_service`», для которого в качестве универсального параметра указывается тип подключаемого сервиса. Данная программная единица реализует интерфейс «`IServiceCollection`», который определяет два доступных свойства: «`ServiceList`» — отдаёт массив записей «`ServiceRecord`» (структура для формирования записей о элементах коллекции: собранный объект сервиса «`ServiceInstance`», тип провайдера «`ServiceProvider`», множество подключённые зависимости «`ServiceDependencies`»), связанных с установленными службами; «`ServiceType`» — индексатор, для получения типа службы, используя её псевдоним.

Класс менеджера «`ServiceManager`» представляет объект для управления сервисами, наследующие абстрактный класс «`ServiceBase`». Экземпляр

предоставляет возможность для доступа к необходимому сервису, его метаданным.

Сущность реализует интерфейс «IServiceManager», который устанавливает поведение для каждой версии менеджера: метод «*get_all_services*» позволяет получить список всех сервисов (их типы данных, классы), установленных в менеджере; метод «*get_service*» — отдаёт под управление конкретный сервис, указав его тип в качестве универсального параметра шаблона, в качестве результата возвращает провайдер, с подключённым к нему запрашиваемой службой; метод «*start_service_handler*» — начинает процесс опрашивания зарегистрированных служб с установленным по времени интервалом, используя для этого отдельный поток из пула, а метод «*stop_service_handler*» — останавливает данный процесс.

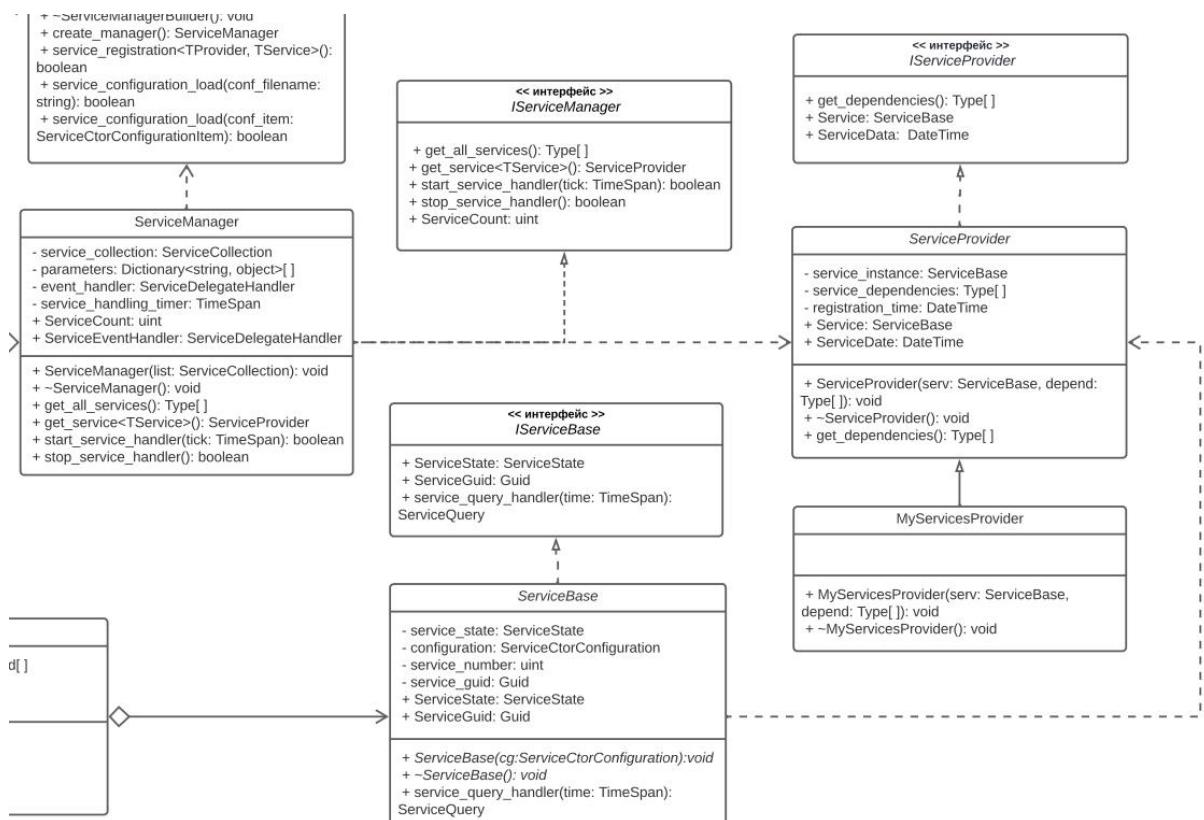


Рисунок 7 — Внутренняя архитектура отношений классов «ServiceManager», «ServiceProvider» и «ServiceBase» (Диаграмма UML)

Абстрактный класс «ServiceBase» представляет собой базовый класс для каждого сервиса, который необходимо присоединить в менеджер служб. Каждый такой объект является основной единицей при проектировании решения, устанавливает необходимый минимум, набор свойств и действий, которые необходимо содержать каждому проектируемому сервису. Реализует интерфейс «IServiceBase», определяющий, какой функционал закладывается в каждый класс-сервис: чисто виртуальный (абстрактный) метод «service_query_handler» определяет действия который должен совершить экземпляр служебного класс в моменте, когда менеджер опрашивает загруженные элементы (данный процесс происходит в отдельном потоке), возвращает объект структуры «ServiceQuery», которая содержит значение: передаваемого сообщения «Message», тип класса сервиса «ServiceType» и состояние службы «State».

Каждая служба имеет свой глобальный идентификатор GUID установленный в момент подключения в коллекцию с помощью создателя менеджера и переменную отвечающую за состояние («Idle» — незанятый, служба не опрашивается; «Running» — служба опрашивается; «Killed» — служба отключена); доступ к GUID службы осуществляется через свойство «ServiceGuid»; доступ к текущему состоянию сервиса — свойство «ServiceState».

Абстрактный класс «ServiceProvider» является родительским классом для всех провайдеров, отвечающих за доступ к службам, предоставляющих полную информацию о элементах коллекции. Реализует интерфейс «IServiceProvider», устанавливает следующее поведение: метод «get_dependencies» позволяет получить все зависимости, встроенные в привязанный сервис; свойство «ServiceDate» возвращает время регистрации службы, а свойство «Service» передаёт доступ к вызванной пользователем вложенной в провайдер службе.

2.4. Разработка классов

При создании приложения, решающее выбранную проблему, используется подход, при котором единое решение строится как набор небольших сервисов, каждый из которых работает в собственном изолированном процессе и коммуницирует с остальными подобными элементами используя легковесные механизмы.

Для создания вышеописанных служб, используется класс «ServiceBase», который выступает в качестве программного элемента от которого наследуются все сервисы. На стадии планирования было выявлены 5 потребностей, которые должны быть решены созданными службами: связь с базой данных, управление моделями в таксопарке, обработка заказов со стороны клиентов, внутренние денежные переводы между водителями и клиентами, администрирование зарегистрированных учётных записей.

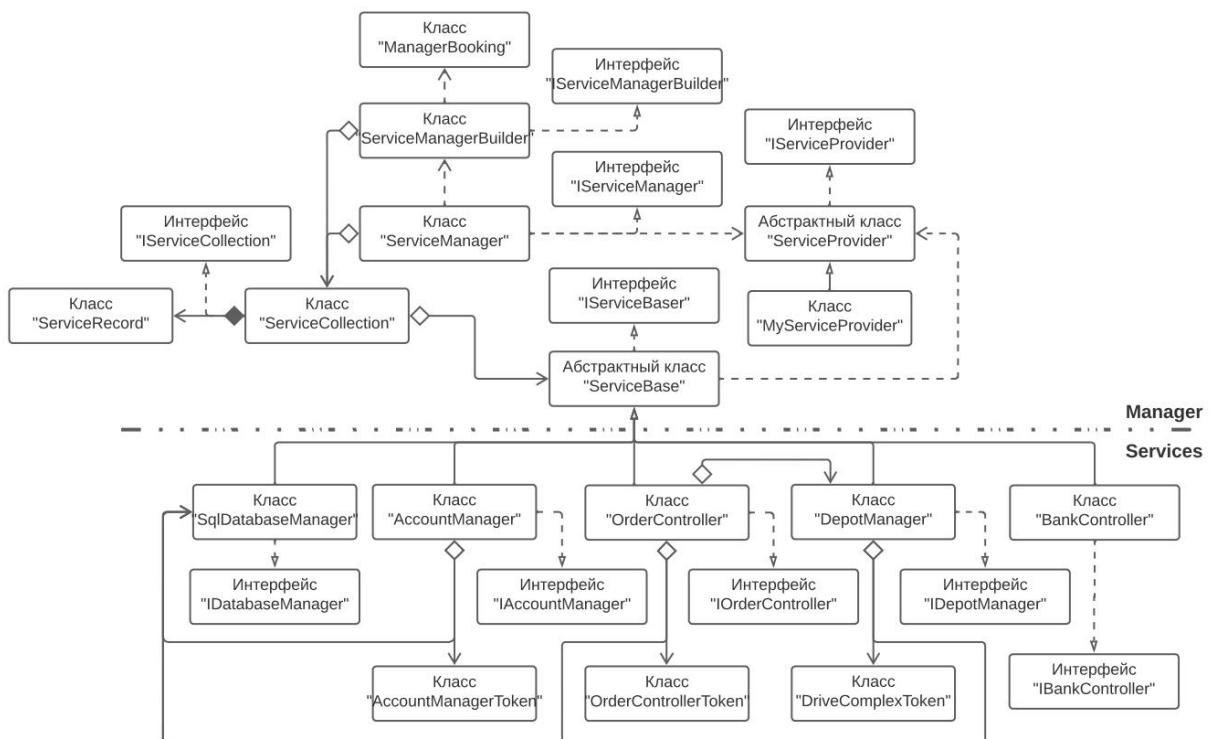


Рисунок 8 — Диаграмма UML для всех доступных классов

Чтобы решить задачу, касательно необходимости создания связи между собранной программой и спроектированной базой данных, разработан сервис «SqlDatabaseManager». Конструктор сервиса использует экземпляр класса «ServiceCtorConfiguration», который представляет собой список необходимых для настройки подключения конфигурации (сервер подключения «connection_server», пользователь БД «connection_user», название базы данных для подключения «connection_database», пароль, предоставляющий доступ «connection_password»), которые установлены в узле, запрашиваемом при помощи атрибута «ServiceConfigurationAttribute». Данный пользовательский атрибут, класс определён в пространстве имён блока Менеджера Служб «Service Manager», устанавливает метку на конструктор, который будет отвечать за настройку сервиса, содержит название установленного блока с конфигурацией, запрашиваемого со стороны создаваемого сервиса (конфигурация для дальнейших сервисов настраивается аналогичным способом).

```
[Manager::ServiceConfigurationAttribute("database_provider")]
SqlDatabaseManager(IServiceBase::ServiceCtorConfiguration^ conf) :
    : Manager::ServiceBase(conf)
{
    this->db = gcnew Threading::Mutex();
    this->coi = gcnew List<SqlDatabaseFieldKey>();
    this->db_build_connection();
}
```

Рисунок 9 — Пример использования атрибута «ServiceConfigurationAttribute»

Данный класс предоставляет инструмент для подключения к выбранным таблицам и создания запросов, реализует интерфейс «IDatabaseManager», устанавливающий список действий, которые должны уметь выполнять любые подписанные версии службы: метод «send_database_data» осуществляет отправку записей в базу данных, на место параметра указывается объект структуры, в который упакованы передаваемые данные, в качестве результата возвращает состояние выполнения; метод «update_database_data» отвечает за

обновление данных в строке выбранной таблицы, в роли входных значений устанавливаются объект структуры с данными и список ключ для определения необходимой позиции; метод «get_database_data» — результатом является контейнер с объектами, собранными из таблицы при поиске записей на основе списка пар ключ-значение; метод «delete_database_data» удаляет строку на основе найденного при поиске значения с установленным фильтром — в указанном столбце (указывается ключ, название).

Чтобы указать с какой из таблиц размещённых в БД необходимо работать, проводить операции (транзакции), запросы используется определённый в классе метод «set_scheme_struct», который позволяет на основе параметра шаблона установить структуру нужной таблицы: название таблицы и её поля. Чтобы класс являлся валидной моделью для работы с методом, он должен в заголовке содержать атрибут типа «SqlDatabaseTableAttribute» (определен в пространстве сервиса), который определяет название таблицы; далее свойствам также устанавливаются метаданные при помощи атрибута «SqlDatabaseFieldAttribute», отвечающего за имена полей таблицы.

```
[Services::SqlDatabaseTableAttribute("account_authentication")]
public ref class AccountAuthenticationDbScheme sealed : ISqlDataBaseSchemaType
{
public: [Services::SqlDatabaseFieldAttribute("account_guid")] → property System::String^ account;
public: [Services::SqlDatabaseFieldAttribute("account_login")] → property System::String^ login;
public: [Services::SqlDatabaseFieldAttribute("account_password")] → property System::String^ password;
public: [Services::SqlDatabaseFieldAttribute("account_type")] → property System::String^ type;
public: [Services::SqlDatabaseFieldAttribute("account_state")] → property System::String^ state;
```

Рисунок 10 — Пример использования атрибутов для создания схемы БД.

Ответственным за мониторинг и управлением гаражом службы такси выступает класс-сервис «DepotManager». Для первоначальной настройки используется конструктор с установленным пользовательским классом, атрибутом «ServiceConfigurationAttribute», в который передаётся конфигурационный объект и зависимости, в лице зарегистрированного сервиса «SqlDatabaseManager», который используется для связи сервиса с соответствующими таблицами в БД.

Чтобы менеджер сервисов понимал, какие зависимости запрашивает сервис и, в соответствии с разработанным списком внедрял их через конструктор, используется пользовательский атрибут «ServiceRequireAttribute». Данный атрибут принимает в качестве входного параметра тип необходимого сервиса, который будет встраиваться в качестве зависимости.

```
[Manager::ServiceAttribute::ServiceRequireAttribute(Services::DepotManager::typeid)]
[Manager::ServiceAttribute::ServiceRequireAttribute(Services::SqlDatabaseManager::typeid)]
public ref class OrderController sealed : Manager::ServiceBase, Services::IOrderController
{
public:
    [Manager::ServiceConfigurationAttribute("order_controller")]
    OrderController(IServiceBase::ServiceCtorConfiguration^ conf, SqlDatabaseManager^ sql_manager,
                    DepotManager^ depot_manager) : Manager::ServiceBase(conf), session_order_count(0), service_sql_manager(sql_manager), service_depot_manager(depot_manager)
    {
        this->order_request_second = System::UInt32::Parse((String^)this->configuration["order_request_second"]);
        this->service_sql_manager = sql_manager; this->service_depot_manager = depot_manager;
    }
}
```

Рисунок 11 — Пример использования атрибута «ServiceRequireAttribute»

Целью для установки атрибута является класс (т. е. атрибут указывается перед заголовком класса). Для создания коллекции зависимостей можно указать последовательно несколько атрибутов «ServiceRequireAttribute».

В процессе внедрения необходимых служб сборщик менеджера получает тип зависимости из доступных атрибутов класса путём «рефлексии» (динамическое выявление типов) и после в конструктор передаёт ссылку на уже ранее созданный сервис (запрашиваемый) из контейнера «ServiceCollection».

Класс «DepotManager» реализует интерфейс «IDepotManager», устанавливающий определённое поведение: метод «add_car_model» добавляет новую модель в гараж, загружая информацию о ней в базу данных, в качестве входного параметра принимает объект дочернего класса «CarBaseModel» (представляет собой структуру модели машины, описывающий её характеристики) и их количество, возвращает состояние ошибки загрузки; метод «delete_car_model» удаляет модели из таксопарка; метод «rent_car_model» позволяет арендовать выбранную машину, закрепив её за водителем службы такси, в роли параметров указываются экземпляр требуемой

модели машины и идентификатор пользователя, возвращает состояние аренды; метод «return_car_model» возвращает арендованную машину обратно в гараж.

Сервис содержит запись о текущем установленном перевозчике «DriveComplexToken». Она содержит информацию о состоянии водителя, его идентификационный номер и модели арендованной машины (если автомобиль под арендой не выбран, то устанавливается значение «Null»).

В качестве методов доступа предложены: «get_all_cars» — возвращает список зарегистрированных моделей авто, «get_all_drivers» — возвращает коллекцию данных о подключённых водителях, «get_driver_complex» — возвращает информацию о водителе с необходимым идентификатором.

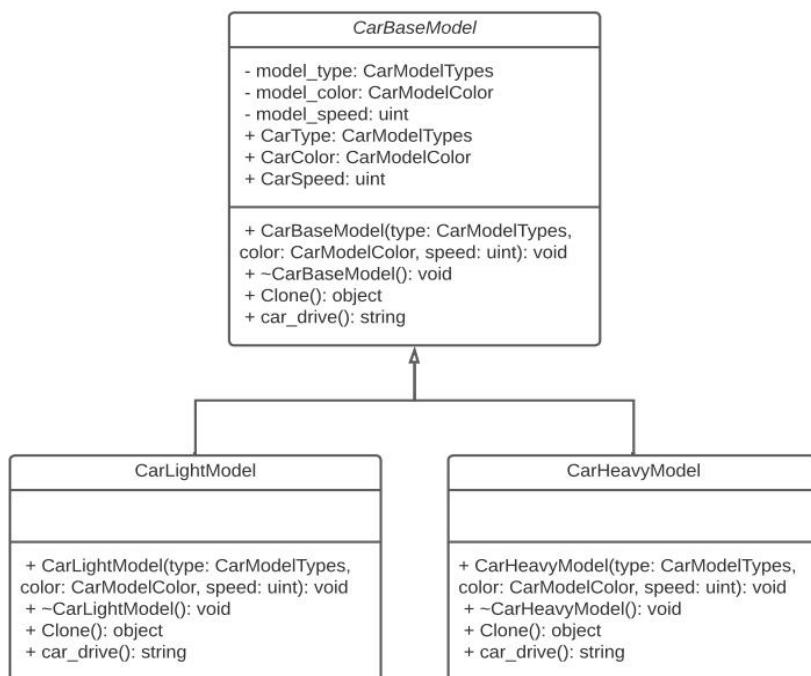


Рисунок 12 — UML диаграмма классов, моделей автомобилей

Для мониторинга заказов используется сервис «OrderController», который позволяет совершать заказы автотранспорта необходимой категории, принимать зарегистрированные заказы и вести их учёт. В качестве зависимостей класс использует сервис для соединения с базой данных

«SqlDatabaseManager» и службу «DepotManager» для проверки закрепления заказа за перевозчиком.

Реализует интерфейс «IOrderController», устанавливающий минимальный необходимый функционал: метод «registration_order» позволяет зарегистрировать заказ, для этого указываются адрес, модель машины и идентификатор клиента (в процессе проверки состояния заказа используется отдельный поток, внутри которого опрашивается база данных на наличие изменения статуса заказа); метод «accept_order» даёт возможность подтвердить заказ со стороны водителя, используя для этого сервис «DepotManager»; метод «cancellation_order» отменяет заказ.

Сервис «AccountManager» предоставляет инструмент для управления учётными записями. Для работы необходимо использовать зависимость в виде службы «SqlDatabaseManager» для управления определённой в схеме таблицей. Реализует интерфейс «IAccountManager», устанавливающий поведение: метод «registration_account» регистрирует аккаунт, в качестве входных значений используется модель аккаунта типа «AccountBaseModel» (устанавливает настройки под выбранный тип аккаунта), строка логина и пароля, также метод устанавливает экземпляр текущего авторизованного аккаунта типа «AccountManagerToken» (структура, характеризующая модель учётной записи: идентификатор, имя, возраст, пол и дополнительные конфигурации в зависимости от аккаунта) в скрытое поле, предоставляя свойства для доступа к необходимым данным; метод «sign_out_account» производит процедуру выхода из аккаунта, очищает токен с данными аккаунта; метод «authorization_account» производит процесс авторизации в выбранную учётную запись, запрашивает в роли параметров строки логина и пароля; метод «delete_account» очищает информацию о аккаунте из базы данных.

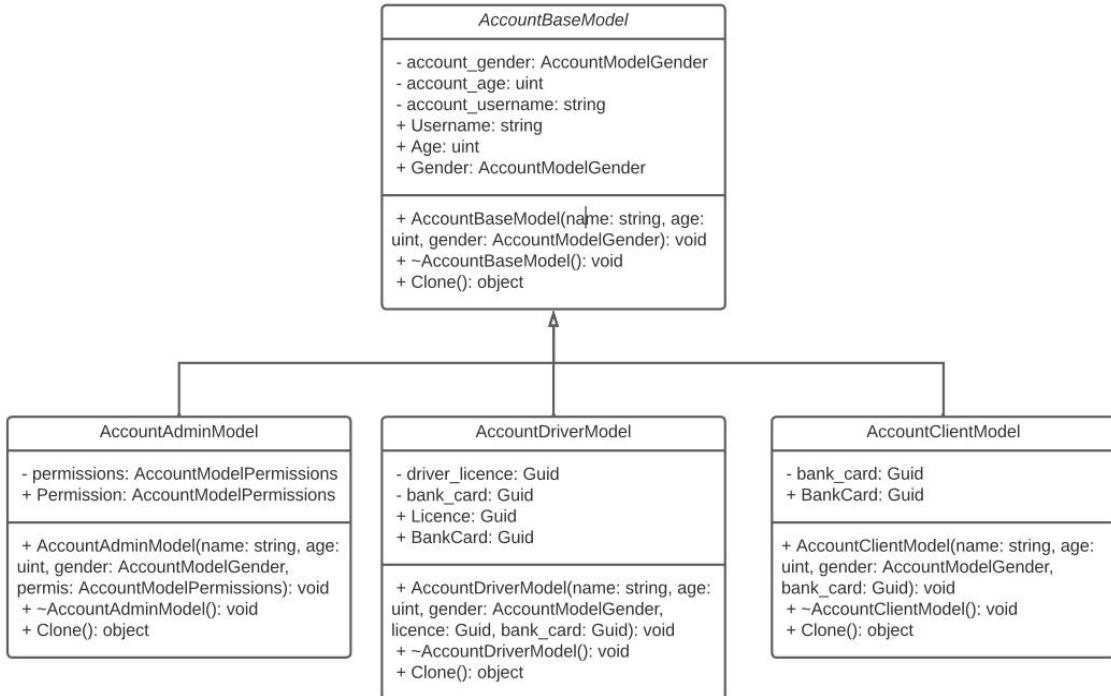


Рисунок 13 — UML диаграмма классов, моделей аккаунтов

Экземпляр менеджера учётных записей службы такси содержит два свойства: «**AccountList**» — возвращает контейнер с выстроенным списком установленных аккаунтов, «**IsInitialized**» — позволяет проверить наличие авторизированной в текущей сессии учётной записи.

Метод доступа «**get_account_scheme**» позволяет получить схему из таблицы БД, с помощью идентификатора; метод «**get_account_status**» — проверяет состояние выбранного аккаунта.

Служба «**BankController**» отвечает за денежные операции сервиса такси. В качестве зависимостей класс использует сервис для соединения с базой данных «**SqlDatabaseManager**». Соединяется с интерфейсом «**IBankController**», устанавливающий некоторый функционал: «**put_money**» положить деньги на выбранный счёт; «**take_money**» снимает деньги со счёта; «**transfer_money**» совершают перевод денежных средств с одного счёта на другой.

Метод «**create_bank_account**» создаёт банковский аккаунт, используя в процессе для этого соединение с соответствующей таблицей в базе данных, на вход принимает идентификатор ячейки, возвращает состояние ошибки. Метод

«load_bank_account» загружается в созданный аккаунт через идентификатор. Метод доступа «get_bank_accounts» возвращает список зарегистрированных банковских счётов.

Банковская служба содержит два свойства: «AccountGuid» возвращает значение идентификатора загруженного банковского профиля, а «AccountMoney» передаёт количество средств от данных на хранение.

3. Реализация системы на общесистемном языке программирования

3.1. Выбор средств программной реализации

Для создания приложения, представляющее агрегатор такси, необходимо определить одну из важнейших составляющих — выбрать платформу разработки, соответствующий API и язык программирования.

Диалог программы с пользователем будет осуществляться при помощи графического пользовательского интерфейса (GUI). В качестве инструмента для создания клиентского оконного приложения будет использоваться набор управляемых библиотек платформы Windows Forms, входящая в состав фреймфорка .NET. Платформа обеспечивает один из самых эффективных способов создания классических приложений с помощью визуального конструктора в Visual Studio IDE, упрощая создание при помощи размещения визуальных элементов управления путём перетаскивания на необходимую форму (представление окна приложения).

Для того чтобы у нас в процессе разработки была возможность работать с технологией Windows Form и использовать C++ в качестве основного языка программирования, необходимо использовать C++ CLI — модификацию ЯП от Microsoft, входящую в состав языков, совместимых с платформой разработки .NET, с возможностью запуска в среде CRL.

В качестве системы для управления базами данных была выбрана платформа Oracle MySql, в составе которой присутствует собранная динамически подгружаемая библиотека классов, методов и интерфейсов для создания связей между СУБД и собранным экземпляром программы.

В роли интегрированной среды разработки устанавливается Visual Studio версии релиза 2022 года от компании Microsoft. Причиной послужила поддержка сборки проектов написанных с использование фреймфорка .NET, интуитивно понятный интерфейс, обширная база инструментов, с помощью которых будет проще проходит отладку промежуточных версий. Также VS имеет инструмент для работы с объектами Windows Form и поддержку средств

сборки и запуска приложений C++ CLI, использующих платформу .NET и среды CLR.

3.2. Алгоритм выполнения программы

Главным классом в выполнении программы является общедоступный тип «Program», который определяет статический публичный метод «Run», который запускает процесс создания модуля сервисов, графического интерфейса и т. д.

В момент запуска функции «Main» вызывается метод «Run», создаётся объект создателя менеджера «ServiceManagerBuilder». После следуют инструкции для загрузки конфигурационных значений дочерних служб из файла с расширение XML. Далее регистрируются разработанные сервисные классы: «SqlDatabaseManager», «DepotManager», «AccountManager», «BankController» и «OrderController».

Когда первоначальная настройка завершается, создаётся экземпляр класса «ServiceManager», отвечающий за распространение сервисов, предоставление к ним доступ из любой части программы. После создания вызывается метод «start_service_handler», который запускает процесс обработки запросов к зарегистрированным службам.

Производится вызов блока настройки отображения GUI при помощи статических методов «EnableVisualStyles» (отображение визуальных стилей системы) и «SetCompatibleTextRenderingDefault» (установка системных совместимых шрифтов для внутреннего текста окон). Далее создаётся объект начальной формы «AuthorizationView», отвечающей за отображение графического представления блока регистрации и авторизации, после чего ссылка на этот экземпляр помещается в качестве входного параметра в статический метод «Run» класса «Application» (ответственный за создание графического интерфейса приложение), который запускает цикл отрисовки окна с подготовленной разметкой.

3.3. Модульная структура программы

Модульное программирование представляет собой принцип организации программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам. Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок.

Система связи водитель пассажир администратор состоит из трёх самостоятельных элементов: клиентское окно, водительское окно и форма администратора.

В зависимости от выбранной конфигурации профиля при регистрации, устанавливается нужное представление в качестве основное окна.

Загруженные сервисы, подгружаются в качестве приватных полей внутри всех классов, наследуемых от базового «Windows::Form». Менеджер служб используется при создании экземпляра формы, в момент вызова конструктора, он предоставляет провайдера на запрашиваемые зависимости.

3.4. Описание диалога с пользователем

При запуске процесса приложения открывается начальное окно регистрации и авторизации. Пользователь может выбрать дальнейшие действия: совершить вход в учётную запись, к которой он имеет доступ, или зарегистрировать новый аккаунт. Чтобы зарегистрировать новый аккаунт, необходимо указать первичные данные о профиле (имя пользователя, возраст, пол), логин и пароль, а после необходимо выбрать какой из типов профилей необходимо создать и в зависимости от выбора установить дополнительные параметры: клиент сервиса такси — номер банковского счёта, водитель — номер лицензии и банковского счёта, администратор — уровень привилегий

(полный доступ, доступ только к складу моделей авто, доступ только к зарегистрированным профилям).

После того как произошёл вход в учётную запись пользователя, загружается соответствующее представление, в зависимости от выбранного типа аккаунта.

Окно клиента службы имеет две вкладки. Вкладка «Заказать машину» позволяет оформить заказ, для этого заполняется предложенная на странице форма: адрес заказа, тип машины (детская, эконом, премиум) и класс транспорта (легковая машина, грузовая машина).

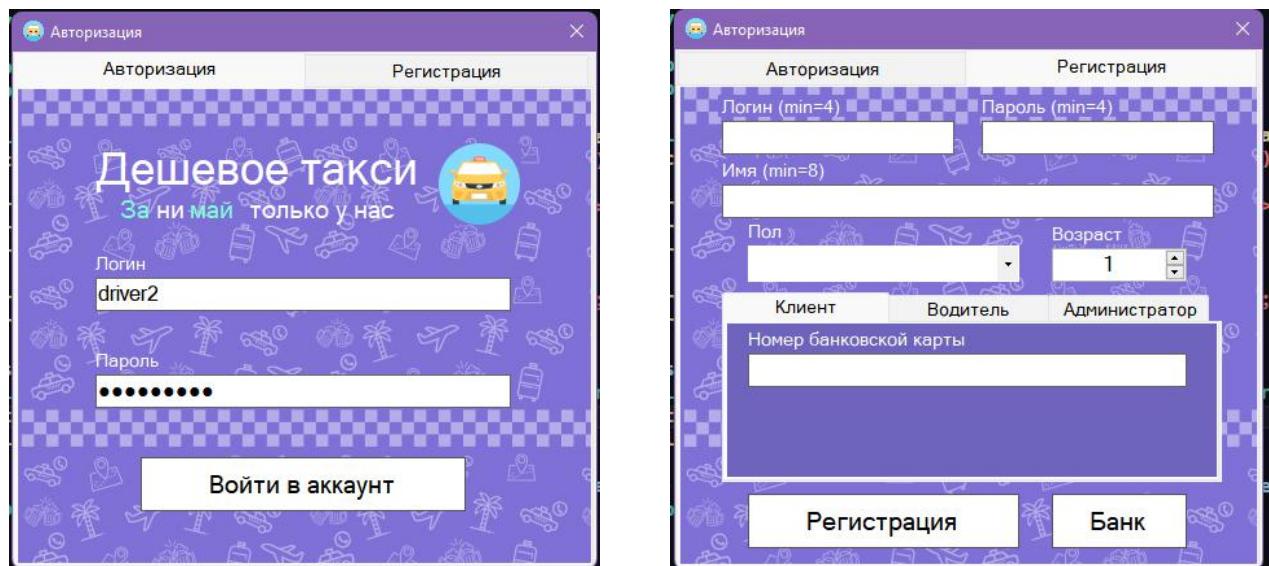


Рисунок 14 — Начальное окно процесса регистрации и авторизации

После того как все необходимые данные установлены, пользователь может нажать кнопку «Заказать машину», после чего откроется процесс ожидания подтверждения со стороны активного водителя, при необходимости есть возможность отменить заказ. Список доступных водителей доступен в данной вкладке и расположен с нижней части формы.

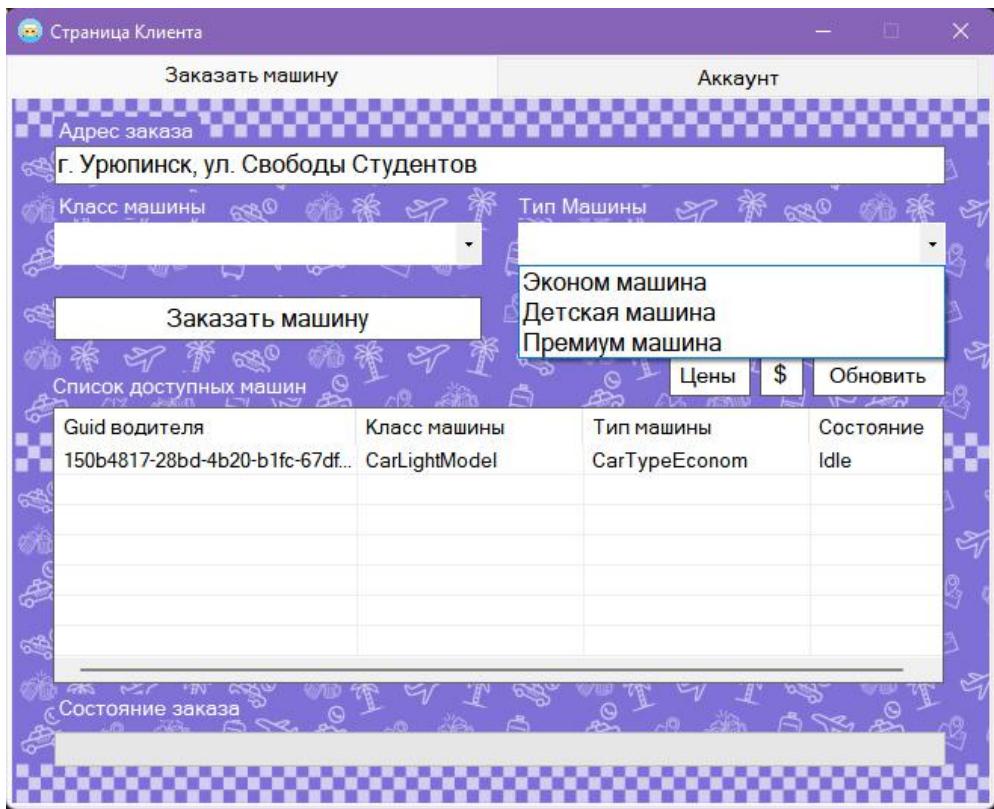


Рисунок 15 — Клиентское окно, вкладка оформления заказов

Используя кнопку «\$» можно проверить доступные средства на банковском счёту. Если нажать кнопку «Цены», то откроется форма с информацией касательно стоимости проезда для разных типов моделей.

Тип машины	Стоимость заказа
CarTypeEconom	200
CarTypePremium	500
CarTypeChild	300

Рисунок 16 — Окно с информацией о установленных ценах проезда

Окно водителя службы такси имеет три вкладки. Вкладка «Поиск» позволяет найти заказ и принять вызов. Список отображает доступные в данный момент зарегистрированные заказы, выделив один из которых и нажав кнопку «Принять заказ» водитель подписывает оформление заказа на свой профиль.

Чуть ниже от списка можно изменить статус перевозчика, выбрав из выпадающего списка необходимое значение и нажав кнопку «Изменить статус». Кнопка «Показать информацию о комплексе» открывает форму с информацией о текущей системе водителя и его арендованной машины. Значок «\$» показывает информацию о текущем банковском балансе.

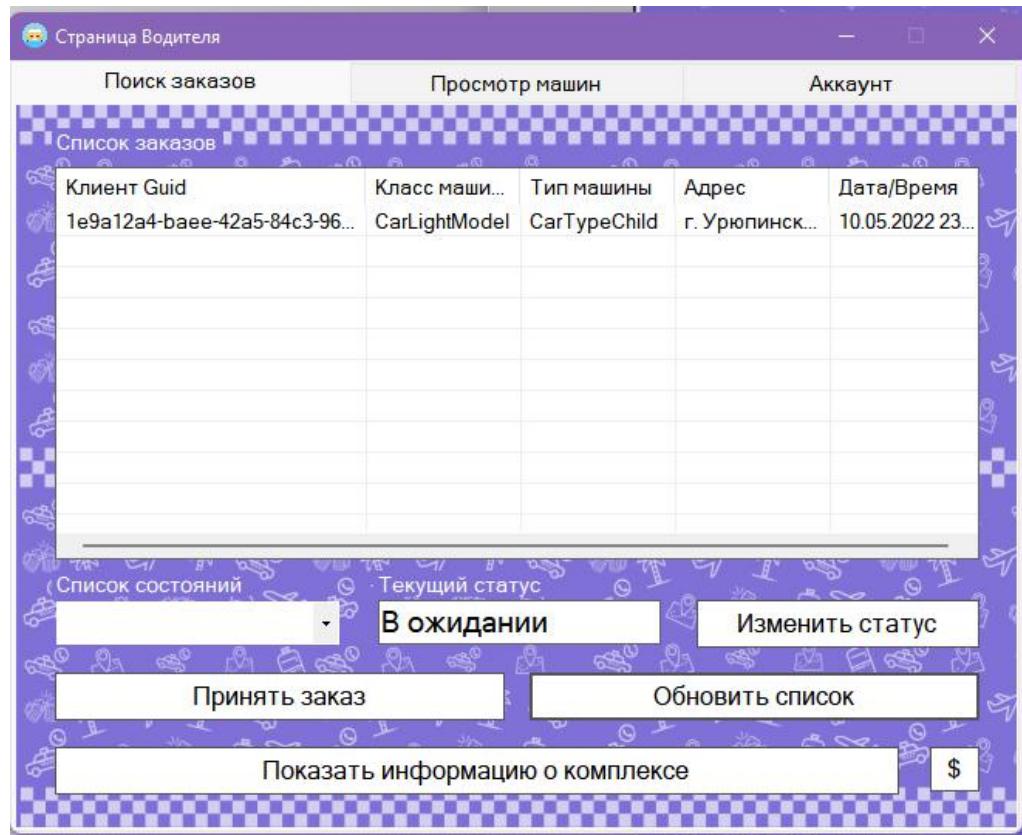


Рисунок 17 — Окно водителя, вкладка поиска заказов

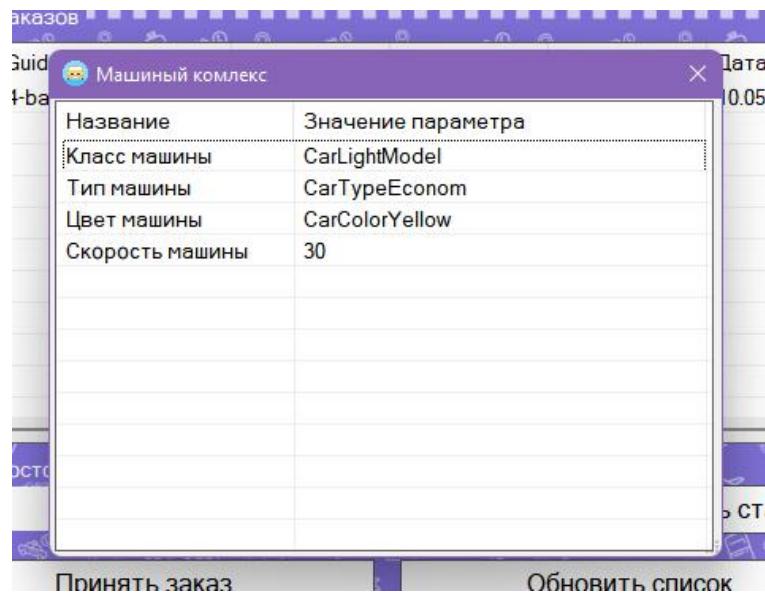


Рисунок 18 — UML диаграмма классов, моделей аккаунтов

Вкладка «Просмотр машин» отвечает за процесс аренды транспорта. Чтобы оформить машину на свой аккаунт, водитель должен выбрать из списка доступных моделей автомобиля один элемент и нажать кнопку «Использовать машину».

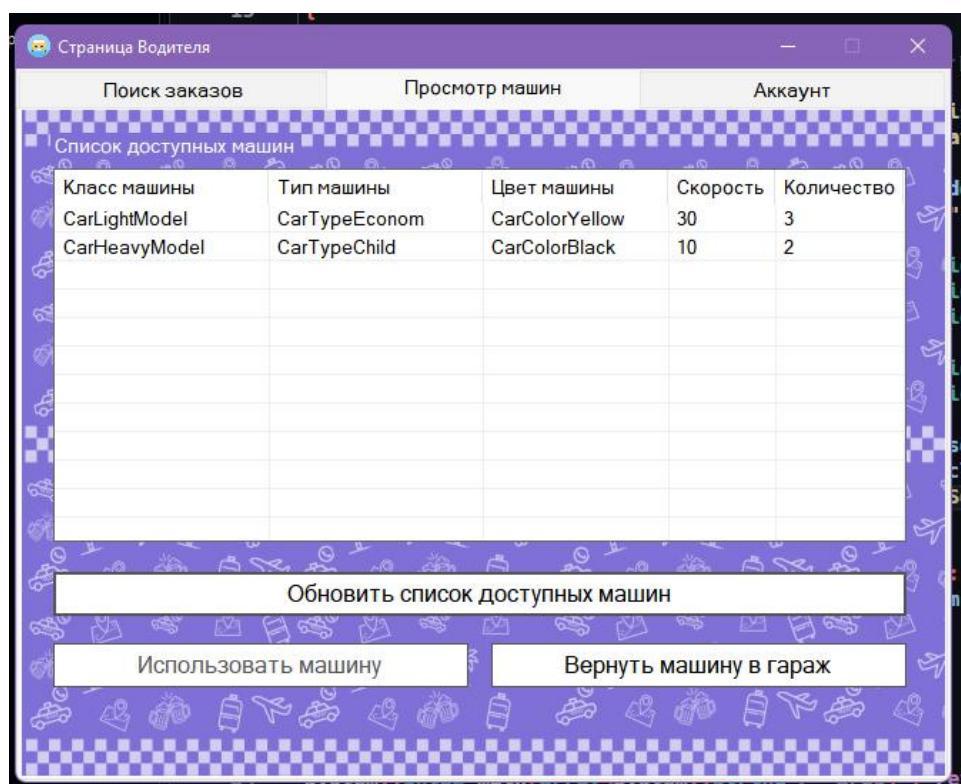


Рисунок 19 — Окно водителя, вкладка аренды транспорта

Вкладка «Аккаунт» демонстрирует данные установленные в учётной записи. Чтобы изменить информацию необходимо установить в checkbox галочку и установить новое значение для выбранного поля и нажать кнопку «Обновить данные».

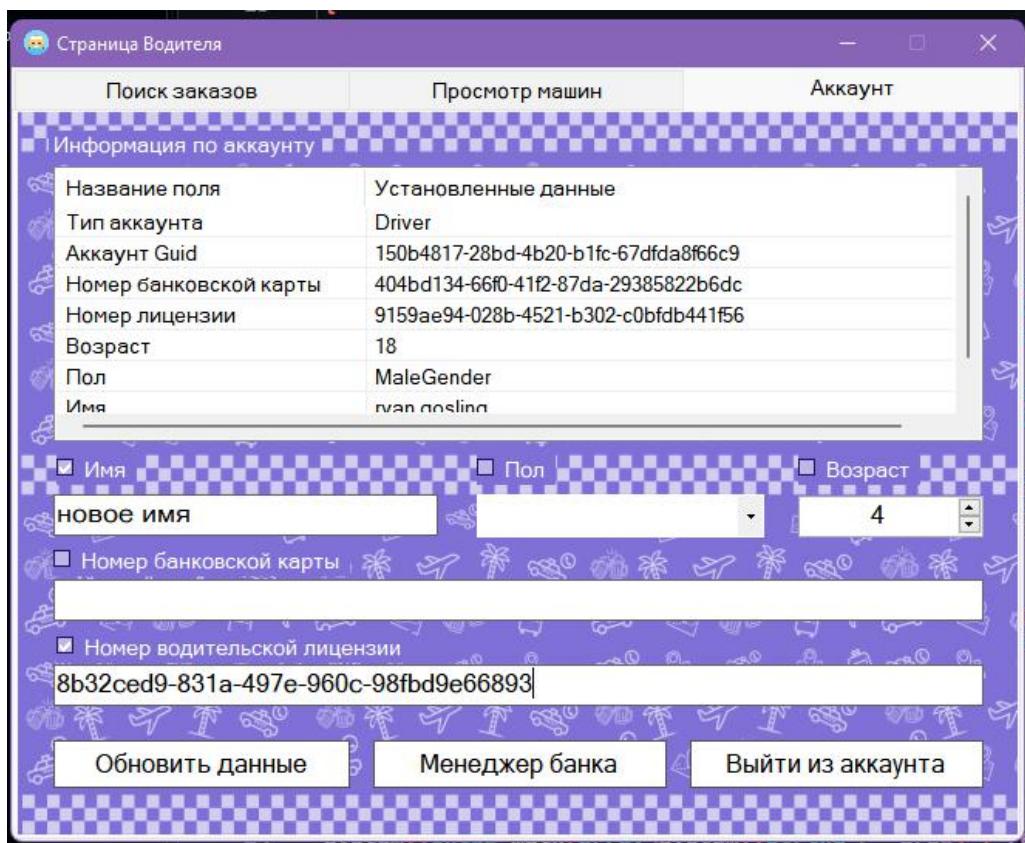


Рисунок 20 — Окно водителя, вкладка с пользовательским профилем

При нажатии на кнопку «Менеджер банка» открывается окно, в котором можно получить информацию о доступных банковских счётах. Для копирования значения идентификатора аккаунта в буфер обмена, необходимо выбрать элемент из списка и нажать кнопку «Копировать счёт».

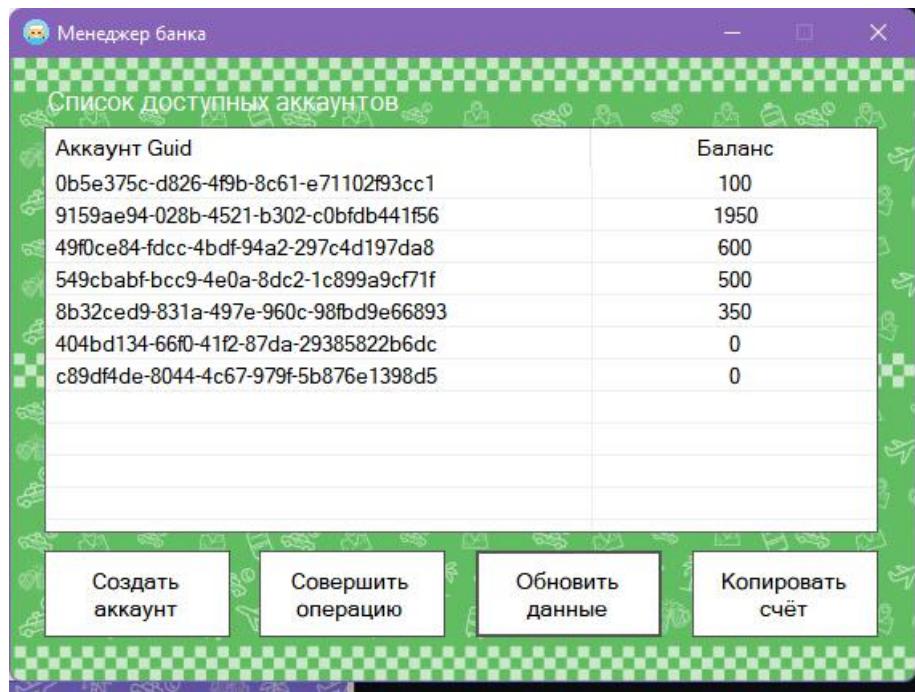


Рисунок 21 — Окно менеджера банка

Если нажать кнопку «Совершить операцию» открывается диалоговое окно, в котором можно совершить банковскую операцию, выбрав из выпадающего списка её тип и ввести сумму в поле, после чего нажать «Совершить операцию».

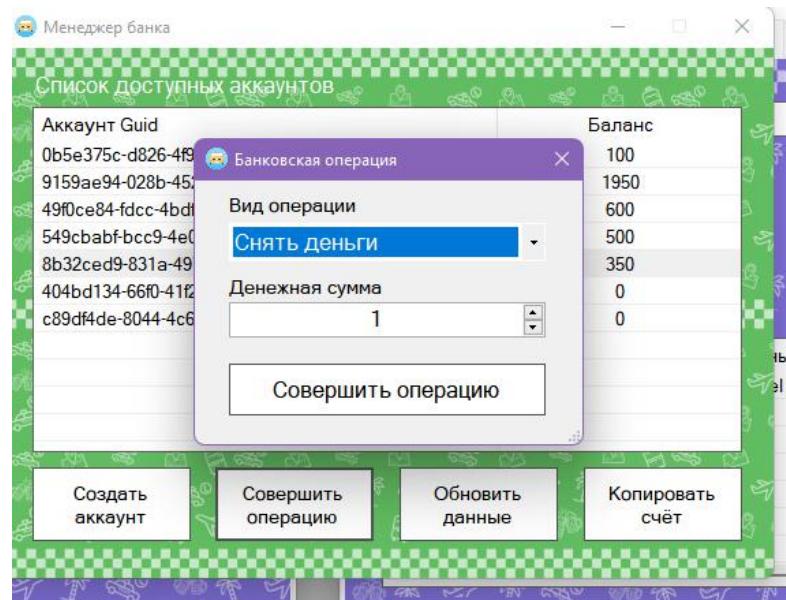


Рисунок 22 — Процесс совершения денежных операций

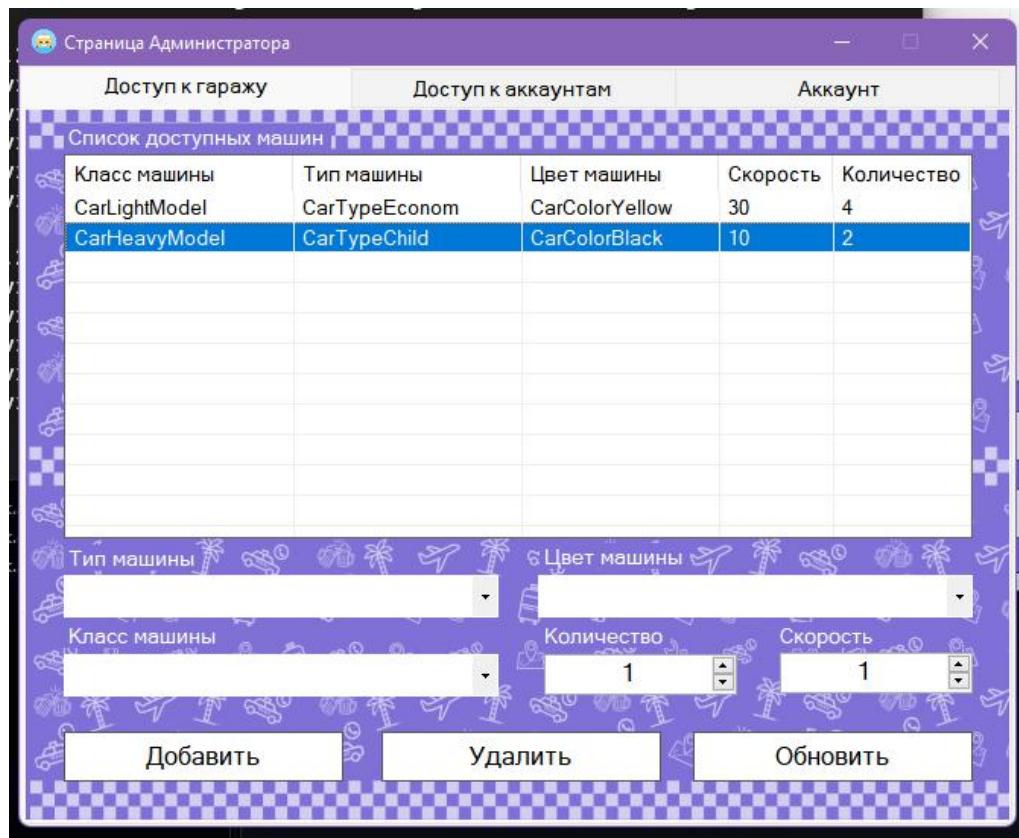


Рисунок 23 — Окно администратора, вкладка менеджмента моделей машин

Окно администратора службы такси содержит 3 вкладки. Вкладка «Доступ к гаражу» предоставляет доступ к настройке коллекций моделей машин таксопарка. Чтобы добавить новый экземпляр транспорта, необходимо установить значения для предоставленных полей (настраивается конфигурация автомобиля) и нажать кнопку «Добавить». Если необходимо удалить модель из таблицы, пользователь должен выделить необходимый элемент из списка и нажать кнопку «Удалить».

Вкладка «Доступ к аккаунтам» позволяет администратору заниматься мониторингом учётных записей пользователей. Если появляется необходимость посмотреть подробную информацию, то необходимо выбрать нужную запись и нажать кнопку «Открыть подробную информацию».

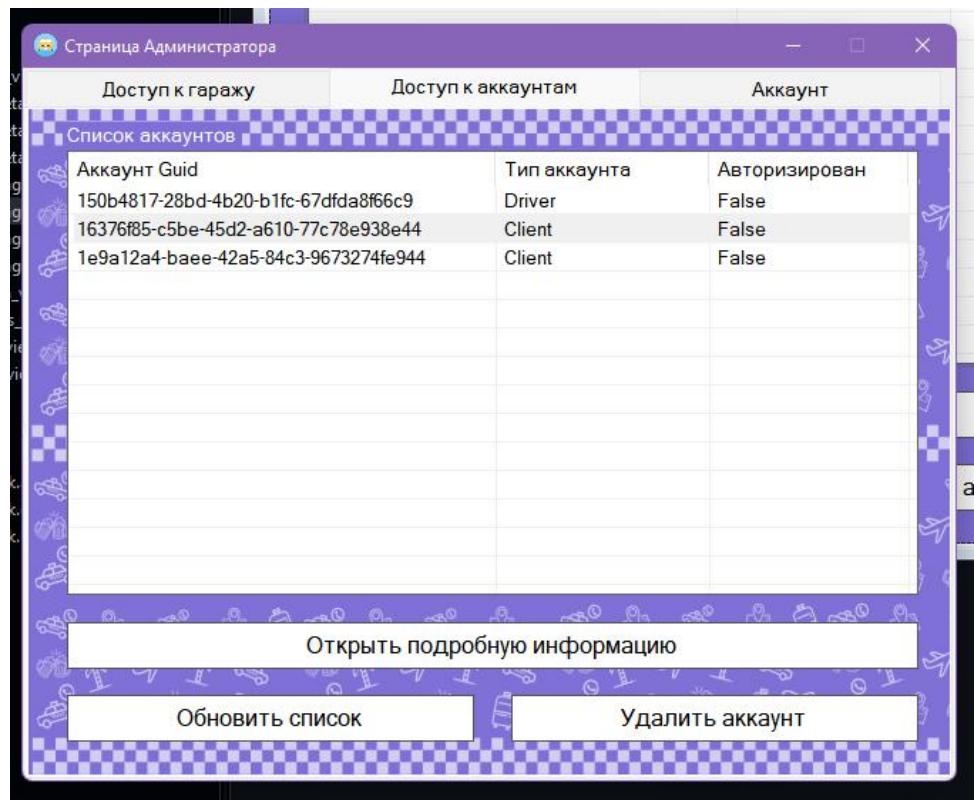


Рисунок 24 — Окно администратора, вкладка менеджмента моделей машин

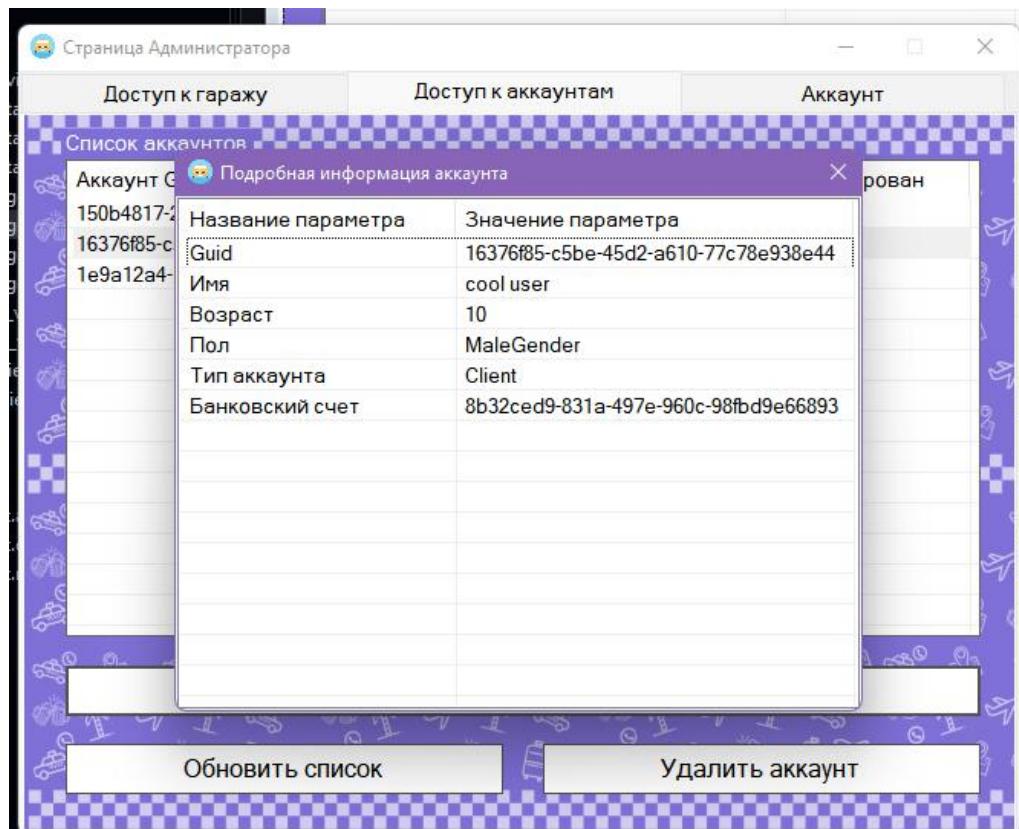


Рисунок 25 — Подробная информация о выбранном профиле

Заключение

В ходе курсового проектирования на языке «C++ CLI» с использованием разработки «Visual Studio 2022» была реализована программа для службы такси.

Разработанная программа предоставляет пользователю систему с удобным интерфейсом для переключения между аккаунтами, для выполнения операций, выбранных пользователем. Организован контроль ошибок при их возникновении.

Все требования к программе выполнены успешно. Реализованы следующие аспекты: иерархия классов, соответствующая и использующая три основных принципа ООП, необходимые для корректной работы функций. Структура, предоставляющая удобное обращение к элементам, добавление новых элементов, удаление, открытие потоков. Использование среды «CLR» и платформы .NET Framework для создания форм пользовательского интерфейса.

Главным составляющей частью проекта является менеджер сервисов, который предоставляет удобный инструмент для менеджмента пользовательских служб, отвечающих за разный задачи внутри решения.

Была разработана схем связанных таблиц с возможностью для хранения необходимой для работоспособности службы информации при помощи системы управления базами данных «Oracle MySql»

В ходе разработки программного решения для выбранной темы были выполнены следующие задачи:

- Предоставление в панели администратора возможностей редактирования наполнения таксопарка: добавление и удаление моделей машин.
- Реализован интуитивно понятного для пользователя графического интерфейса приложения.
- Обеспечение пользователя сервиса возможностью регистрации нового аккаунт службы такси.
- Приложение предоставляет возможность оформление заказов со стороны клиента сервиса.

- Программное решение предоставляет возможность обрабатывать запросы перевозчика на аренду необходимой модели машины из таксопарка.
- Программа обеспечивает всех водителей сервиса возможностью просматривать доступные, свободные клиентские заказы на транспортировку.
- Соблюдение основополагающих принципов объектно-ориентированного программирования при разработке программного решения.
- Приложение выполняет проверка валидности совершения операций оформления заказа, создания аккаунта и разработки модели транспорта.
- Бал создан инструмент для разработки микросервисных модулей с управляющей сущностью в роли менеджера служб.

Список использованной литературы

1. Документация по языку Си++ - <https://docs.microsoft.com/ru-ru/cpp>
2. Норенков И.П. Основы автоматизированного проектирования: Учеб. для вузов. – 3-е изд., перераб. и доп. / И.П. Норенков. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2006. – 448с.
3. Лафоре Роберт Объектно-Ориентированное программирование в C++: Учеб для вузов. – 4-е издание.
4. Курипта О.В. Основы программирования и алгоритмизации: практикум / О.В.Курипта, О.В. Минакова, Д.К. Проскурин; Воронежский ГАСУ. – Воронеж, 2015. – 132 с.
5. Надейкина Л.А. Программирование: учебное пособие. – М.: МГТУ ГА, 2017. – 84 с.
6. Документация по среде CLR - <https://docs.microsoft.com/ru-ru/dotnet/standard/clr>

Приложение

Листинг программы на C++

Файл «/main.cpp»

```
#include "manager/manager.h"
#include "services/services.h"
#include "views/authorization_view/authorization_view.h"

using namespace System;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Xml;

using namespace Services;
using namespace Manager;
using namespace Models;

public ref class Program sealed
{
public:
    static System::Int32 Run(array<System::String^>^ params)
    {
        System::Windows::Forms::Application::EnableVisualStyles();

        System::Windows::Forms::Application::SetCompatibleTextRenderingDefault(false)
    ;

        Manager::ServiceManagerBuilder^ service_builder =
Manager::ManagerBooking::create_builder();
        service_builder-
>service_configuration_load("./services/services_settings.xml");

        service_builder->service_registration<MyServiceProvider^,
SqlDatabaseManager^>();
        service_builder->service_registration<MyServiceProvider^,
BankController^>();
        service_builder->service_registration<MyServiceProvider^,
DepotManager^>();

        service_builder->service_registration<MyServiceProvider^,
OrderController^>();
        service_builder->service_registration<MyServiceProvider^,
AccountManager^>();

        Manager::ServiceManager^ services_manager = service_builder-
>create_manager();
        Manager::ServiceManagerCli^ service_manager_cli = gcnew
Manager::ServiceManagerCli(
            services_manager, System::TimeSpan::FromSeconds(5));
        service_manager_cli->service_manager_run();

        Views::AuthorizationView^ form = gcnew
Views::AuthorizationView(services_manager);
        System::Windows::Forms::Application::Run(form);

        return System::Int32(0);
    }
}
```

```

    }

};

[STAThreadAttribute]
System::Int32 main(array<System::String^>^ args) { return Program::Run(args); }

```

Файл «/manager/manager.h»

```

#pragma once

#ifndef MANAGER_INCLUDE_STUFF
#define MANAGER_INCLUDE_STUFF

#include "service_collection/service_collection.h"
#include "service_manager/service_manager.h"
#include "service_manager_builder/service_manager_builder.h"
#include "service_attribute/service_attribute.h"
#include "service_items/service_items.h"

namespace Manager
{
    public ref class ManagerBooking sealed
    {
        public:
            inline static ServiceManagerBuilder^ create_builder(System::Void)
            { return gcnew Manager::ServiceManagerBuilder(); }

            explicit ManagerBooking(System::Void) { }
            virtual ~ManagerBooking(System::Void) { }

        public ref class ServiceManagerCli sealed
        {
            private: Manager::ServiceManager^ service_manager = nullptr;
            private: System::TimeSpan handling_timer;
            private:
                System::Void service_manager_help(System::Void)
                {
                    Console::WriteLine("[$] - | manager_help | - [Помощник по командам
менеджера]");
                    Console::WriteLine("[$] - | run_handling | - [Запуск обработки
сервисов]");
                    Console::WriteLine("[$] - | service_list | - [Список
зарегистрированных сервисов]");
                    Console::WriteLine("[$] - | manager_exit | - [Список
зарегистрированных сервисов]");
                }
                System::Void service_manager_process(System::Void)
                {
                    Console::BackgroundColor = ConsoleColor::DarkYellow;
                    Console::WriteLine("|=====| SERVICE MANAGER CLIENT
|=====|");
                    Console::BackgroundColor = ConsoleColor::Black;    this-
>service_manager_help();
                    while (true)
                    {
                        Console::Write("[$] > "); System::String^ manager_command =
Console::ReadLine();
                        if (manager_command == System::String::Empty) continue;
                        array<System::String^>^ command_arg_list = manager_command-
>Split(' ');

```

```

        if (command_arg_list[0] == "run_handling")
    {
        Console::WriteLine("[$] - [Для остановки нажмите любую
клавишу]");
        >handling_timer);
        >stop_service_handler();
    }
    else if (command_arg_list[0] == "manager_help") { this-
>service_manager_help(); }
    else if (command_arg_list[0] == "service_list")
    {
        for each (auto item in this->service_manager-
>get_all_services())
            { Console::WriteLine("[$] - [" + item->FullName +
"]"); }
    }
    else if (command_arg_list[0] == "manager_exit")
{ Console::Clear(); break; }
        else Console::WriteLine("[$] - [Команда была не распознана]");
    }
}
public:
    explicit ServiceManagerCli(Manager::ServiceManager^ service_manager,
System::TimeSpan handling_timer)
    { this->service_manager = service_manager; this->handling_timer =
handling_timer; }
    virtual ~ServiceManagerCli(System::Void) { }

    System::Void service_manager_run(System::Void)
    {
        Threading::Thread^ manager_thread = gcnew Threading::Thread(
            gcnew Threading::ThreadStart(this,
&ServiceManagerCli::service_manager_process));
        manager_thread->Start();
    }
};

#endif // !MANAGER_INCLUDE_STUFF

```

Файл «/manager/service_attribute/service_attribute.h»

```

#pragma once

namespace Manager
{
    using namespace System;
    using namespace System::Reflection;

    inline namespace ServiceAttribute
    {
        [System::AttributeUsage(System::AttributeTargets::Class, AllowMultiple =
true)]
        public ref class ServiceRequireAttribute sealed : System::Attribute
        {
            private:    System::Type^ requirement = nullptr;
            public:
                property System::Type^ Requirement
                {

```

```

        public: System::Type^ get(System::Void) { return this-
>requirement; }
        private: System::Void set(System::Type^ value) { this->requirement
= value; }
    };
    public:
        explicit ServiceRequireAttribute(System::Type^ list) :
System::Attribute()
        { this->Requirement = list; }
        virtual ~ServiceRequireAttribute(System::Void) { delete this-
>requirement; }
    };

    [System::AttributeUsage(System::AttributeTargets::Constructor,
AllowMultiple = false)]
    public ref class ServiceConfigurationAttribute sealed : System::Attribute
    {
        private: System::String^ configuration_name = nullptr;
        public:
            property System::String^ Configuration
            {
                public: System::String^ get(System::Void) { return this-
>configuration_name; }
                private: System::Void set(System::String^ value) { this-
>configuration_name = value; }
            }
        public:
            explicit ServiceConfigurationAttribute(System::String^ conf) :
System::Attribute()
            { this->configuration_name = conf; }

            virtual ~ServiceConfigurationAttribute(System::Void) { delete
configuration_name; }
        };
    };
}

```

Файл «/manager/service_collection/service_collection.h»

```

#pragma once
#include "../service_items/service_items.h"

namespace Manager
{
    using namespace System;
    using namespace System::Collections::Generic;
    using namespace System::Collections;

    private ref struct ServiceRecord sealed
    {
        public:           property System::Type^ ServiceProvider;
        public:           property Generic::List<System::Type^>^ ServiceDependencies;
                    property ServiceBase^ ServiceInstance;
        public:
            ServiceRecord(System::Type^ provider, ServiceBase^ service,
List<System::Type^>^ dependencies)
            {
                this->ServiceDependencies = dependencies;   this->ServiceProvider
= provider;           this->ServiceInstance = service;
            }
            ~ServiceRecord(System::Void) { delete ServiceProvider, ServiceInstance,
ServiceDependencies; }
    };
}

```

```

public interface class IServiceCollection
{
    public: property Collections::Generic::List<Manager::ServiceRecord^>^
ServiceList
        { Collections::Generic::List<ServiceRecord^>^ get(System::Void)
abstract; }

    public: property System::Type^ ServiceType[System::String^]
        { System::Type^ get(System::String^ index) abstract; }
};

private ref class ServiceCollection: Manager::IServiceCollection
{
    public: using ServiceListType =
Collections::Generic::List<Manager::ServiceRecord^>;
    private: ServiceListType^ services_collection = nullptr;

    public:     virtual property Manager::ServiceCollection::ServiceListType^
ServiceList
        { public: ServiceListType^ get(System::Void) override { return
services_collection; } }

    public:     virtual property System::Type^ ServiceType[System::String^]
        { public: System::Type ^ get(System::String ^ index) override; }

    public:
        ServiceCollection(System::Void) { services_collection = gcnew
ServiceListType(); }
        virtual ~ServiceCollection(System::Void) { delete services_collection; }

    generic <class TService> where TService : Manager::IServiceBase
        System::Boolean add_service(Manager::ServiceRecord^ service_record);
};

}

```

Файл «/manager/service_collection/service_collection.cpp»

```

#pragma once
#include "service_collection.h"

using namespace Manager;

generic <class TService> where TService : Manager::IServiceBase
    System::Boolean ServiceCollection::add_service(Manager::ServiceRecord^
service_record)
{
    // проверка на наличие добавляемого сервиса в коллекции
    for each (Manager::ServiceRecord^ item in this->services_collection)
    {
        try { TService check = safe_cast<TService>(item->ServiceInstance); return
false; }
        catch (System::InvalidOperationException^ error) { }
    }
    this->services_collection->Add(service_record);
    return true;
}

System::Type^ ServiceCollection::ServiceType::get(System::String^ index)
{
    System::Type^ service_result = nullptr;
    for each (Manager::ServiceRecord ^ item in this->services_collection)
    {
        if (index == item->ServiceInstance->GetType()->Name ||

```

```

        index == item->ServiceInstance->ToString())
        { service_result = item->ServiceInstance->GetType(); }
    }
    return service_result;
}

```

Файл «/manager/service_items/service_items.h»

```

#pragma once

namespace Manager
{
    using namespace System;
    using namespace System::Reflection;
    using namespace System::Collections;

    public enum class ServiceState : System::UInt32 { Idle, Running, Killed };

    public interface class IServiceBase
    {
        public: using ServiceCtorConfiguration = Generic::Dictionary<System::String^,
System::Object^>;
        public: value struct ServiceQuery { Type^ ServiceType; String^ Message;
ServiceState State; };
        public: virtual ServiceQuery^ service_query_handler(System::TimeSpan work_time)
abstract;
    };

    property Manager::ServiceState ServiceState { Manager::ServiceState
get(System::Void) abstract; }
    property System::Guid ServiceGuid { System::Guid get(System::Void)
abstract; }
};

public ref class ServiceBase abstract : Manager::IServiceBase
{
private: Manager::ServiceState service_state;
        System::Guid service_guid = System::Guid::Empty;

protected: IServiceBase::ServiceCtorConfiguration^ configuration = nullptr;
private: static System::UInt32 service_number = 0;
public:
    virtual property Manager::ServiceState ServiceState
    {
        public: Manager::ServiceState get(System::Void) override { return this-
>service_state; }
        public: System::Void set(Manager::ServiceState value) override { this-
>service_state = value; }
    };
    virtual property System::Guid ServiceGuid
    { public: System::Guid get(System::Void) override { return this-
>service_guid; } }
public:
    explicit ServiceBase(IServiceBase::ServiceCtorConfiguration^
configuration)
        : service_guid(System::Guid::.NewGuid()),
service_state(Manager::ServiceState::Idle)
        { this->service_number++; this->configuration = configuration; }

    virtual ~ServiceBase(System::Void) { }
    virtual IServiceBase::ServiceQuery^
service_query_handler(System::TimeSpan work_time) abstract;
};

public interface class IServiceProvider

```

```

{
    public: // переделать get_dependencies под IEnumerable (IEnumerator)
        Collections::Generic::List<System::Type^>^ get_dependencies(System::Void);
    }

    property System::DateTime ServiceDate { System::DateTime get(System::Void) abstract; }
    property ServiceBase^ Service { ServiceBase^ get(System::Void) abstract; }
};

public ref class ServiceProvider abstract : Manager::IServiceProvider
{
private: Manager::ServiceBase^ service_instance = nullptr;
        Generic::List<System::Type^>^ service_dependencies = nullptr;

private: System::DateTime registration_time;
public:
    virtual property Manager::ServiceBase^ Service
    {
        private: System::Void set(Manager::ServiceBase^ value) override { this->service_instance = value; }
        public: Manager::ServiceBase^ get(System::Void) override { return this->service_instance; }
    }

    virtual property System::DateTime ServiceDate
    {
        public: System::DateTime get(System::Void) override { return this->registration_time; }
    }

public:
    explicit ServiceProvider(Manager::ServiceBase^ service,
Generic::List<System::Type^>^ dependencies)
        : registration_time(System::DateTime::Now)
        { this->service_dependencies = dependencies; this->service_instance = service; }

    virtual ~ServiceProvider(System::Void) { delete this->service_dependencies; }
    virtual Generic::List<System::Type^>^ get_dependencies(System::Void) override;
};

}

```

Файл «/manager/service_items/service_items.cpp»

```

#include "service_items.h"

using namespace Manager;

Generic::List<System::Type^>^ ServiceProvider::get_dependencies(System::Void)
{
    Generic::List<System::Type^>^ result = gcnew Generic::List<System::Type^>(
        this->service_dependencies->Count);

    for each (System::Type^ item in this->service_dependencies) result->Add(item);
    return result;
}

```

Файл «/manager/service_manager/service_manager.h»

```

#pragma once
#include "../service_items/service_items.h"
#include "../service_collection/service_collection.h"

```

```

namespace Manager
{
    using namespace System::Reflection;
    using namespace System::Collections;
    using namespace System::Collections::Generic;
    using namespace System::Threading;
    using namespace System;

    public interface class IServiceManager
    {
        public: generic <class TService> where TService : Manager::IServiceBase
            Manager::ServiceProvider^ get_service(System::Void);

            property System::UInt16 ServiceCount { System::UInt16 get(System::Void)
abstract; } virtual Generic::List<System::Type^>^ get_all_services(System::Void)
abstract;

            virtual Tasks::Task^ start_service_handler(System::TimeSpan tick_timer)
abstract; virtual System::Boolean stop_service_handler(System::Void) abstract;
};

        public ref class ServiceManager sealed : Manager::IServiceManager
        {
            public:     delegate IServiceBase::ServiceQuery^
ServiceDelegateHandler(System::TimeSpan);

            private:     Manager::ServiceCollection^ service_collection = nullptr;
            ServiceManager::ServiceDelegateHandler^ services_event_handler = nullptr;

            System::Threading::CancellationTokenSource^ cancel_source = nullptr;
            System::Threading:: CancellationToken cancellation_token;
            System::TimeSpan^ service_handling_timer = nullptr;

            System::Void service_handling_process(System::Void);
            System::Boolean service_handling_connection(System::Void);
public:
            virtual property System::UInt16 ServiceCount
            { public: System::UInt16 get(System::Void) { return service_collection-
>ServiceList->Count; } }

            event ServiceManager::ServiceDelegateHandler^ ServiceEventHandler
            {
                public: System::Void add(ServiceDelegateHandler^ value)
{ services_event_handler += value; }
                protected: System::Void remove(ServiceDelegateHandler^ value)
{ services_event_handler -= value; }
            };
            public:
                explicit ServiceManager(Manager::ServiceCollection^ collection)
                {
                    this->service_collection = collection;
                    if (service_handling_connection() != true) throw gcnew
System::Exception("Service handling error");
                }
                virtual ~ServiceManager(System::Void)
                { delete this->service_collection, this->services_event_handler; }

            virtual Generic::List<System::Type^>^ get_all_services(System::Void)
override;

            generic <class TService> where TService : Manager::IServiceBase
                virtual Manager::ServiceProvider^ get_service(System::Void)
override;

```

```

        virtual Tasks::Task^ start_service_handler(System::TimeSpan tick_timer);
        virtual System::Boolean stop_service_handler(System::Void);
    };
}

```

Файл «/manager/service_manager/service_manager.cpp»

```

#pragma once
#include "service_manager.h"

using namespace Manager;

generic <class TService> where TService : IServiceBase ServiceProvider^
ServiceManager::get_service(System::Void)
{
    System::Type^ service_require = TService::typeid;
    Manager::ServiceProvider^ provider = nullptr;

    for each (Manager::ServiceRecord^ srv in this->service_collection->ServiceList)
    {
        if (srv->ServiceInstance->GetType() == service_require)
        {
            array<System::Object^>^ params = gcnew array<System::Object^>
{ srv->ServiceInstance, srv->ServiceDependencies };

            try { provider =
safe_cast<Manager::ServiceProvider^>(Activator::CreateInstance(srv->ServiceProvider,
params)); }
            catch (System::MissingMethodException^ error)
{ Console::WriteLine(error->Message); break; }
        }
    }
    return provider;
}

Generic::List<System::Type^>^ ServiceManager::get_all_services(System::Void)
{
    Generic::List<System::Type^>^ result_collection = gcnew
Generic::List<System::Type^>();

    for each (Manager::ServiceRecord ^ item in this->service_collection->ServiceList)
    { result_collection->Add(item->ServiceInstance->GetType()); }

    return result_collection;
}

System::Boolean ServiceManager::service_handling_connection(System::Void)
{
    for each (Manager::ServiceRecord^ service_item in this->service_collection-
>ServiceList)
    {
        try { this->ServiceEventHandler += gcnew
ServiceManager::ServiceDelegateHandler(
            service_item->ServiceInstance,
&IServiceBase::service_query_handler); }
        catch (System::Exception^ error) { Console::WriteLine(error->Message);
return false; }
    }
    return true;
}

System::Void console_service_state(IServiceBase::ServiceQuery^ query)

```

```

{
    switch (query->State)
    {
        case ServiceState::Idle;; Console::BackgroundColor = ConsoleColor::Gray; break;
        case ServiceState::Killed;; Console::BackgroundColor = ConsoleColor::Magenta;
    break;
        case ServiceState::Running;; Console::BackgroundColor = ConsoleColor::DarkGreen;
    break;
    }

    Console::Write("[ " + query->State.ToString() + " ]");
    Console::BackgroundColor = ConsoleColor::Black;

    Console::WriteLine(" [ " + query->ServiceType->ToString() + " ] : " + query-
>Message);
}

System::Void ServiceManager::service_handling_process(System::Void)
{
    System::UInt32 tick_millisecond = this->service_handling_timer-
>TotalMilliseconds;
    for(System::UInt64 iteration_tick = 0; true; iteration_tick++)
    {
        Console::WriteLine("===== [ " + System::DateTime::Now + " ] ###");
Сообщение от Менеджера сервисов =====;
        for each (auto handler in this->services_event_handler-
>GetInvocationList())
        {
            ServiceManager::ServiceDelegateHandler^ handler_delegate = nullptr;
            try { handler_delegate =
safe_cast<ServiceManager::ServiceDelegateHandler^>(handler); }
            catch (System::Exception^ error)
            { Console::WriteLine(error->Message); this->stop_service_handler();
break; }

            IServiceProvider::ServiceQuery^ query = handler_delegate-
>Invoke(System::TimeSpan::FromSeconds(iteration_tick));
            console_service_state(query);
        }
        Console::WriteLine();

        if (this->cancellation_token.IsCancellationRequested) return;
        Thread::Sleep(tick_millisecond);
    }
}

Tasks::Task^ ServiceManager::start_service_handler(System::TimeSpan tick_timer)
{
    this->cancel_source = gcnew CancellationTokenSource();
    this->cancellation_token = this->cancel_source->Token;
    this->service_handling_timer = tick_timer;

    Tasks::Task^ service_handling_task = gcnew Tasks::Task(
        gcnew Action(this, &ServiceManager::service_handling_process), this-
>cancel_source->Token);
    service_handling_task->Start();

    if (service_handling_task != nullptr)
    {
        for each (auto service_item in this->service_collection->ServiceList)
            service_item->ServiceInstance->ServiceState =
Manager::ServiceState::Running;
    }

    return service_handling_task;
}

```

```

}

System::Boolean ServiceManager::stop_service_handler(System::Void)
{
    try { this->cancel_source->Cancel(); }
    catch (System::Exception^ error) { Console::WriteLine(error->Message); return
false; }

    for each (auto service_item in this->service_collection->ServiceList)
        service_item->ServiceInstance->ServiceState = Manager::ServiceState::Idle;

    this->cancel_source = nullptr;
    return true;
}

```

Файл «/manager/service_manager_builder/service_manager_builder.h»

```

#pragma once
#include "../service_items/service_items.h"
#include "../service_collection/service_collection.h"
#include "../service_manager/service_manager.h"
#include "../service_attribute/service_attribute.h"

namespace Manager
{
    using namespace System;
    using namespace System::Collections::Generic;
    using namespace System::Collections;

    public ref class ServiceManagerBuilderException sealed : System::Exception
    {
    private:    System::Type^ exception_attach_from = nullptr;
    public:
        property System::Type^ AttachFrom
        { public: System::Type^ get(System::Void) { return this-
>exception_attach_from; } }

        virtual property System::String^ Message
        {
            public: System::String^ get(System::Void) override
                { return "From " + exception_attach_from->ToString() + ": " +
Exception::Message; }
        }
    public:
        ServiceManagerBuilderException(System::Type^ from, System::String^
message) : System::Exception(message)
        { this->exception_attach_from = from; }

        virtual ~ServiceManagerBuilderException(System::Void) { delete
exception_attach_from; }
    };

    public interface class IServiceManagerBuilder
    {
    public: value struct ServiceCtorConfigurationItem { System::String^ Key;
        IServiceBase::ServiceCtorConfiguration^ Configuration; };

    public:     Manager::ServiceManager^ create_manager(System::Void);
    public: generic<class TProvider, class TService> where TService:
Manager::IServiceBase
        System::Boolean service_registration(System::Void);
    };
}

```

```

        System::Boolean service_configuration_load(ServiceCtorConfigurationItem^
configuration_item);
        System::Boolean service_configuration_load(System::String^
configuration_filename);
    };

    public ref class ServiceManagerBuilder sealed : Manager::IServiceManagerBuilder
    {
    public:     using ServiceRequire = ServiceAttribute::ServiceRequireAttribute;
                value struct ServiceConstructor { ConstructorInfo^ Ctor; String^
Configuration; };

        Dictionary<String^, IServiceBase::ServiceCtorConfiguration^>^
configurations = nullptr;
        private:     System::Boolean manager_is_created;
        private:     System::UInt16 registration_count;
        private:     Manager::ServiceCollection^ collection = nullptr;
        public:
            virtual property System::UInt16 ServiceCount
            {
                public: System::UInt16 get(System::Void) override { return
registration_count; }
                private: System::Void set(System::UInt16 value) override
{ registration_count = value; }
            }
            private: generic <class TService> where TService: Manager::IServiceBase
                System::Tuple<TService, List<System::Type^>^>^
dependency_injection(System::Void);

            ServiceManagerBuilder::ServiceConstructor^
get_service_configuration(System::Type^ obj);
            System::Void service_configuration_setup(System::Void);
        public:
            explicit ServiceManagerBuilder(System::Void) : manager_is_created(false),
registration_count(0)
            { this->collection = gcnew ServiceCollection(); this-
>service_configuration_setup(); }
            virtual ~ServiceManagerBuilder(System::Void) { }

            virtual Manager::ServiceManager^ create_manager(System::Void) override;

            generic<class TProvider, class TService> where TService:
Manager::IServiceBase
                virtual System::Boolean service_registration(System::Void) override;

                virtual System::Boolean service_configuration_load(System::String^
configuration_filename);
                virtual System::Boolean service_configuration_load(
                    IServiceManagerBuilder::ServiceCtorConfigurationItem^
configuration_item) override;
            };
    };
}

```

Файл «/manager/service_manager_builder/service_manager_builder.cpp»

```

#pragma once
#include "service_manager_builder.h"

using namespace Manager;
using namespace ServiceAttribute;

ServiceManagerBuilder::ServiceConstructor^
ServiceManagerBuilder::get_service_configuration(System::Type^ obj)
{

```

```

ServiceManagerBuilder::ServiceConstructor^ selected_configuration = nullptr;
array<ConstructorInfo^>^ ctor_list = obj->GetConstructors();

for(System::UInt32 index = 0; index < ctor_list->Length; index++)
{
    Reflection::ConstructorInfo^ selected_ctor = ctor_list[index];
    ServiceConfigurationAttribute^ ctor_attribute = nullptr;

    try { ctor_attribute = safe_cast<ServiceConfigurationAttribute^>(
        Attribute::GetCustomAttribute(selected_ctor,
        ServiceConfigurationAttribute::typeid)); }
    catch (System::Exception^ error) { Console::WriteLine(error->Message);
continue; }

    if (ctor_attribute != nullptr)
    {
        selected_configuration = gcnew
ServiceManagerBuilder::ServiceConstructor();
        selected_configuration->Configuration = ctor_attribute-
>Configuration;
        selected_configuration->Ctor = selected_ctor;      break;
    }
}
return selected_configuration;
}

generic <class TService> where TService: IServiceProvider
System::Tuple<TService, List<System::Type^>^>^
ServiceManagerBuilder::dependency_injection(System::Void)
{
    array<ServiceRequirement^>^ service_requirement =
safe_cast<array<ServiceRequirement^>^>(
    Attribute::GetCustomAttributes(TService::typeid, ServiceRequirement::typeid));

    ServiceConstructor^ service_ctor = this-
>get_service_configuration(TService::typeid);
    if (service_ctor == nullptr) { return nullptr; }
    array<ParameterInfo^>^ ctor_parameter = service_ctor->Ctor->GetParameters();

    if (ctor_parameter->Length <= 0 || ctor_parameter[0]->ParameterType !=
IServiceBase::ServiceCtorConfiguration::typeid
        || !this->configurations->ContainsKey(service_ctor->Configuration))
    { Console::WriteLine(TService::typeid->FullName + " Parameter Error"); return
nullptr; }

    List<System::Type^>^ service_dependencies = gcnew List<System::Type^>(); // для
сервис провайдера
    array<Object^>^ service_includes = gcnew array<Object^>(service_requirement-
>Length + 1); // для создания сервиса (в конструктор)

    service_includes[0] = this->configurations[service_ctor->Configuration];
    System::UInt16 dependencies_collected(0);

    if (service_requirement != nullptr && service_requirement->Length > 0)
    {
        for (System::UInt16 index = 0; index < service_requirement->Length;
index++)
        {
            for each (auto service_in_collection in this->collection-
>ServiceList)
            {
                if (service_requirement[index]->Requirement ==
service_in_collection->ServiceInstance->GetType())
                    { service_includes[dependencies_collected++ + 1] =
service_in_collection->ServiceInstance; }
            }
        }
    }
}

```

```

        }
        service_dependencies->Add(service_requirement[index]->Requirement);
    }
}
Tuple<TService, List<System::Type^>>^ service_instance = nullptr;
// если не удалось найти некоторые зависимости в контейнере
if (dependencies_collected != service_requirement->Length)
{
    Console::WriteLine("dependencies_collected != service_requirement->Length");
    return service_instance;
}
try {
    TService service = safe_cast<TService>(service_ctor->Ctor->Invoke(service_includes));
    service_instance = Tuple::Create(service, service_dependencies);
}
catch (System::Exception^ error) { Console::WriteLine(error->Message + "\nFrom \"ServiceManagerBuilder\""); }

return service_instance;
}

generic<class TProvider, class TService> where TService: Manager::IServiceBase
System::Boolean ServiceManagerBuilder::service_registration(System::Void)
{
    System::Type^ provider_interface = TProvider::typeid->GetInterface("IServiceProvider");
    System::Type^ service_interface = TService::typeid->GetInterface("IServiceBase");

    if (provider_interface != nullptr && service_interface != nullptr)
    {
        System::Tuple<TService, Generic::List<System::Type^>>^ service_instance
= nullptr;
        try { service_instance = dependency_injection<TService>(); }
        catch (System::Exception^ error) { Console::WriteLine(error->Message); }
        return false; }

        if (service_instance != nullptr)
        {
            return this->collection->add_service<TService>(gcnew
Manager::ServiceRecord(TProvider::typeid,
(Manager::ServiceBase^)(service_instance->Item1),
service_instance->Item2));
        }
    }
    return false;
}

System::Boolean ServiceManagerBuilder::service_configuration_load(
    IServiceManagerBuilder::ServiceCtorConfigurationItem^ configuration_item)
{
    if (configuration_item == nullptr || configuration_item->Configuration ==
nullptr) return false;
    if (this->configurations->ContainsKey(configuration_item->Key))
    {
        for each (Generic::KeyValuePair<String^, Object^> item in
configuration_item->Configuration)
        {
            try { this->configurations[configuration_item->Key]->Add(item.Key,
item.Value); }
            catch (System::Exception^ error) { Console::WriteLine(error-
>Message); return false; }
        }
    }
}

```

```

        else { this->configurations->Add(configuration_item->Key, configuration_item-
>Configuration); }
        return true;
    }

System::Boolean ServiceManagerBuilder::service_configuration_load(System::String^
configuration_filename)
{
    Xml::XmlDocument^ parameters_document = gcnew Xml::XmlDocument();
    parameters_document->Load(configuration_filename);

    Xml::XmlElement^ doc_root = parameters_document->DocumentElement;
    if (doc_root == nullptr) return false;

    for each (Xml::XmlElement^ doc_element in doc_root)
    {
        XmlNode^ node_attr = doc_element->Attributes-
>GetNamedItem("service_name");
        if (node_attr == nullptr) continue;

        auto configuration_item = gcnew
IServiceManagerBuilder::ServiceCtorConfigurationItem();
        configuration_item->Key = node_attr->Value;
        configuration_item->Configuration = gcnew
IServiceBase::ServiceCtorConfiguration();

        for each (XmlNode ^ childnode in doc_element->ChildNodes)
        { configuration_item->Configuration->Add(childnode->Name, childnode-
>InnerText); }
        this->service_configuration_load(configuration_item);
    }
    return true;
}

System::Void ServiceManagerBuilder::service_configuration_setup(System::Void)
{
    this->configurations = gcnew Dictionary<String^,
IServiceBase::ServiceCtorConfiguration^>();
    this->configurations->Add("none", nullptr);
}

Manager::ServiceManager^ ServiceManagerBuilder::create_manager(System::Void)
{
    if (manager_is_created != false)
    { throw gcnew ServiceManagerBuilderException(ServiceManagerBuilder::typeid,
"Менеджер уже создан"); }

    manager_is_created = !manager_is_created;
    return gcnew ServiceManager(this->collection);
}

```

Файл «/models/cars_model/cars_model.h»

```

#pragma once
#include "../account_model/account_model.h"

namespace Models
{
    public enum class CarModelTypes : System::UInt16 { CarTypeEconom, CarTypePremium,
CarTypeChild };
    public enum class CarModelColor : System::UInt16 { CarColorBlack, CarColorWhite,
CarColorRed, CarColorYellow };

    public ref class CarBaseModel abstract : System::ICloneable

```

```

{
private:     Models::CarModelTypes model_type;
private:     Models::CarModelColor model_color;
private:     System::UInt32 model_speed;
public:
    property Models::CarModelTypes CarType
    { public: CarModelTypes get(System::Void) { return this->model_type; } }

    property Models::CarModelColor CarColor
    { public: CarModelColor get(System::Void) { return this->model_color; } }

    property System::UInt32 CarSpeed
    { public: System::UInt32 get(System::Void) { return this->model_speed; } }
public:
    CarBaseModel(Models::CarModelTypes car_type, Models::CarModelColor color,
System::UInt32 speed)
        : model_type(car_type), model_color(color), model_speed(speed) { }
    virtual ~CarBaseModel(System::Void) { }

    virtual System::Object^ Clone(System::Void) abstract;
    virtual System::String^ car_drive(System::Void) abstract;
};

public ref class CarLightModel sealed : CarBaseModel
{
public:
    CarLightModel(Models::CarModelTypes car_type, Models::CarModelColor color,
System::UInt32 speed)
        : CarBaseModel(car_type, color, speed) { }
    virtual ~CarLightModel(System::Void) { }

    virtual System::Object^ Clone(System::Void) override { return gcnew
CarLightModel(CarType, CarColor, CarSpeed); }
    virtual System::String^ car_drive(System::Void) override { return "Идем
на легковой машине"; }
};

public ref class CarHeavyModel sealed : CarBaseModel
{
public:
    CarHeavyModel(Models::CarModelTypes car_type, Models::CarModelColor color,
System::UInt32 speed)
        : CarBaseModel(car_type, color, speed) { }
    virtual ~CarHeavyModel(System::Void) { }

    virtual System::Object^ Clone(System::Void) override { return gcnew
CarHeavyModel(CarType, CarColor, CarSpeed); }
    virtual System::String^ car_drive(System::Void) override { return "Идем
на грузовой машине"; }
};

}

```

Файл «/models/account_model/account_model.h»

```

#pragma once

namespace Models
{
    using namespace System;
    using namespace System::Collections::Generic;

    public enum class AccountModelGender : System::UInt16 { MaleGender,
FemaleGender };

```

```

public enum class AccountModelPermissions : System::UInt16
{ FullPermission = 0b00000011, CarGarage = 0b00000001, AccountList = 0b00000010 };

public ref class AccountBaseModel abstract : System::ICloneable
{
private:    Models::AccountModelGender account_gender;
private:    System::UInt32 account_age;

private:    System::String^ account_username = nullptr;
public:
    property System::String^ Username
    {
        public: System::String^ get(System::Void) { return this->account_username; }
        private: System::Void set(System::String^ value) { this->account_username = value; }
    }

    property AccountModelGender Gender { AccountModelGender get(System::Void) { return account_gender; } }
    property System::UInt32 Age { System::UInt32 get(System::Void) { return this->account_age; } }

public:
    AccountBaseModel(System::String^ name, System::UInt32 age,
Models::AccountModelGender gender)
        : account_age(age), account_gender(gender) { this->account_username = name; }
    virtual ~AccountBaseModel(System::Void) { delete account_username; }

    virtual System::Object^ Clone(System::Void) abstract;
};

public ref class AccountAdminModel sealed : Models::AccountBaseModel
{
    Models::AccountModelPermissions permissions;
public:
    property Models::AccountModelPermissions Permissions
    { public: AccountModelPermissions get(System::Void) { return this->permissions; } }
    public:
        AccountAdminModel(String^ name, UInt32 age, AccountModelGender gender,
AccountModelPermissions permissions)
            : Models::AccountBaseModel(name, age, gender),
permissions(permissions) { }
        virtual ~AccountAdminModel(System::Void)
{ AccountBaseModel::~AccountBaseModel(); }

    virtual System::Object^ Clone(System::Void) override
    { return gcnew AccountAdminModel((System::String^)Username->Clone(), Age,
Gender, permissions); }
};

public ref class AccountDriverModel sealed : Models::AccountBaseModel
{
    // лицензия, права (машины),
    System::Guid driver_licence = System::Guid::Empty;
    System::Guid bank_card = System::Guid::Empty;
public:
    property System::Guid Licence { System::Guid get(System::Void) { return this->driver_licence; } }
    property System::Guid BankCard { System::Guid get(System::Void) { return this->bank_card; } }
};

```

```

public:
    AccountDriverModel(String^ name, UInt32 age, AccountModelGender gender,
Guid licence, Guid bank_card)
        : Models::AccountBaseModel(name, age, gender)
    {
        this->bank_card = bank_card;
        this->driver_licence = licence;
    }
    virtual ~AccountDriverModel(System::Void)
{ AccountBaseModel::~AccountBaseModel(); }

    virtual System::Object^ Clone(System::Void) override
    { return gcnew AccountDriverModel((String^)Username->Clone(), Age, Gender,
driver_licence, bank_card); }
};

public ref class AccountClientModel sealed : Models::AccountBaseModel
{
    System::Guid bank_card = System::Guid::Empty;
    public: property System::Guid BankCard { System::Guid get(System::Void) { return
this->bank_card; } }
    public:
        AccountClientModel(System::String^ name, System::UInt32 age,
AccountModelGender gender,
                           System::Guid bank_card) : AccountBaseModel(name, age, gender),
bank_card(bank_card) { }
        virtual ~AccountClientModel(System::Void) { delete bank_card;
AccountBaseModel::~AccountBaseModel(); }

    virtual System::Object^ Clone(System::Void) override
    { return gcnew AccountClientModel((String^)Username->Clone(), Age, Gender,
bank_card); }
};
}

```

Файл «/services/account_manager/account_manager.h»

```

#pragma once
#include "../database_provider/database_provider.h"
#include "../../manager/manager.h"
#include "account_manager_scheme.h"
#include "account_manager_token.h"

namespace Services
{
    using namespace System;
    using namespace Manager;
    using namespace System::Collections::Generic;

    public interface class IAccountManager
    {
    public:
        generic <class TAccountModel> where TAccountModel :
Models::AccountBaseModel
            System::Boolean registration_account(System::String^ login,
System::String^ password,
                           TAccountModel account_model);
            System::Boolean authorization_account(System::String^ login,
System::String^ password);

            System::Boolean delete_account(System::Guid guid);
            System::Boolean sign_out_account(System::Void);
    };
}

```

```

[Manager::ServiceAttribute::ServiceRequireAttribute(Services::SqlDatabaseManager:
:typeid)]
    public ref class AccountManager sealed : Manager::ServiceBase,
Services::IAccountManager
{
    public:     value struct AccountInfo { System::Boolean State;
        System::Guid Guid; System::String^ Name;
AccountManagerToken::AccountManagerType Type; };

    private:    Services::SqlDatabaseManager^ service_sql_manager = nullptr;
                System::Boolean service_disposed = false;

    private:    System::Boolean account_initialized;
    private:    Services::AccountManagerToken account_token;
    public:
        property System::Boolean IsInitialized { System::Boolean get(System::Void)
{ return this->account_initialized; } }
        property List<AccountManager::AccountInfo>^ AccountList
{ List<AccountManager::AccountInfo>^ get(System::Void); }

        property Services::AccountManagerToken AccountToken
    {
        public:    Services::AccountManagerToken get(System::Void)
        {
            if (this->account_initialized) return this->account_token;
            throw gcnew
AccountManagerTokenException(AccountManagerToken::typeid, "token not init");
        }
        protected: System::Void set(Services::AccountManagerToken value) { this-
>account_token = value; }
        }
    public:
        [Manager::ServiceConfigurationAttribute("none")]
        AccountManager(IServiceBase::ServiceCtorConfiguration^ configuration,
SqlDatabaseManager^ sql_manager)
            : Manager::ServiceBase(configuration), account_initialized(false)
        { this->service_sql_manager = sql_manager; }

        virtual ~AccountManager(System::Void)
{ AccountManager::!AccountManager(); }
        !AccountManager(System::Void) { if (!service_disposed) { this-
>sign_out_account(); service_disposed = true; } }

        generic <class TAccountModel> where TAccountModel :
Models::AccountBaseModel
            virtual System::Boolean registration_account(System::String^ login,
System::String^ password,
TAccountModel account_model) override;

        virtual System::Boolean authorization_account(System::String^ login,
System::String^ password) override;

        generic <class TAccountModel> where TAccountModel :
Models::AccountBaseModel
            System::Boolean update_account(TAccountModel account_model);

        generic <class TAccountScheme> where TAccountScheme : Services::
AccountClassesDbScheme
            TAccountScheme get_account_scheme(System::Guid account_guid);

        virtual IServiceBase::ServiceQuery^
service_query_handler(System::TimeSpan work_time) override;

```

```

        System::Nullable<System::Boolean> get_account_status(System::Guid
account_guid);

        virtual System::Boolean delete_account(System::Guid guid) override;
        virtual System::Boolean sign_out_account(System::Void) override;
    };
}

```

Файл «/services/account_manager/account_manager.cpp»

```

#include "account_manager.h"

using namespace Services;

generic <class TEnum> TEnum convert_to_enum_cool(System::String^ value)
{
    return safe_cast<TEnum>(System::Enum::Parse(TEnum::typeid, value, true));
}

Manager::IServiceBase::ServiceQuery^
AccountManager::service_query_handler(System::TimeSpan work_time)
{
    Manager::IServiceBase::ServiceQuery^ service_query = gcnew
Manager::IServiceBase::ServiceQuery();

    service_query->Message = "Message from Account Manager";
    service_query->ServiceType = this->GetType();
    service_query->State = this->ServiceState;

    return service_query;
}

System::Nullable<System::Boolean> AccountManager::get_account_status(System::Guid
account_guid)
{
    List<IDatabaseManager::KeyValuePair^>^ request_list = gcnew
List<IDatabaseManager::KeyValuePair^>();
    request_list->Add(gcnew IDatabaseManager::KeyValuePair("account_guid",
account_guid.ToString()));

    List<IDatabaseManager::RequestRow^>^ response_list = this->service_sql_manager
        ->set_scheme_struct<AccountAuthenticationDbScheme^>()->
get_database_data(request_list, false);

    if (response_list == nullptr || response_list->Count <= 0) return
System::Nullable<System::Boolean>();
    AccountAuthenticationDbScheme^ scheme =
cli::safe_cast<AccountAuthenticationDbScheme^>(response_list[0]);

    System::Boolean account_status(false);
    try { account_status = System::Boolean::Parse(scheme->state); }
    catch (System::Exception^ error)
    { Console::WriteLine(error->Message); return
System::Nullable<System::Boolean>(); }

    return System::Nullable<System::Boolean>(account_status);
}

generic <class TAccountScheme> where TAccountScheme : Services::AccountClassesDbScheme
TAccountScheme AccountManager::get_account_scheme(System::Guid account_guid)
{
    TAccountScheme result_scheme;

```

```

        List<IDatabaseManager::KeyValuePair^>^ request_list = gcnew
List<IDatabaseManager::KeyValuePair^>();
    request_list->Add(gcnew IDatabaseManager::KeyValuePair("account_guid",
account_guid.ToString()));

        List<IDatabaseManager::RequestRow^>^ response_list = this->service_sql_manager
            ->set_scheme_struct<TAccountScheme>()->get_database_data(request_list,
false);
    if (response_list == nullptr || response_list->Count <= 0) return result_scheme;

        try { result_scheme = cli::safe_cast<TAccountScheme>(response_list-
>default[0]); }
        catch (System::Exception^ error) { Console::WriteLine(error->Message); }

        return result_scheme;
}

List<AccountManager::AccountInfo>^ AccountManager::AccountList::get(System::Void)
{
    List<IDatabaseManager::KeyValuePair^>^ request_list = gcnew
List<IDatabaseManager::KeyValuePair^>();
    request_list->Add(gcnew IDatabaseManager::KeyValuePair("account_state", "True"));
    request_list->Add(gcnew IDatabaseManager::KeyValuePair("account_state",
"False"));

    List<IDatabaseManager::RequestRow^>^ response_list = this->service_sql_manager
        ->set_scheme_struct<AccountAuthenticationDbScheme^>()->
get_database_data(request_list, true);
    if (response_list == nullptr)
        throw gcnew AccountManagerTokenException(AccountManager::typeid,
"AccountList::get DB error");

    List<AccountManager::AccountInfo>^ result = gcnew
List<AccountManager::AccountInfo>();
    for each (IDatabaseManager::RequestRow ^ item in response_list)
    {
        AccountAuthenticationDbScheme^ account_scheme =
(AccountAuthenticationDbScheme^)item;
        try {
            AccountManager::AccountInfo account_info_item {};
            account_info_item.Guid = System::Guid::Parse(account_scheme-
>account_guid);
            account_info_item.Type = convert_to_enum_cool
                <AccountManagerToken::AccountManagerType>(account_scheme-
>type);

            account_info_item.State = System::Boolean::Parse(account_scheme-
>state);
            account_info_item.Name = account_scheme->login;
            result->Add(account_info_item);
        }
        catch (System::Exception^ error) { Console::WriteLine(error->Message); }
    }

    return result;
}

generic <class TAccountModel> where TAccountModel : Models::AccountBaseModel
    System::Boolean AccountManager::registration_account(System::String^ login,
System::String^ password,
    TAccountModel account_model)
{
    if (this->account_initialized == true) return false;
}

```

```

        List<IDatabaseManager::KeyValuePair^>^ request_list = gcnew
List<IDatabaseManager::KeyValuePair^>();
    request_list->Add(gcnew IDatabaseManager::KeyValuePair("account_login", login));

        List<IDatabaseManager::RequestRow^>^ response_list = this->service_sql_manager
            ->set_scheme_struct<AccountAuthenticationDbScheme^>()->
>get_database_data(request_list, false);
    if (response_list == nullptr || response_list->Count > 0) return false;

        AccountManagerToken::AccountManagerType account_type;
        if (TAccountModel::typeid->Name == "AccountAdminModel") account_type =
AccountManagerToken::AccountManagerType::Admin;
        else if (TAccountModel::typeid->Name == "AccountDriverModel") account_type =
AccountManagerToken::AccountManagerType::Driver;
        else if (TAccountModel::typeid->Name == "AccountClientModel") account_type =
AccountManagerToken::AccountManagerType::Client;
        else return false;

        System::Guid account_guid = System::Guid::NewGuid();
        AccountAuthenticationDbScheme^ registration_list = gcnew
AccountAuthenticationDbScheme(
            account_guid.ToString(), login, password, account_type.ToString(),
true.ToString());

        System::Boolean registration_request = this->service_sql_manager
            ->set_scheme_struct<AccountAuthenticationDbScheme^>()->
>send_database_data(registration_list);
    if (registration_request != true) return false;

        System::Boolean add_account_request;
        switch (account_type)
        {
            case AccountManagerToken::AccountManagerType::Admin:
            {
                Models::AccountAdminModel^ cast_model =
(Models::AccountAdminModel^)account_model;
                AccountAdminsDbScheme^ account_scheme =
AccountAdminsDbScheme::cast_to_scheme(cast_model, account_guid);

                add_account_request = this->service_sql_manager-
>set_scheme_struct<AccountAdminsDbScheme^>()
                    ->send_database_data(account_scheme);
            }
            break;
            case AccountManagerToken::AccountManagerType::Driver:
            {
                Models::AccountDriverModel^ cast_model =
(Models::AccountDriverModel^)account_model;
                AccountDriversDbScheme^ account_scheme =
AccountDriversDbScheme::cast_to_scheme(cast_model, account_guid);

                add_account_request = this->service_sql_manager-
>set_scheme_struct<AccountDriversDbScheme^>()
                    ->send_database_data(account_scheme);
            }
            break;
            case AccountManagerToken::AccountManagerType::Client:
            {
                Models::AccountClientModel^ cast_model =
(Models::AccountClientModel^)account_model;
                AccountClientsDbScheme^ account_scheme =
AccountClientsDbScheme::cast_to_scheme(cast_model, account_guid);

                add_account_request = this->service_sql_manager-
>set_scheme_struct<AccountClientsDbScheme^>()

```

```

        ->send_database_data(account_scheme);
    }
    break;
}

if (add_account_request != true) return false;
this->AccountToken = AccountManagerToken(account_type, account_model,
account_guid);
this->account_initialized = true;

return true;
}

System::Boolean AccountManager::authorization_account(System::String^ login,
System::String^ password)
{
    if (this->account_initialized == true) return false;
    List<IDatabaseManager::KeyValuePair^>^ request_list = gcnew
List<IDatabaseManager::KeyValuePair^>();
    request_list->Add(gcnew IDatabaseManager::KeyValuePair("account_login", login));
    request_list->Add(gcnew IDatabaseManager::KeyValuePair("account_password",
password));

    List<IDatabaseManager::RequestRow^>^ response_list = this->service_sql_manager
        ->set_scheme_struct<AccountAuthenticationDbScheme^>()->
get_database_data(request_list, false); // <----- если шо
    if (response_list == nullptr || response_list->Count != 1) return false;

    AccountAuthenticationDbScheme^ account_auth =
safe_cast<AccountAuthenticationDbScheme^>(response_list[0]);
    try { if (System::Boolean::Parse(account_auth->state) == true) throw gcnew
System::Exception("already login"); }
    catch (System::Exception^ error) { Console::WriteLine(error->Message); return
false; }

    System::Guid account_guid = System::Guid::Empty;
    AccountManagerToken::AccountManagerType account_type;

    try {
        account_guid = System::Guid::Parse(account_auth->account_guid);
        account_type =
convert_to_enum_cool<AccountManagerToken::AccountManagerType>(account_auth->type);
    }
    catch (System::Exception^ error) { Console::WriteLine(error->Message); return
false; }

    List<IDatabaseManager::KeyValuePair^>^ request_key = gcnew
List<IDatabaseManager::KeyValuePair^>();
    List<IDatabaseManager::RequestRow^>^ response_obj = gcnew
List<IDatabaseManager::RequestRow^>();
    request_key->Add(gcnew IDatabaseManager::KeyValuePair("account_guid",
account_auth->account_guid));

    Models::AccountBaseModel^ account_model = nullptr;
    switch (account_type)
    {
        case AccountManagerToken::AccountManagerType::Admin:

            response_obj = this->service_sql_manager-
        >set_scheme_struct<AccountAdminsDbScheme^>()
            ->get_database_data(request_key, false);
            if (response_obj == nullptr || response_obj->Count != 1) return false;
            account_model =
AccountAdminsDbScheme::cast_to_model((AccountAdminsDbScheme^)response_obj[0]);
            break;
    }
}

```

```

        case AccountManagerToken::AccountManagerType::Driver:
            response_obj = this->service_sql_manager-
>set_scheme_struct<AccountDriversDbScheme^>()
                ->get_database_data(request_key, false);
            if (response_obj == nullptr || response_obj->Count != 1) return false;
            account_model =
AccountDriversDbScheme::cast_to_model((AccountDriversDbScheme^)response_obj[0]);
            break;
        case AccountManagerToken::AccountManagerType::Client:
            response_obj = this->service_sql_manager-
>set_scheme_struct<AccountClientsDbScheme^>()
                ->get_database_data(request_key, false);
            if (response_obj == nullptr || response_obj->Count != 1) return false;
            account_model =
AccountClientsDbScheme::cast_to_model((AccountClientsDbScheme^)response_obj[0]);
            break;
        }

        if (account_model == nullptr) return false;

        account_auth->state = true.ToString();
        System::Boolean account_state_update = this->service_sql_manager-
>set_scheme_struct<AccountAuthenticationDbScheme^>()
                ->update_database_date(account_auth, gcnew
IDatabaseManager::KeyValuePair("account_guid", account_auth->account_guid));
        if (account_state_update != true) return false;

        this->AccountToken = AccountManagerToken(account_type, account_model,
account_guid);
        this->account_initialized = true;
        return true;
    }

System::Boolean AccountManager::sign_out_account(System::Void)
{
    if (this->account_initialized != true) return false;
    List<IDatabaseManager::KeyValuePair^>^ request_list = gcnew
List<IDatabaseManager::KeyValuePair^>();
    request_list->Add(gcnew IDatabaseManager::KeyValuePair("account_guid", this-
>AccountToken.AccountGuid.ToString()));

    List<IDatabaseManager::RequestRow^>^ response_list = this->service_sql_manager
        ->set_scheme_struct<AccountAuthenticationDbScheme^>()->
get_database_data(request_list, false);

    if (response_list == nullptr || response_list->Count != 1) return false;
    AccountAuthenticationDbScheme^ account_auth =
safe_cast<AccountAuthenticationDbScheme^>(response_list[0]);

    account_auth->state = false.ToString();
    System::Boolean account_state_update = this->service_sql_manager-
>set_scheme_struct<AccountAuthenticationDbScheme^>()
                ->update_database_date(account_auth, gcnew
IDatabaseManager::KeyValuePair("account_guid", account_auth->account_guid));
    if (account_state_update != true) return false;

    AccountManagerToken::AccountManagerType account_enum_type;
    try {
        account_enum_type =
convert_to_enum_cool<AccountManagerToken::AccountManagerType>(account_auth->type);
    }
    catch (System::Exception^ error) { Console::WriteLine(error->Message); return
false; }
    this->account_initialized = false;
}

```

```

        this->AccountToken = AccountManagerToken(account_enum_type, nullptr,
System::Guid::Empty);
        return true;
    }

generic <class TAccountModel> where TAccountModel : Models::AccountBaseModel
{
    if (this->IsInitialized != true) return false;
    IDatabaseManager::KeyValuePair^ upload_key = gcnew
IDatabaseManager::KeyValuePair(
    "account_guid", this->AccountToken.AccountGuid.ToString());
    switch (this->AccountToken.AccountType)
    {
        case AccountManagerToken::AccountManagerType::Admin: {
            if (TAccountModel::typeid->Name != "AccountAdminModel") return false;
            IDatabaseManager::RequestRow^ upload_model =
AccountAdminsDbScheme::cast_to_scheme(
                (Models::AccountAdminModel^)account_model, this-
>AccountToken.AccountGuid);

            System::Boolean upload_check = this->service_sql_manager-
>set_scheme_struct<AccountAdminsDbScheme^>()
                ->update_database_date(upload_model, upload_key);
            if (upload_check != true) return false;
        } break;
        case AccountManagerToken::AccountManagerType::Client: {
            if (TAccountModel::typeid->Name != "AccountClientModel") return false;
            IDatabaseManager::RequestRow^ upload_model =
AccountClientsDbScheme::cast_to_scheme(
                (Models::AccountClientModel^)account_model, this-
>AccountToken.AccountGuid);

            System::Boolean upload_check = this->service_sql_manager-
>set_scheme_struct<AccountClientsDbScheme^>()
                ->update_database_date(upload_model, upload_key);
            if (upload_check != true) return false;
        } break;
        case AccountManagerToken::AccountManagerType::Driver: {
            if (TAccountModel::typeid->Name != "AccountDriverModel") return false;
            IDatabaseManager::RequestRow^ upload_model =
AccountDriversDbScheme::cast_to_scheme(
                (Models::AccountDriverModel^)account_model, this-
>AccountToken.AccountGuid);

            System::Boolean upload_check = this->service_sql_manager-
>set_scheme_struct<AccountDriversDbScheme^>()
                ->update_database_date(upload_model, upload_key);
            if (upload_check != true) return false;
        } break;
    }

    this->AccountToken = AccountManagerToken(AccountToken.AccountType, account_model,
AccountToken.AccountGuid);
    return true;
}

System::Boolean AccountManager::delete_account(System::Guid guid)
{
    if (guid == this->AccountToken.AccountGuid && this->IsInitialized != true)
return false;
    else {
        System::Nullable<System::Boolean> check = this->get_account_status(guid);
        if (check.HasValue != true || check.Value == true) return false;
    }
}

```

```

IDatabaseManager::KeyValuePair^ request_key = gcnew
IDatabaseManager::KeyValuePair("account_guid", guid.ToString());
List<IDatabaseManager::KeyValuePair^>^ request_list = gcnew
List<IDatabaseManager::KeyValuePair^>();
request_list->Add(request_key);

List<IDatabaseManager::RequestRow^>^ response_list = this->service_sql_manager
->set_scheme_struct<AccountAuthenticationDbScheme^>()->
get_database_data(request_list, false);

if (response_list == nullptr || response_list->Count != 1) return false;
AccountAuthenticationDbScheme^ account_auth =
safe_cast<AccountAuthenticationDbScheme^>(response_list[0]);

AccountManagerToken::AccountManagerType account_enum_type;
try {
    account_enum_type =
convert_to_enum_cool<AccountManagerToken::AccountManagerType>(account_auth->type);
}
catch (System::Exception^ error) { Console::WriteLine(error->Message); return
false; }
switch (account_enum_type)
{
case AccountManagerToken::AccountManagerType::Admin:
    if (this->service_sql_manager->set_scheme_struct<AccountAdminsDbScheme^>()
        ->delete_database_data(request_key) != true) return false;
    break;
case AccountManagerToken::AccountManagerType::Driver:
    if (this->service_sql_manager-
>set_scheme_struct<AccountDriversDbScheme^>()
        ->delete_database_data(request_key) != true) return false;
    break;
case AccountManagerToken::AccountManagerType::Client:
    if (this->service_sql_manager-
>set_scheme_struct<AccountClientsDbScheme^>()
        ->delete_database_data(request_key) != true) return false;
    break;
}

this->service_sql_manager->set_scheme_struct<AccountAuthenticationDbScheme^>()
->delete_database_data(request_key);
if (guid == this->AccountToken.AccountGuid)
{
    this->account_initialized = false;
    this->AccountToken = AccountManagerToken(account_enum_type, nullptr,
System::Guid::Empty);
}

return true;
}

```

Файл «/services/account_manager/account_manager_scheme.h»

```

#pragma once
#include "../database_provider/database_provider.h"
#include "../database_provider/database_provider_attribute.h"
#include "../../models/account_model/account_model.h"

namespace Services
{
    using namespace System;
    using namespace System::Collections::Generic;

```

```

public interface class AccountClassesDbScheme {};

[Services::SqlDatabaseTableAttribute("account_authentication")]
public ref class AccountAuthenticationDbScheme sealed :
SqlDatabaseManager::ISqlDataBaseSchemaType
{
    public: [Services::SqlDatabaseFieldAttribute("account_guid")]           property
System::String^ account_guid;
    public: [Services::SqlDatabaseFieldAttribute("account_login")]           property
System::String^ login;
    public: [Services::SqlDatabaseFieldAttribute("account_password")]         property
System::String^ password;
    public: [Services::SqlDatabaseFieldAttribute("account_type")]            property
System::String^ type;
    public: [Services::SqlDatabaseFieldAttribute("account_state")]           property
System::String^ state;

        AccountAuthenticationDbScheme(System::String^ p1, System::String^ p2,
System::String^ p3,
                                         System::String^ p4, System::String^ p5)
        { this->account_guid = p1;   this->login = p2;   this->password = p3;
this->type = p4;   this->state = p5; }
        AccountAuthenticationDbScheme(System::Void)
        { this->account_guid = "";   this->login = "";   this->password = "";
this->type = "";   this->state = ""; }

    virtual ~AccountAuthenticationDbScheme(System::Void) { delete
account_guid, login, password; }
};

[Services::SqlDatabaseTableAttribute("account_clients")]
public ref class AccountClientsDbScheme sealed :
SqlDatabaseManager::ISqlDataBaseSchemaType, AccountClassesDbScheme
{
    public: [Services::SqlDatabaseFieldAttribute("account_guid")]           property
System::String^ account_guid;
    public: [Services::SqlDatabaseFieldAttribute("username")]                property
System::String^ username;
    public: [Services::SqlDatabaseFieldAttribute("age")]                      property
System::String^ age;
    public: [Services::SqlDatabaseFieldAttribute("gender")]                 property
System::String^ gender;
    public: [Services::SqlDatabaseFieldAttribute("bank_card")]              property
System::String^ bank_card;

        AccountClientsDbScheme(System::String^ p1, System::String^ p2,
System::String^ p3, System::String^ p4, System::String^ p5)
        {
            this->account_guid = p1;   this->username = p2;   this->age = p3;
this->gender = p4;   this->bank_card = p5;
        }
        AccountClientsDbScheme(System::Void)
        {
            this->account_guid = "";   this->username = "";   this->age = "";
this->gender = "";   this->bank_card = "";
        }

    virtual ~AccountClientsDbScheme(System::Void) { delete account_guid,
username, age, gender, bank_card; }
    static Models::AccountClientModel^ cast_to_model(AccountClientsDbScheme^
scheme);
    static AccountClientsDbScheme^ cast_to_scheme(Models::AccountClientModel^
model, System::Guid guid);
};

```

```

[Services::SqlDatabaseTableAttribute("account_drivers")]
public ref class AccountDriversDbScheme sealed :
SqlDatabaseManager::ISqlDataBaseSchemeType, AccountClassesDbScheme
{
    public: [Services::SqlDatabaseFieldAttribute("account_guid")]
System::String^ account_guid;
    public: [Services::SqlDatabaseFieldAttribute("username")]
System::String^ username;
    public: [Services::SqlDatabaseFieldAttribute("age")]
    property System::String^ age;
    public: [Services::SqlDatabaseFieldAttribute("gender")]
    property System::String^ gender;
    public: [Services::SqlDatabaseFieldAttribute("driver_licence")]
System::String^ licence;
    public: [Services::SqlDatabaseFieldAttribute("bank_card")]
    property System::String^ bank_card;

        AccountDriversDbScheme(System::String^ p1, System::String^ p2,
System::String^ p3, System::String^ p4, System::String^ p5, System::String^ p6)
    {
        this->account_guid = p1;      this->username = p2;      this->age
= p3;  this->gender = p4;          this->licence = p5;          this->bank_card = p6;
    }
        AccountDriversDbScheme(System::Void)
    {
        this->account_guid = "";      this->username = "";      this->age
= "";  this->gender = "";          this->licence = "";          this->bank_card = "";
    }

        virtual ~AccountDriversDbScheme(System::Void) { delete account_guid,
username, age, gender, licence, bank_card; }
        static Models::AccountDriverModel^
cast_to_model(AccountDriversDbScheme^ scheme);
        static AccountDriversDbScheme^
cast_to_scheme(Models::AccountDriverModel^ model, System::Guid guid);
    };

[Services::SqlDatabaseTableAttribute("account_admins")]
public ref class AccountAdminsDbScheme sealed :
SqlDatabaseManager::ISqlDataBaseSchemeType, AccountClassesDbScheme
{
    public: [Services::SqlDatabaseFieldAttribute("account_guid")]
System::String^ account_guid;
    public: [Services::SqlDatabaseFieldAttribute("username")]
System::String^ username;
    public: [Services::SqlDatabaseFieldAttribute("age")]
    property System::String^ age;
    public: [Services::SqlDatabaseFieldAttribute("gender")]
    property System::String^ gender;
    public: [Services::SqlDatabaseFieldAttribute("permissions")]
System::String^ permissions;

        AccountAdminsDbScheme(System::String^ p1, System::String^ p2,
System::String^ p3, System::String^ p4, System::String^ p5)
    {
        this->account_guid = p1;      this->username = p2;      this->age = p3;
        this->gender = p4;  this->permissions = p5;  }
        AccountAdminsDbScheme(System::Void)
    {
        this->account_guid = "";      this->username = "";      this->age = "";
        this->gender = "";  this->permissions = "";  }

        virtual ~AccountAdminsDbScheme(System::Void) { delete account_guid,
username, age, gender, permissions; }

```

```

        static Models::AccountAdminModel^ cast_to_model(AccountAdminsDbScheme^
scheme);
        static AccountAdminsDbScheme^ cast_to_scheme(Models::AccountAdminModel^
model, System::Guid guid);
    };
}

```

Файл «/services/account_manager/account_manager_scheme.cpp»

```

#include "account_manager_scheme.h"

using namespace Services;

generic <class TEnum> TEnum fixedconvert_to_enum(System::String^ value)
{
    return safe_cast<TEnum>(System::Enum::Parse(TEnum::typeid, value, true));
}

Models::AccountClientModel^
AccountClientsDbScheme::AccountClientsDbScheme::cast_to_model(AccountClientsDbScheme^
scheme)
{
    Models::AccountClientModel^ account_model = nullptr;
    try {
        account_model = gcnew Models::AccountClientModel(scheme->username,
System::UInt32::Parse(scheme->age),
                fixedconvert_to_enum<Models::AccountModelGender>(scheme->gender),
System::Guid::Parse(scheme->bank_card));
    }
    catch (System::Exception^ error) { Console::WriteLine(error->Message); return
nullptr; }
    return account_model;
}

Models::AccountAdminModel^ AccountAdminsDbScheme::cast_to_model(AccountAdminsDbScheme^
scheme)
{
    Models::AccountAdminModel^ account_model = nullptr;
    try {
        account_model = gcnew Models::AccountAdminModel(scheme->username,
System::UInt32::Parse(scheme->age),
                fixedconvert_to_enum<Models::AccountModelGender>(scheme->gender),
                fixedconvert_to_enum<Models::AccountModelPermissions>(scheme-
>permissions));
    }
    catch (System::Exception^ error) { Console::WriteLine(error->Message); return
nullptr; }
    return account_model;
}

Models::AccountDriverModel^
AccountDriversDbScheme::cast_to_model(AccountDriversDbScheme^ scheme)
{
    Models::AccountDriverModel^ account_model = nullptr;
    try {
        account_model = gcnew Models::AccountDriverModel(scheme->username,
System::UInt32::Parse(scheme->age),
                fixedconvert_to_enum<Models::AccountModelGender>(scheme->gender),
                System::Guid::Parse(scheme->licence), System::Guid::Parse(scheme-
>bank_card));
    }
    catch (System::Exception^ error) { Console::WriteLine(error->Message); return
nullptr; }
    return account_model;
}

```

```

}

AccountDriversDbScheme^
AccountDriversDbScheme::cast_to_scheme(Models::AccountDriverModel^ model, System::Guid
guid)
{
    AccountDriversDbScheme^ account_scheme = gcnew AccountDriversDbScheme();

    account_scheme->account_guid = guid.ToString();
    account_scheme->age = model->Age.ToString();
    account_scheme->username = model->Username;
    account_scheme->gender = model->Gender.ToString();
    account_scheme->bank_card = model->BankCard.ToString();
    account_scheme->licence = model->Licence.ToString();

    return account_scheme;
}

AccountAdminsDbScheme^
AccountAdminsDbScheme::cast_to_scheme(Models::AccountAdminModel^ model, System::Guid
guid)
{
    AccountAdminsDbScheme^ account_scheme = gcnew AccountAdminsDbScheme();

    account_scheme->account_guid = guid.ToString();
    account_scheme->username = model->Username;
    account_scheme->age = model->Age.ToString();
    account_scheme->gender = model->Gender.ToString();
    account_scheme->permissions = model->Permissions.ToString();

    return account_scheme;
}

AccountClientsDbScheme^
AccountClientsDbScheme::cast_to_scheme(Models::AccountClientModel^ model, System::Guid
guid)
{
    AccountClientsDbScheme^ account_scheme = gcnew AccountClientsDbScheme();

    account_scheme->account_guid = guid.ToString();
    account_scheme->age = model->Age.ToString();
    account_scheme->username = model->Username;
    account_scheme->gender = model->Gender.ToString();
    account_scheme->bank_card = model->BankCard.ToString();

    return account_scheme;
}

```

Файл «/services/account_manager/account_manager_token.h»

```

#pragma once
#include "../models/account_model/account_model.h"
#include "../models/cars_model/cars_model.h"

namespace Services
{
    using namespace System;
    using namespace Models;
    using namespace System::Collections::Generic;

    public ref class AccountManagerTokenException sealed : System::Exception
    {
        private:     System::Type^ exception_attach_from = nullptr;
        public:
            property System::Type^ AttachFrom

```

```

    { public: System::Type^ get(System::Void) { return this->exception_attach_from; } }

    virtual property System::String^ Message
    {
        public: System::String^ get(System::Void) override
        { return "From " + exception_attach_from->ToString() + ":" + Exception::Message; }
    }
    public: AccountManagerTokenException(System::Type^ from, System::String^ message) : System::Exception(message)
        { this->exception_attach_from = from; }
    virtual ~AccountManagerTokenException(System::Void) { delete exception_attach_from; }
};

public value struct AccountManagerToken sealed
{
    public: enum class AccountManagerType : System::UInt16 { Admin, Driver, Client };
private: System::Guid account_guid;
private: AccountManagerToken::AccountManagerType account_type;
private: Models::AccountBaseModel^ account_model;

public:
    property System::Guid AccountGuid
    { public: System::Guid get(System::Void) { return this->account_guid; } };

    property Models::AccountBaseModel^ AccountModel
    { AccountBaseModel^ get(System::Void) { return (AccountBaseModel^)account_model->Clone(); } };

    property Services::AccountManagerToken::AccountManagerType AccountType
    { AccountManagerToken::AccountManagerType get(System::Void) { return account_type; } };

    public: AccountManagerToken(AccountManagerType type, Models::AccountBaseModel^ model, Guid id)
        : account_guid(id), account_type(type)
    {
        if (model != nullptr) this->account_model = (AccountBaseModel^)model->Clone();
        else this->account_model = nullptr;
    }
};
}

```

Файл «/services/bank_controller/bank_controller.h»

```

#pragma once
#include "../database_provider/database_provider.h"
#include "../../manager/manager.h"
#include "bank_controller_scheme.h"

namespace Services
{
    using namespace System;
    using namespace System::Collections;
    using namespace Manager;

    public interface class IBankController
    {

```

```

        public:      System::Boolean transfer_money(System::Guid payee_guid,
System::Int32 money);
        public:      System::Boolean take_money(System::Int32 money);
        public:      System::Boolean put_money(System::Int32 money);
    };

    [Manager::ServiceAttribute::ServiceRequireAttribute(Services::SqlDatabaseManager:
:typeid)]
    public ref class BankController sealed : Manager::ServiceBase,
Services::IBankController
    {
        public: value struct BankAccount { System::Guid AccountGuid; System::Int32
Money; };

        private:     Services::SqlDatabaseManager^ service_sql_manager = nullptr;
        private:     BankController::BankAccount^ bank_account = nullptr;

        BankAccountDbScheme^ account_check(System::Guid account_guid);
        public:
            property System::Guid AccountGuid { public: System::Guid
get(System::Void); }
            property System::Int32 AccountMoney { public: System::Int32
get(System::Void); }
        public:
            [Manager::ServiceConfigurationAttribute("none")]
            BankController(IServiceBase::ServiceCtorConfiguration^ configuration,
SqlDatabaseManager^ sql_manager)
                : Manager::ServiceBase(configuration) { this->service_sql_manager =
sql_manager; }

            virtual ~BankController(System::Void) { ServiceBase::~ServiceBase(); }
            virtual System::Boolean transfer_money(System::Guid payee_guid,
System::Int32 money) override;

            virtual IServiceBase::ServiceQuery^
service_query_handler(System::TimeSpan work_time) override;

            System::Boolean load_bank_account(System::Guid account_guid);
            System::Boolean create_bank_account(System::Guid account_guid);
            Generic::List<System::Guid>^ get_bank_accounts(System::Void);

            virtual System::Boolean take_money(System::Int32 money) override;
            virtual System::Boolean put_money(System::Int32 money) override;
    };
}

```

Файл «/services/bank_controller/bank_controller.cpp»

```

#include "bank_controller.h"

using namespace Services;

System::Guid BankController::AccountGuid::get(System::Void)
{
    if (this->bank_account != nullptr) return this->bank_account->AccountGuid;
    return System::Guid::Empty;
}

System::Int32 BankController::AccountMoney::get(System::Void)
{
    if (this->bank_account != nullptr) return this->bank_account->Money;
    return System::Int32(0);
}

```

```

Manager::IServiceBase::ServiceQuery^
BankController::service_query_handler(System::TimeSpan work_time)
{
    Manager::IServiceBase::ServiceQuery^ service_query = gcnew
Manager::IServiceBase::ServiceQuery();

    service_query->Message = "Message from Bank Controller";
    service_query->ServiceType = this->GetType();
    service_query->State = this->ServiceState;

    return service_query;
}

System::Boolean BankController::transfer_money(System::Guid payee_guid, System::Int32
money)
{
    if (this->bank_account == nullptr || money <= 0) return false;

    List<IDatabaseManager::KeyValuePair^>^ receiver_key = gcnew
List<IDatabaseManager::KeyValuePair^>();
    receiver_key->Add(gcnew IDatabaseManager::KeyValuePair("account_guid",
payee_guid.ToString()));

    auto receiver_items = this->service_sql_manager-
>set_scheme_struct<BankAccountDbScheme^>()->get_database_data(receiver_key, false);
    if (receiver_items == nullptr || receiver_items->Count <= 0) { return false; }

    if (this->take_money(money) != true) return false;

    BankAccountDbScheme^ receiver_model =
safe_cast<BankAccountDbScheme^>(receiver_items[0]);
    System::Int32 receiver_money(0);
    System::Boolean transfer_checker(true);

    try { receiver_money = System::UInt32::Parse(receiver_model->account_money); }
    catch (System::Exception^ error) { this->put_money(money); return false; }

    receiver_model->account_money = System::Int32(receiver_money + money).ToString();

    System::Boolean update_receiver = this->service_sql_manager-
>set_scheme_struct<BankAccountDbScheme^>()
    ->update_database_date(receiver_model, gcnew
IDatabaseManager::KeyValuePair("account_guid", receiver_model->account_guid));

    if (update_receiver != true) { this->put_money(money); return false; }
    return true;
}

System::Boolean BankController::create_bank_account(System::Guid account_guid)
{
    List<IDatabaseManager::KeyValuePair^>^ request_key = gcnew
List<IDatabaseManager::KeyValuePair^>();
    request_key->Add(gcnew IDatabaseManager::KeyValuePair("account_guid",
account_guid.ToString()));

    auto request_items = this->service_sql_manager-
>set_scheme_struct<BankAccountDbScheme^>()
    ->get_database_data(request_key, false);
    if (request_items != nullptr && request_items->Count > 0) return false;

    BankAccountDbScheme^ account_model = gcnew BankAccountDbScheme();
    account_model->account_guid = account_guid.ToString();
    account_model->account_money = System::Int32(0).ToString();
    account_model->account_marker = System::Boolean(true).ToString();
}

```

```

        if (!this->service_sql_manager->set_scheme_struct<BankAccountDbScheme^>()->send_database_data(account_model)) return false;
        return true;
    }

BankAccountDbScheme^ BankController::account_check(System::Guid account_guid)
{
    List<IDatabaseManager::KeyValuePair^>^ request_key = gcnew
List<IDatabaseManager::KeyValuePair^>();
    request_key->Add(gcnew IDatabaseManager::KeyValuePair("account_guid",
account_guid.ToString()));

    auto request_items = this->service_sql_manager-
>set_scheme_struct<BankAccountDbScheme^>()
    ->get_database_data(request_key, false);
    if (request_items == nullptr || request_items->Count <= 0) return nullptr;
    return safe_cast<BankAccountDbScheme^>(request_items[0]);
}

System::Boolean BankController::take_money(System::Int32 money)
{
    if (this->bank_account == nullptr) return false;
    BankAccountDbScheme^ item = this->account_check(this->AccountGuid);
    if (item == nullptr) return false;

    System::Int32 account_money(0);
    try { account_money = System::Int32::Parse(item->account_money); }
    catch (System::Exception^ error) { Console::WriteLine(error->Message); return
false; }

    if (account_money - money < 0) return false;

    this->bank_account->Money = System::Int32(account_money - money);
    item->account_money = this->bank_account->Money.ToString();

    if (!this->service_sql_manager->set_scheme_struct<BankAccountDbScheme^>()
        ->update_database_date(item, gcnew
IDatabaseManager::KeyValuePair("account_guid", item->account_guid))) return false;

    return true;
}

System::Boolean BankController::put_money(System::Int32 money)
{
    if (this->bank_account == nullptr) return false;
    BankAccountDbScheme^ item = this->account_check(this->AccountGuid);
    if (item == nullptr) return false;

    System::Int32 account_money(0);
    try { account_money = System::Int32::Parse(item->account_money); }
    catch (System::Exception^ error) { Console::WriteLine(error->Message); return
false; }

    this->bank_account->Money = System::Int32(account_money + money);
    item->account_money = this->bank_account->Money.ToString();

    if (!this->service_sql_manager->set_scheme_struct<BankAccountDbScheme^>()
        ->update_database_date(item, gcnew
IDatabaseManager::KeyValuePair("account_guid", item->account_guid))) return false;

    return true;
}

System::Boolean BankController::load_bank_account(System::Guid account_guid)
{

```

```

        BankAccountDbScheme^ item = this->account_check(account_guid);
        if (item != nullptr) {
            this->bank_account = gcnew BankController::BankAccount();
            try {
                this->bank_account->Money = System::UInt32::Parse(item-
>account_money);
                this->bank_account->AccountGuid = System::Guid::Parse(item-
>account_guid);
            }
            catch (System::Exception^ error) { this->bank_account = nullptr; return
false; }
        }
        else return false;
    }

    return true;
}

Generic::List<System::Guid>^ BankController::get_bank_accounts(System::Void)
{
    List<IDatabaseManager::KeyValuePair>^ request_key = gcnew
List<IDatabaseManager::KeyValuePair>();
    request_key->Add(gcnew IDatabaseManager::KeyValuePair("account_marker", "True"));

    auto request_items = this->service_sql_manager-
>set_scheme_struct<BankAccountDbScheme>()
        ->get_database_data(request_key, false);
    if (request_items == nullptr || request_items->Count <= 0) return nullptr;

    List<System::Guid>^ result_list = gcnew List<System::Guid>();
    for each (auto item in request_items)
    {
        System::Guid item_guid;
        try {
            BankAccountDbScheme^ scheme = safe_cast<BankAccountDbScheme>(item);
            item_guid = System::Guid::Parse(scheme->account_guid);
        }
        catch (System::Exception^ error) { Console::WriteLine(error->Message);
continue; }

        result_list->Add(item_guid);
    }

    return result_list;
}

```

Файл «/services/bank_controller/bank_controller_scheme.h»

```

#pragma once
#include "../database_provider/database_provider.h"
#include "../database_provider/database_provider_attribute.h"

namespace Services
{
    [Services::SqlDatabaseTableAttribute("bank_accounts")]
    public ref struct BankAccountDbScheme sealed :
SqlDatabaseManager::ISqlDataBaseSchemaType
    {
        public: [Services::SqlDatabaseFieldAttribute("account_guid")] property
System::String^ account_guid;
        public: [Services::SqlDatabaseFieldAttribute("account_money")] property
System::String^ account_money;
        public: [Services::SqlDatabaseFieldAttribute("account_marker")] property
System::String^ account_marker;

```

```

        BankAccountDbScheme(System::String^ p1, System::String^ p2,
System::String^ p3)
        { this->account_guid = p1; this->account_money = p2; this->account_marker
= p3; }

        BankAccountDbScheme(System::Void) { this->account_guid = ""; this-
>account_money = ""; this->account_marker = ""; }
        virtual ~BankAccountDbScheme(System::Void) { delete account_guid,
account_money, account_marker; }
    };
}

```

Файл «/services/database_provider/database_provider.h»

```

#pragma once
#include "../manager/manager.h"
#include "database_provider_attribute.h"

namespace Services
{
    using namespace System;
    using namespace System::Reflection;
    using namespace System::Collections::Generic;
    using namespace MySql::Data;
    using namespace Manager;

    public interface class IDatabaseManager
    {
        public: using KeyValuePair = System::Tuple<System::String^, System::String^>;
        public: using RequestRow = System::Object;

        List<RequestRow^>^ get_database_data(List<KeyValuePair^>^ searching_param,
System::Boolean merging);
        System::Boolean update_database_date(RequestRow^ request_param,
KeyValuePair^ searching_param);

        System::Boolean send_database_data(IDatabaseManager::RequestRow^
request_param);
        System::Boolean delete_database_data(IDatabaseManager::KeyValuePair^
searching_param);
    };

    public ref class SqlDatabaseManagerException sealed : System::Exception
    {
        private: System::Type^ exception_attach_from = nullptr;
        public:
            property System::Type^ AttachFrom
            { public: System::Type^ get(System::Void) override
                { return "From " + exception_attach_from->ToString() + ": " +
Exception::Message; } }

            virtual property System::String^ Message
            {
                public: System::String^ get(System::Void) override
                { return "From " + exception_attach_from->ToString() + ": " +
Exception::Message; }
            }
        public:
            SqlDatabaseManagerException(System::Type^ from, System::String^ message) :
System::Exception(message)
            { this->exception_attach_from = from; }
            virtual ~SqlDatabaseManagerException(System::Void) { delete
exception_attach_from; }
    };
}

```

```

public ref class SqlDatabaseManager sealed : Manager::ServiceBase,
IDatabaseManager
{
public: interface class ISqlDataBaseSchemeType { };
    value struct SqlDatabaseFieldKey { System::String^ Attribute;
System::String^ ClassField; };
private:
    MySqlClient::MySqlConnection^ db_connection = nullptr;
    List<SqlDatabaseFieldKey>^ db_keys_name = nullptr;

private: Threading::Mutex^ connection_mutex = nullptr;
private: System::String^ db_table_name = nullptr;
private: System::Type^ db_scheme_type = nullptr;
public:
    property System::String^ TableName
    {
        public: System::String^ get(System::Void) { return this->db_table_name; }
        public: System::Void set(System::String^ value) { this->db_table_name =
value; }
    }

    property System::String^ SchemeName
    { public: System::String^ get(System::Void) { return this-
>db_scheme_type->ToString(); } }

    property List<System::String^>^ CurrentScheme { List<System::String^>^
get(System::Void); }
private: System::Void db_build_connection(System::Void);
public:
    [Manager::ServiceConfigurationAttribute("database_provider")]
    SqlDatabaseManager(IServiceBase::ServiceCtorConfiguration^ conf) :
Manager::ServiceBase(conf)
    {
        this->db_keys_name = gcnew List<SqlDatabaseFieldKey>();
        this->connection_mutex = gcnew Threading::Mutex();
        this->db_build_connection();
    }
    virtual ~SqlDatabaseManager(System::Void)
    {
        delete this->db_connection, this->db_table_name, this->db_keys_name;
        Manager::ServiceBase::~ServiceBase();
    }
    generic <class TSchemeStruct> Services::SqlDatabaseManager^
set_scheme_struct(System::Void);

    virtual System::Boolean
update_database_date(IDatabaseManager::RequestRow^ request_param,
    IDatabaseManager::KeyValuePair^ searching_param) override;

    virtual System::Boolean send_database_data(IDatabaseManager::RequestRow^
request_param);
    virtual System::Boolean
delete_database_data(IDatabaseManager::KeyValuePair^ searching_param);

    virtual List<IDatabaseManager::RequestRow^>^ get_database_data(
        List<IDatabaseManager::KeyValuePair^>^ searching_param,
        System::Boolean mergering) override;

    virtual IServiceBase::ServiceQuery^
service_query_handler(System::TimeSpan work_time) override;
};

}

```

Файл «/services/database_provider/database_provider.cpp»

```

#pragma once
#include "database_provider.h"

using namespace Services;
using namespace System;

List<System::String^>^ SqlDatabaseManager::CurrentScheme::get(System::Void)
{
    List<System::String^>^ copy_list = gcnew List<System::String^>();
    for each (auto item in this->db_keys_name) { copy_list-
>Add(safe_cast<System::String^>(item.Attribute->Clone())); }
    return copy_list;
}

System::Void SqlDatabaseManager::db_build_connection(System::Void)
{
    // "server=localhost;user=root;database=test;password=prolodgy778"
    System::String^ database_connection_string = "server=" + this-
>configuration["sql_connection_server"]
    + ";user=" + this->configuration["sql_connection_user"]
    + ";database=" + this->configuration["sql_connection_database"]
    + ";password=" + this->configuration["sql_connection_password"];

    this->db_connection = gcnew
MySqlConnection(database_connection_string);
}

IServiceBase::ServiceQuery^ SqlDatabaseManager::service_query_handler(System::TimeSpan
work_time)
{
    Manager::IServiceBase::ServiceQuery^ service_query = gcnew
Manager::IServiceBase::ServiceQuery();

    service_query->Message = "Message from Sql Database Manager";
    service_query->ServiceType = this->GetType();
    service_query->State = this->ServiceState;

    return service_query;
}

generic <class TSchemeStruct> Services::SqlDatabaseManager^
SqlDatabaseManager::set_scheme_struct(System::Void)
{
    System::Type^ scheme_deconstruct = TSchemeStruct::typeid;
    SqlDatabaseTableAttribute^ scheme_attribute =
safe_cast<SqlDatabaseTableAttribute^>(
        System::Attribute::GetCustomAttribute(scheme_deconstruct,
SqlDatabaseTableAttribute::typeid));
    this->TableName = scheme_attribute->TableName;

    if(scheme_deconstruct->GetInterface("ISqlDataBaseSchemeType") == nullptr)
        throw gcnew SqlDatabaseManagerException(SqlDatabaseManager::typeid, "Does
Not Implement ISqlDataBaseSchemeType");
    array<System::Reflection::PropertyInfo^>^ property_list = scheme_deconstruct-
>GetProperties();

    this->db_scheme_type = scheme_deconstruct;
    this->db_keys_name->Clear();

    for (int i = 0; i < property_list->Length; i++)
    {
        SqlDatabaseFieldAttribute^ field_attribute =
safe_cast<SqlDatabaseFieldAttribute^>(
            System::Attribute::GetCustomAttribute(property_list[i],
SqlDatabaseFieldAttribute::typeid));

```

```

        SqlDatabaseFieldKey database_key{ field_attribute->FieldName,
property_list[i]->Name };

        if(field_attribute != nullptr) this->db_keys_name->Add(database_key);
    }
    return this;
}

System::String^ build_request_string(IDatabaseManager::RequestRow^ request_type,
System::String^ request_key)
{
    System::Reflection::PropertyInfo^ prop_info = nullptr;

    try { prop_info = request_type->GetType()->GetProperty(request_key,
BindingFlags::Public | BindingFlags::Instance); }
    catch (System::Exception^ error) { Console::WriteLine(error->Message); return
nullptr; }

    return prop_info->GetValue(request_type)->ToString();
}

System::Boolean SqlDatabaseManager::update_database_date(IDatabaseManager::RequestRow^
request_param,
    IDatabaseManager::KeyValuePair^ searching_param)
{
    if (request_param == nullptr || searching_param == nullptr || db_keys_name-
>Count == 0) return false;
    this->connection_mutex->WaitOne();
    this->db_connection->Open();
    try
    {
        System::String^ request_string = "UPDATE " + this->db_table_name + " SET
";
        for(int i = 0; i < db_keys_name->Count; i++)
        {
            request_string = String::Concat(request_string,
db_keys_name[i].Attribute, " = \"",
                build_request_string(request_param,
db_keys_name[i].ClassField), "\" ");
            if (i != db_keys_name->Count - 1) request_string += ", ";
        }
        request_string = String::Concat(request_string, " WHERE ",
searching_param->Item1, " = \"", searching_param->Item2, "\" ");

        MySqlCommand::MySqlCommand^ sql_command = gcnew
MySqlCommand(request_string, this->db_connection);
        sql_command->ExecuteNonQuery();
    }
    catch (MySqlClient::MySqlException^ error) { System::Console::WriteLine(error-
>Message); return false; }
    finally { this->db_connection->Close(); this->connection_mutex->ReleaseMutex(); }

    return true;
}

List<IDatabaseManager::RequestRow^>^ SqlDatabaseManager::get_database_data(
    List<IDatabaseManager::KeyValuePair^>^ searching_param, System::Boolean
mergering)
{
    List<IDatabaseManager::RequestRow^>^ request_result = gcnew
List<IDatabaseManager::RequestRow^>();
    if (searching_param == nullptr || db_keys_name->Count == 0) return
request_result;

    this->connection_mutex->WaitOne();

```

```

        this->db_connection->Open();
        try {
            System::String^ request = "SELECT * FROM " + this->db_table_name + "
WHERE 1";
            for each (auto item in searching_param)
                { request = System::String::Concat(request, mergering ? " OR " : " AND ",
item->Item1, " = \"", item->Item2, "\""); }

            MySqlClient::MySqlCommand^ sql_command = gcnew
MySqlClient::MySqlCommand(request, this->db_connection);
            MySqlClient::MySqlDataReader^ sql_reader = sql_command->ExecuteReader();

            while (sql_reader->Read())
            {
                IDatabaseManager::RequestRow^ request_row = nullptr;
                array<System::Object^>^ param = gcnew
array<System::Object^>(sql_reader->FieldCount - 1);
                for (int i = 1; i < sql_reader->FieldCount; i++) { param[i - 1] =
sql_reader[i]->ToString(); }

                try { request_row = System::Activator::CreateInstance(this-
>db_scheme_type, param); }
                catch (System::Exception^ error)
                { sql_reader->Close(); this->db_connection->Close(); return
nullptr; }

                request_result->Add(request_row);
            }
            sql_reader->Close();
        }
        catch (MySqlClient::MySqlException^ error) { System::Console::WriteLine(error-
>Message); return nullptr; }
        finally { this->db_connection->Close(); this->connection_mutex->ReleaseMutex(); }

        return request_result;
    }

System::Boolean SqlDatabaseManager::send_database_data(IDatabaseManager::RequestRow^
request_param)
{
    System::Type^ request_type = request_param->GetType();
    if (request_param == nullptr || request_type != this->db_scheme_type ||
db_keys_name->Count == 0) return false;
    System::Boolean request_result = true;

    this->connection_mutex->WaitOne();
    this->db_connection->Open();
    try {
        MySqlClient::MySqlCommand^ sql_command = this->db_connection-
>CreateCommand();
        sql_command->CommandText = "INSERT INTO " + this->db_table_name + " (" +
db_keys_name[0].Attribute;
        for (int i = 1; i < db_keys_name->Count; i++)
        {
            sql_command->CommandText += System::String::Concat(",",
db_keys_name[i].Attribute);
        }
        sql_command->CommandText += System::String::Concat(") VALUES (\"",
build_request_string(request_param, db_keys_name[0].ClassField),
"\")");

        for(int i = 1; i < db_keys_name->Count; i++)
        {
            sql_command->CommandText += System::String::Concat(", ?",
db_keys_name[i].Attribute);
        }
    }
}
```

```

        sql_command->Parameters->Add("?" + db_keys_name[i].Attribute,
            MySqlClient::MySqlDbType::VarChar)->Value
            = build_request_string(request_param,
db_keys_name[i].ClassField);
        }
        sql_command->CommandText += ");";
        sql_command->ExecuteNonQuery();
    }
    catch (System::Exception^ error) { System::Console::WriteLine(error->Message);
request_result = false; }
finally { this->db_connection->Close(); this->connection_mutex->ReleaseMutex(); }

    return request_result;
}

System::Boolean
SqlDatabaseManager::delete_database_data(IDatabaseManager::KeyValuePair^
searching_param)
{
    if (searching_param == nullptr || db_keys_name->Count == 0) return false;

    this->connection_mutex->WaitOne();
    this->db_connection->Open();
    try {
        System::String^ request = System::String::Concat( "DELETE FROM " + this-
>db_table_name + " WHERE ",
            searching_param->Item1, " = \"", searching_param->Item2, "\"");

        MySqlClient::MySqlCommand^ sql_command = gcnew
MySqlClient::MySqlCommand(request, this->db_connection);
        sql_command->ExecuteNonQuery();
    }
    catch (MySqlClient::MySqlException^ error) { System::Console::WriteLine(error-
>Message); return false; }
finally { this->db_connection->Close(); this->connection_mutex->ReleaseMutex(); }

    return true;
}

```

Файл «/services/database_provider/database_provider_attribute.h»

```

#pragma once

namespace Services
{
    [System::AttributeUsage(System::AttributeTargets::Property, AllowMultiple =
false)]
    public ref class SqlDatabaseFieldAttribute sealed : System::Attribute
    {
        System::String^ database_field_name = nullptr;
    public:
        SqlDatabaseFieldAttribute(System::String^ field_name) :
System::Attribute()
        {
            this->database_field_name = field_name;
            virtual ~SqlDatabaseFieldAttribute(System::Void) { delete this-
>database_field_name; }

            property System::String^ FieldName
            {
                System::String^ get(System::Void) { return this-
>database_field_name; }
            }
        };
    [System::AttributeUsage(System::AttributeTargets::Class, AllowMultiple =
false)]

```

```

public ref class SqlDatabaseTableAttribute sealed : System::Attribute
{
    System::String^ database_table_name = nullptr;
public:
    SqlDatabaseTableAttribute(System::String^ table_name) :
System::Attribute()
    {
        this->database_table_name = table_name;
    }
    virtual ~SqlDatabaseTableAttribute(System::Void) { delete this-
>database_table_name; }

    property System::String^ TableName
    { System::String^ get(System::Void) { return this-
>database_table_name; } }
};

}

```

Файл «/services/depot_manager/depot_manager.h»

```

#pragma once
#include "../manager/manager.h"
#include "../models/cars_model/cars_model.h"
#include "../database_provider/database_provider.h"
#include "depot_manager_token.h"
#include "depot_manager_scheme.h"

namespace Services
{
    using namespace System;
    using namespace System::Collections::Generic;
    using namespace System::Collections;
    using namespace Manager;

    public interface class IDepotManager
    {
        public:     System::Boolean return_car_model(System::Void);
        public:
            generic <class TCarClass> where TCarClass : Models::CarBaseModel
                System::Boolean rent_car_model(TCarClass car_model, System::Guid
driver_guid);

            generic <class TCarClass> where TCarClass : Models::CarBaseModel
                System::Boolean add_car_model(TCarClass car_model, System::UInt16
count);

            generic <class TCarClass> where TCarClass : Models::CarBaseModel
                System::Boolean delete_car_model(TCarClass car_model);
    };

    [Manager::ServiceAttribute::ServiceRequireAttribute(Services::SqlDatabaseManager:
:typeid)]
    public ref class DepotManager sealed : Manager::ServiceBase,
Services::IDepotManager
    {
        public:     value struct CarGarageItems
        { System::Guid Guid; Models::CarBaseModel^ CarModel; System::Type^ CarClass;
System::UInt32 CarCount; };

        private:    Services::DriveComplexToken^ drive_complex = nullptr;
        private:    Services::SqlDatabaseManager^ service_sql_manager = nullptr;

        private:    System::Boolean service_disposed;
                    System::UInt32 garage_collection_size;
    };
}

```

```

public:
    property System::Guid DriverGuid { public: System::Guid
get(System::Void); }
        property Models::CarBaseModel^ CarModel { public: Models::CarBaseModel^
get(System::Void); }

        property System::Type^ CarModelType { public: System::Type^
get(System::Void); }
        property System::UInt32 CarTypePrice[Models::CarModelTypes]
        { public: System::UInt32 get(Models::CarModelTypes value); }

        property System::Boolean IsBuilded
        { public: System::Boolean get(System::Void) { return (this-
>drive_complex != nullptr); } }

        property DriveComplexToken::DriverStateType DriverState
        { public: DriveComplexToken::DriverStateType get(System::Void); }

private: System::Boolean delete_car_process(Services::CarModelDbScheme^ item);
public:
    [Manager::ServiceConfigurationAttribute("depot_manager")]
    DepotManager(IServiceBase::ServiceCtorConfiguration^ configuration,
Services::SqlDatabaseManager^ db_manager)
        : Manager::ServiceBase(configuration), service_disposed(false)
    {
        if (!configuration->ContainsKey("car_child_type_price")
|| !configuration->ContainsKey("car_econom_type_price")
        || !configuration->ContainsKey("car_premium_type_price"))
throw gcnew System::Exception("Parameter Error");

        this->garage_collection_size =
System::UInt32::Parse((String^)configuration["garage_collection_size"]);
        this->service_sql_manager = db_manager;
    }
    virtual ~DepotManager(System::Void) { delete this->drive_complex;
ServiceBase::~ServiceBase(); }
    !DepotManager(System::Void)
    { if(!service_disposed && drive_complex != nullptr) { return_car_model();
service_disposed = true; } }

    Services::DriveComplexDbScheme^ get_driver_complexs(System::Guid
driver_guid);
    System::Boolean update_drive_state(DriveComplexToken::DriverStateType
state);

    Generic::List<System::Guid>^ get_all_drivers(System::Void);
    Generic::List<DepotManager::CarGarageItems>^ get_all_cars(System::Void);

    generic <class TCarClass> where TCarClass : Models::CarBaseModel
        virtual System::Boolean add_car_model(TCarClass car_model, UInt16
count) override;

    generic <class TCarClass> where TCarClass : Models::CarBaseModel
        virtual System::Boolean rent_car_model(TCarClass car_model,
System::Guid driver_guid);

    generic <class TCarClass> where TCarClass : Models::CarBaseModel
        virtual System::Boolean delete_car_model(TCarClass car_model)
override;
    virtual System::Boolean return_car_model(System::Void) override;

    virtual IServiceBase::ServiceQuery^
service_query_handler(TimeSpan work_time) override;
    System::Boolean delete_car_model_by_guid(System::Guid group_id);

```

```
    };
```

Файл «/services/depot_manager/depot_manager.cpp»

```
#include "depot_manager.h"

using namespace Services;

generic <class TEnum> TEnum convert_to_enum(System::String^ value)
{ return safe_cast<TEnum>(System::Enum::Parse(TEnum::typeid, value, true)); }

List<IDatabaseManager::KeyValuePair^>^ create_car_request(Models::CarBaseModel^ car_model, System::Type^ car_class)
{
    List<IDatabaseManager::KeyValuePair^>^ key_pair = gcnew List<IDatabaseManager::KeyValuePair^>();
    key_pair->Add(gcnew IDatabaseManager::KeyValuePair("car_class", car_class->Name));
    key_pair->Add(gcnew IDatabaseManager::KeyValuePair("car_type", car_model->CarType.ToString()));
    key_pair->Add(gcnew IDatabaseManager::KeyValuePair("car_speed", car_model->CarSpeed.ToString()));
    key_pair->Add(gcnew IDatabaseManager::KeyValuePair("car_color", car_model->CarColor.ToString()));

    return key_pair;
}

System::UInt32 DepotManager::CarTypePrice::get(Models::CarModelTypes car_type_value)
{
    switch (car_type_value)
    {
        case Models::CarModelTypes::CarTypeChild:
            return System::UInt32::Parse((System::String^)this->configuration["car_child_type_price"]);
        case Models::CarModelTypes::CarTypeEconom:
            return System::UInt32::Parse((System::String^)this->configuration["car_econom_type_price"]);
        case Models::CarModelTypes::CarTypePremium:
            return System::UInt32::Parse((System::String^)this->configuration["car_premium_type_price"]);
    }
}

Manager::IServiceBase::ServiceQuery^
DepotManager::service_query_handler(System::TimeSpan work_time)
{
    Manager::IServiceBase::ServiceQuery^ service_query = gcnew Manager::IServiceBase::ServiceQuery();

    service_query->Message = "Message from Depot Manager";
    service_query->ServiceType = this->GetType();
    service_query->State = this->ServiceState;

    return service_query;
}

System::Boolean DepotManager::delete_car_process(Services::CarModelDbScheme^ item)
{
    System::Int32 garage_car_count(0);
    try { garage_car_count = System::Int32::Parse(item->car_count); }
    catch (System::Exception^ error) { Console::WriteLine(error->Message); return false; }
```

```

        System::Boolean delete_check(true);
        if (garage_car_count <= 1)
        {
            delete_check = this->service_sql_manager-
>set_scheme_struct<Services::CarModelDbSchema^>()
                ->delete_database_data(gcnew
IDatabaseManager::KeyValuePair("group_guid", item->group_guid));
        }
        else
        {
            item->car_count = System::Int32(garage_car_count - 1).ToString();
            delete_check = this->service_sql_manager-
>set_scheme_struct<Services::CarModelDbSchema^>()
                ->update_database_date(item, gcnew
IDatabaseManager::KeyValuePair("group_guid", item->group_guid));
        }

        return delete_check;
    }

System::Boolean DepotManager::delete_car_model(System::Guid group_id)
{
    List<IDatabaseManager::KeyValuePair^>^ key_pair = gcnew
List<IDatabaseManager::KeyValuePair^>();
    key_pair->Add(gcnew IDatabaseManager::KeyValuePair("group_guid",
group_id.ToString()));

    auto request_items = this->service_sql_manager-
>set_scheme_struct<Services::CarModelDbSchema^>()
                ->get_database_data(key_pair, false);

    if (request_items == nullptr || request_items->Count <= 0) return false;
    Services::CarModelDbSchema^ item =
safe_cast<Services::CarModelDbSchema^>(request_items[0]);

    return this->delete_car_process(item);
}

generic <class TCarClass> where TCarClass : Models::CarBaseModel
    System::Boolean DepotManager::delete_car_model(TCarClass car_model)
{
    List<IDatabaseManager::KeyValuePair^>^ key_pair = create_car_request(car_model,
TCarClass::typeid);
    auto request_items = service_sql_manager-
>set_scheme_struct<Services::CarModelDbSchema^>()
                ->get_database_data(key_pair, false);

    if (request_items == nullptr || request_items->Count <= 0) return false;
    Services::CarModelDbSchema^ item =
safe_cast<Services::CarModelDbSchema^>(request_items[0]);

    return this->delete_car_process(item);
}

generic <class TCarClass> where TCarClass : Models::CarBaseModel
    System::Boolean DepotManager::rent_car_model(TCarClass car_model, System::Guid
driver_guid)
{
    if (this->drive_complex != nullptr) return false;
    List<IDatabaseManager::KeyValuePair^>^ key_pair = create_car_request(car_model,
TCarClass::typeid);

    auto request_items = service_sql_manager-
>set_scheme_struct<Services::CarModelDbSchema^>()

```

```

        ->get_database_data(key_pair, false);
    if (request_items == nullptr || request_items->Count <= 0) return false;
    Services::CarModelDbScheme^ item =
safe_cast<Services::CarModelDbScheme^>(request_items[0]);

    System::Int32 db_model_count(0);
    try { db_model_count = System::Int32::Parse(item->car_count) - 1; }
    catch (System::Exception^ error) { Console::WriteLine(error->Message); return
false; }

    if (db_model_count > 0)
    {
        item->car_count = db_model_count.ToString();
        if (!this->service_sql_manager-
>set_scheme_struct<Services::CarModelDbScheme^>()
            ->update_database_date(item, gcnew
IDatabaseManager::KeyValuePair("group_guid", item->group_guid)))
            return false;
    }
    else
    {
        if (!service_sql_manager-
>set_scheme_struct<Services::CarModelDbScheme^>()->delete_database_data(
            gcnew IDatabaseManager::KeyValuePair("group_guid", item-
>group_guid))) return false;
    }
    Services::DriveComplexDbScheme^ scheme = gcnew Services::DriveComplexDbScheme();
    scheme->driver_state = DriveComplexToken::DriverStateType::Idle.ToString();
    scheme->car_class = TCarClass::typeid->Name;
    scheme->car_type = car_model->CarType.ToString();
    scheme->driver_guid = driver_guid.ToString();

    if (this->service_sql_manager-
>set_scheme_struct<Services::DriveComplexDbScheme^>()
            ->send_database_data(scheme) != true)
        throw gcnew Services::DriveComplexTokenException(DepotManager::typeid,
"rent_car_model");

    this->drive_complex = gcnew DriveComplexToken(car_model, driver_guid,
TCarClass::typeid);
    return true;
}

System::Boolean DepotManager::return_car_model(System::Void)
{
    if (this->drive_complex == nullptr) return false;
    List<IDatabaseManager::KeyValuePair^>^ key_pair
        = create_car_request(this->drive_complex->CarModel, this->drive_complex-
>CarModelType);

    auto request_items = service_sql_manager-
>set_scheme_struct<Services::CarModelDbScheme^>()
        ->get_database_data(key_pair, false);
    if (request_items != nullptr && request_items->Count > 0)
    {
        Services::CarModelDbScheme^ item =
safe_cast<Services::CarModelDbScheme^>(request_items[0]);
        System::Int32 new_model_count(1);
        try { new_model_count = System::Int32::Parse(item->car_count) + 1; }
        catch (System::Exception^ error) { Console::WriteLine(error->Message);
return false; }

        item->car_count = new_model_count.ToString();
        if (!service_sql_manager->set_scheme_struct<Services::CarModelDbScheme^>()

```

```

        ->update_database_date(item, gcnew
IDatabaseManager::KeyValuePair("group_guid", item->group_guid)))
            return false;
    }
    else
    {
        Services::CarModelDbScheme^ model = gcnew Services::CarModelDbScheme();
        model->car_class = this->drive_complex->CarModelType->Name;
        model->car_type = this->drive_complex->CarModel->CarType.ToString();
        model->car_speed = this->drive_complex->CarModel->CarSpeed.ToString();
        model->car_color = this->drive_complex->CarModel->CarColor.ToString();
        model->car_count = System::UInt32(1).ToString();
        model->group_guid = System::Guid::NewGuid().ToString();

        if(!service_sql_manager-
>set_scheme_struct<Services::CarModelDbScheme^>()>send_database_data(model))
            return false;
    }

    if (!service_sql_manager->set_scheme_struct<Services::DriveComplexDbScheme^>()
        ->delete_database_data(gcnew IDatabaseManager::KeyValuePair("driver_guid",
this->DriverGuid.ToString())))
        return false;

    this->drive_complex = nullptr;
    return true;
}

System::Boolean DepotManager::update_drive_state(DriveComplexToken::DriverStateType
state)
{
    if (this->drive_complex == nullptr) return false;

    List<IDatabaseManager::KeyValuePair^>^ key_pair = gcnew
List<IDatabaseManager::KeyValuePair^>();
    key_pair->Add(gcnew IDatabaseManager::KeyValuePair("driver_guid", this-
>DriverGuid.ToString()));

    auto request_items = service_sql_manager-
>set_scheme_struct<Services::DriveComplexDbScheme^>()
        ->get_database_data(key_pair, false);
    if (request_items == nullptr || request_items->Count <= 0) return false;

    Services::DriveComplexDbScheme^ response_item = nullptr;
    try { response_item =
safe_cast<Services::DriveComplexDbScheme^>(request_items[0]); }
    catch (System::Exception^ error) { Console::WriteLine(error->Message); return
false; }

    response_item->driver_state = state.ToString();
    if (!service_sql_manager->set_scheme_struct<Services::DriveComplexDbScheme^>()
        ->update_database_date(response_item, gcnew
IDatabaseManager::KeyValuePair("driver_guid", response_item->driver_guid)))
        return false;

    this->drive_complex->DriverState = state;
    return true;
}

generic <class TCarClass> where TCarClass : Models::CarBaseModel
    System::Boolean DepotManager::add_car_model(TCarClass car_model, System::UInt16
count)
{
    List<IDatabaseManager::KeyValuePair^>^ key_pair = create_car_request(car_model,
TCarClass::typeid);

```

```

        auto request_items = service_sql_manager-
>set_scheme_struct<Services::CarModelDbScheme^>()
->get_database_data(key_pair, false);

    if (request_items == nullptr || request_items->Count <= 0)
    {
        Services::CarModelDbScheme^ model = gcnew Services::CarModelDbScheme();
        model->car_class = TCarClass::typeid->Name;
        model->car_type = car_model->CarType.ToString();
        model->car_speed = car_model->CarSpeed.ToString();
        model->car_color = car_model->CarColor.ToString();
        model->car_count = count.ToString();
        model->group_guid = System::Guid::NewGuid().ToString();

        if (service_sql_manager->set_scheme_struct<Services::CarModelDbScheme^>()
            ->send_database_data(model) != true) return false;
    }
    else
    {
        Services::CarModelDbScheme^ item =
safe_cast<Services::CarModelDbScheme^>(request_items[0]);
        System::Int32 new_model_count(0);

        try { new_model_count = System::Int32::Parse(item->car_count) + count; }
        catch (System::Exception^ error) { Console::WriteLine(error->Message);
return false; }

        item->car_count = new_model_count.ToString();
        if (!service_sql_manager->set_scheme_struct<Services::CarModelDbScheme^>()
            ->update_database_date(item, gcnew
IDatabaseManager::KeyValuePair("group_guid", item->group_guid)))
            return false;
    }
    return true;
}

Models::CarBaseModel^ DepotManager::CarModel::get(System::Void)
{
    if (this->drive_complex == nullptr) return nullptr;
    return (Models::CarBaseModel^)this->drive_complex->CarModel->Clone();
}

System::Type^ DepotManager::CarModelType::get(System::Void)
{
    if (this->drive_complex == nullptr) return nullptr;
    return this->drive_complex->CarModelType;
}

System::Guid DepotManager::DriverGuid::get(System::Void)
{
    if (this->drive_complex != nullptr) return this->drive_complex->DriverGuid;
    return System::Guid::Empty;
}

DriveComplexToken::DriverStateType DepotManager::DriverState::get(System::Void)
{
    if (this->drive_complex != nullptr) return this->drive_complex->DriverState;
    throw gcnew Services::DriveComplexTokenException(DepotManager::typeid, "drive
complex not found");
}

Generic::List<System::Guid>^ DepotManager::get_all_drivers(System::Void)
{
    List<System::Guid>^ driver_list = gcnew List<System::Guid>();

```

```

        List<IDatabaseManager::KeyValuePair^>^ request_key = gcnew
List<IDatabaseManager::KeyValuePair^>();

        request_key->Add(gcnew IDatabaseManager::KeyValuePair("car_class",
"CarLightModel"));
        request_key->Add(gcnew IDatabaseManager::KeyValuePair("car_class",
"CarHeavyModel"));

        auto response_param = this->service_sql_manager-
>set_scheme_struct<Services::DriveComplexDbScheme^>()
    ->get_database_data(request_key, true);
    for each (IDatabaseManager::RequestRow^ item in response_param)
    {
        try {
            System::Guid value =
System::Guid::Parse(safe_cast<Services::DriveComplexDbScheme^>(item)->driver_guid);
            driver_list->Add(value);
        }
        catch (System::Exception^ error) { Console::WriteLine(error->Message); }
    }
    return driver_list;
}

Generic::List<DepotManager::CarGarageItems^>^ DepotManager::get_all_cars(System::Void)
{
    List<DepotManager::CarGarageItems^>^ car_list = gcnew
List<DepotManager::CarGarageItems^>();
    List<IDatabaseManager::KeyValuePair^>^ request_key = gcnew
List<IDatabaseManager::KeyValuePair^>();

    request_key->Add(gcnew IDatabaseManager::KeyValuePair("car_class",
"CarLightModel"));
    request_key->Add(gcnew IDatabaseManager::KeyValuePair("car_class",
"CarHeavyModel"));

    auto response_param = this->service_sql_manager-
>set_scheme_struct<Services::CarModelDbScheme^>()
    ->get_database_data(request_key, true);
    if (response_param == nullptr || response_param->Count <= 0) return nullptr;

    for each (auto item in response_param)
    {
        Services::CarModelDbScheme^ car_model = nullptr;
        Services::DepotManager::CarGarageItems garage_item;

        Models::CarModelTypes garage_item_type;
        Models::CarModelColor garage_item_color;
        System::UInt32 garage_item_speed(0);

        try {
            car_model = safe_cast<Services::CarModelDbScheme^>(item);
            garage_item.Guid = System::Guid::Parse(car_model->group_guid);
            garage_item_type =
convert_to_enum<Models::CarModelTypes>(car_model->car_type);
            garage_item_color =
convert_to_enum<Models::CarModelColor>(car_model->car_color);
            garage_item_speed = System::UInt32::Parse(car_model->car_speed);
            garage_item.CarCount = System::UInt32::Parse(car_model->car_count);
        }
        catch (System::Exception^ error) { Console::WriteLine(error->Message);
continue; }

        if (car_model->car_class == "CarHeavyModel")
        {

```

```

garage_item.CarModel = gcnew Models::CarHeavyModel(garage_item_type,
garage_item_color, garage_item_speed);
garage_item.CarClass = Models::CarHeavyModel::typeid;

}
else if (car_model->car_class == "CarLightModel")
{
    garage_item.CarModel = gcnew Models::CarLightModel(garage_item_type,
garage_item_color, garage_item_speed);
    garage_item.CarClass = Models::CarLightModel::typeid;
}
else continue;
car_list->Add(garage_item);
}

return car_list;
}

Services::DriveComplexDbScheme^ DepotManager::get_driver_complexes(System::Guid
driver_guid)
{
    List<IDatabaseManager::KeyValuePair^>^ request_param = gcnew
List<IDatabaseManager::KeyValuePair^>();
    request_param->Add(gcnew IDatabaseManager::KeyValuePair("driver_guid",
driver_guid.ToString()));

    auto response_param = this->service_sql_manager-
>set_scheme_struct<Services::DriveComplexDbScheme^>()
    ->get_database_data(request_param, false);

    if (request_param == nullptr || request_param->Count > 1)
        throw gcnew Services::DriveComplexTokenException(DepotManager::typeid,
"get_driver_complexes");
    return safe_cast<Services::DriveComplexDbScheme^>(response_param[0]);
}

```

Файл «/services/depot_manager/depot_manager_scheme.h»

```

#pragma once
#include "../database_provider/database_provider.h"
#include "../database_provider/database_provider_attribute.h"

namespace Services
{
    [Services::SqlDatabaseTableAttribute("drive_complex")]
    public ref class DriveComplexDbScheme sealed :
SqlDatabaseManager::ISqlDataBaseSchemeType
    {
        public: [Services::SqlDatabaseFieldAttribute("driver_guid")] property
System::String^ driver_guid;
        public: [Services::SqlDatabaseFieldAttribute("car_type")] property
System::String^ car_type;
        public: [Services::SqlDatabaseFieldAttribute("car_class")] property
System::String^ car_class;
        public: [Services::SqlDatabaseFieldAttribute("driver_status")] property
System::String^ driver_state;

        DriveComplexDbScheme(System::String^ p1, System::String^ p2,
System::String^ p3, System::String^ p4)
        {
            this->driver_guid = p1; this->car_type = p2;      this->car_class = p3;
            this->driver_state = p4; }
        DriveComplexDbScheme(System::Void)
        {
            this->driver_guid = ""; this->car_type = "";      this->car_class = "";
            this->driver_state = ""; }
    }
}

```

```

        virtual ~DriveComplexDbScheme(System::Void) { delete driver_guid,
car_type, car_class, driver_state; }

};

[Services::SqlDatabaseTableAttribute("car_garage")]
public ref class CarModelDbScheme sealed :
SqlDatabaseManager::ISqlDataBaseSchemeType
{
    public: [Services::SqlDatabaseFieldAttribute("group_guid")] property
System::String^ group_guid;
    public: [Services::SqlDatabaseFieldAttribute("car_type")] property
System::String^ car_type;
    public: [Services::SqlDatabaseFieldAttribute("car_class")] property
System::String^ car_class;
    public: [Services::SqlDatabaseFieldAttribute("car_speed")] property
System::String^ car_speed;
    public: [Services::SqlDatabaseFieldAttribute("car_color")] property
System::String^ car_color;
    public: [Services::SqlDatabaseFieldAttribute("car_count")] property
System::String^ car_count;

    CarModelDbScheme(System::String^ p1, System::String^ p2, System::String^
p3, System::String^ p4,
                    System::String^ p5, System::String^ p6)
    {
        this->car_type = p2;      this->car_class = p3;      this->car_speed
= p4;  this->car_color = p5;
        this->car_count = p6;      this->group_guid = p1;
    }
    CarModelDbScheme(System::Void)
    {
        this->car_type = "";      this->car_class = "";      this->car_speed
= "";  this->car_color = "";
        this->car_count = "";      this->group_guid = "";
    }

    virtual ~CarModelDbScheme(System::Void)
        { delete car_count, car_type, car_class, car_color, car_speed,
group_guid; }
    };
}

```

Файл «/services/depot_manager/depot_manager_token.h»

```

#pragma once
#include "../models/cars_model/cars_model.h"

namespace Services
{
    using namespace System;
    using namespace System::Collections;

    public ref class DriveComplexTokenException sealed : System::Exception
    {
private:     System::Type^ exception_attach_from = nullptr;
public:
    property System::Type^ AttachFrom
        { public: System::Type^ get(System::Void) { return this-
>exception_attach_from; } }

    virtual property System::String^ Message
    {
        public: System::String^ get(System::Void) override

```

```

        { return "From " + exception_attach_from->ToString() + ": " +
Exception::Message; }
    }
public:
    DriveComplexTokenException(System::Type^ from, System::String^ message) :
System::Exception(message)
    { this->exception_attach_from = from; }
    virtual ~DriveComplexTokenException(System::Void) { delete
exception_attach_from; }
};

public value struct DriveComplexToken sealed : System::ICloneable
{
public: enum class DriverStateType : System::UInt16 { Busy, Ready, Idle };
public: property Models::CarBaseModel^ CarModel;
public: property System::Type^ CarModelType;

private: Services::DriveComplexToken::DriverStateType driver_state;
private: System::Guid driver_guid;
public:
    property System::Guid DriverGuid
    {
        public: System::Guid get(System::Void) { return this-
>driver_guid; }
        protected: System::Void set(System::Guid value) { this->driver_guid =
value; }
    };
    property Services::DriveComplexToken::DriverStateType DriverState
    {
        public: DriverStateType get(System::Void) { return this-
>driver_state; }
        public: System::Void set(DriverStateType value) { this->driver_state =
value; }
    };

public:
    DriveComplexToken(Models::CarBaseModel^ car_model, Guid driver_guid,
Type^ car_model_type)
        : driver_guid(driver_guid), driver_state(DriverStateType::Idle)
    { this->CarModel = car_model; this->CarModelType = car_model_type; }

    virtual System::Object^ Clone(System::Void) override
    {
        Services::DriveComplexToken^ token = gcnew
Services::DriveComplexToken(
            (Models::CarBaseModel^)CarModel->Clone(), this->driver_guid,
CarModelType);
        token->DriverState = this->driver_state;
        return token;
    }
};
}

```

Файл «/services/order controller/order controller.h»

```
#pragma once
#include "../manager/manager.h"
#include "../depot_manager/depot_manager.h"
#include "../models/cars_model/cars_model.h"
#include "order_controller_token.h"

//#define ORDER_REQUEST_SECOND 15

namespace Services
```

```

{
    using namespace System;
    using namespace System::Collections::Generic;
    using namespace System::Threading;
    using namespace System::Threading::Tasks;
    using namespace Manager;

    public interface class IOrderController
    {
        public: generic<class TCarModelClass> where TCarModelClass :
Models::CarBaseModel
            System::Boolean registration_order(String^ request_address,
Models::CarModelTypes car_type,
            System::Guid client_guid);

        public:           System::Boolean accept_request(System::Guid order_id,
System::Guid driver_id);
        public:           System::Boolean cancellation_order(System::Void);
    };

    [Manager::ServiceAttribute::ServiceRequireAttribute(Services::DepotManager::typeid)]
    [Manager::ServiceAttribute::ServiceRequireAttribute(Services::SqlDatabaseManager::typeid)]
    public ref class OrderController sealed : Manager::ServiceBase,
Services::IOrderController
    {
        public:           value struct RequestAcceptToken { System::Boolean Status;
System::Guid ConnectionGuid; };
        public:           using OrderPairConnection = System::Tuple<System::Guid,
System::Guid>;
        public:           delegate System::Void
RequestCallback(OrderController::RequestAcceptToken);
        private:
            System::UInt32 session_order_count, order_request_second;
            System::Boolean service_disposed;

            Services::SqlDatabaseManager^ service_sql_manager = nullptr;
            Services::DepotManager^ service_depot_manager = nullptr;
            Services::OrderControllerToken^ order_token = nullptr;

            System::Threading::CancellationTokenSource^ cancel_source = nullptr;
            System::Threading::CancellationToken request_cancel_token;
            OrderController::RequestCallback^ request_callback = nullptr;
        public:
            event RequestCallback^ OrderRequestCallback
            {
                System::Void add(RequestCallback^ handler) { this-
>request_callback += handler; }
                System::Void remove(RequestCallback^ handler) { this-
>request_callback -= handler; }
            }
            property System::UInt32 SessionOrderCount
            { public: System::UInt32 get(System::Void) { return this-
>session_order_count; } }

            property System::UInt32 OrderRequestSecond
            { public: System::UInt32 get (System::Void) { return this-
>order_request_second; } }
        private:   System::Boolean add_request(System::Void);
        private:   RequestAcceptToken order_process(System::Void);

        System::Void order_callback(Task<RequestAcceptToken>^ task)
    };
}

```

```

        { this->request_callback(task->Result); this->request_callback =
nullptr; }
    public:
        property Services::OrderControllerToken^ OrderToken
        { public: OrderControllerToken^ get(System::Void) { return this-
>order_token; } }

        property List<Services::OrderControllerDbScheme^>^ OrderList
        { public: List<Services::OrderControllerDbScheme^>^ get(System::Void); }

public:
    [Manager::ServiceConfigurationAttribute("order_controller")]
    OrderController(IServiceBase::ServiceCtorConfiguration^ conf,
SqlDatabaseManager^ sql_manager,
DepotManager^ depot_manager) : Manager::ServiceBase(conf),
session_order_count(0), service_disposed(false)
{
    this->order_request_second = System::UInt32::Parse((String^)this-
>configuration["order_request_wait_second"]);
    this->service_sql_manager = sql_manager; this-
>service_depot_manager = depot_manager;
}
virtual ~OrderController(System::Void) { delete cancel_source;
Manager::ServiceBase::~ServiceBase(); }
!OrderController(System::Void)
{ if (!service_disposed && order_token != nullptr) { cancellation_order();
service_disposed = true; } }

generic<class TCarModelClass> where TCarModelClass: Models::CarBaseModel
virtual System::Boolean registration_order(System::String^
request_address, Models::CarModelTypes car_type,
System::Guid client_guid) override;

virtual System::Boolean accept_request(Guid order_id, Guid driver_id)
override;
virtual IServiceBase::ServiceQuery^
service_query_handler(System::TimeSpan work_time) override;

virtual System::Boolean cancellation_order(System::Void) override;
System::Boolean cancellation_token_push(System::Void);
};

}

```

Файл «/services/order_controller/order_controller.cpp»

```

#include "order_controller.h"

using namespace Services;

List<Services::OrderControllerDbScheme^>^ OrderController::OrderList::get(System::Void)
{
    List<IDatabaseManager::KeyValuePair^>^ request_data = gcnew
List<IDatabaseManager::KeyValuePair^>();
    request_data->Add(gcnew IDatabaseManager::KeyValuePair("order_status", "False"));
    request_data->Add(gcnew IDatabaseManager::KeyValuePair("order_status", "True"));
    auto request_result = this->service_sql_manager
        ->set_scheme_struct<Services::OrderControllerDbScheme^>()
->get_database_data(request_data, true);

    List<Services::OrderControllerDbScheme^>^ function_result = gcnew
List<Services::OrderControllerDbScheme^>();
    for each (auto item in request_result)
    { function_result->Add(safe_cast<Services::OrderControllerDbScheme^>(item)); }

```

```

        return function_result;
    }

System::Boolean OrderController::add_request(System::Void)
{
    this->session_order_count++;
    Services::OrderControllerDbScheme^ request_data = gcnew
Services::OrderControllerDbScheme();

    request_data->address = this->order_token->OrderAddress;
    request_data->car_class = this->order_token->CarModelClass->Name;
    request_data->car_type = this->order_token->CarModelType.ToString();

    request_data->driver_guid = this->order_token->DriverGuid.ToString();
    request_data->client_guid = this->order_token->ClientGuid.ToString();
    request_data->date_time = this->order_token->OrderDate.ToString();
    request_data->order_status = Convert::.ToBoolean(0).ToString();

    return this->service_sql_manager-
>set_scheme_struct<Services::OrderControllerDbScheme^>()
    ->send_database_data(request_data);
}

System::Boolean OrderController::accept_request(System::Guid order_id, System::Guid
driver_id)
{
    Services::DriveComplexDbScheme^ request_driver = this->service_depot_manager-
>get_driver_complexs(driver_id);
    auto request_keys_list = gcnew List<IDatabaseManager::KeyValuePair^>();
    request_keys_list->Add(gcnew IDatabaseManager::KeyValuePair("client_guid",
order_id.ToString()));

    List<IDatabaseManager::RequestRow^>^ request_row = this->service_sql_manager
        ->set_scheme_struct<Services::OrderControllerDbScheme^>()->
get_database_data(request_keys_list, true);

    //изменено request_row->Count > 1
    if (request_row->Count != 1 || request_row == nullptr)
        throw gcnew
Services::OrderControllerTokenException(OrderController::typeid, "order_guid
accept_request");

    OrderControllerDbScheme^ request_result =
safe_cast<OrderControllerDbScheme^>(request_row[0]);
    // проверка валидности принятого заказа с текущим водителем
    if (request_driver->car_class != request_result->car_class ||
        request_driver->car_type != request_result->car_type) return false;

    request_result->driver_guid = driver_id.ToString();
    request_result->order_status = Convert::.ToBoolean(1).ToString();
    bool update_check = this->service_sql_manager-
>set_scheme_struct<Services::OrderControllerDbScheme^>()
    ->update_database_date(request_result, gcnew
IDatabaseManager::KeyValuePair("client_guid", order_id.ToString()));

    if (update_check != true) return false;

    this->order_token = OrderControllerToken::create_from_dbscheme(request_result);
    return true;
}

generic<class TCarModelClass> where TCarModelClass: Models::CarBaseModel
System::Boolean OrderController::registration_order(System::String^ request_address,
Models::CarModelTypes car_type,
System::Guid client_guid)

```

```

{
    if (this->order_token != nullptr) return false;

    this->order_token = gcnew OrderControllerToken(request_address, car_type,
TCarModelClass::typeid, client_guid);
    this->cancel_source = gcnew CancellationTokenSource();
    this->request_cancel_token = cancel_source->Token;

    if (this->add_request() != true) { this->order_token = nullptr; return false; }
    Task<RequestAcceptToken>^ order_processing = gcnew Task<RequestAcceptToken>(
        gcnew System::Func<RequestAcceptToken>(this,
&Services::OrderController::order_process), this->request_cancel_token);

    order_processing->ContinueWith(gcnew
System::Action<Task<RequestAcceptToken>>^(this,
&Services::OrderController::order_callback));
    order_processing->Start();

    return true;
}

System::Boolean OrderController::cancellation_order(System::Void)
{
    if (this->order_token == nullptr) return false;
    System::Boolean check = this->service_sql_manager-
>set_scheme_struct<Services::OrderControllerDbScheme>()
    ->delete_database_data(gcnew IDatabaseManager::KeyValuePair("client_guid",
this->order_token->ClientGuid.ToString()));

    this->order_token = nullptr;
    return check;
}

System::Boolean OrderController::cancellation_token_push(System::Void)
{
    try { this->cancel_source->Cancel(); }
    catch (System::Exception^ error) { Console::WriteLine(error->Message); return
false; }
    return true;
}

OrderController::RequestAcceptToken OrderController::order_process(System::Void)
{
    for (System::UInt32 seconds = 0; seconds < this->order_request_second; seconds++)
    {
        if (this->request_cancel_token.IsCancellationRequested) { break; }
        List<IDatabaseManager::KeyValuePair>^ search_param = gcnew
List<IDatabaseManager::KeyValuePair>();
        search_param->Add(gcnew IDatabaseManager::KeyValuePair("client_guid",
this->order_token->ClientGuid.ToString()));

        List<IDatabaseManager::RequestRow>^ request_result = this-
>service_sql_manager
            ->set_scheme_struct<Services::OrderControllerDbScheme>()
            ->get_database_data(search_param, false);
        if (request_result == nullptr) return
OrderController::RequestAcceptToken{ false, System::Guid::Empty};

        // забираю первое вхождение в выборку (т.к guid уникальный ключ)
        IDatabaseManager::RequestRow^ request_row = request_result->default[0];
        Services::OrderControllerDbScheme^ request_obj =
safe_cast<Services::OrderControllerDbScheme>(request_row);

        if (System::Boolean::Parse(request_obj->order_status) == true)
        {

```

```

        System::Guid driver_id = System::Guid::Empty;

        try { driver_id = System::Guid::Parse(request_obj->driver_guid); }
        catch (System::Exception^)
        {
            return OrderController::RequestAcceptToken{ false,
System::Guid::Empty };
        }

        return OrderController::RequestAcceptToken{ true, driver_id };

    }

    Thread::Sleep(System::TimeSpan(0, 0, 1));
}
return OrderController::RequestAcceptToken{ false, System::Guid::Empty };
}

Manager::IServiceBase::ServiceQuery^
OrderController::service_query_handler(System::TimeSpan work_time)
{
    Manager::IServiceBase::ServiceQuery^ service_query = gcnew
Manager::IServiceBase::ServiceQuery();

    service_query->Message = "Message from Order Controller";
    service_query->ServiceType = this->GetType();
    service_query->State = this->ServiceState;

    return service_query;
}

```

Файл «/services/order_controller/order_controller_scheme.h»

```

#pragma once
#include "../database_provider/database_provider.h"
#include "../database_provider/database_provider_attribute.h"

namespace Services
{
    [Services::SqlDatabaseTableAttribute("order_collection")]
    public ref struct OrderControllerDbScheme sealed :
SqlDatabaseManager::ISqlDataBaseSchemeType
    {
        public: [Services::SqlDatabaseFieldAttribute("client_guid")] property
System::String^ client_guid;
        public: [Services::SqlDatabaseFieldAttribute("driver_guid")] property
System::String^ driver_guid;
        public: [Services::SqlDatabaseFieldAttribute("car_type")] property
System::String^ car_type;
        public: [Services::SqlDatabaseFieldAttribute("car_class")] property
System::String^ car_class;

        public: [Services::SqlDatabaseFieldAttribute("address")] property
System::String^ address;
        public: [Services::SqlDatabaseFieldAttribute("date_time")] property
System::String^ date_time;
        public: [Services::SqlDatabaseFieldAttribute("order_status")] property
System::String^ order_status;

        OrderControllerDbScheme(System::String^ p1, System::String^ p2,
System::String^ p3, System::String^ p4,
                           System::String^ p5, System::String^ p6, System::String^ p7)
        {
            this->client_guid = p1; this->driver_guid = p2; this->car_type =
p3; this->car_class = p4;
            this->address = p5;           this->date_time = p6;   this-
>order_status = p7;
        }
    }
}

```

```

        }
        OrderControllerDbScheme(System::Void)
    {
        this->client_guid = ""; this->driver_guid = ""; this->car_type =
""; this->car_class = "";
        this->address = ""; this->date_time = ""; this-
>order_status = "";
    }
    virtual ~OrderControllerDbScheme(System::Void)
    {
        delete order_status, client_guid, driver_guid, car_class,
car_type, address, date_time;
    }
}

```

Файл «/services/order_controller/order_controller_token.h»

```

#pragma once
#include "../models/cars_model/cars_model.h"
#include "order_controller_scheme.h"

namespace Services
{
    using namespace System;
    using namespace System::Collections::Generic;
    using namespace System::Reflection;

    public ref class OrderControllerTokenException sealed : System::Exception
    {
    private:    System::Type^ exception_attach_from = nullptr;
    public:
        property System::Type^ AttachFrom
        { public: System::Type^ get(System::Void) { return this-
>exception_attach_from; } }

        virtual property System::String^ Message
        {
            public: System::String^ get(System::Void) override
            { return "From " + exception_attach_from->ToString() + ":" + +
Exception::Message; }
        }
    public:
        OrderControllerTokenException(System::Type^ from, System::String^
message) : System::Exception(message)
        { this->exception_attach_from = from; }
        virtual ~OrderControllerTokenException(System::Void) { delete
exception_attach_from; }

    public value struct OrderControllerToken sealed
    {
        public:          System::Boolean OrderStatus;
        public:          System::DateTime OrderDate;
        public:          System::Guid ClientGuid, DriverGuid;
        public:          Models::CarModelTypes CarModelType;

        public: property System::Type^ CarModelClass;
        public:     property System::String^ OrderAddress;
        public:
            OrderControllerToken(String^ order_address, Models::CarModelTypes
car_model, Type^ car_class, Guid client)
                : ClientGuid(client), DriverGuid(Guid::Empty),
OrderDate(System::DateTime::Now), OrderStatus(false)
    };
}

```

```

    {
        if (car_class->IsSubclassOf(Models::CarBaseModel::typeid) != true)
        { throw gcnew
OrderControllerTokenException(OrderControllerToken::typeid, "car_class"); }

        this->OrderAddress = order_address;
        this->CarModelClass = car_class;
        this->CarModelType = car_model;
    }

    public: static OrderControllerToken
create_from_dbscheme(OrderControllerDbScheme^ scheme)
{
    System::Type^ car_class = scheme->car_class == "CarLightModel" ?
Models::CarLightModel::typeid :
                scheme->car_class == "CarHeavyModel" ?
Models::CarHeavyModel::typeid : nullptr;

    Models::CarModelTypes car_model;
    if (scheme->car_type == "CarTypeEconom") car_model =
Models::CarModelTypes::CarTypeEconom;
    else if (scheme->car_type == "CarTypePremium") car_model =
Models::CarModelTypes::CarTypePremium;
    else if (scheme->car_type == "CarTypeChild") car_model =
Models::CarModelTypes::CarTypeChild;
    else throw gcnew
OrderControllerTokenException(OrderControllerToken::typeid, "create_from_dbscheme");

    System::Guid driver_guid = System::Guid::Parse(scheme->driver_guid);
    System::Guid client_guid = System::Guid::Parse(scheme->client_guid);
    OrderControllerToken token(scheme->address, car_model, car_class,
client_guid);

    token.OrderStatus = Convert::.ToBoolean(scheme->order_status);
    token.OrderDate = System::DateTime::Parse(scheme->date_time);
    return token;
}
};

}

```

Файл «/services/services.h»

```

#pragma once

#ifndef SERVICES_INCLUDE_STUFF
#define SERVICES_INCLUDE_STUFF

#include "database_provider/database_provider.h"
#include "depot_manager/depot_manager.h"
#include "order_controller/order_controller.h"
#include "account_manager/account_manager.h"
#include "bank_controller/bank_controller.h"

namespace Services
{
    public ref class MyServiceProvider sealed : Manager::ServiceProvider
    {
    public:
        explicit MyServiceProvider(Manager::ServiceBase^ service, List<Type^>^
dependencies)
            : Manager::ServiceProvider(service, dependencies) { }
        virtual ~MyServiceProvider(System::Void) { }
    };
}

```

```

}

#endif // !SERVICES_INCLUDE_STUFF

```

Файл «/services/services_settings.xml»

```

<?xml version="1.0" encoding="utf-8"?>
<service_parameters_list>
    <service_parameters service_name="database_provider">
        <sql_connection_server>localhost</sql_connection_server>
        <sql_connection_user>root</sql_connection_user>
        <sql_connection_database>test</sql_connection_database>
        <sql_connection_password>password</sql_connection_password>
    </service_parameters>

    <service_parameters service_name="depot_manager">
        <garage_collection_size>100</garage_collection_size>
        <car_child_type_price>300</car_child_type_price>
        <car_premium_type_price>500</car_premium_type_price>
        <car_econom_type_price>200</car_econom_type_price>
    </service_parameters>

    <service_parameters service_name="order_controller">
        <order_request_wait_second>20</order_request_wait_second>
    </service_parameters>
</service_parameters_list>

```

Файл «/views/admin_page_view/admin_details_view.h»

```

#pragma once
#include "../models/account_model/account_model.h"
#include "../services/account_manager/account_manager_token.h"

namespace Views
{
    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Сводка для MyForm
    /// </summary>
    public ref class AdminDetailsView: public System::Windows::Forms::Form
    {
        Models::AccountBaseModel^ model = nullptr;
        Services::AccountManagerToken::AccountManagerType type;
        System::Guid guid = System::Guid::Empty;

    public:
        AdminDetailsView(System::Void) { InitializeComponent(); }

        AdminDetailsView(Models::AccountBaseModel^ model, System::Guid guid,
                        Services::AccountManagerToken::AccountManagerType type) :
guid(guid), type(type)
        {
            InitializeComponent();
            this->Icon = gcnew
System::Drawing::Icon(L"./assets/my_app_icon.ico");
            this->model = (Models::AccountBaseModel^)model->Clone();
            this->account_data_load();
        }
    };
}

```

```

    }

protected:
    /// <summary>
    /// Освободить все используемые ресурсы.
    /// </summary>
    ~AdminDetailsView(System::Void) { if (components) delete components; }
private: System::Windows::Forms::ListView^ account_listview;
private: System::Windows::Forms::ColumnHeader^ param_name;
private: System::Windows::Forms::ColumnHeader^ param_value;
protected:

protected:

private:
    /// <summary>
    /// Обязательная переменная конструктора.
    /// </summary>
    System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Требуемый метод для поддержки конструктора – не изменяйте
    /// содержимое этого метода с помощью редактора кода.
    /// </summary>
    void InitializeComponent(void);
#pragma endregion

private: System::Void account_data_load(System::Void)
{
    this->account_listview->FullRowSelect = true;
    this->account_listview->Items->Clear();

    ListViewItem^ list_item = gcnew ListViewItem("Guid");
    list_item->SubItems->Add(this->guid.ToString());
    this->account_listview->Items->Add(list_item);

    list_item = gcnew ListViewItem("Имя");
    list_item->SubItems->Add(this->model->Username);
    this->account_listview->Items->Add(list_item);

    list_item = gcnew ListViewItem("Возраст");
    list_item->SubItems->Add(this->model->Age.ToString());
    this->account_listview->Items->Add(list_item);

    list_item = gcnew ListViewItem("Пол");
    list_item->SubItems->Add(this->model->Gender.ToString());
    this->account_listview->Items->Add(list_item);

    list_item = gcnew ListViewItem("Тип аккаунта");
    list_item->SubItems->Add(this->type.ToString());
    this->account_listview->Items->Add(list_item);

    list_item = gcnew ListViewItem("Банковский счет");
    Models::AccountDriverModel^ full_model = nullptr;

    switch (this->type)
    {
        case Services::AccountManagerToken::AccountManagerType::Client:

            Models::AccountClientModel^ full_client_model;

            try { full_client_model =
safe_cast<Models::AccountClientModel^>(model); }

```

```

        catch (System::Exception^) { MessageBox::Show("Не удалось
загрузить банковский счет", "Ошибка"); return; }

    list_item->SubItems->Add(full_client_model-
>BankCard.ToString()); break;
    case Services::AccountManagerToken::AccountManagerType::Driver:::

        Models::AccountDriverModel^ full_driver_model;

        try { full_driver_model =
safe_cast<Models::AccountDriverModel^>(model); }
        catch (System::Exception^) { MessageBox::Show("Не удалось
загрузить банковский счет", "Ошибка"); return; }

    list_item->SubItems->Add(full_driver_model-
>BankCard.ToString()); break;
}
this->account_listview->Items->Add(list_item);
}
};

}

}

```

Файл «/views/admin_page_view/admin_page_view.h»

```

#pragma once
#include "../manager/manager.h"
#include "../services/services.h"
#include "admin_details_view.h"

namespace Views
{

using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;

using namespace Models;
using namespace Services;

/// <summary>
/// Сводка для AdminPageView
/// </summary>
public ref class AdminPageView : public System::Windows::Forms::Form
{
    Manager::ServiceManager^ service_manager = nullptr;
    Windows::Forms::Form^ start_page = nullptr;

    Services::AccountManager^ service_account_manager = nullptr;
    Services::DepotManager^ service_depot_manager = nullptr;
private: System::Windows::Forms::ColumnHeader^ garage_car_groupguid;
private: System::Windows::Forms::Label^ admin_label_carspeed;
private: System::Windows::Forms::NumericUpDown^ admin_numeric_carspeed;
    Services::OrderController^ service_order_controller = nullptr;
public:
    AdminPageView(System::Void) { InitializeComponent(); }

    AdminPageView(Windows::Forms::Form^ start_page, Manager::ServiceManager^
service_manager)
    {
        InitializeComponent();

```

```

        this->Icon = gcnew
System::Drawing::Icon(L"./assets/my_app_icon.ico");

        this->admin_page_account->BackgroundImage =
System::Drawing::Image::FromFile("./assets/background.png");
        this->admin_page_account->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;

        this->admin_page_clients->BackgroundImage =
System::Drawing::Image::FromFile("./assets/background.png");
        this->admin_page_clients->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;

        this->admin_page_garage->BackgroundImage =
System::Drawing::Image::FromFile("./assets/background.png");
        this->admin_page_garage->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;

        this->service_manager = service_manager;
        this->start_page = start_page;

        Manager::IServiceProvider^ provider_account_manager = this-
>service_manager->get_service<AccountManager^>();
        Manager::IServiceProvider^ provider_depot_manager = this-
>service_manager->get_service<DepotManager^>();
        Manager::IServiceProvider^ provider_order_controller = this-
>service_manager->get_service<OrderController^>();

        this->service_account_manager =
(Services::AccountManager^)provider_account_manager->Service;
        this->service_depot_manager =
(Services::DepotManager^)provider_depot_manager->Service;
        this->service_order_controller =
(Services::OrderController^)provider_order_controller->Service;
        this->set_tabs_limitation();
        this->account_list_initialize();
    }

protected:
    /// <summary>
    /// Освободить все используемые ресурсы.
    /// </summary>
~AdminPageView(System::Void)
{
    if (components) delete components;
    if (this->service_account_manager->IsInitialized) this-
>service_account_manager->sign_out_account();
}

private: System::Windows::Forms::TabControl^ admin_tabcontrol;
private: System::Windows::Forms::TabPage^ admin_page_garage;
private: System::Windows::Forms::TabPage^ admin_page_account;
private: System::Windows::Forms::CheckBox^ admin_checkbox_age;
private: System::Windows::Forms::CheckBox^ admin_checkbox_gender;
private: System::Windows::Forms::CheckBox^ admin_checkbox_username;

private: System::Windows::Forms::Label^ admin_label_account;
private: System::Windows::Forms::NumericUpDown^ admin_numeric_age;
private: System::Windows::Forms::ComboBox^ admin_combobox_gender;
private: System::Windows::Forms::TextBox^ admin_textbox_username;
private: System::Windows::Forms::Button^ admin_button_update;
private: System::Windows::Forms::Button^ admin_button_logout;
private: System::Windows::Forms::ListView^ admin_listview_account;

private: System::Windows::Forms::ColumnHeader^ account_field;

```

```

private: System::Windows::Forms::ColumnHeader^ account_data;
private: System::Windows::Forms::TabPage^ admin_page_clients;
private: System::Windows::Forms::ListView^ admin_listview_carlist;
private: System::Windows::Forms::ColumnHeader^ garage_car_class;
private: System::Windows::Forms::ColumnHeader^ garage_car_type;
private: System::Windows::Forms::ColumnHeader^ garage_car_color;
private: System::Windows::Forms::ColumnHeader^ garage_car_speed;
private: System::Windows::Forms::ColumnHeader^ garage_car_count;
private: System::Windows::Forms::Button^ admin_button_cardelete;
private: System::Windows::Forms::Button^ admin_button_caradd;
private: System::Windows::Forms::Button^ admin_button_upd;

private: System::Windows::Forms::ComboBox^ admin_combobox_carclass;
private: System::Windows::Forms::ComboBox^ admin_combobox_carcolor;
private: System::Windows::Forms::ComboBox^ admin_combobox_cartype;
private: System::Windows::Forms::Label^ admin_label_carcount;
private: System::Windows::Forms::Label^ admin_label_carclass;
private: System::Windows::Forms::Label^ admin_label_carcolor;
private: System::Windows::Forms::Label^ admin_label_cartype;

private: System::Windows::Forms::Label^ admin_label_carlist;
private: System::Windows::Forms::NumericUpDown^ admin_numeric_carcount;
private: System::Windows::Forms::Label^ admin_label_accounts;
private: System::Windows::Forms::Button^ admin_button_refresh;
private: System::Windows::Forms::Button^ admin_button_delete;

private: System::Windows::Forms::ListView^ admin_listview_accounts;
private: System::Windows::Forms::Button^ admin_button_accountinfo;
private: System::Windows::Forms::ColumnHeader^ account_guid;
private: System::Windows::Forms::ColumnHeader^ account_type;
private: System::Windows::Forms::ColumnHeader^ account_state;

private:
    /// <summary>
    /// Обязательная переменная конструктора.
    /// </summary>
    System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Требуемый метод для поддержки конструктора – не изменяйте
    /// содержимое этого метода с помощью редактора кода.
    /// </summary>
    void InitializeComponent(void);
#pragma endregion

    private: System::Void set_tabs_limitation(System::Void);

    private: generic <class TEnum> TEnum convert_to_enum(System::String^
value);

    private: System::Void account_list_initialize(System::Void);

    private: System::Void admin_button_caradd_Click(System::Object^ sender,
System::EventArgs^ e);

    private: System::Void admin_button_cardelete_Click(System::Object^ sender,
System::EventArgs^ e);

    private: System::Void admin_button_upd_Click(System::Object^ sender,
System::EventArgs^ e);

    private: System::Void admin_button_accountinfo_Click(System::Object^
sender, System::EventArgs^ e);

```

```

    private: System::Void admin_button_update_Click(System::Object^ sender,
System::EventArgs^ e);

    private: System::Void admin_button_logout_Click(System::Object^ sender,
System::EventArgs^ e);

    private: System::Void admin_button_refresh_Click(System::Object^ sender,
System::EventArgs^ e);

    private: System::Void admin_button_delete_Click(System::Object^ sender,
System::EventArgs^ e);
};

}

```

Файл «/views/admin_page_view/admin_page_view_callbacks.h»

```

#include "admin_page_view.h"

using namespace Views;

generic <class TEnum> TEnum AdminPageView::convert_to_enum(System::String^ value)
{
    return safe_cast<TEnum>(System::Enum::Parse(TEnum::typeid, value, true));
}

System::Void AdminPageView::set_tabs_limitation(System::Void)
{
    Models::AccountAdminModel^ admin_model = nullptr;
    try { admin_model = safe_cast<Models::AccountAdminModel^>(this-
>service_account_manager->AccountToken.AccountModel); }
    catch (System::Exception^) { MessageBox::Show("Невозможно установить
ограничения", "Ошибка"); }

    this->admin_page_garage->Enabled = false;
    this->admin_page_clients->Enabled = false;

    if (admin_model->Permissions == Models::AccountModelPermissions::AccountList
        || admin_model->Permissions ==
Models::AccountModelPermissions::FullPermission)
        this->admin_page_clients->Enabled = true;

    if (admin_model->Permissions == Models::AccountModelPermissions::FullPermission
        || admin_model->Permissions == Models::AccountModelPermissions::CarGarage)
        this->admin_page_garage->Enabled = true;
}

System::Void AdminPageView::admin_button_upd_Click(System::Object^ sender,
System::EventArgs^ e)
{
    List<DepotManager::CarGarageItems>^ car_list = this->service_depot_manager-
>get_all_cars();
    this->admin_listview_carlist->FullRowSelect = true;
    this->admin_listview_carlist->Items->Clear();

    if (car_list == nullptr || car_list->Count <= 0)
    { MessageBox::Show("Невозможно получить список машин", "Ошибка"); return; }

    for each (auto item in car_list)
    {
        ListViewItem^ list_item = gcnew ListViewItem(item.Guid.ToString());
        list_item->SubItems->Add(item.CarClass->Name);
        list_item->SubItems->Add(item.CarModel->CarType.ToString());
        list_item->SubItems->Add(item.CarModel->CarColor.ToString());
    }
}

```

```

        list_item->SubItems->Add(item.CarModel->CarSpeed.ToString());
        list_item->SubItems->Add(item.CarCount.ToString());
        this->admin_listview_carlist->Items->Add(list_item);
    }
}

System::Void AdminPageView::admin_button_caradd_Click(System::Object^ sender,
System::EventArgs^ e)
{
    System::Int32 car_count = System::Decimal::ToInt32(this->admin_numeric_carcount-
>Value);
    System::Int32 car_speed = System::Decimal::ToInt32(this->admin_numeric_carspeed-
>Value);

    Models::CarModelTypes car_type;
    switch(this->admin_combobox_cartype->SelectedIndex)
    {
        case 0: car_type = Models::CarModelTypes::CarTypeEconom; break;
        case 1: car_type = Models::CarModelTypes::CarTypeChild; break;
        case 2: car_type = Models::CarModelTypes::CarTypePremium; break;
    }

    Models::CarModelColor car_color;
    switch (this->admin_combobox_carcolor->SelectedIndex)
    {
        case 0: car_color = Models::CarModelColor::CarColorBlack; break;
        case 1: car_color = Models::CarModelColor::CarColorWhite; break;
        case 2: car_color = Models::CarModelColor::CarColorRed; break;
        case 3: car_color = Models::CarModelColor::CarColorYellow; break;
    }

    System::Boolean car_add_check(true);
    switch (this->admin_combobox_carclass->SelectedIndex)
    {
        case 0:;
            Models::CarLightModel^ light_model;
            light_model = gcnew Models::CarLightModel(car_type, car_color, car_speed);
            car_add_check = this->service_depot_manager-
>add_car_model<Models::CarLightModel^>(light_model, car_count);
            break;
        case 1:;
            Models::CarHeavyModel^ heavy_model;
            heavy_model = gcnew Models::CarHeavyModel(car_type, car_color, car_speed);
            car_add_check = this->service_depot_manager-
>add_car_model<Models::CarHeavyModel^>(heavy_model, car_count);
            break;
    }
    if (car_add_check != true) { MessageBox::Show("Невозможно добавить машину(ы)", "Ошибка"); return; }
    this->admin_button_upd_Click(sender, e);
    MessageBox::Show("Машина(ы) успешно добавлены", "Готово"); return;
}

System::Void AdminPageView::admin_button_cardelete_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (this->admin_listview_carlist->SelectedItems->Count > 0)
    {
        System::Guid carmodel_guid;
        try { carmodel_guid = System::Guid::Parse(this->admin_listview_carlist-
>SelectedItems[0]->Text); }
        catch (System::Exception^ error) { MessageBox::Show("Данные повреждены", "Ошибка"); return; }
    }
}

```

```

        if (!this->service_depot_manager->delete_car_model_by_guid(carmodel_guid))
        { MessageBox::Show("Невозможно удалить машину", "Ошибка"); return; }

        this->admin_button_upd_Click(sender, e);
        MessageBox::Show("Машина успешно удалена", "Готово");
    }
    else MessageBox::Show("Выберите строку с машиной", "Ошибка");
}

System::Void AdminPageView::admin_button_delete_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (this->admin_listview_accounts->SelectedItems->Count > 0)
    {
        System::Guid account_guid;
        Services::AccountManagerToken::AccountManagerType account_type_new;
        try {
            account_guid = System::Guid::Parse(this->admin_listview_accounts-
>SelectedItems[0]->Text);
            account_type_new = this-
>convert_to_enum<Services::AccountManagerToken::AccountManagerType>(
                this->admin_listview_accounts->SelectedItems[0]->SubItems[1]->Text);
        }
        catch (System::Exception^) { MessageBox::Show("Данные повреждены",
"Ошибка"); return; }

        System::Nullable<System::Boolean> check = this->service_account_manager-
>get_account_status(account_guid);
        if (check.HasValue != true) { MessageBox::Show("Данные повреждены",
"Ошибка"); return; }
        if (check.Value == true) { MessageBox::Show("Аккаунт активен –
недоступно", "Ошибка"); return; }

        if (!this->service_account_manager->delete_account(account_guid))
        { MessageBox::Show("Невозможно удалить аккаунт", "Ошибка"); return; }

        MessageBox::Show("Аккаунт успешно удален", "Готово");
    }
    else MessageBox::Show("Выберите строку с аккаунтом", "Ошибка");
}

System::Void AdminPageView::admin_button_refresh_Click(System::Object^ sender,
System::EventArgs^ e)
{
    Generic::List<AccountManager::AccountInfo>^ account_list = this-
>service_account_manager->AccountList;
    if (account_list == nullptr || account_list->Count <= 0)
    { MessageBox::Show("Возникла ошибка при обновлении", "Уведомление"); return; }

    this->admin_listview_accounts->Items->Clear();
    this->admin_listview_accounts->FullRowSelect = true;

    for each (auto item in account_list)
    {
        if (item.Type == Services::AccountManagerToken::AccountManagerType::Admin)
        continue;

        ListViewItem^ list_item = gcnew ListViewItem(item.Guid.ToString());
        list_item->SubItems->Add(item.Type.ToString());
        list_item->SubItems->Add(item.State.ToString());
        admin_listview_accounts->Items->Add(list_item);
    }

    MessageBox::Show("Данные обновлены", "Уведомление");
}

```

```

}

System::Void AdminPageView::admin_button_accountinfo_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (this->admin_listview_accounts->SelectedItems->Count > 0)
    {
        System::Guid account_guid;
        Services::AccountManagerToken::AccountManagerType account_type_new;
        try {
            account_guid = System::Guid::Parse(this->admin_listview_accounts-
>SelectedItems[0]->Text);
            account_type_new = this-
>convert_to_enum<Services::AccountManagerToken::AccountManagerType>(
                this->admin_listview_accounts->SelectedItems[0]->SubItems[1]->Text);
        }
        catch (System::Exception^ error) { MessageBox::Show("Данные повреждены",
"Ошибка"); return; }

        Models::AccountBaseModel^ model = nullptr;
        switch (account_type_new)
        {
            case Services::AccountManagerToken::AccountManagerType::Client:::
                Services::AccountClientsDbScheme^ client_scheme;
                client_scheme = this->service_account_manager-
>get_account_scheme<Services::AccountClientsDbScheme^>(account_guid);
                if (client_scheme == nullptr) { MessageBox::Show("Не удалось
сформировать модель", "Ошибка"); return; }

                model =
Services::AccountClientsDbScheme::cast_to_model(client_scheme); break;

            case Services::AccountManagerToken::AccountManagerType::Driver:::
                Services::AccountDriversDbScheme^ driver_scheme;
                driver_scheme = this->service_account_manager-
>get_account_scheme<Services::AccountDriversDbScheme^>(account_guid);
                if (driver_scheme == nullptr) { MessageBox::Show("Не удалось
сформировать модель", "Ошибка"); return; }

                model =
Services::AccountDriversDbScheme::cast_to_model(driver_scheme); break;
        }

        Windows::Forms::Form^ form = gcnew Views::AdminDetailsView(model,
account_guid, account_type_new);
        form->ShowDialog();
    }
    else MessageBox::Show("Выберите строку с аккаунтом", "Ошибка");
}

System::Void AdminPageView::account_list_initialize(System::Void)
{
    this->admin_listview_account->FullRowSelect = true;
    this->admin_listview_account->Items->Clear();

    ListViewItem^ list_item = gcnew ListViewItem("Тип аккаунта");
    list_item->SubItems->Add(this->service_account_manager-
>AccountToken.AccountType.ToString());
    this->admin_listview_account->Items->Add(list_item);

    list_item = gcnew ListViewItem("Аккаунт Guid");
}

```

```

list_item->SubItems->Add(this->service_account_manager-
>AccountToken.AccountGuid.ToString());
this->admin_listview_account->Items->Add(list_item);

Models::AccountAdminModel^ account_model = nullptr;
try {
    account_model = safe_cast<Models::AccountAdminModel^>(this-
>service_account_manager
    ->AccountToken.AccountModel);
}
catch (System::Exception^) { MessageBox::Show("Невозможно определить модель
аккаунта", "Ошибка"); return; }

list_item = gcnew ListViewItem("Возраст");
list_item->SubItems->Add(account_model->Age.ToString());
this->admin_listview_account->Items->Add(list_item);

list_item = gcnew ListViewItem("Пол");
list_item->SubItems->Add(account_model->Gender.ToString());
this->admin_listview_account->Items->Add(list_item);

list_item = gcnew ListViewItem("Имя");
list_item->SubItems->Add(account_model->Username);
this->admin_listview_account->Items->Add(list_item);
}

System::Void AdminPageView::admin_button_update_Click(System::Object^ sender,
System::EventArgs^ e)
{
    Models::AccountAdminModel^ current_model =
safe_cast<Models::AccountAdminModel^>(
    this->service_account_manager->AccountToken.AccountModel);

    System::Int32 age_field = this->admin_checkbox_age->Checked ?
Decimal::ToInt32(this->admin_numeric_age->Value)
    : current_model->Age;

    System::String^ username_field = this->admin_checkbox_username->Checked ?
    this->admin_textbox_username->Text : current_model->Username;
    if (username_field == System::String::Empty) { MessageBox::Show("Неверный формат
имени", "Ошибка"); return; }

    Models::AccountModelGender gender_field;
    if (this->admin_checkbox_gender->Checked)
    {
        switch (this->admin_combobox_gender->SelectedIndex)
        {
            case 0: gender_field = Models::AccountModelGender::MaleGender; break;
            case 1: gender_field = Models::AccountModelGender::FemaleGender; break;
        }
    }
    else gender_field = current_model->Gender;

    Models::AccountAdminModel^ model = gcnew Models::AccountAdminModel(
        username_field, age_field, gender_field, current_model->Permissions);

    System::Boolean update_check = this->service_account_manager-
>update_account(model);
    if (update_check != true) { MessageBox::Show("Не удалось обновить данные
аккаунта", "Ошибка"); return; }

    MessageBox::Show("Данные успешно обновлены", "Готово");
    this->account_list_initialize();
}

```

```

System::Void AdminPageView::admin_button_logout_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (MessageBox::Show("Вы уверены?", "Подтверждение", MessageBoxButtons::YesNo,
MessageBoxIcon::Question)
    == ::DialogResult::Yes)
    {
        System::Boolean logout_check = this->service_account_manager-
>sign_out_account();
        this->Close();
    }
}

```

Файл «/views/authorization_view/authorization_view.h»

```

#pragma once
#include "../manager/manager.h"
#include "../services/services.h"

#include "../bank_settings_view/bank_settings_view.h"
#include "../admin_page_view/admin_page_view.h"
#include "../driver_page_view/driver_page_view.h"
#include "../client_page_view/client_page_view.h"

namespace Views
{
    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    using namespace Services;
    using namespace Models;
    /// <summary>
    /// Сводка для AuthorizationView
    /// </summary>
public ref class AuthorizationView : public System::Windows::Forms::Form
{
    Manager::ServiceManager^ service_manager = nullptr;

    Services::BankController^ bank_controller = nullptr;
private: System::Windows::Forms::Label^ page1_label_mainlogo;
private: System::Windows::Forms::Label^ page1_label_suplogo1;

private: System::Windows::Forms::PictureBox^ page1_picturebox_imagelogo;
private: System::Windows::Forms::Label^ page1_label_suplogo2;
private: System::Windows::Forms::Label^ page1_label_suplogo3;

    Services::AccountManager^ account_manager = nullptr;
public:
    AuthorizationView(System::Void) { InitializeComponent(); }

    AuthorizationView(Manager::ServiceManager^ service_manager)
    {
        InitializeComponent();
        this->Icon = gcnew
System::Drawing::Icon(L"./assets/my_app_icon.ico");
        this->SetStyle(ControlStyles::AllPaintingInWmPaint |
ControlStyles::UserPaint | ControlStyles::DoubleBuffer, true);

        this->page1_picturebox_imagelogo->Image =
System::Drawing::Image::FromFile("./assets/my_app_logo.png");
    }
}

```

```

        this->page1_picturebox_imagelogo->SizeMode =
PictureBoxSizeMode::Zoom;

        this->page_authorization->BackgroundImage =
System::Drawing::Image::FromFile("./assets/background2.png");
        this->page_authorization->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Tile;

        this->page_registration->BackgroundImage =
System::Drawing::Image::FromFile("./assets/background2.png");
        this->page_registration->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Tile;

        this->service_manager = service_manager;
        Manager::IServiceProvider^ bank_controller_provider = this-
>service_manager->get_service<Services::BankController^>();
        Manager::IServiceProvider^ account_manager_provider = this-
>service_manager->get_service<Services::AccountManager^>();

        this->account_manager =
(Services::AccountManager^)account_manager_provider->Service;
        this->bank_controller =
(Services::BankController^)bank_controller_provider->Service;
    }

protected:
    /// <summary>
    /// Освободить все используемые ресурсы.
    /// </summary>
    ~AuthorizationView(System::Void)
    {
        if (components) delete components;
        if(this->account_manager->IsInitialized) this->account_manager-
>sign_out_account();
        Application::Exit();
    }

private: System::Windows::Forms::TabControl^ page_list;
private: System::Windows::Forms::TabPage^ page_authorization;
private: System::Windows::Forms::TabPage^ page_registration;
private: System::Windows::Forms::Label^ page1_label_login;
private: System::Windows::Forms::TextBox^ page1_textbox_login;
private: System::Windows::Forms::Button^ page1_button_authorize;
private: System::Windows::Forms::Label^ page1_label_password;
private: System::Windows::Forms::TextBox^ page1_textbox_password;
private: System::Windows::Forms::Label^ page2_label_login;

private: System::Windows::Forms::TextBox^ page2_textbox_login;
private: System::Windows::Forms::Label^ page2_label_password;
private: System::Windows::Forms::TextBox^ page2_textbox_password;
private: System::Windows::Forms::Label^ page2_label_username;
private: System::Windows::Forms::TextBox^ page2_textbox_username;
private: System::Windows::Forms::TabControl^ page2_tabcontrol_info;
private: System::Windows::Forms::TabPage^ page2_tabpage_client;
private: System::Windows::Forms::TabPage^ page2_tabpage_driver;

private: System::Windows::Forms::Label^ page2_label_age;
private: System::Windows::Forms::NumericUpDown^ page2_numeric_age;
private: System::Windows::Forms::Label^ page2_label_gender;
private: System::Windows::Forms::ComboBox^ page2_combobox_gender;
private: System::Windows::Forms::Button^ page2_button_registration;
private: System::Windows::Forms::TabPage^ page2_tabpage_admin;
private: System::Windows::Forms::Label^ pageclient_label_bankcard;
private: System::Windows::Forms::TextBox^ pageclient_textbox_bankcard;
private: System::Windows::Forms::Button^ page2_button_banksettings;

```

```

private: System::Windows::Forms::Label^ pagedriver_label_bankcard;
private: System::Windows::Forms::TextBox^ pagedriver_textbox_bankcard;
private: System::Windows::Forms::Label^ pagedriver_label_licence;
private: System::Windows::Forms::TextBox^ pagedriver_textbox_licence;
private: System::Windows::Forms::Label^ pageadmin_label_permissions;
private: System::Windows::Forms::ComboBox^ pageadmin_combobox_permissions;

private:
    /// <summary>
    /// Обязательная переменная конструктора.
    /// </summary>
    System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Требуемый метод для поддержки конструктора – не изменяйте
    /// содержимое этого метода с помощью редактора кода.
    /// </summary>
    void InitializeComponent(void);
#pragma endregion

private: System::Void clear_all_textboxes(System::Void);
private: System::Void page1_button_authorize_Click(System::Object^ sender,
System::EventArgs^ e);

private: System::Void page2_button_registration_Click(System::Object^
sender, System::EventArgs^ e);

private: System::Void page2_button_banksettings_Click(System::Object^
sender, System::EventArgs^ e);

private: System::Void form_closed(System::Object^ sender,
FormClosedEventArgs^ e);
private: System::Void page1_label_suplogo1_Click(System::Object^ sender,
System::EventArgs^ e) {
}
};

}

```

Файл «/views/authorization_view/authorization_view_callbacks.cpp»

```

#include "authorization_view.h"

using namespace Views;

System::Void AuthorizationView::page1_button_authorize_Click(System::Object^ sender,
System::EventArgs^ e)
{
    System::String^ login_text = this->page1_textbox_login->Text;
    System::String^ password_text = this->page1_textbox_password->Text;

    System::Boolean login_check = account_manager->authorization_account(login_text,
password_text);
    if (login_check != true) { MessageBox::Show("Невозможно войти в аккаунт",
"Ошибка входа"); return; }
    this->clear_all_textboxes();

    Windows::Forms::Form^ view_preparation = nullptr;
    switch (this->account_manager->AccountToken.AccountType)
    {
        case AccountManagerToken::AccountManagerType::Admin:

```

```

        view_preparation = gcnew Views::AdminPageView(this, this-
>service_manager); break;
    case AccountManagerToken::AccountManagerType::Client:
        view_preparation = gcnew Views::ClientPageView(this, this-
>service_manager); break;
    case AccountManagerToken::AccountManagerType::Driver:
        view_preparation = gcnew Views::DriverPageView(this, this-
>service_manager); break;
    }
    view_preparation->FormClosed += gcnew FormClosedEventHandler(this,
&AuthorizationView::form_closed);
    this->Hide();      view_preparation->Show();
}

System::Void AuthorizationView::clear_all_textboxes(System::Void)
{
    for (int i = 0; i < this->page_list->Controls->Count; i++)
    {
        Control^ check_list = this->page_list->Controls[i];
        for each (Control ^ item in check_list->Controls)
        {
            try { TextBox^ textbox = safe_cast<TextBox^>(item); textbox-
>Clear(); }
            catch (System::Exception^ error) { continue; }
        }
    }
}

System::Void AuthorizationView::page2_button_banksettings_Click(System::Object^ sender,
System::EventArgs^ e)
{
    Views::BankSettingsView^ bank_page = gcnew
Views::BankSettingsView(bank_controller);
    bank_page->>ShowDialog();
}

System::Void AuthorizationView::form_closed(System::Object^ sender,
FormClosedEventArgs^ e) { this->Show(); }

System::Void AuthorizationView::page2_button_registration_Click(System::Object^ sender,
System::EventArgs^ e)
{
    System::String^ login_field = this->page2_textbox_login->Text;
    System::String^ password_field = this->page2_textbox_password->Text;
    System::String^ username_field = this->page2_textbox_username->Text;

    if (login_field->Length < 4 || password_field->Length < 4 || username_field-
>Length < 8)
        { MessageBox::Show("Неверно заполнены текстовые поля", "Ошибка"); return; }

    System::UInt32 age_field = System::Decimal::ToUInt32(this->page2_numeric_age-
>Value);
    Models::AccountModelGender gender_field;

    if (this->page2_combobox_gender->SelectedIndex == 0) gender_field =
Models::AccountModelGender::MaleGender;
    else gender_field = Models::AccountModelGender::FemaleGender;

    System::Boolean registration_check(true);
    Windows::Forms::Form^ view_preparation = nullptr;

    switch (this->page2_tabcontrol_info->SelectedIndex)
    {
    case 0: {
        System::Guid bank_card_number = System::Guid::Empty;

```

```

        try { bank_card_number = System::Guid::Parse(this-
>pageclient_textbox_bankcard->Text); }
        catch (System::Exception^ error) { MessageBox::Show("Номер банковской
карты невалиден", "Ошибка заполнения"); return; }

        if (!this->bank_controller->load_bank_account(bank_card_number))
        { MessageBox::Show("Банковский аккаунт не найден", "Ошибка"); return; }

        Models::AccountClientModel^ model = gcnew Models::AccountClientModel(
            username_field, age_field, gender_field, bank_card_number);

        registration_check = this->account_manager-
>registration_account<Models::AccountClientModel^>(
            login_field, password_field, model);
        if (registration_check) view_preparation = gcnew
Views::ClientPageView(this, this->service_manager);
    } break;
    case 1: {
        System::Guid bank_card_number = System::Guid::Empty;
        System::Guid licence_number = System::Guid::Empty;
        try {
            bank_card_number = System::Guid::Parse(this-
>pagedriver_textbox_bankcard->Text);
            licence_number = System::Guid::Parse(this-
>pagedriver_textbox_licence->Text);
        }
        catch (System::Exception^ error) {
            MessageBox::Show("Номер банковской карты или лицензии невалиден",
"Ошибка заполнения"); return;
        }

        if (!this->bank_controller->load_bank_account(bank_card_number))
        { MessageBox::Show("Банковский аккаунт не найден", "Ошибка"); return; }

        Models::AccountDriverModel^ model = gcnew Models::AccountDriverModel(
            username_field, age_field, gender_field, licence_number,
            bank_card_number);

        registration_check = this->account_manager-
>registration_account<Models::AccountDriverModel^>(
            login_field, password_field, model);
        if (registration_check) view_preparation = gcnew
Views::DriverPageView(this, this->service_manager);
    } break;
    case 2: {
        Models::AccountModelPermissions model_permission;
        switch (pageadmin_combobox_permissions->SelectedIndex)
        {
            case 0: model_permission =
Models::AccountModelPermissions::FullPermission; break;
            case 1: model_permission = Models::AccountModelPermissions::CarGarage;
break;
            case 2: model_permission = Models::AccountModelPermissions::AccountList;
break;
        }
        Models::AccountAdminModel^ model = gcnew Models::AccountAdminModel(
            username_field, age_field, gender_field, model_permission);

        registration_check = this->account_manager-
>registration_account<Models::AccountAdminModel^>(
            login_field, password_field, model);
        if (registration_check) view_preparation = gcnew
Views::AdminPageView(this, this->service_manager);
    } break;
}

```

```

    if (registration_check != true) { MessageBox::Show("Невозможно
зарегистрироваться", "Ошибка"); return; }

    this->clear_all_textboxes();
    view_preparation->FormClosed += gcnew FormClosedEventHandler(this,
&AuthorizationView::form_closed);
    this->Hide();      view_preparation->Show();
}

```

Файл «/views/bank_settings_view/bank_settings_operation.h»

```

#pragma once
#include "../manager/manager.h"
#include "../services/services.h"

namespace Views
{

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Сводка для MyForm
    /// </summary>
public ref class BankSettingsOperationView : public System::Windows::Forms::Form
{
    Services::BankController^ bank_controller = nullptr;
public:
    BankSettingsOperationView(System::Void) { InitializeComponent(); }

    BankSettingsOperationView(Services::BankController^ bank_controller)
    {
        InitializeComponent();
        this->Icon = gcnew
System::Drawing::Icon(L"./assets/my_app_icon.ico");
        this->bank_controller = bank_controller;
    }

protected:
    /// <summary>
    /// Освободить все используемые ресурсы.
    /// </summary>
    ~BankSettingsOperationView(System::Void) { if (components) delete
components; }
    private: System::Windows::Forms::ComboBox^ combobox_operation_type;
    private: System::Windows::Forms::NumericUpDown^ numeric_money_value;
    private: System::Windows::Forms::Button^ button_operation;
protected:

protected:

private: System::Windows::Forms::Label^ label_operation_type;
private: System::Windows::Forms::Label^ label_money_value;

private:
    /// <summary>

```

```

/// Обязательная переменная конструктора.
/// </summary>
System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Требуемый метод для поддержки конструктора – не изменяйте
    /// содержимое этого метода с помощью редактора кода.
    /// </summary>
    void InitializeComponent(void);
#pragma endregion
    private: System::Void button_operation_Click(System::Object^ sender,
System::EventArgs^ e)
    {
        System::Int32 money = System::Decimal::ToInt32(this-
>numeric_money_value->Value);
        System::Boolean operation_check;

        if (this->combobox_operation_type->SelectedIndex == 0)
operation_check = this->bank_controller->take_money(money);
        else operation_check = this->bank_controller->put_money(money);

        if (operation_check != true) { MessageBox::Show("Невозможно
свершить операцию", "Ошибка"); return; }
        MessageBox::Show("Операция совершена", "Готово");
        this->Close();
    }
}
}

```

Файл «/views/bank_settings_view/bank_settings_view.h»

```

#pragma once
#include "../manager/manager.h"
#include "../services/services.h"
#include "bank_settings_operation.h"

namespace Views
{

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Сводка для MyForm
    /// </summary>
public ref class BankSettingsView : public System::Windows::Forms::Form
{
    Services::BankController^ bank_controller = nullptr;
public:
    BankSettingsView(System::Void) { InitializeComponent(); }

    BankSettingsView(Services::BankController^ bank_controller)
    {
        InitializeComponent();
        this->BackgroundImage =
System::Drawing::Image::FromFile("./assets/background.png");
        this->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
    }
}

```

```

        this->Icon = gcnew
System::Drawing::Icon(L"./assets/my_app_icon.ico");
        this->bank_controller = bank_controller;
    }

protected:
    /// <summary>
    /// Освободить все используемые ресурсы.
    /// </summary>
~BankSettingsView(System::Void) { if (components) delete components; }

protected:

private: System::Windows::Forms::Button^ button_create_bank;
private: System::Windows::Forms::Button^ button_reload;
private: System::Windows::Forms::Label^ label_bank;
private: System::Windows::Forms::ListView^ listview_bank;
private: System::Windows::Forms::ColumnHeader^ column_guid;
private: System::Windows::Forms::ColumnHeader^ column_money;
private: System::Windows::Forms::Button^ button_operation;
private: System::Windows::Forms::Button^ button_copy;

private:
    /// <summary>
    /// Обязательная переменная конструктора.
    /// </summary>
System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Требуемый метод для поддержки конструктора – не изменяйте
    /// содержимое этого метода с помощью редактора кода.
    /// </summary>
    void InitializeComponent(void);
#pragma endregion
    private: System::Void button_create_bank_Click(System::Object^ sender,
System::EventArgs^ e)
    {
        if (!this->bank_controller-
>create_bank_account(System::Guid::.NewGuid()))
            MessageBox::Show("Возникла ошибка при создании счёта",
"Уведомление");
        else
            MessageBox::Show("Банковский счёт успешно создан", "Уведомление");
    }

    private: System::Void button_reload_Click(System::Object^ sender,
System::EventArgs^ e)
    {
        Generic::List<System::Guid>^ account_list = this->bank_controller-
>get_bank_accounts();
        if(account_list == nullptr) MessageBox::Show("Возникла ошибка при
обновлении", "Уведомление");

        this->listview_bank->Items->Clear();
        this->listview_bank->FullRowSelect = true;

        for each (System::Guid item in account_list)
        {
            if(!this->bank_controller->load_bank_account(item)) continue;
            ListViewItem^ list_item = gcnew ListViewItem(this-
>bank_controller->AccountGuid.ToString());
            list_item->SubItems->Add(this->bank_controller-
>AccountMoney.ToString());
            this->listview_bank->Items->Add(list_item);
        }
    }

```

```

        MessageBox::Show("Данные обновлены", "Уведомление");
    }

    private: System::Void button_operation_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (this->listview_bank->SelectedItems->Count > 0)
    {
        System::Guid account_guid;
        try { account_guid = System::Guid::Parse(this->listview_bank-
>SelectedItems[0]->Text); }
        catch (System::Exception^ error) { MessageBox::Show("Данные
повреждены", "Ошибка"); return; }

        if (!this->bank_controller->load_bank_account(account_guid))
        {
            MessageBox::Show("Не удалось загрузить данные
аккаунта", "Ошибка"); return;
        }

        Views::BankSettingsOperationView^ form = gcnew
Views::BankSettingsOperationView(this->bank_controller);
        form->>ShowDialog();
    }
    else MessageBox::Show("Выберите строку с аккаунтом", "Ошибка");
}

private: System::Void button_copy_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (this->listview_bank->SelectedItems->Count > 0)
    {
        Clipboard::SetText(this->listview_bank->SelectedItems[0]-
>Text);
        MessageBox::Show("Данные помещены в буфер обмена", "Успешно");
    }
    else MessageBox::Show("Выберите строку с аккаунтом", "Ошибка");
}
};
}

```

Файл «/views/client_page_view/client_page_view.h»

```

#pragma once
#include "../manager/manager.h"
#include "../services/services.h"
#include "../models/cars_model/cars_model.h"
#include "../models/account_model/account_model.h"
#include "../bank_settings_view/bank_settings_view.h"
#include "client_pricelist_view.h"

namespace Views
{
    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    using namespace System::Threading::Tasks;
    using namespace System::Threading;

    using namespace Models;

```

```

using namespace Services;
/// <summary>
/// Сводка для ClientPageView
/// </summary>
public ref class ClientPageView : public System::Windows::Forms::Form
{
    Manager::ServiceManager^ service_manager = nullptr;
    Windows::Forms::Form^ start_page = nullptr;
    System::Boolean order_proccess;

    Services::AccountManager^ service_account_manager = nullptr;
    Services::DepotManager^ service_depot_manager = nullptr;
    Services::OrderController^ service_order_controller = nullptr;
    Services::BankController^ service_bank_controller = nullptr;
public:
    ClientPageView(System::Void) { InitializeComponent(); }

    ClientPageView(Windows::Forms::Form^ start_page, Manager::ServiceManager^
service_manager) : order_proccess(false)
    {
        InitializeComponent();
        this->Icon = gcnew
System::Drawing::Icon(L"./assets/my_app_icon.ico");

        this->client_page_account->BackgroundImage =
System::Drawing::Image::FromFile("./assets/background.png");
        this->client_page_account->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;

        this->client_page_order->BackgroundImage =
System::Drawing::Image::FromFile("./assets/background.png");
        this->client_page_order->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;

        this->service_manager = service_manager;
        this->start_page = start_page;

        Manager::IServiceProvider^ provider_account_manager = this-
>service_manager->get_service<AccountManager^>();
        Manager::IServiceProvider^ provider_depot_manager = this-
>service_manager->get_service<DepotManager^>();
        Manager::IServiceProvider^ provider_order_controller = this-
>service_manager->get_service<OrderController^>();
        Manager::IServiceProvider^ provider_bank_controller = this-
>service_manager->get_service<BankController^>();

        this->service_account_manager =
(Services::AccountManager^)provider_account_manager->Service;
        this->service_depot_manager =
(Services::DepotManager^)provider_depot_manager->Service;
        this->service_order_controller =
(Services::OrderController^)provider_order_controller->Service;
        this->service_bank_controller =
(Services::BankController^)provider_bank_controller->Service;

        this->account_list_initialize();
    }

protected:
    /// <summary>
    /// Освободить все используемые ресурсы.
    /// </summary>
    ~ClientPageView(System::Void)
    {

```

```

        if (this->service_account_manager->IsInitialized) this-
>service_account_manager->sign_out_account();
        if (components) delete components;
    }

private: System::Windows::Forms::CheckBox^ client_checkbox_bankcard;
private: System::Windows::Forms::CheckBox^ client_checkbox_age;
private: System::Windows::Forms::CheckBox^ client_checkbox_gender;
private: System::Windows::Forms::CheckBox^ client_checkbox_username;

private: System::Windows::Forms::Button^ client_button_money;
private: System::Windows::Forms::Button^ client_button_price;
private: System::Windows::Forms::TabControl^ client_tabcontrol;
private: System::Windows::Forms::TabPage^ client_page_order;
private: System::Windows::Forms::Label^ client_label_carlist;
private: System::Windows::Forms::ComboBox^ client_combobox_cartype;
private: System::Windows::Forms::ProgressBar^ client_progressbar_waiting;
private: System::Windows::Forms::ListView^ client_listview_car;
private: System::Windows::Forms::TabPage^ client_page_account;

private: System::Windows::Forms::Button^ client_button_refresh;
private: System::Windows::Forms::ColumnHeader^ driver_state;
private: System::Windows::Forms::Button^ client_button_cancel;
private: System::Windows::Forms::Button^ client_button_bank;

private: System::Windows::Forms::TextBox^ client_textbox_address;
private: System::Windows::Forms::Button^ client_button_order;
private: System::Windows::Forms::Label^ client_label_cartype;

private: System::Windows::Forms::Label^ client_label_address;
private: System::Windows::Forms::Label^ client_label_carclass;
private: System::Windows::Forms::ComboBox^ client_combobox_carclass;
private: System::Windows::Forms::ColumnHeader^ driver_guid;

private: System::Windows::Forms::ColumnHeader^ car_class;
private: System::Windows::Forms::ColumnHeader^ car_type;
private: System::Windows::Forms::Label^ client_label_waiting;
private: System::Windows::Forms::TextBox^ client_textbox_username;
private: System::Windows::Forms::Button^ client_button_update;
private: System::Windows::Forms::Button^ client_button_logout;
private: System::Windows::Forms::ListView^ client_listview_account;

private: System::Windows::Forms::Label^ client_label_account;
private: System::Windows::Forms::NumericUpDown^ client_numeric_age;
private: System::Windows::Forms::TextBox^ client_textbox_bankcard;
private: System::Windows::Forms::ComboBox^ client_combobox_gender;
private: System::Windows::Forms::ColumnHeader^ account_field;
private: System::Windows::Forms::ColumnHeader^ account_data;

/// <summary>
/// Обязательная переменная конструктора.
/// </summary>
System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code
/// <summary>
/// Требуемый метод для поддержки конструктора – не изменяйте
/// содержимое этого метода с помощью редактора кода.
/// </summary>
void InitializeComponent(void);
#pragma endregion
private: generic <class TEnum> TEnum convert_to_enum(System::String^
value);

```

```

    private: System::Void
order_request_callback(OrderController::RequestAcceptToken token);

    private: System::Void progressbar_proceed(System::Void);

    private: System::Void client_button_order_Click(System::Object^ sender,
System::EventArgs^ e);

    private: System::Void client_button_cancel_Click(System::Object^ sender,
System::EventArgs^ e);

    private: System::Void client_button_refresh_Click(System::Object^ sender,
System::EventArgs^ e);

    private: System::Void client_button_update_Click(System::Object^ sender,
System::EventArgs^ e);

    private: System::Void client_button_logout_Click(System::Object^ sender,
System::EventArgs^ e);

    private: System::Void account_list_initialize(System::Void);

    private: System::Void client_button_bank_Click(System::Object^ sender,
System::EventArgs^ e);

    private: System::Void client_button_price_Click(System::Object^ sender,
System::EventArgs^ e);

    private: System::Void client_button_money_Click(System::Object^ sender,
System::EventArgs^ e);
};

}

```

Файл «/views/client_page_view/client_page_view_callbacks.cpp»

```

#include "client_page_view.h"

using namespace Views;

generic <class TEnum> TEnum ClientPageView::convert_to_enum(System::String^ value)
{ return safe_cast<TEnum>(System::Enum::Parse(TEnum::typeid, value, true)); }

System::Void
ClientPageView::order_request_callback(OrderController::RequestAcceptToken token)
{
    if (!this->service_order_controller->cancellation_order())
        MessageBox::Show("Произошла ошибка при отмене заказа", "Ошибка");

    System::Boolean transfer_checker(true);

    if (token.Status == true)
    {
        Services::DriveComplexDbScheme^ complex_scheme = this-
>service_depot_manager->get_driver_complexes(token.ConnectionGuid);
        Services::AccountDriversDbScheme^ driver_scheme = this-
>service_account_manager
        -
>get_account_scheme<Services::AccountDriversDbScheme^>(token.ConnectionGuid);

        System::Guid transfer_guid = System::Guid::Empty;
        System::Int32 transfer_price(0);

        try {

```

```

        Models::CarModelTypes car_model_type =
convert_to_enum<Models::CarModelTypes>(complex_scheme->car_type);
        transfer_price = this->service_depot_manager-
>CarTypePrice[car_model_type];
        transfer_guid = System::Guid::Parse(driver_scheme->bank_card);
    }
    catch (System::Exception^) { MessageBox::Show("Произошла ошибка при
обработки подготовке к транзакции", "Ошибка"); }
    transfer_checker = this->service_bank_controller-
>transfer_money(transfer_guid, transfer_price);
}

this->order_proccess = false;
this->client_label_waiting->Text = "Состояние заказа";

if(transfer_checker)MessageBox::Show("Состояние заказа: " +
token.Status.ToString(), "Готово");
else MessageBox::Show("Произошла ошибка при оплате поездки", "Ошибка");

this->client_button_cancel->Enabled = false;
this->client_button_order->Enabled = true;
}

System::Void ClientPageView::client_button_money_Click(System::Object^ sender,
System::EventArgs^ e)
{
    try {
        Models::AccountClientModel^ account_model =
safe_cast<Models::AccountClientModel^>(
            this->service_account_manager->AccountToken.AccountModel);

        if (!this->service_bank_controller->load_bank_account(account_model-
>BankCard))
            throw gcnew
Services::AccountManagerTokenException(ClientPageView::typeid, "cant load bank
account");
        }
        catch (System::Exception^) { MessageBox::Show("Невозможно подключиться к
банковскому аккаунту", "Ошибка"); return; }

        MessageBox::Show(System::String::Concat("Текущий баланс: ", this-
>service_bank_controller->AccountMoney), "Готово");
    }
}

System::Void ClientPageView::progressbar_proceed(System::Void)
{
    this->client_progressbar_waiting->Value = 100;
    for (System::Int32 time_index = 0; time_index < 100; time_index++)
    {
        if (this->order_proccess != true) break;
        this->client_progressbar_waiting->Value -= 1;
        Thread::Sleep(this->service_order_controller->OrderRequestSecond * 10);
    }
    this->client_progressbar_waiting->Value = 0;
}

System::Void ClientPageView::client_button_order_Click(System::Object^ sender,
System::EventArgs^ e)
{
    Models::AccountClientModel^ client_model = nullptr;
    try { client_model = cli::safe_cast<Models::AccountClientModel^>(this-
>service_account_manager->AccountToken.AccountModel); }
    catch (System::Exception^) { MessageBox::Show("Невозможно загрузить данные
аккаунта"); return; }
}

```

```

if (!this->service_bank_controller->load_bank_account(client_model->BankCard))
{ MessageBox::Show("Невозможно загрузить банковские данные"); return; }

System::Int32 balance = this->service_bank_controller->AccountMoney;

if (this->client_textbox_address->Text == System::String::Empty)
{ MessageBox::Show("Заполните текстовое поле для адреса", "Требование");
return; }

try {
    Models::AccountClientModel^ account_model =
safe_cast<Models::AccountClientModel^>(
        this->service_account_manager->AccountToken.AccountModel);

    if (!this->service_bank_controller->load_bank_account(account_model-
>BankCard))
        throw gcnew
Services::AccountManagerTokenException(ClientPageView::typeid, "cant load bank
account");
    }
    catch (System::Exception^) { MessageBox::Show("Невозможно подключиться к
банковскому аккаунту", "Ошибка"); return; }

System::String^ address_field = this->client_textbox_address->Text;
Models::CarModelTypes cartype_field;
System::Int32 request_price(0);

switch (this->client_combobox_cartype->SelectedIndex)
{
case 0: cartype_field = Models::CarModelTypes::CarTypeEconom; break;
case 1: cartype_field = Models::CarModelTypes::CarTypeChild; break;
case 2: cartype_field = Models::CarModelTypes::CarTypePremium; break;
}
request_price = this->service_depot_manager->CarTypePrice[cartype_field];
if (balance < request_price) { MessageBox::Show("Недостаточно средств");
return; }

this->service_order_controller->OrderRequestCallback += gcnew
OrderController::RequestCallback(
    this, &ClientPageView::order_request_callback);

System::Guid client_guid = this->service_account_manager-
>AccountToken.AccountGuid;
System::Boolean registered_order;

if (this->client_combobox_carclass->SelectedIndex == 0)
{
    registered_order = this->service_order_controller-
>registration_order<Models::CarLightModel^>(
        address_field, cartype_field, client_guid);
}
else
{
    registered_order = this->service_order_controller-
>registration_order<Models::CarHeavyModel^>(
        address_field, cartype_field, client_guid);
}

if (registered_order != true) { MessageBox::Show("Невозможно создать заказ",
"Ошибка"); return; }

this->client_button_cancel->Enabled = true;
this->client_button_order->Enabled = false;

this->client_label_waiting->Text = "Ожидание принятия заказа...";

```

```

    this->order_proccess = true;

    Task::Run(gcnew Action(this, &ClientPageView::progressbar_proceed));
}

System::Void ClientPageView::client_button_cancel_Click(System::Object^ sender,
System::EventArgs^ e)
{
    this->service_order_controller->cancellation_token_push();
    return System::Void();
}

System::Void ClientPageView::client_button_bank_Click(System::Object^ sender,
System::EventArgs^ e)
{
    Windows::Forms::Form^ bank_page = gcnew Views::BankSettingsView(this-
>service_bank_controller);
    bank_page->>ShowDialog();
}

System::Void ClientPageView::client_button_logout_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (MessageBox::Show("Вы уверены?", "Подтверждение", MessageBoxButtons::YesNo,
MessageBoxIcon::Question)
        == ::DialogResult::Yes)
    {
        System::Boolean logout_check = this->service_account_manager-
>sign_out_account();
        this->Close();
    }
}

System::Void ClientPageView::account_list_initialize(System::Void)
{
    this->client_listview_account->FullRowSelect = true;
    this->client_listview_account->Items->Clear();

    ListViewItem^ list_item = gcnew ListViewItem("Тип аккаунта");
    list_item->SubItems->Add(this->service_account_manager-
>AccountToken.AccountType.ToString());
    this->client_listview_account->Items->Add(list_item);

    list_item = gcnew ListViewItem("Аккаунт Guid");
    list_item->SubItems->Add(this->service_account_manager-
>AccountToken.AccountGuid.ToString());
    this->client_listview_account->Items->Add(list_item);

    Models::AccountClientModel^ account_model = nullptr;
    try {
        account_model = safe_cast<Models::AccountClientModel^>(this-
>service_account_manager
            ->AccountToken.AccountModel);
    }
    catch (System::Exception^) { MessageBox::Show("Невозможно определить модель
аккаунта", "Ошибка"); return; }

    list_item = gcnew ListViewItem("Номер банковской карты");
    list_item->SubItems->Add(account_model->BankCard.ToString());
    this->client_listview_account->Items->Add(list_item);

    list_item = gcnew ListViewItem("Возраст");
    list_item->SubItems->Add(account_model->Age.ToString());
    this->client_listview_account->Items->Add(list_item);
}

```

```

list_item = gcnew ListViewItem("Пол");
list_item->SubItems->Add(account_model->Gender.ToString());
this->client_listview_account->Items->Add(list_item);

list_item = gcnew ListViewItem("Имя");
list_item->SubItems->Add(account_model->Username);
this->client_listview_account->Items->Add(list_item);
}

System::Void ClientPageView::client_button_update_Click(System::Object^ sender,
System::EventArgs^ e)
{
    Models::AccountClientModel^ current_model =
safe_cast<Models::AccountClientModel^>(
    this->service_account_manager->AccountToken.AccountModel);

    System::Int32 age_field = this->client_checkbox_age->Checked ?
Decimal::ToInt32(this->client_numeric_age->Value)
    : current_model->Age;

    System::String^ username_field = this->client_checkbox_username->Checked ?
        this->client_textbox_username->Text : current_model->Username;
    if (username_field == System::String::Empty) { MessageBox::Show("Неверный формат
имени", "Ошибка"); return; }

    Models::AccountModelGender gender_field;
    if (this->client_checkbox_gender->Checked)
    {
        switch (this->client_combobox_gender->SelectedIndex)
        {
            case 0: gender_field = Models::AccountModelGender::MaleGender; break;
            case 1: gender_field = Models::AccountModelGender::FemaleGender; break;
        }
    }
    else gender_field = current_model->Gender;

    System::Guid bankcard_field = System::Guid::Empty;
    if (this->client_checkbox_bankcard->Checked)
    {
        try { bankcard_field = System::Guid::Parse(this->client_textbox_bankcard-
>Text); }
        catch (System::Exception^) { MessageBox::Show("Неверный формат банковской
карты", "Ошибка"); return; }

        if (!this->service_bank_controller->load_bank_account(bankcard_field))
            { MessageBox::Show("Банковский аккаунт не найден", "Ошибка"); return; }
    }
    else bankcard_field = current_model->BankCard;

    Models::AccountClientModel^ model = gcnew Models::AccountClientModel(
        username_field, age_field, gender_field, bankcard_field);
    System::Boolean update_check = this->service_account_manager-
>update_account(model);
    if (update_check != true) { MessageBox::Show("Не удалось обновить данные
аккаунта", "Ошибка"); return; }

    MessageBox::Show("Данные успешно обновлены", "Готово");
    this->account_list_initialize();
}

System::Void ClientPageView::client_button_refresh_Click(System::Object^ sender,
System::EventArgs^ e)
{
    Generic::List<System::Guid>^ drivers_list = this->service_depot_manager-
>get_all_drivers();
}

```

```

    if (drivers_list == nullptr) { MessageBox::Show("Не удалось получить информацию
о водителях", "Ошибка"); return; }

    this->client_listview_car->FullRowSelect = true;
    this->client_listview_car->Items->Clear();

    for each (auto item in drivers_list)
    {
        Services::DriveComplexDbScheme^ complex = this->service_depot_manager-
>get_driver_complexes(item);
        if (complex == nullptr) continue;

        ListViewItem^ list_item = gcnew ListViewItem(complex->driver_guid);
        list_item->SubItems->Add(complex->car_class);
        list_item->SubItems->Add(complex->car_type);
        list_item->SubItems->Add(complex->driver_state);

        this->client_listview_car->Items->Add(list_item);
    }
    MessageBox::Show("Список доступных машин обновлен", "Готово");
}

System::Void ClientPageView::client_button_price_Click(System::Object^ sender,
System::EventArgs^ e)
{
    Windows::Forms::Form^ form = gcnew Views::ClientPriceListView(this-
>service_depot_manager);
    form->>ShowDialog();
}

```

Файл «/views/client_page_view/client_pricelist_view.h»

```

#pragma once
#include "../services/services.h"
#include "../models/cars_model/cars_model.h"

namespace Views
{
    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Сводка для MyForm
    /// </summary>
    public ref class ClientPriceListView : public System::Windows::Forms::Form
    {
        Services::DepotManager^ service_depot_manager = nullptr;
    public:
        ClientPriceListView(Services::DepotManager^ depot_manager)
        {
            InitializeComponent();
            this->Icon = gcnew
System::Drawing::Icon(L"./assets/my_app_icon.ico");
            this->service_depot_manager = depot_manager; this-
>listview_initialize();
        }

    protected:
        /// <summary>
        /// Освободить все используемые ресурсы.

```

```

    /// </summary>
    ~ClientPriceListView(System::Void) { if (components) delete components; }
private: System::Windows::Forms::ListView^ list_view;
protected:
private: System::Windows::Forms::ColumnHeader^ car_type;
private: System::Windows::Forms::ColumnHeader^ car_price;

private:
    /// <summary>
    /// Обязательная переменная конструктора.
    /// </summary>
    System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Требуемый метод для поддержки конструктора – не изменяйте
    /// содержимое этого метода с помощью редактора кода.
    /// </summary>
    void InitializeComponent(void);
#pragma endregion
    private: System::Void listview_initialize(System::Void)
    {
        this->list_view->FullRowSelect = true;
        this->list_view->Items->Clear();

        ListViewItem^ list_item = gcnew ListViewItem("CarTypeEconom");
        list_item->SubItems->Add(this->service_depot_manager

>CarTypePrice[Models::CarModelTypes::CarTypeEconom].ToString());
        this->list_view->Items->Add(list_item);

        list_item = gcnew ListViewItem("CarTypePremium");
        list_item->SubItems->Add(this->service_depot_manager

>CarTypePrice[Models::CarModelTypes::CarTypePremium].ToString());
        this->list_view->Items->Add(list_item);

        list_item = gcnew ListViewItem("CarTypeChild ");
        list_item->SubItems->Add(this->service_depot_manager

>CarTypePrice[Models::CarModelTypes::CarTypeChild].ToString());
        this->list_view->Items->Add(list_item);
    }
}
}

```

Файл «/views/driver_page_view/driver_page_view.h»

```

#pragma once

#include "../manager/manager.h"
#include "../services/services.h"

#include "../bank_settings_view/bank_settings_view.h"
#include "../admin_page_view/admin_page_view.h"
#include "../driver_page_view/driver_page_view.h"
#include "../client_page_view/client_page_view.h"
#include "driver_dialog_view.h"
#include "driver_complex_view.h"

namespace Views
{
    using namespace System;

```

```

using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;

using namespace Models;
using namespace Services;

/// <summary>
/// Сводка для DriverPageView
/// </summary>
public ref class DriverPageView : public System::Windows::Forms::Form
{
    Manager::ServiceManager^ service_manager = nullptr;
    Windows::Forms::Form^ start_page = nullptr;

    Services::AccountManager^ service_account_manager = nullptr;
    Services::DepotManager^ service_depot_manager = nullptr;
    Services::OrderController^ service_order_controller = nullptr;
private: System::Windows::Forms::CheckBox^ driver_checkbox_licence;
private: System::Windows::Forms::CheckBox^ driver_checkbox_bankcard;
private: System::Windows::Forms::CheckBox^ driver_checkbox_age;
private: System::Windows::Forms::CheckBox^ driver_checkbox_gender;
private: System::Windows::Forms::CheckBox^ driver_checkbox_username;
    Services::BankController^ service_bank_controller = nullptr;

public:
    DriverPageView(System::Void) { InitializeComponent(); }

    DriverPageView(Windows::Forms::Form^ start_page, Manager::ServiceManager^
service_manager)
    {
        InitializeComponent();
        this->Icon = gcnew
System::Drawing::Icon(L"./assets/my_app_icon.ico");

        this->driver_page_account->BackgroundImage =
System::Drawing::Image::FromFile("./assets/background.png");
        this->driver_page_account->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;

        this->driver_page_order->BackgroundImage =
System::Drawing::Image::FromFile("./assets/background.png");
        this->driver_page_order->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;

        this->driver_page_garage->BackgroundImage =
System::Drawing::Image::FromFile("./assets/background.png");
        this->driver_page_garage->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;

        this->service_manager = service_manager;
        this->start_page = start_page;

        Manager::IServiceProvider^ provider_account_manager = this-
>service_manager->get_service<AccountManager^>();
        Manager::IServiceProvider^ provider_depot_manager = this-
>service_manager->get_service<DepotManager^>();
        Manager::IServiceProvider^ provider_order_controller = this-
>service_manager->get_service<OrderController^>();
        Manager::IServiceProvider^ provider_bank_controller = this-
>service_manager->get_service<BankController^>();
    }
}

```

```

        this->service_account_manager =
(Services::AccountManager^)provider_account_manager->Service;
        this->service_depot_manager =
(Services::DepotManager^)provider_depot_manager->Service;
        this->service_order_controller =
(Services::OrderController^)provider_order_controller->Service;
        this->service_bank_controller =
(Services::BankController^)provider_bank_controller->Service;

        this->account_list_initialize();
        this->driver_complex_close();
    }

protected:
    /// <summary>
    /// Освободить все используемые ресурсы.
    /// </summary>
~DriverPageView(System::Void)
{
    if (this->service_account_manager->IsInitialized) this-
>service_account_manager->sign_out_account();
    if (this->service_depot_manager->IsBuilded) this-
>service_depot_manager->return_car_model();
    if (components) delete components;
}

private: System::Windows::Forms::Label^ driver_label_currentstatus;
private: System::Windows::Forms::Label^ driver_statuslist;
private: System::Windows::Forms::TextBox^ driver_textbox_status;
private: System::Windows::Forms::Button^ driver_button_status;
private: System::Windows::Forms::ComboBox^ driver_combobox_status;
private: System::Windows::Forms::Button^ driver_button_bankmoney;
private: System::Windows::Forms::Label^ driver_orders_list;
private: System::Windows::Forms::Button^ driver_button_update_orders;
private: System::Windows::Forms::Button^ driver_button_accept_order;
private: System::Windows::Forms::ColumnHeader^ order_client_guid;
private: System::Windows::Forms::ColumnHeader^ order_car_class;
private: System::Windows::Forms::ColumnHeader^ order_car_type;
private: System::Windows::Forms::ColumnHeader^ order_address;
private: System::Windows::Forms::Button^ driver_button_complexinfo;

private: System::Windows::Forms::ColumnHeader^ order_date;
private: System::Windows::Forms::Button^ driver_button_update_garage;
private: System::Windows::Forms::Label^ driver_label_garage;
private: System::Windows::Forms::Button^ driver_button_return;
private: System::Windows::Forms::Button^ driver_button_rent;
private: System::Windows::Forms::ListView^ driver_listview_garage;
private: System::Windows::Forms::ColumnHeader^ garage_car_class;
private: System::Windows::Forms::ColumnHeader^ garage_car_type;
private: System::Windows::Forms::ColumnHeader^ garage_car_color;
private: System::Windows::Forms::ColumnHeader^ garage_car_speed;
private: System::Windows::Forms::ColumnHeader^ garage_car_count;

private: System::Windows::Forms::TabControl^ driver_tabcontrol;
private: System::Windows::Forms::TabPage^ driver_page_order;
private: System::Windows::Forms::TabPage^ driver_page_account;

private: System::Windows::Forms::TextBox^ driver_textbox_licence;
private: System::Windows::Forms::Button^ driver_button_bank;

private: System::Windows::Forms::Label^ driver_label_account;

```

```

private: System::Windows::Forms::NumericUpDown^ driver_numeric_age;
private: System::Windows::Forms::TextBox^ driver_textbox_bankcard;

private: System::Windows::Forms::ComboBox^ driver_combobox_gender;
private: System::Windows::Forms::TextBox^ driver_textbox_username;
private: System::Windows::Forms::Button^ driver_button_update;
private: System::Windows::Forms::Button^ driver_button_logout;
private: System::Windows::Forms::ListView^ driver_listview_account;
private: System::Windows::Forms::ColumnHeader^ account_field;
private: System::Windows::Forms::ColumnHeader^ account_data;
private: System::Windows::Forms::ListView^ driver_listview_orders;

private: System::Windows::Forms::TabPage^ driver_page_garage;

private:

    /// <summary>
    /// Обязательная переменная конструктора.
    /// </summary>
    System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Требуемый метод для поддержки конструктора – не изменяйте
    /// содержимое этого метода с помощью редактора кода.
    /// </summary>
    void InitializeComponent(void);
#pragma endregion

private: generic <class TEnum> TEnum convert_to_enum(System::String^
value);

private: System::Void driver_complex_open(System::Void);
private: System::Void driver_complex_close(System::Void);

private: System::Void driver_button_accept_order_Click(System::Object^
sender, System::EventArgs^ e);

private: System::Void driver_button_update_orders_Click(System::Object^
sender, System::EventArgs^ e);

private: System::Void driver_button_update_garage_Click(System::Object^
sender, System::EventArgs^ e);

private: System::Void driver_button_rent_Click(System::Object^ sender,
System::EventArgs^ e);

private: System::Void driver_button_return_Click(System::Object^ sender,
System::EventArgs^ e);

private: System::Void account_list_initialize(System::Void);

private: System::Void driver_button_update_Click(System::Object^ sender,
System::EventArgs^ e);

private: System::Void driver_button_bank_Click(System::Object^ sender,
System::EventArgs^ e);

private: System::Void driver_button_logout_Click(System::Object^ sender,
System::EventArgs^ e);

private: System::Void driver_button_complexinfo_Click(System::Object^
sender, System::EventArgs^ e);

```

```

    private: System::Void driver_button_bankmoney_Click(System::Object^ sender, System::EventArgs^ e);

    private: System::Void driver_button_status_Click(System::Object^ sender, System::EventArgs^ e);
}
}

```

Файл «/views/driver_page_view/driver_page_view_callbacks.cpp»

```

#include "driver_page_view.h"

using namespace Views;

generic <class TEnum> TEnum DriverPageView::convert_to_enum(System::String ^ value)
{
    return safe_cast<TEnum>(System::Enum::Parse(TEnum::typeid, value, true));
}

System::Void DriverPageView::driver_complex_close(System::Void)
{
    this->driver_textbox_status->Text = "";
    this->driver_button_status->Enabled = false;
    this->driver_button_rent->Enabled = true;
    this->driver_button_return->Enabled = false;
}

System::Void DriverPageView::driver_complex_open(System::Void)
{
    this->driver_textbox_status->Text = "В ожидании";
    this->driver_button_status->Enabled = true;
    this->driver_button_rent->Enabled = false;
    this->driver_button_return->Enabled = true;
}

System::Void DriverPageView::driver_button_status_Click(System::Object^ sender, System::EventArgs^ e)
{
    Services::DriveComplexToken::DriverStateType new_driver_status;
    System::String^ new_driver_status_text = nullptr;

    switch (this->driver_combobox_status->SelectedIndex)
    {
        case 0:
            new_driver_status = Services::DriveComplexToken::DriverStateType::Ready;
            new_driver_status_text = "Готов"; break;
        case 1: new_driver_status = Services::DriveComplexToken::DriverStateType::Busy;
            new_driver_status_text = "Занят"; break;
        case 2: new_driver_status = Services::DriveComplexToken::DriverStateType::Idle;
            new_driver_status_text = "В ожидании"; break;
    }

    if (!this->service_depot_manager->update_drive_state(new_driver_status))
    { MessageBox::Show("Не удалось изменить статус", "Ошибка"); return; }

    this->driver_textbox_status->Text = new_driver_status_text;
}

System::Void DriverPageView::driver_button_bankmoney_Click(System::Object^ sender, System::EventArgs^ e)
{
    try {

```

```

        Models::AccountDriverModel^ account_model =
safe_cast<Models::AccountDriverModel^>(
    this->service_account_manager->AccountToken.AccountModel);

    if (!this->service_bank_controller->load_bank_account(account_model-
>BankCard))
        throw gcnew
Services::AccountManagerTokenException(ClientPageView::typeid, "cant load bank
account");
    }
    catch (System::Exception^) { MessageBox::Show("Невозможно подключиться к
банковскому аккаунту", "Ошибка"); return; }

    MessageBox::Show(System::String::Concat("Текущий баланс: ", this-
>service_bank_controller->AccountMoney), "Готово");
}

System::Void DriverPageView::driver_button_accept_order_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (this->service_depot_manager->IsBuilded != true)
    { MessageBox::Show("Для начала нужно арендовать машину", "Ошибка"); return; }

    if (this->driver_listview_orders->SelectedItems->Count <= 0)
    { MessageBox::Show("Выберите строку с заказом", "Ошибка"); return; }

    System::String^ selected_order = this->driver_listview_orders->SelectedItems[0]-
>Text;
    System::Guid client_guid;

    try { client_guid = System::Guid::Parse(selected_order); }
    catch (System::Exception^) { MessageBox::Show("Невозможно прочитать Guid заказа",
"Ошибка"); return; }

    System::Boolean accept_check(false);
    try {
        accept_check = this->service_order_controller->accept_request(client_guid,
            this->service_depot_manager->DriverGuid);
    }
    catch (Services::OrderControllerTokenException^) {}

    if (accept_check != true) MessageBox::Show("Невозможно принять заказ", "Ошибка");
    else {
        Windows::Forms::Form^ form = gcnew Views::DriverDialogView(this-
>service_depot_manager->CarModel);
        form->>ShowDialog();
    }
}

System::Void DriverPageView::driver_button_update_orders_Click(System::Object^ sender,
System::EventArgs^ e)
{
    List<Services::OrderControllerDbScheme^>^ order_list = this-
>service_order_controller->OrderList;
    if (order_list == nullptr) { MessageBox::Show("Невозможно получить список
заказов", "Ошибка"); return; }

    this->driver_listview_orders->FullRowSelect = true;
    this->driver_listview_orders->Items->Clear();
    for each (auto item in order_list)
    {
        ListViewItem^ list_item = gcnew ListViewItem(item->client_guid);
        list_item->SubItems->Add(item->car_class);
        list_item->SubItems->Add(item->car_type);
    }
}

```

```

        list_item->SubItems->Add(item->address);
        list_item->SubItems->Add(item->date_time);

        this->driver_listview_orders->Items->Add(list_item);
    }

}

System::Void DriverPageView::driver_button_update_garage_Click(System::Object^ sender,
System::EventArgs^ e)
{
    List<DepotManager::CarGarageItems>^ car_list = this->service_depot_manager-
>get_all_cars();
    if (car_list == nullptr) { MessageBox::Show("Невозможно получить список машин",
"Ошибка"); return; }

    this->driver_listview_garage->FullRowSelect = true;
    this->driver_listview_garage->Items->Clear();

    for each (auto item in car_list)
    {
        ListViewItem^ list_item = gcnew ListViewItem(item.CarClass->Name);
        list_item->SubItems->Add(item.CarModel->CarType.ToString());
        list_item->SubItems->Add(item.CarModel->CarColor.ToString());

        list_item->SubItems->Add(item.CarModel->CarSpeed.ToString());
        list_item->SubItems->Add(item.CarCount.ToString());
        this->driver_listview_garage->Items->Add(list_item);
    }
}

System::Void DriverPageView::driver_button_rent_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (this->driver_listview_garage->SelectedItems->Count <= 0)
    { MessageBox::Show("Выберите строку с машиной", "Ошибка"); return; }

    if (this->service_depot_manager->IsBuilded)
    { MessageBox::Show("Машина уже выбрана", "Ошибка"); return; }

    auto selected_list = this->driver_listview_garage->SelectedItems[0]->SubItems;
    System::String^ car_class_string = selected_list[0]->Text;
    System::String^ car_type_string = selected_list[1]->Text;
    System::String^ car_color_string = selected_list[2]->Text;
    System::String^ car_speed_string = selected_list[3]->Text;

    System::UInt32 car_speed;
    Models::CarModelColor car_color;
    Models::CarModelTypes car_type;

    try {
        car_speed = System::UInt32::Parse(car_speed_string);
        car_color = this-
>convert_to_enum<Models::CarModelColor>(car_color_string);
        car_type = this->convert_to_enum<Models::CarModelTypes>(car_type_string);
    }
    catch (System::Exception^) { MessageBox::Show("Невозможно составить запрос",
"Ошибка"); return; }

    System::Boolean rent_request(true);
    if (car_class_string == Models::CarLightModel::typeid->Name)
    {
        Models::CarLightModel^ model = gcnew Models::CarLightModel(car_type,
car_color, car_speed);
        rent_request = this->service_depot_manager-
>rent_car_model<Models::CarLightModel^>(

```

```

                model, this->service_account_manager->AccountToken.AccountGuid);
}
else if (car_class_string == Models::CarHeavyModel::typeid->Name)
{
    Models::CarHeavyModel^ model = gcnew Models::CarHeavyModel(car_type,
car_color, car_speed);
    rent_request = this->service_depot_manager-
>rent_car_model<Models::CarHeavyModel^>(
        model, this->service_account_manager->AccountToken.AccountGuid);
}
else { MessageBox::Show("Невозможно определить класс машины", "Ошибка");
return; }

if (rent_request != true) { MessageBox::Show("Запрос на аренду машины отклонен",
"Ошибка"); }
else
{
    MessageBox::Show("Машина успешно арендована", "Готово");
    this->driver_complex_open();
}
}

System::Void DriverPageView::driver_button_return_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (!this->service_depot_manager->IsBuilded)
    { MessageBox::Show("Машина еще не выбрана", "Ошибка"); return; }

    if (!this->service_depot_manager->return_car_model()) MessageBox::Show("Запрос
на возврат отклонен", "Ошибка");
    else
    {
        MessageBox::Show("Машина возвращена в гараж", "Готово");
        this->driver_complex_close();
    }
}

System::Void DriverPageView::account_list_initialize(System::Void)
{
    this->driver_listview_account->FullRowSelect = true;
    this->driver_listview_account->Items->Clear();

    ListViewItem^ list_item = gcnew ListViewItem("Тип аккаунта");
    list_item->SubItems->Add(this->service_account_manager-
>AccountToken.AccountType.ToString());
    this->driver_listview_account->Items->Add(list_item);

    list_item = gcnew ListViewItem("Аккаунт Guid");
    list_item->SubItems->Add(this->service_account_manager-
>AccountToken.AccountGuid.ToString());
    this->driver_listview_account->Items->Add(list_item);

    Models::AccountDriverModel^ account_model = nullptr;
    try {
        account_model = safe_cast<Models::AccountDriverModel^>(this-
>service_account_manager
            ->AccountToken.AccountModel);
    }
    catch (System::Exception^) { MessageBox::Show("Невозможно определить модель
аккаунта", "Ошибка"); return; }

    list_item = gcnew ListViewItem("Номер банковской карты");
    list_item->SubItems->Add(account_model->BankCard.ToString());
    this->driver_listview_account->Items->Add(list_item);
}

```

```

list_item = gcnew ListViewItem("Номер лицензии");
list_item->SubItems->Add(account_model->Licence.ToString());
this->driver_listview_account->Items->Add(list_item);

list_item = gcnew ListViewItem("Возраст");
list_item->SubItems->Add(account_model->Age.ToString());
this->driver_listview_account->Items->Add(list_item);

list_item = gcnew ListViewItem("Пол");
list_item->SubItems->Add(account_model->Gender.ToString());
this->driver_listview_account->Items->Add(list_item);

list_item = gcnew ListViewItem("Имя");
list_item->SubItems->Add(account_model->Username);
this->driver_listview_account->Items->Add(list_item);
}

System::Void DriverPageView::driver_button_update_Click(System::Object^ sender,
System::EventArgs^ e)
{
    Models::AccountDriverModel^ current_model =
safe_cast<Models::AccountDriverModel^>(
    this->service_account_manager->AccountToken.AccountModel);

    System::Int32 age_field = this->driver_checkbox_age->Checked ?
Decimal::ToInt32(this->driver_numeric_age->Value)
    : current_model->Age;

    System::String^ username_field = this->driver_checkbox_username->Checked ?
        this->driver_textbox_username->Text : current_model->Username;
    if (username_field == System::String::Empty) { MessageBox::Show("Неверный формат
имени", "Ошибка"); return; }

    Models::AccountModelGender gender_field;
    if (this->driver_checkbox_gender->Checked)
    {
        switch (this->driver_combobox_gender->SelectedIndex)
        {
            case 0: gender_field = Models::AccountModelGender::MaleGender; break;
            case 1: gender_field = Models::AccountModelGender::FemaleGender; break;
        }
    }
    else gender_field = current_model->Gender;

    System::Guid bankcard_field, licence_field;
    if (this->driver_checkbox_bankcard->Checked)
    {
        try { bankcard_field = System::Guid::Parse(this->driver_textbox_bankcard-
>Text); }
        catch (System::Exception^) { MessageBox::Show("Неверный формат банковской
карты", "Ошибка"); return; }

        if (!this->service_bank_controller->load_bank_account(bankcard_field))
            { MessageBox::Show("Банковский аккаунт не найден", "Ошибка"); return; }
    }
    else bankcard_field = current_model->BankCard;

    if (this->driver_checkbox_licence->Checked)
    {
        try { licence_field = System::Guid::Parse(this->driver_textbox_licence-
>Text); }
        catch (System::Exception^) { MessageBox::Show("Неверный формат лицензии",
"Ошибка"); return; }
    }
    else licence_field = current_model->Licence;
}

```

```

Models::AccountDriverModel^ model = gcnew Models::AccountDriverModel(
    username_field, age_field, gender_field, licence_field, bankcard_field);

System::Boolean update_check = this->service_account_manager-
>update_account(model);
if (update_check != true) MessageBox::Show("Не удалось обновить данные аккаунта",
"Ошибка");

MessageBox::Show("Данные успешно обновлены", "Готово");
this->account_list_initialize();
}

System::Void DriverPageView::driver_button_bank_Click(System::Object^ sender,
System::EventArgs^ e)
{
    Windows::Forms::Form^ bank_page = gcnew Views::BankSettingsView(this-
>service_bank_controller);
    bank_page->ShowDialog();
}

System::Void DriverPageView::driver_button_logout_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (MessageBox::Show("Вы уверены?", "Подтверждение", MessageBoxButtons::YesNo,
MessageBoxIcon::Question)
        == ::DialogResult::Yes)
    {
        System::Boolean logout_check = this->service_account_manager-
>sign_out_account();
        if (this->service_depot_manager->IsBuilded) this->service_depot_manager-
>return_car_model();
        this->Close();
    }
}

System::Void DriverPageView::driver_button_complexinfo_Click(System::Object^ sender,
System::EventArgs^ e)
{
    if (!this->service_depot_manager->IsBuilded)
    { MessageBox::Show("Для начала арендуйте машину", "Ошибка"); return; }

    Windows::Forms::Form^ form = gcnew Views::DriverComplexView(this-
>service_depot_manager->CarModel,
    this->service_depot_manager->CarModelType);
    form->ShowDialog();
}

```

Файл «/views/driver_page_view/driver_dialog_view.h»

```

#pragma once
#include "../models/cars_model/cars_model.h"

#define ORDER_COMPLITION_TIME 5

namespace Views
{
    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

```

```

using namespace System::Threading::Tasks;
using namespace System::Threading;
/// <summary>
/// Сводка для MyForm
/// </summary>
public ref class DriverDialogView : public System::Windows::Forms::Form
{
    Models::CarBaseModel^ car_model = nullptr;

public:
    DriverDialogView(System::Void) { InitializeComponent(); }

    DriverDialogView(Models::CarBaseModel^ car_model)
    {
        InitializeComponent();
        this->Icon = gcnew
System::Drawing::Icon(L"./assets/my_app_icon.ico");
        this->list_view->Items->Add(gcnew ListViewItem(car_model-
>car_drive()));
        this->car_model = car_model;
    }

protected:
    /// <summary>
    /// Освободить все используемые ресурсы.
    /// </summary>
    ~DriverDialogView(System::Void) { if (components) delete components; }
private: System::Windows::Forms::ProgressBar^ progress_bar;
private: System::Windows::Forms::Button^ button;

private: System::Windows::Forms::ListView^ list_view;
private: System::Windows::Forms::ColumnHeader^ column_header;

private:
    /// <summary>
    /// Обязательная переменная конструктора.
    /// </summary>
    System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Требуемый метод для поддержки конструктора – не изменяйте
    /// содержимое этого метода с помощью редактора кода.
    /// </summary>
    void InitializeComponent(void);
#pragma endregion
    private: System::Void order_processing_bar(System::Void)
    {
        for (System::Int16 i = 0; i < 100; i++)
        {
            this->progress_bar->Value += 1;
            Thread::Sleep(ORDER_COMPLITION_TIME* 10);
        }
        this->Hide();
        MessageBox::Show("Поездка закончилась", "Готово");
        this->Close();
    }

    private: System::Void button_Click(System::Object^ sender,
System::EventArgs^ e)
    {
        this->button->Enabled = false;
        Task::Run(gcnew System::Action(this,
&DriverDialogView::order_processing_bar));
    }

```

```
    };
```

Файл «/views/driver_page_view/driver_complex_view.h»

```
#pragma once
#include "../services/depot_manager/depot_manager.h"
#include "../services/depot_manager/depot_manager_token.h"
#include "../models/cars_model/cars_model.h"

namespace Views
{

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Сводка для MyForm
    /// </summary>
public ref class DriverComplexView : public System::Windows::Forms::Form
{
    Models::CarBaseModel^ car_model = nullptr;
    System::Type^ car_model_class = nullptr;
public:
    DriverComplexView(System::Void) { InitializeComponent(); }

    DriverComplexView(Models::CarBaseModel^ car_model, System::Type^
car_model_class)
    {
        InitializeComponent();
        this->Icon = gcnew
System::Drawing::Icon(L"./assets/my_app_icon.ico");
        this->car_model_class = car_model_class;
        this->car_model = car_model;
        this->update_listview();
    }

protected:
    /// <summary>
    /// Освободить все используемые ресурсы.
    /// </summary>
~DriverComplexView(System::Void) { if (components) delete components; }

private: System::Windows::Forms::ListView^ listview;

private: System::Windows::Forms::ColumnHeader^ listview_name;
private: System::Windows::Forms::ColumnHeader^ listview_value;
private:
    /// <summary>
    /// Обязательная переменная конструктора.
    /// </summary>
    System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Требуемый метод для поддержки конструктора – не изменяйте
    /// содержимое этого метода с помощью редактора кода.
    /// </summary>
void InitializeComponent(void);
```

```
#pragma endregion
    private: System::Void update_listview()
    {
        this->listview->Items->Clear();
        this->listview->FullRowSelect = true;

        ListViewItem^ list_item = gcnew ListViewItem("Класс машины");
        list_item->SubItems->Add(this->car_model_class->Name);
        this->listview->Items->Add(list_item);

        list_item = gcnew ListViewItem("Тип машины");
        list_item->SubItems->Add(this->car_model->CarType.ToString());
        this->listview->Items->Add(list_item);

        list_item = gcnew ListViewItem("Цвет машины");
        list_item->SubItems->Add(this->car_model->CarColor.ToString());
        this->listview->Items->Add(list_item);

        list_item = gcnew ListViewItem("Скорость машины");
        list_item->SubItems->Add(this->car_model->CarSpeed.ToString());
        this->listview->Items->Add(list_item);
    }
};
```