

**КУРСОВОЙ ПРОЕКТ**

**По дисциплине: МДК 01.01 Разработка программных модулей**

**Тема: Разработка системы классов «Автосалон»**

**Специальность 09.02.07 «Информационные системы и программирование»**

|  |  |  |
| --- | --- | --- |
| **Выполнил студент(ка) группы 31ИС-23** | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | **М.С. Рудько** |
| **Руководитель** | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Л.Б. Гусятинер** |

**Москва 2024**



**УТВЕРЖДАЮ**

**Зам. директора КМПО**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_С.Ф. Гасанов**

**«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.**

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

**По дисциплине: МДК 01.01 Разработка программных модулей**

**Специальность 09.02.07 «Информационные системы и программирование»**

**Студент группы 302ИС-22 Рудько Матвей**

**ТЕМА: Разработка программного модуля информационной**

**системы «Автосалон»**

Дата выдачи задания «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г.

Срок сдачи проекта «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2024 г.

**Москва 2024**

**Содержание**

**ВВЕДЕНИЕ4**

ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ6

**1.1 Изучение предметной области6**

1.2 Изучение существующих разработок8

ГЛАВА 2. ПРОЕКТНАЯ ЧАСТЬ9

**2.1 Построение диаграммы прецедентов9**

2.2 Выбор инструментов10

2.3 Проектирование сценария12

2.4 Построение базы данных14

ГЛАВА 3. ЧАСТЬ РЕАЛИЗАЦИИ15

3.1 Начало разработки15

3.2 Описание главного модуля17

3.3 Описание спецификаций модулей22

3.4 Описание модуля «Клиент»23

3.5 Описание модуля «Сотрудник»28

3.6 Описание модуля «Машина»29

3.7 Описание модуля «Заказ»35

3.8 Описание модуля «Модели»38

3.9 Описание модуля «Настройки»39

3.10 Описание модуля «Поиск»46

3.11 Описание модуля «Перечисления»48

3.12 Описание модуля «Валидация»49

3.13 Описание модуля «prelude»51

3.14 Описание модуля «Приостановка программы»52

ГЛАВА 4. ЧАСТЬ ТЕСТИРОВАНИЯ53

4.1 Тестирование53

4.2 Отладка59

ГЛАВА 5. ЭКСПЛУАТАЦИОННАЯ ЧАСТЬ60

5.1 Руководство оператора60

5.2 To-Do List63

ЗАКЛЮЧЕНИЕ64

СПИСОК ЛИТЕРАТУРЫ И ИНТЕРНЕТ-ИСТОЧНИКОВ66

**Введение**

Целью моего курсового проекта является разработка консольного приложения для управления данными автосалона. Это приложение полезно для эффективного обрабатывания информации о клиентской базе, автомобилях и заказа, а также обеспечивает структурированный подход к учету и работе с этими данными. Современные технологии программирования и автоматизации делают такие решения крайне актуальными, особенно в условиях активного роста рынка автомобилей и увеличения объемов информации, с которой приходится работать автосалонам.

Для реализации данного проекта выбран язык программирования Rust. Этот выбор обусловлен рядом факторов. Во-первых, высокая производительность и строгая система управления памятью, которая обеспечивает надежность и безопасность, что особенно важно для обработки данных и выполнения критически важных операций. Во-вторых, поддержка многопоточности и асинхронного программирования позволяет разрабатывать приложения, способные обрабатывать множество задач одновременно, что крайне важно для автоматизации процессов в автосалоне.

Приложение представляет собой консольную программу с удобным интерфейсом, которая будет включать следующие функции:

* управление базой данных работников, включая добавление, удаление и обновление информации;
* управление базой данных клиентов, включая добавление, удаление и обновление информации;
* учет автомобилей, включая добавление новых моделей, редактирование и удаление;
* обработка заказов, включая создание, изменение статуса и удаление заказов;
* поиск и фильтрация данных по заданным критериям.

В первой части курсового проекта будет рассмотрена актуальность темы и проанализирована предметная область. Будут описаны основные требования к приложению, проблемы, которые оно решает, и его потенциальная практическая значимость.

Во второй части работы будет представлена архитектура приложения, выбранные технологии и инструменты, включая язык Rust и используемые библиотеки, а также база данных PostgreSQL. Особое внимание будет уделено структуре программы, принципам модульности.

В третьей части курсового проекта будет представлена реализация приложения. Оно будет включать подробное описание функций, структуру взаимодействия между основными и вспомогательными модулями, а также описание используемых алгоритмов. Будут приведены примеры кода, демонстрирующие работу ключевых компонентов программы, таких как управление базами данных и обработка пользовательского ввода.

В четвертой части курсового проекта будут представлены тесты и отладка программы.

В пятой части курсового проекта будет представлено руководство пользователя. Оно будет включать подробное описание процесса установки программы, использования её основных функций, а также демонстрацию возможностей интерфейса. Дополнительно будут представлены планы по дальнейшему развитию программы, включая возможные улучшения и расширения функционала.

Заключительная часть проекта будет посвящена анализу достигнутых результатов. Будут подведены итоги проделанной работы, оценена эффективность выбранных инструментов и технологий, а также сделаны выводы о практической значимости разработанного приложения.

Таким образом, данный курсовой проект направлен на создание удобного и производительного инструмента для автоматизации работы автосалона.

**Глава 1.**

**Теоретическая часть**

* 1. **Изучение предметной области**

Современный рынок автомобилей требует от автосалонов высокой скорости обслуживания клиентов, точного учета автомобилей и гибкости в управлении заказами. Успешное управление этими процессами становится возможным только благодаря использованию современных информационных систем. Именно поэтому создание консольного приложения для автоматизации деятельности автосалона является актуальной задачей.

Приложение для автосалона предоставляет пользователю удобный интерфейс для работы с базой данных автомобилей, клиентов, заказов и сотрудников. Оно позволяет решать задачи учета, поиска и анализа данных, минимизируя ошибки и упрощая рутинные процессы. Особенно важно, что приложение доступно для использования как небольшими, так и крупными автосалонами благодаря своей гибкости и масштабируемости.

Простота и эффективность интерфейса такого приложения играют ключевую роль. Оно должно быть интуитивно понятным и не требовать от пользователя глубокой технической подготовки. Основной функционал приложения сосредоточен на автоматизации учета автомобилей, заказов и работы с клиентами.

**Цели использования приложения для автосалона:**

1. Автоматизация процессов управления автосалоном.
2. Единый стандарт учета данных о клиентах, заказах и автомобилях.
3. Повышение скорости и точности обработки данных.
4. Упрощение поиска информации по заданным критериям.

**Основные требования к приложению:**

1. Простота использования и интуитивно понятный интерфейс.
2. Высокая производительность при работе с большими объемами данных.
3. Возможность работы с данными клиентов, автомобилей и заказов через единую систему.

В автосалонах ежедневно ведется учет автомобилей, доступных для продажи, обработка заявок от клиентов и оформление заказов. Эти процессы требуют значительных временных и трудозатрат, особенно при использовании традиционных методов, таких как бумажные журналы или электронные таблицы. Автоматизация позволяет не только ускорить выполнение этих задач, но и исключить ошибки, связанные с человеческим фактором.

* 1. **Изучение существующих разработок**

В рамках изучения существующих разработок я провёл поиск консольных приложений, ориентированных на автоматизацию работы автосалонов. Основное внимание было уделено приложениям, которые могли бы выполнять задачи управления сотрудниками, клиентами, автомобилями и заказами. Также были рассмотрены решения, поддерживающие базовые функции, такие как добавление, редактирование, удаление данных и обработка заказов.

Однако в ходе исследования не удалось найти готового решения, полностью соответствующего поставленным задачам.

Исходя из этого, было принято решение о разработке собственного консольного приложения, которое будет включать все необходимые функции и удовлетворять требованиям, сформулированным для данного курсового проекта.

**Глава 2**

**Проектная часть**

**2.1 Построение диаграммы прецедентов**

**2.2 Выбор инструментов**

При выборе языков программирования, я исходил из своих критериев.

Важность критерия выбрана из: низкая, ниже средней, средняя, выше средней, высокая.

*Таблица 1*

Критерии выбора языка программирования

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Критерий | Опыт | Простота сопровождения | Понятный синтаксис | Скорость разработки | Перспективы дальнейшего изучения | Скорость выполнения кода |
| Важность критерия | Высокая | Выше среднего | Выше среднего | Средняя | Высокая | Выше среднего |

Исходя из этих критериев, я сравнил 3 языка программирования от 0 до 10 баллов за критерий, где 0 – самый низкий балл и 10 – самый высокий.

*Таблица 2*

Сравнительный анализ языков программирования, основанный на собственных предпочтениях.

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий/Язык программирования | C | Python | Rust |
| Опыт | 2 | 6 | 10 |
| Простота сопровождения | 3 | 6 | 10 |
| Понятный синтаксис | 6 | 8 | 10 |
| Скорость разработки | 2 | 5 | 10 |
| Перспективы дальнейшего изучения | 5 | 10 | 10 |
| Скорость выполнения кода | 10 | 4 | 10 |
| Итого баллов | 28 | 39 | 60 |

По результатам сравнения я выбрал язык программирования Rust.

При выборе базы данных, я исходил из своих критериев.

Важность критерия выбрана из: низкая, ниже средней, средняя, выше средней, высокая.

*Таблица 3*

Критерии выбора языка программирования

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Критерий | Опыт | Производительность при обработке данных | Поддержка сложных операций и запросов | Надёжность и соответствие ACID | Расширяемость и функциональные возможности | Масштабируемость и работа с большими данными |
| Важность критерия | Высокая | Выше среднего | Средняя | Средняя | Выше среднего | Высокая |

Исходя из этих критериев, я сравнил 3 языка программирования от 0 до 10 баллов за критерий, где 0 – самый низкий балл и 10 – самый высокий.

*Таблица 4*

Сравнительный анализ баз данных, основанный на собственных предпочтениях.

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий/Язык программирования | МуSQL | PostgreSQL | MongoDB |
| Опыт | 5 | 10 | 8 |
| Производительность при обработке данных | 7 | 10 | 8 |
| Поддержка сложных операций и запросов | 6 | 10 | 5 |
| Надёжность и соответствие ACID | 8 | 10 | 6 |
| Расширяемость и функциональные возможности | 7 | 10 | 8 |
| Масштабируемость и работа с большими данными | 7 | 10 | 8 |
| Итого баллов | 40 | 60 | 43 |

По результатам сравнения я выбрал базу данных PostgreSQL.

**2.3 Проектирование сценария**

В данном разделе приведен сценарий использования программы пользователем.

До

От

Проверка

Ввод пароля

Заказ

Машина

Работник

Клиент

Новое имя

Новая должность

Новая зарплата

Новая дата трудоустройства

Ошибка

Успешно

Новая цена

Новое состояние

Новая дата

Новый employee\_id

Новый customer\_id

Новый car\_id

Новое состояние

Новый пробег

Новая цена

Новый год машины

Новая модель

Новое марка

|  |
| --- |
|  |
|  | C:\Users\RUDKO\AppData\Local\Microsoft\Windows\Clipboard\HistoryData\{99A71424-CFFE-40AD-BC9D-BF5312743783}\{F5D53D6F-290D-46C8-9A14-834368272A48}\ResourceMap\{A621DE14-45C7-4DE5-9D09-44A6BA1B92D3} |

Ввод id

Новое имя

Новый номер

Email

Бюджет

Выход

Удалить

Редактировать

Выход

Заказ

Автомобиль

Работник

Клиент

Вывод искомого объекта

Ввод id

Не найден

Выход

Поиск заказа по id

Поиск машины по id

Поиск работника по id

Поиск покупателя по id

Ошибка

Автомобиль создан

Ввод состояния

Ввод пробега

Ввод цены

Ввод года выпуска

Ввод модели

Ввод марки

Ввод состояния

Ввод марки

Вывод машин

Ввод id

Ввод пробега

Ввод цены

Ввод года выпуска

Ввод модели

Ввод цены

Выход

По бюджету

По фильтрам

Ввод неверного id

Клиент выбран

Ввод id

Ввод бюджета

Ввод email

Ввод телефона

Ввод имени

Ввод id

Найти по фильтрам

Вывести всех клиентов

Введите бюджет

Клиент создан

Введите email

Введите телефон

Введите имя клиента

Ввод неверного id

Закрытие заказа

Ввод id заказа

Ввод цены

Ввод id машины

Не выбран id клиента

Ввод неверного id

Выход из программы

Ввод верного id

Подобрать автомобиль

Добавить машину

Поиск

Настройки

Выход

Выбрать клиента

Добавить клиента

Закрыть заказ

Создать заказ

Создание нового работника

Войти

Запуск программы

Ниже приведу пример одного из множества вариантов использования приложения. Задача: нужно удалить работника с id1 8, т.к. его уволили.

Пользователь запускает программу, выбирает «войти» (у него уже есть своя учетная запись) и вводит свой id. Затем выбирает «настройки», «работник» и «редактировать». Вводит 8 (это id работника, которого нужно удалить), программа проверяет под каким id зашел пользователь, т.к. невозможно удалить учетную запись под которой он зашел. Если id не совпадают, то пользователь вводит пароль для удаления сотрудника и происходит удаление (если ввели неправильный пароль, то программа возвращает в главное меню и удаление не происходит). Затем программа выводит его в главное меню, и он нажимает «выход».

id1 - аббревиатура от английского слова identifier, которое переводится как «идентификатор». Индивидуальный номер.

**2.4 Построение базы данных**

Построим схему базы данных. В моей базе данных 4 таблицы: клиент, сотрудник, автомобиль, заказ.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | Клиент | | id | | Имя | | Телефон | | Email\* | | Бюджет | | |  | | --- | | Сотрудник | | id | | Имя | | Должность | | Зарплата | | Дата трудоустройства | | |  | | --- | | Машина | | id | | Марка | | Модель | | Год | | Цена | | Пробег | | Состояние | | |  | | --- | | Заказ | | id | | id машины | | id покупателя | | id сотрудника | | Дата создания | | Цена | | Актуальность | |

Из выше указанных таблиц, делаем вывод, что:

1) Для каждого клиента необходимо заполнить поля «Имя», «Телефон и «Бюджет». Email является опциональным.

2) Для каждого работника необходимо заполнить поля «Имя», «Должность», «Зарплата» и «Дата трудоустройства».

3) Для каждой машины необходимо заполнить поля «Марка», «Модель», «Год», «Цена», «Пробег» и «Состояние».

4) Для каждого заказа необходимо заполнить поля «id машины», «id покупателя», «id сотрудника», «Дата создания», «Цена» и «Актуальность».

5) Вводить id не нужно, т.к. с новой записью он автоматически увеличивается на 1.

**ГЛАВА 3**

**ЧАСТЬ РЕАЛИЗАЦИИ**

**3.1 Начало разработки**

Для реализации проекта «Автосалон» будет использована специализированная интегрированная среда разработки (IDE) от компании JetBrains, предназначенная для работы с языком программирования Rust.

RustRover имеет глубокую интеграцию с языком Rust и имеет множество встроенных инструментов для упрощения написания кода.

Для начала в файле Cargo.toml в разделе «dependencies» нужно написать библиотеки, которые будут использованы в проекте:

Листинг 1

[dependencies]  
dotenv = **"0.15" //для переменных из .env**tokio = { version = **"1"**, features = [**"full"**] } **//асинхронное программирование**  
tokio-postgres = **"0.7" //асинхронный клиент для PostreSQL**serde = { version = **"1"**, features = [**"derive"**] } **//для сериализации и десериализации данных**sqlx = { version = **"0.6"**, features = [**"postgres"**, **"runtime-tokio-native-tls"**, **"migrate"**] } **//ORM**  
async-std = **"1.13.0" //альтернатива tokio**chrono = **"0.4.38" //для работы с датами**

Определение функции в main.rs, которая будет подключаться к базе данных PostgreSQL и возвращать объект подключения:

Листинг 2

use tokio\_postgres::NoTls

use dotenv::dotenv;  
use std::env;

let **database\_url** = env::var(**"DATABASE\_URL"**).expect(**"DATABASE\_URL must be set in .env file"**);  
  
let (**client**, **connection**) = tokio\_postgres::connect(&**database\_url**, NoTls).await?;  
tokio::spawn(async move **{** if let ***Err***(**e**) = **connection**.await **{** eprintln!(**"Connection error: {}"**, **e**);  
 **}  
}**);

Переменная *connection* представляет соединение с базой данных, которая обрабатывает входящие и исходящие данные.

Переменная *client* используется для выполнения запросов к базе данных.

**3.2 Описание главного модуля**

В главный модуль подключаются 7 основных модулей: модуль автомобиля, модуль клиента, модуль сотрудника, модуль заказа, модуль моделей, модуль настроек и модуль поиска. Так же в этом модуле происходит подключение к базе данных, создание таблиц (если они не созданы) и создание учетной записи или вход в учетную запись сотрудника.

Кроме того, подключаются 4 вспомогательных модуля: модуль перечислений, модуль prelude (для удобного импорта модулей), модуль валидации и модуль приостановления программы.

Главный модуль объединяет работу всех основных и вспомогательных модулей, обеспечивая их взаимодействие и инкапсулируя общую логику программы.

Данный модуль состоит из одного кода, который приведен в листинге 1

***Листинг 1***

use tokio\_postgres::**{**NoTls, Error**}**;  
use std::io::**{**self, Write**}**;  
use dotenv::dotenv;  
use std::env;  
use chrono::Local;  
  
mod prelude;  
use prelude::prelude\_main::\*;  
  
  
mod customer;  
mod employee;  
mod order;  
mod car;  
mod models;  
mod validation;  
mod enums;  
mod settings;  
mod sleep;  
  
#[tokio::main]  
async fn main() -> Result<(), Error> **{** dotenv().ok();  
  
 let **database\_url** = env::var(**"DATABASE\_URL"**).expect(**"DATABASE\_URL must be set in .env file"**);  
  
 let (**client**, **connection**) = tokio\_postgres::connect(&**database\_url**, NoTls).await?;  
 tokio::spawn(async move **{** if let ***Err***(**e**) = **connection**.await **{** eprintln!(**"Connection error: {}"**, **e**);  
 **}  
 }**);  
  
 let mut **current\_id\_empl**: i32 = *0*;  
 let mut **current\_id\_cust**: i32 = *0*;  
  
 // Employee  
 **client**.execute(  
 **"CREATE TABLE IF NOT EXISTS employee (  
 id SERIAL PRIMARY KEY,  
 name VARCHAR(100) NOT NULL,  
 position VARCHAR(50) NOT NULL,  
 salary INTEGER NOT NULL,  
 hire\_date VARCHAR(20) NOT NULL  
 )"**,  
 &[],  
 ).await?;  
  
 // Customer  
 **client**.execute(  
 **"CREATE TABLE IF NOT EXISTS customer (  
 id SERIAL PRIMARY KEY,  
 name VARCHAR(100) NOT NULL,  
 phone VARCHAR(50) NOT NULL,  
 email VARCHAR(100) ,  
 budget INTEGER NOT NULL  
 )"**,  
 &[],  
 ).await?;  
  
 // Car  
 **client**.execute(  
 **"CREATE TABLE IF NOT EXISTS car (  
 id SERIAL PRIMARY KEY,  
 brand VARCHAR(100) NOT NULL,  
 model VARCHAR(100) NOT NULL,  
 year INTEGER NOT NULL,  
 price INTEGER NOT NULL,  
 mileage INTEGER NOT NULL,  
 is\_new BOOLEAN NOT NULL DEFAULT FALSE  
 )"**,  
 &[],  
 ).await?;  
  
 // Order  
 **client**.execute(  
 **"CREATE TABLE IF NOT EXISTS orders (  
 id SERIAL PRIMARY KEY,  
 car\_id INTEGER NOT NULL,  
 customer\_id INTEGER NOT NULL,  
 employee\_id INTEGER NOT NULL,  
 order\_date VARCHAR(20) NOT NULL,  
 price INTEGER NOT NULL,  
 is\_active BOOLEAN DEFAULT true  
 )"**,  
 &[],  
 ).await?;  
  
 loop **{** let mut **input** = String::*new*();  
  
 println!(**"1 - Войти**\n**2 - Создать работника"**);  
 io::stdin().read\_line(&mut **input**).expect(**"Ошибка при чтении"**);  
  
 match **input**.as\_str().trim() **{  
 "1"** => **{** //войти  
 println!(**"Напишите ваш id: "**);  
 let mut **id** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **id**).unwrap();  
 **id** = **id**.trim().parse().unwrap();  
 **current\_id\_empl** = **id**.trim().parse().expect(**"Ошибка"**);  
  
 let **name** = get\_employee(&**client**, **current\_id\_empl**).await;  
  
 match **name {  
 *Ok***(***Some***(**employee**)) => **{** println!(**"**\n\n\n\n**id: {}"**, **employee**.**id**.unwrap\_or\_default());  
 println!(**"name: {}"**, **employee**.**name**);  
 println!(**"position: {}"**, **employee**.**position**);  
 println!(**"salary: {}"**, **employee**.**salary**);  
 println!(**"hire\_date: {}"**, **employee**.**hire\_date**);  
 **}**,  
 ***Ok***(***None***) => **{** println!(**"Такого сотрудника не существует."**);  
 std::process::exit(*1*);  
 **}**,  
 ***Err***(**err**) => **{** println!(**"Ошибка при выполнении запроса: {:?}"**, **err**);  
 std::process::exit(*1*);  
 **}  
 }** break;  
 **}**,  
 **"2"** => **{**//создать работника  
 let **name1** = loop **{** let mut **input** = String::*new*();  
 println!(**"**\n\n\n**Введите Ваше имя:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 let **trimmed** = **input**.trim();  
 if !**trimmed**.is\_empty() **{** break **trimmed**.to\_string();  
 **}** else **{** println!(**"**\n\n**Ошибка: Ваше имя не может быть пустым. Пожалуйста, повторите ввод."**);  
 **}  
 }**;  
  
 let **position1** = loop **{** let mut **input** = String::*new*();  
 println!(**"**\n\n\n**Введите Вашу должность:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 let **trimmed** = **input**.trim();  
 if !**trimmed**.is\_empty() **{** break **trimmed**.to\_string();  
 **}** else **{** println!(**"**\n\n**Ошибка: Ваша должность не может быть пустой. Пожалуйста, повторите ввод."**);  
 **}  
 }**;  
  
 let **salary1** = loop **{** let mut **input** = String::*new*();  
 println!(**"**\n\n\n**Введите Вашу зарплату:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 match **input**.trim().parse::<i32>() **{  
 *Ok***(**salary**) => break **salary**,  
 ***Err***(\_) => println!(**"**\n\n**Ошибка: зарплата должна быть числом. Пожалуйста, повторите ввод.**\n\n**"**),  
 **}  
 }**;  
  
 let **date** = Local::*now*().format(**"%d.%m.%Y"**).to\_string();  
  
 let **new\_empl** = Employee **{  
 id**: ***None***,  
 **name**: **name1**,  
 **position**: **position1**,  
 **salary**: **salary1**,  
 **hire\_date**: **date**,  
 **}**;  
  
 match create\_employee(&**client**, **new\_empl**).await **{  
 *Ok***(**employee\_id**) => **{** println!(**"**\n\n\n**Новый сотрудник создан с ID: {}"**, **employee\_id**);  
 **current\_id\_empl** = **employee\_id**;  
 **}**,  
 ***Err***(**err**) => eprintln!(**"**\n\n\n**Ошибка при добавлении сотрудника: {:?}"**, **err**),  
 **}** break;  
 **}**,  
 \_ => **{** println!(**"**\n\n\n**Неверный ввод**\n\n\n**"**);  
 **input**.clear();  
 **}  
 }** sleep\_700mil().await;  
 **}** sleep\_700mil().await;  
  
 loop **{** let mut **choose** = String::*new*();  
  
 println!(**"**\n\n\n**1 - Создать заказ"**);  
 println!(**"2 - Закрыть заказ"**);  
 println!(**"3 - Добавить клиента"**);  
 println!(**"4 - Выбрать клиента"**);  
 println!(**"5 - Подобрать автомобиль"**);  
 println!(**"6 - Добавить автомобиль"**);  
 println!(**"7 - Поиск"**);  
 println!(**"8 - Настройки"**);  
 println!(**"9 - Выход"**);  
  
 io::stdin().read\_line(&mut **choose**).expect(**"Ошибка"**);  
  
 match **choose**.as\_str().trim() **{  
 "1"** => **{** //создать заказ  
 loop **{** match new\_order(**current\_id\_cust**, **current\_id\_empl**, &**client**).await **{  
 *Ok***(**order\_id**) => **{** if **order\_id** != *0* **{** sleep\_700mil().await;  
 break  
 **}** else **{** println!(**"Заказ не был создан."**);  
 sleep\_700mil().await;  
 break  
 **}  
 }  
 *Err***(**err**) => **{** eprintln!(**"Ошибка при создании заказа: {:?}"**, **err**);  
 break  
 **}  
 }  
 }  
 }**,  
 **"2"** => **{**//закрыть заказ  
 match close\_order(&**client**).await **{  
 *Ok***(\_) => sleep\_700mil().await,  
 ***Err***(\_) => sleep\_700mil().await,  
 **}  
 }**,  
 **"3"** => **{**//добавить клиента  
 loop **{** match add\_customer(&**client**, &mut **current\_id\_cust**).await **{  
 *Ok***(**customer\_id**) => **{  
 current\_id\_cust** = **customer\_id**;  
 sleep\_700mil().await;  
 break  
 **}**,  
 ***Err***(**err**) => **{** eprintln!(**"Не удалось добавить клиента: {:?}"**, **err**);  
 break  
 **}**,  
 **}  
 }  
 }**,  
 **"4"** => **{** //выбрать клиента  
 loop **{** match choose\_customer(&**client**, &mut **current\_id\_cust**).await **{  
 *Ok***(***Some***(**customer**)) => **{** println!(**"**\n\n\n**Выбран клиент:"**);  
 println!(**"id: {}"**, **customer**.**id**.unwrap\_or(*0*));  
 println!(**"Имя: {}"**, **customer**.**name**);  
 println!(**"Телефон: {}"**, **customer**.**phone**);  
 println!(  
 **"Email: {}"**,  
 **customer**.**email**.unwrap\_or\_else(|| **"Не указан"**.to\_string())  
 );  
 println!(**"Бюджет: {}"**, **customer**.**budget**);  
 break;  
 **}  
 *Ok***(***None***) => **{** println!(**"**\n\n\n**Клиент не был выбран. Попробуйте снова."**);  
 **}  
 *Err***(**err**) => **{** eprintln!(**"**\n\n\n**Ошибка при выборе клиента: {:?}"**, **err**);  
 break;  
 **}  
 }  
 }  
 }**,  
 **"5"** => **{** //подобрать автомобиль  
 loop **{** match find\_car(&**client**).await **{  
 *Ok***(()) => **{** println!(**"**\n\n\n**Подбор автомобиля завершён."**);  
 break;  
 **}  
 *Err***(**err**) => **{** eprintln!(**"**\n\n\n**Ошибка при подборе автомобиля: {:?}"**, **err**);  
 break;  
 **}  
 }  
 }  
 }**,  
 **"6"** => **{** //добавить автомобиль  
 loop **{** match add\_car(&**client**).await **{  
 *Ok***(\_) => **{** sleep\_700mil().await;  
 break;  
 **}  
 *Err***(**err**) => **{** eprintln!(**"Ошибка при добавлении автомобиля: {:?}"**, **err**);  
 break;  
 **}  
 }  
 }  
 }**,  
 **"7"** => **{** //поиск  
 loop **{** match search(&**client**).await **{  
 *Ok***(\_) => break,  
 ***Err***(**err**) => **{** eprintln!(**"Ошибка при выполнении поиска: {:?}"**, **err**);  
 break;  
 **}  
 }  
 }  
 }**,  
 **"8"** => **{**//настройки  
 loop **{** match settings(&**client**, **current\_id\_empl**).await **{  
 *Ok***(\_) => **{** println!(**"Возврат в главное меню..."**);  
 sleep\_700mil().await;  
 break;  
 **}  
 *Err***(**err**) => **{** eprintln!(**"Ошибка в настройках: {:?}"**, **err**);  
 sleep\_700mil().await;  
 break;  
 **}  
 }  
 }  
 }**,  
 **"9"** => **{** println!(**"**\n\n\n**Всего хорошего! До свидания!"**);  
 break  
 **}** \_ => input\_error().await  
 **}  
 }  
 *Ok***(())  
**}**

**3.3 Описание спецификаций к модулям**

Всего разработано 11 модулей, 7 из которых основных, а 4 второстепенных.

7 основных:

* Модуль автомобиля
* Модуль клиента
* Модуль сотрудника
* Модуль заказа
* Модуль моделей
* Модуль настроек
* Модуль поиска

4 второстепенных:

* Модуль перечислений
* Модуль prelude (для удобного импорта модулей)
* Модуль валидации
* Модуль приостановки программы

**3.4 Описание модуля «Клиент»**

В этом модуле описаны функции, которые относятся к клиенту.

*Листинг 1. Функция создания*

pub async fn create\_customer(client: &Client, customer: Customer) -> Result<i32, Error> **{** let **row** = client.query\_one(  
 **"INSERT INTO customer (name, phone, email, budget) VALUES ($1, $2, $3, $4) RETURNING id"**,  
 &[&customer.**name**, &customer.**phone**, &customer.**email**, &customer.**budget**],  
 ).await?;  
 ***Ok***(**row**.get(*0*))  
**}**

*Листинг 2. Функция редактирования*

pub async fn create\_customer(client: &Client, customer: Customer) -> Result<i32, Error> **{** let **row** = client.query\_one(  
 **"INSERT INTO customer (name, phone, email, budget) VALUES ($1, $2, $3, $4) RETURNING id"**,  
 &[&customer.**name**, &customer.**phone**, &customer.**email**, &customer.**budget**],  
 ).await?;  
 ***Ok***(**row**.get(*0*))  
**}**

*Листинг 3. Функция удаления*

pub async fn delete\_customer(client: &Client, customer\_id: i32) -> Result<String, Error> **{** let **result** = client.execute(  
 **"DELETE FROM customer WHERE id = $1"**,  
 &[&customer\_id],  
 ).await?;  
  
 if **result** > *0* **{  
 *Ok***(**"**\n\n\n**Изменения успешно сохранены"**.to\_string())  
 **}** else **{  
 *Ok***(**"**\n\n\n**Удаление не выполнено"**.to\_string())  
 **}  
}**

*Листинг 4. Функция поиска по id*

pub async fn get\_customer(client: &Client, customer\_id: i32) -> Result<Option<Customer>, Error> **{** let **row** = client.query\_opt(  
 **"SELECT id, name, phone, email, budget FROM customer WHERE id = $1"**,  
 &[&customer\_id],  
 ).await?;  
  
 if let ***Some***(**row**) = **row {  
 *Ok***(***Some***(Customer **{  
 id**: **row**.get(*0*),  
 **name**: **row**.get(*1*),  
 **phone**: **row**.get(*2*),  
 **email**: **row**.get(*3*),  
 **budget**: **row**.get(*4*),  
 **}**))  
 **}** else **{  
 *Ok***(***None***)  
 **}  
}**

*Листинг 5. Функция получения всех клиентов*

pub async fn get\_customer(client: &Client, customer\_id: i32) -> Result<Option<Customer>, Error> **{** let **row** = client.query\_opt(  
 **"SELECT id, name, phone, email, budget FROM customer WHERE id = $1"**,  
 &[&customer\_id],  
 ).await?;  
  
 if let ***Some***(**row**) = **row {  
 *Ok***(***Some***(Customer **{  
 id**: **row**.get(*0*),  
 **name**: **row**.get(*1*),  
 **phone**: **row**.get(*2*),  
 **email**: **row**.get(*3*),  
 **budget**: **row**.get(*4*),  
 **}**))  
 **}** else **{  
 *Ok***(***None***)  
 **}  
}**

*Листинг 6. Функция поиска по фильтрам*

pub async fn search\_customer(  
 client: &Client,  
 id: Option<i32>,  
 name: Option<String>,  
 phone: Option<String>,  
 email: Option<String>,  
 budget: Option<i32>,  
) -> Result<Vec<Customer>, Error> **{** let mut **query** = String::*from*(  
 **"SELECT id, name, phone, email, budget FROM customer WHERE 1=1"** );  
 let mut **params**: Vec<&(dyn tokio\_postgres::types::ToSql + Sync)> = Vec::*new*();  
  
 let mut **id1** = ***None***;  
 let **name1**;  
 let **phone1**;  
 let **email1**;  
 let **budget1**;  
  
 if let ***Some***(**id**) = id **{  
 query**.push\_str(&format!(**" AND id = ${}"**, **params**.len() + *1*));  
 **id1** = ***Some***(**id**);  
 **params**.push(&**id1**);  
 **}** if let ***Some***(**name**) = name.as\_deref() **{  
 query**.push\_str(&format!(**" AND name = ${}"**, **params**.len() + *1*));  
 **name1** = **name**.to\_string();  
 **params**.push(&**name1**);  
 **}** if let ***Some***(**phone**) = phone.as\_deref() **{  
 query**.push\_str(&format!(**" AND phone = ${}"**, **params**.len() + *1*));  
 **phone1** = **phone**.to\_string();  
 **params**.push(&**phone1**);  
 **}** if let ***Some***(**email**) = email.as\_deref() **{  
 query**.push\_str(&format!(**" AND email = ${}"**, **params**.len() + *1*));  
 **email1** = **email**.to\_string();  
 **params**.push(&**email1**);  
 **}** if let ***Some***(**budget**) = budget **{  
 query**.push\_str(&format!(**" AND budget = ${}"**, **params**.len() + *1*));  
 **budget1** = ***Some***(**budget**);  
 **params**.push(&**budget1**);  
 **}** let **rows** = client.query(&**query**, &**params**).await?;  
  
 let **customers** = **rows** .iter()  
 .map(|row| Customer **{  
 id**: row.get(*0*),  
 **name**: row.get(*1*),  
 **phone**: row.get(*2*),  
 **email**: row.get(*3*),  
 **budget**: row.get(*4*),  
 **}**)  
 .collect();  
  
 ***Ok***(**customers**)  
**}**

*Листинг 7. Функция создания с интерфейсом*

pub async fn add\_customer(client: &Client, *current\_id\_cust*: &mut i32) -> Result<i32, Error> **{** let **name1** = loop **{** let mut **input** = String::*new*();  
 println!(**"**\n\n\n**Введите имя клиента:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 let **trimmed** = **input**.trim();  
 if !**trimmed**.is\_empty() **{** break **trimmed**.to\_string();  
 **}** else **{** println!(**"**\n\n**Ошибка: имя клиента не может быть пустым. Пожалуйста, повторите ввод."**);  
 **}  
 }**;  
  
 let **phone1** = loop **{** let mut **input** = String::*new*();  
 println!(**"Введите телефон клиента:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 let **trimmed** = **input**.trim();  
 if !**trimmed**.is\_empty() **{** break **trimmed**.to\_string();  
 **}** else **{** println!(**"**\n\n**Ошибка: телефон клиента не может быть пустым. Пожалуйста, повторите ввод.**\n\n**"**);  
 **}  
 }**;  
  
 let **email1** = loop **{** let mut **input** = String::*new*();  
 println!(**"Введите email клиента(необязательно):"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 let **trimmed** = **input**.trim();  
 if **trimmed**.is\_empty() **{** break ***None***;  
 **}** else **{** match valid\_email(**trimmed**.to\_string()).await **{  
 *Ok***(\_) => break ***Some***(**trimmed**.to\_string()),  
 ***Err***(**err**) => println!(**"**\n\n**Ошибка: {}. Пожалуйста, повторите ввод.**\n\n**"**, **err**),  
 **}  
 }  
 }**;  
  
 let **budget1** = loop **{** let mut **input** = String::*new*();  
 println!(**"Введите бюджет клиента:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 match **input**.trim().parse::<i32>() **{  
 *Ok***(**budget**) => break **budget**,  
 ***Err***(\_) => println!(**"**\n\n**Ошибка: бюджет должен быть числом. Пожалуйста, повторите ввод.**\n\n**"**),  
 **}  
 }**;  
  
 let **new\_cust** = Customer **{  
 id**: ***None***,  
 **name**: **name1**,  
 **phone**: **phone1**,  
 **email**: **email1**,  
 **budget**: **budget1**,  
 **}**;  
  
 match create\_customer(&client, **new\_cust**).await **{  
 *Ok***(**customer\_id**) => **{** println!(**"**\n\n\n**Создан новый клиент с id: {}"**, **customer\_id**);  
 \**current\_id\_cust* = **customer\_id**;  
 ***Ok***(**customer\_id**)  
 **}**,  
 ***Err***(**err**) => **{** println!(**"Ошибка при создании нового клиента: {}"**, **err**);  
 ***Err***(**err**)  
 **}  
 }  
}**

*Листинг 8. Выбор клиента по фильтрам с интерфейсом*

pub async fn choose\_customer(client: &Client, *current\_id\_cust*: &mut i32) -> Result<Option<Customer>, Error> **{** let mut **choice** = String::*new*(); //***TODO*** println!(**"**\n\n\n**1 - Вывести всех клиентов"**);  
 println!(**"2 - Найти клиента по фильтрам"**);  
  
 io::stdin().read\_line(&mut **choice**).expect(**"Ошибка"**);  
  
 match **choice**.as\_str().trim() **{  
 "1"** => **{** //вывести всех клиентов  
 match get\_all\_customers(&client).await **{  
 *Ok***(**customers**) => **{** if **customers**.is\_empty() **{** println!(**"Клиенты отсутствуют."**);  
 return ***Ok***(***None***);  
 **}** for **customer** in &**customers {** println!(**"**\n\n**id: {}"**, &**customer**.**id**.unwrap\_or(*0*));  
 println!(**"Имя: {}"**, &**customer**.**name**);  
 println!(**"Телефон: {}"**, &**customer**.**phone**);  
 println!(  
 **"email: {}"**,  
 **customer**.**email**.clone().unwrap\_or\_else(|| **"Не указан"**.to\_string())  
 );  
 println!(**"Бюджет: {}**\n\n**"**, **customer**.**budget**);  
 **}** println!(**"**\n\n\n**Введите id клиента для выбора:"**);  
 let mut **id\_input** = String::*new*();  
 io::stdin().read\_line(&mut **id\_input**).unwrap();  
 let **id** = **id\_input**.trim().parse::<i32>().ok();  
  
  
 if let ***Some***(**id**) = **id {** if let ***Some***(**customer**) = **customers**.into\_iter().find(|c| c.**id** == ***Some***(**id**)) **{** \**current\_id\_cust* = **id**;  
 ***Ok***(***Some***(**customer**))  
 **}** else **{** println!(**"Клиент с таким id не найден."**);  
 ***Ok***(***None***)  
 **}  
 }** else **{** println!(**"Некорректный ввод id."**);  
 ***Ok***(***None***)  
 **}  
 }  
 *Err***(**err**) => **{** eprintln!(**"Ошибка при получении клиентов: {:?}"**, **err**);  
 ***Err***(**err**)  
 **}  
 }  
 }**,  
 **"2"** => **{** //клиент по фильтрам  
 let mut **input** = String::*new*();  
  
 let mut **id1**: Option<i32> = ***None***;  
 let mut **name1**: Option<String> = ***None***;  
 let mut **phone1**: Option<String> = ***None***;  
 let mut **email1**: Option<String> = ***None***;  
 let mut **budget1**: Option<i32> = ***None***;  
 loop **{** println!(**"**\n\n\n**1) id: {:?}"**, **id1**);  
 println!(**"2) name: {:?}"**, **name1**);  
 println!(**"3) phone: {:?}"**, **phone1**);  
 println!(**"4) email: {:?}"**, **email1**);  
 println!(**"5) budget: {:?}"**, **budget1**);  
 println!(**"6) Готово"**);  
  
 **input**.clear();  
 io::stdin().read\_line(&mut **input**).unwrap();  
  
 match **input**.as\_str().trim() **{  
 "1"** => **{** let mut **id\_str** = String::*new*();  
 println!(**"Введите id:"**);  
 io::stdin().read\_line(&mut **id\_str**).unwrap();  
 **id1** = **id\_str**.trim().parse::<i32>().ok();  
 **}**,  
 **"2"** => **{** let mut **name\_str** = String::*new*();  
 println!(**"Введите имя:"**);  
 io::stdin().read\_line(&mut **name\_str**).unwrap();  
 **name1** = ***Some***(**name\_str**.trim().to\_string());  
 **}**,  
 **"3"** => **{** let mut **phone\_str** = String::*new*();  
 println!(**"Введите телефон:"**);  
 io::stdin().read\_line(&mut **phone\_str**).unwrap();  
 **phone1** = ***Some***(**phone\_str**.trim().to\_string());  
 **}**,  
 **"4"** => **{** let mut **email\_str** = String::*new*();  
 println!(**"Введите email:"**);  
 io::stdin().read\_line(&mut **email\_str**).unwrap();  
 **email1** = ***Some***(**email\_str**.trim().to\_string());  
 **}  
 "5"** => **{** let mut **budget\_str** = String::*new*();  
 println!(**"Введите бюджет:"**);  
 io::stdin().read\_line(&mut **budget\_str**).unwrap();  
 **budget1** = **budget\_str**.trim().parse::<i32>().ok();  
 **}**,  
 **"6"** => break,  
 \_ => println!(**"Неверный ввод"**)  
 **}  
 }** match search\_customer(&client, **id1**, **name1**, **phone1**, **email1**, **budget1**).await **{  
 *Ok***(**customers**) => **{** if **customers**.is\_empty() **{** println!(**"Клиенты не найдены."**);  
 return ***Ok***(***None***);  
 **}** else **{** for **customer** in &**customers {** println!(**"{}"**, **customer**.pretty\_print\_customers());  
 **}** println!(**"**\n\n\n**Введите id клиента для выбора:"**);  
 let mut **id\_input** = String::*new*();  
 io::stdin().read\_line(&mut **id\_input**).unwrap();  
 let **id** = **id\_input**.trim().parse::<i32>().ok();  
 \**current\_id\_cust* = **id**.clone().unwrap();  
  
 if let ***Some***(**id**) = **id {** if let ***Some***(**customer**) = **customers**.into\_iter().find(|c| c.**id** == ***Some***(**id**)) **{** return ***Ok***(***Some***(**customer**));  
 **}** else **{** println!(**"Клиент с таким id не найден."**);  
 **}  
 }** else **{** println!(**"Некорректный ввод id."**);  
 **}  
 }  
 }  
 *Err***(**err**) => **{** eprintln!(**"Ошибка при поиске клиентов: {:?}"**, **err**);  
 return ***Err***(**err**)  
 **}  
 }  
 *Ok***(***None***)  
 **}**,  
 \_ => **{** input\_error().await;  
 ***Ok***(***None***)  
 **}  
 }  
}**

*Листинг 9. mod.rs для экспорта модуля*

pub mod customer;  
pub mod add\_customer;  
pub mod choose\_customer;

**3.5 Описание модуля «Сотрудник»**

В этом модуле описаны функции, которые относятся к сотруднику.

*Листинг 1. Функция создания*

pub async fn create\_employee(client: &Client, employee: Employee) -> Result<i32, Error> **{** let **row** = client.query\_one(  
 **"INSERT INTO employee (name, position, salary, hire\_date) VALUES ($1, $2, $3, $4) RETURNING id"**,  
 &[&employee.**name**, &employee.**position**, &employee.**salary**, &employee.**hire\_date**],  
 ).await?;  
 ***Ok***(**row**.get(*0*))  
**}**

*Листинг 2. Функция редактирования*

pub async fn update\_employee(client: &Client, employee\_id: i32, updated\_employee: Employee) -> Result<String, Error> **{** let **result** = client.execute(  
 **"UPDATE employee SET name = $2, position = $3, salary = $4, hire\_date = $5 WHERE id = $1"**,  
 &[&employee\_id, &updated\_employee.**name**, &updated\_employee.**position**, &updated\_employee.**salary**, &updated\_employee.**hire\_date**],  
 ).await?;  
 if **result** > *0* **{  
 *Ok***(**"**\n\n\n**Изменения успешно сохранены"**.to\_string())  
 **}** else **{  
 *Ok***(**"**\n\n\n**Изменения не применены"**.to\_string())  
 **}  
}**

*Листинг 3. Функция удаления*

pub async fn delete\_employee(client: &Client, employee\_id: i32) -> Result<String, Error> **{** let **result** = client.execute(  
 **"DELETE FROM employee WHERE id = $1"**,  
 &[&employee\_id],  
 ).await?;  
  
 if **result** > *0* **{  
 *Ok***(**"**\n\n\n**Изменения успешно сохранены"**.to\_string())  
 **}** else **{  
 *Ok***(**"**\n\n\n**Удаление не выполнено"**.to\_string())  
 **}  
}**

*Листинг 4. Функция поиска по id*

pub async fn get\_employee(client: &Client, employee\_id: i32) -> Result<Option<Employee>, Error> **{** let **row** = client.query\_opt(  
 **"SELECT id, name, position, salary, hire\_date FROM employee WHERE id = $1"**,  
 &[&employee\_id],  
 ).await?;  
  
 if let ***Some***(**row**) = **row {  
 *Ok***(***Some***(Employee **{  
 id**: **row**.get(*0*),  
 **name**: **row**.get(*1*),  
 **position**: **row**.get(*2*),  
 **salary**: **row**.get(*3*),  
 **hire\_date**: **row**.get(*4*),  
 **}**))  
 **}** else **{  
 *Ok***(***None***)  
 **}  
}**

*Листинг 5. mod.rs для экспорта модуля*

pub mod employee;

**3.6 Описание модуля «Автомобиль»**

В этом модуле описаны функции, которые относятся к автомобилю.

*Листинг 1. Функция создания*

pub async fn create\_car(client: &Client, car: Car) -> Result<i32, Error> **{** let **row** = client.query\_one(  
 **"INSERT INTO car (brand, model, year, price, mileage, is\_new) VALUES ($1, $2, $3, $4, $5, $6) RETURNING id"**,  
 &[&car.**brand**, &car.**model**, &car.**year**, &car.**price**, &car.**mileage**, &car.**is\_new**],  
 ).await?;  
 ***Ok***(**row**.get(*0*))  
**}**

*Листинг 2. Функция редактирования*

pub async fn update\_car(client: &Client, car\_id: i32, updated\_car: Car) -> Result<String, Error> **{** let **result** = client.execute(  
 **"UPDATE car SET brand = $2, model = $3, year = $4, price = $5, mileage = $6, is\_new = $7 WHERE id = $1"**,  
 &[&car\_id, &updated\_car.**brand**, &updated\_car.**model**, &updated\_car.**year**, &updated\_car.**price**, &updated\_car.**mileage**, &updated\_car.**is\_new**],  
 ).await?;  
 if **result** > *0* **{  
 *Ok***(**"**\n\n\n**Изменения успешно сохранены"**.to\_string())  
 **}** else **{  
 *Ok***(**"**\n\n\n**Изменения не применены"**.to\_string())  
 **}  
}**

*Листинг 3. Функция удаления*

pub async fn delete\_car(client: &Client, car\_id: i32) -> Result<String, Error> **{** let **result** = client.execute(  
 **"DELETE FROM car WHERE id = $1"**,  
 &[&car\_id],  
 ).await?;  
 if **result** > *0* **{  
 *Ok***(**"**\n\n\n**Изменения успешно сохранены"**.to\_string())  
 **}** else **{  
 *Ok***(**"**\n\n\n**Удаление не выполнено"**.to\_string())  
 **}  
}**

*Листинг 4. Функция поиска по id*

pub async fn get\_car(client: &Client, car\_id: i32) -> Result<Option<Car>, Error> **{** let **row** = client.query\_opt(  
 **"SELECT id, brand, model, year, price, mileage, is\_new FROM car WHERE id = $1"**,  
 &[&car\_id],  
 ).await?;  
  
 if let ***Some***(**row**) = **row {  
 *Ok***(***Some***(Car **{  
 id**: **row**.get(*0*),  
 **brand**: **row**.get(*1*),  
 **model**: **row**.get(*2*),  
 **year**: **row**.get(*3*),  
 **price**: **row**.get(*4*),  
 **mileage**: **row**.get(*5*),  
 **is\_new**: **row**.get(*6*),  
 **}**))  
 **}** else **{  
 *Ok***(***None***)  
 **}  
}**

*Листинг 5. Функция поиска ниже определенной стоимости*

pub async fn budget\_car\_under(client: &Client, budget: i32) -> Result<Vec<Car>, Error> **{** let **query** = **"SELECT id, brand, model, year, price, mileage, is\_new FROM car WHERE price <= $1"**;  
  
 let **rows** = client.query(**query**, &[&budget]).await?;  
  
 let **cars** = **rows** .iter()  
 .map(|row| Car **{  
 id**: row.get(*0*),  
 **brand**: row.get(*1*),  
 **model**: row.get(*2*),  
 **year**: row.get(*3*),  
 **price**: row.get(*4*),  
 **mileage**: row.get(*5*),  
 **is\_new**: row.get(*6*),  
 **}**)  
 .collect();  
  
 ***Ok***(**cars**)  
**}**

*Листинг 6. Функция поиска выше определенной стоимости*

pub async fn budget\_car\_upper(client: &Client, budget: i32) -> Result<Vec<Car>, Error> **{** let **query** = **"SELECT id, brand, model, year, price, mileage, is\_new FROM car WHERE price >= $1"**;  
  
 let **rows** = client.query(**query**, &[&budget]).await?;  
  
 let **cars** = **rows** .iter()  
 .map(|row| Car **{  
 id**: row.get(*0*),  
 **brand**: row.get(*1*),  
 **model**: row.get(*2*),  
 **year**: row.get(*3*),  
 **price**: row.get(*4*),  
 **mileage**: row.get(*5*),  
 **is\_new**: row.get(*6*),  
 **}**)  
 .collect();  
  
 ***Ok***(**cars**)  
**}**

*Листинг 7. Функция поиска по фильтрам*

pub async fn search\_cars(  
 client: &Client,  
 id: Option<i32>,  
 brand: Option<String>,  
 model: Option<String>,  
 year: Option<i32>,  
 price: Option<i32>,  
 mileage: Option<i32>,  
 is\_new: Option<bool>,  
) -> Result<Vec<Car>, Error> **{** let mut **query** = String::*from*(  
 **"SELECT id, brand, model, year, price, mileage, is\_new FROM car WHERE 1=1"** );  
 let mut **params**: Vec<&(dyn tokio\_postgres::types::ToSql + Sync)> = Vec::*new*();  
  
 let mut **id1** = ***None***;  
 let **brand1** ;  
 let **model1** ;  
 let **year1**;  
 let **price1**;  
 let **mileage1** ;  
 let **is\_new1** ;  
  
 if let ***Some***(**id**) = id **{  
 query**.push\_str(&format!(**" AND id = {}"**, **params**.len() + *1*));  
 **id1** = ***Some***(**id**);  
 **params**.push(&**id1**);  
 **}** if let ***Some***(**brand**) = brand **{  
 query**.push\_str(&format!(**" AND brand ILIKE ${}"**, **params**.len() + *1*));  
 **brand1** = format!(**"%{}%"**, **brand**.trim());  
 **params**.push(&**brand1**);  
 **}** if let ***Some***(**model**) = model **{  
 query**.push\_str(&format!(**" AND model ILIKE ${}"**, **params**.len() + *1*));  
 **model1** = format!(**"%{}%"**, **model**.trim());  
 **params**.push(&**model1**);  
 **}** if let ***Some***(**year**) = year **{  
 query**.push\_str(&format!(**" AND year = ${}"**, **params**.len() + *1*));  
 **year1** = ***Some***(**year**);  
 **params**.push(&**year1**);  
 **}** if let ***Some***(**price**) = price **{  
 query**.push\_str(&format!(**" AND price = ${}"**, **params**.len() + *1*));  
 **price1** = ***Some***(**price**);  
 **params**.push(&**price1**);  
 **}** if let ***Some***(**mileage**) = mileage **{  
 query**.push\_str(&format!(**" AND mileage = ${}"**, **params**.len() + *1*));  
 **mileage1** = ***Some***(**mileage**);  
 **params**.push(&**mileage1**);  
 **}** if let ***Some***(**is\_new**) = is\_new **{  
 query**.push\_str(&format!(**" AND is\_new = ${}"**, **params**.len() + *1*));  
 **is\_new1** = ***Some***(**is\_new**);  
 **params**.push(&**is\_new1**);  
 **}** let **rows** = client.query(&**query**, &**params**).await?;  
  
 let **cars** = **rows** .iter()  
 .map(|row| Car **{  
 id**: row.get(*0*),  
 **brand**: row.get(*1*),  
 **model**: row.get(*2*),  
 **year**: row.get(*3*),  
 **price**: row.get(*4*),  
 **mileage**: row.get(*5*),  
 **is\_new**: row.get(*6*),  
 **}**)  
 .collect();  
  
 ***Ok***(**cars**)  
**}**

*Листинг 8. Создание с интерфейсом*

pub async fn add\_car(client: &Client) -> Result<(), Error> **{** let **brand** = loop **{** let mut **input** = String::*new*();  
 println!(**"**\n\n\n**Введите марку автомобиля:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 let **trimmed** = **input**.trim();  
 if !**trimmed**.is\_empty() **{** break **trimmed**.to\_string();  
 **}** else **{** println!(**"**\n\n**Ошибка: марка автомобиля не может быть пустой. Пожалуйста, повторите ввод."**);  
 **}  
 }**;  
  
 let **model** = loop **{** let mut **input** = String::*new*();  
 println!(**"Введите модель автомобиля:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 let **trimmed** = **input**.trim();  
 if !**trimmed**.is\_empty() **{** break **trimmed**.to\_string();  
 **}** else **{** println!(**"**\n\n**Ошибка: модель автомобиля не может быть пустой. Пожалуйста, повторите ввод.**\n\n**"**);  
 **}  
 }**;  
  
 let **year** = loop **{** let mut **input** = String::*new*();  
 println!(**"Введите год производства:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 match **input**.trim().parse::<i32>() **{  
 *Ok***(**year1**) => break **year1**,  
 ***Err***(\_) => println!(**"**\n\n**Ошибка: год производства должен быть числом. Пожалуйста, повторите ввод.**\n\n**"**),  
 **}  
 }**;  
  
 let **price** = loop **{** let mut **input** = String::*new*();  
 println!(**"Введите цену автомобиля:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 match **input**.trim().parse::<i32>() **{  
 *Ok***(**price1**) => break **price1**,  
 ***Err***(\_) => println!(**"**\n\n**Ошибка: цена автомобиля должна быть числом. Пожалуйста, повторите ввод.**\n\n**"**),  
 **}  
 }**;  
  
 let **mileage** = loop **{** let mut **input** = String::*new*();  
 println!(**"Введите пробег автомобиля:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 match **input**.trim().parse::<i32>() **{  
 *Ok***(**mileage1**) => break **mileage1**,  
 ***Err***(\_) => println!(**"**\n\n**Ошибка: пробег автомобиля должен быть числом. Пожалуйста, повторите ввод.**\n\n**"**),  
 **}  
 }**;  
  
 let **is\_new1** = loop **{** let mut **is\_new\_str** = String::*new*();  
 println!(**"Введите 0 (Новый автомобиль) или 1 (Не новый автомобиль):"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **is\_new\_str**).unwrap();  
  
 match **is\_new\_str**.trim() **{  
 "0"** => break ***Some***(true),  
 **"1"** => break ***Some***(false),  
 \_ => **{** println!(**"**\n\n**Ошибка: неверный ввод. Введите 0 или 1.**\n\n**"**);  
 **}  
 }  
 }**;  
  
 let **is\_new** = **is\_new1**.is\_some();  
  
 let **new\_car** = Car **{  
 id**: ***None***,  
 brand,  
 model,  
 year,  
 price,  
 mileage,  
 is\_new,  
 **}**;  
  
 match create\_car(&client, **new\_car**).await **{  
 *Ok***(**car\_id**) => **{** println!(**"**\n\n\n**Создан новый автомобиль с id: {}"**, **car\_id**);  
 ***Ok***(())  
 **}**,  
 ***Err***(**err**) => **{** eprintln!(**"**\n\n\n**Ошибка при добавлении автомобиля: {}"**, **err**);  
 ***Err***(**err**)  
 **}  
 }  
}**

*Листинг 9. Поиск по фильтрам с интерфейсом*

pub async fn find\_car(client: &Client, ) -> Result<(), Error> **{** let mut **int** = String::*new*();  
 println!(**"**\n\n\n**1 - По бюджету"**);  
 println!(**"2 - По определенным фильтрам"**);  
 println!(**"3 - Выход"**);  
  
 io::stdin().read\_line(&mut **int**).expect(**"Ошибка"**);  
  
  
  
 match **int**.as\_str().trim() **{  
 "1"** => **{** //по бюджету  
 let mut **int1** = String::*new*();  
  
 println!(**"1 - До"**);  
 println!(**"2 - От"**);  
  
 io::stdin().read\_line(&mut **int1**).expect(**"Ошибка"**);  
  
 match **int1**.as\_str().trim() **{  
 "1"** => **{** //до  
 let mut **budget1** = String::*new*();  
 println!(**"Введите сумму до которой Вы готовы приобрести автомобиль:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **budget1**).expect(**"Ошибка при чтении ввода"**);  
  
 match **budget1**.trim().parse::<i32>() **{  
 *Ok***(**budget**) => **{** match budget\_car\_under(&client, **budget**).await **{  
 *Ok***(**cars**) => **{** if **cars**.is\_empty() **{** println!(**"Нет автомобилей с ценой до {}"**, **budget**);  
 **}** else **{** println!(**"Автомобили в бюджете до {}:"**, **budget**);  
 for **car** in **cars {** println!(**"id: {}"**, **car**.**id**.unwrap\_or(*0*));  
 println!(**"Марка: {}"**, **car**.**brand**);  
 println!(**"Модель: {}"**, **car**.**model**);  
 println!(**"Год: {}"**, **car**.**year**);  
 println!(**"Цена: {}"**, **car**.**price**);  
 println!(**"Пробег: {}"**, **car**.**mileage**);  
 println!(**"Новый: {}"**, if **car**.**is\_new { "Да" }** else **{ "Нет" }**);  
 println!();  
 **}  
 }  
 }  
 *Err***(**err**) => eprintln!(**"Ошибка при получении автомобилей: {:?}"**, **err**),  
 **}  
 }  
 *Err***(\_) => **{** println!(**"Пожалуйста, введите корректное число."**);  
 **}  
 }  
 }**,  
 **"2"** => **{** //от  
 let mut **budget1** = String::*new*();  
 println!(**"Введите сумму от которой Вы готовы приобрести автомобиль:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **budget1**).expect(**"Ошибка при чтении ввода"**);  
  
 match **budget1**.trim().parse::<i32>() **{  
 *Ok***(**budget**) => **{** match budget\_car\_upper(&client, **budget**).await **{  
 *Ok***(**cars**) => **{** if **cars**.is\_empty() **{** println!(**"Нет автомобилей с ценой от {}"**, **budget**);  
 **}** else **{** println!(**"Автомобили в бюджете от {}:"**, **budget**);  
 for **car** in **cars {** println!(**"id: {}"**, **car**.**id**.unwrap\_or(*0*));  
 println!(**"Марка: {}"**, **car**.**brand**);  
 println!(**"Модель: {}"**, **car**.**model**);  
 println!(**"Год: {}"**, **car**.**year**);  
 println!(**"Цена: {}"**, **car**.**price**);  
 println!(**"Пробег: {}"**, **car**.**mileage**);  
 println!(**"Новый: {}"**, if **car**.**is\_new { "Да" }** else **{ "Нет" }**);  
 println!();  
 **}  
 }  
 }  
 *Err***(**err**) => eprintln!(**"Ошибка при получении автомобилей: {:?}"**, **err**),  
 **}  
 }  
 *Err***(\_) => **{** println!(**"Пожалуйста, введите корректное число."**);  
 **}  
 }  
 }**,  
 \_ => println!(**"Введите корректное значение"**)  
 **}  
 }**,  
 **"2"** => **{** //определенные фильтры для автомобиля  
 let mut **input** = String::*new*();  
  
 let mut **id1**: Option<i32> = ***None***;  
 let mut **brand1**: Option<String> = ***None***;  
 let mut **model1**: Option<String> = ***None***;  
 let mut **year1**: Option<i32> = ***None***;  
 let mut **price1**: Option<i32> = ***None***;  
 let mut **mileage1**: Option<i32> = ***None***;  
 let mut **is\_new**: Option<bool> = ***None***;  
  
 loop **{** println!(**"**\n\n\n**1) id: {:?}"**, **id1**);  
 println!(**"2) Марка: {:?}"**, **brand1**);  
 println!(**"3) Модель: {:?}"**, **model1**);  
 println!(**"4) Год: {:?}"**, **year1**);  
 println!(**"5) Цена: {:?}"**, **price1**);  
 println!(**"6) Пробег: {:?}"**, **mileage1**);  
 println!(**"7) Новый: {:?}"**, **is\_new**);  
 println!(**"8) Готово"**);  
  
 **input**.clear();  
 io::stdin().read\_line(&mut **input**).unwrap();  
  
 match **input**.as\_str().trim() **{  
 "1"** => **{** let mut **id\_str** = String::*new*();  
 println!(**"Введите id:"**);  
 io::stdin().read\_line(&mut **id\_str**).unwrap();  
 **id1** = **id\_str**.trim().parse::<i32>().ok();  
 **}**,  
 **"2"** => **{** let mut **brand\_str** = String::*new*();  
 println!(**"Введите марку:"**);  
 io::stdin().read\_line(&mut **brand\_str**).unwrap();  
 **brand1** = ***Some***(**brand\_str**.trim().to\_string());  
 **}**,  
 **"3"** => **{** let mut **model\_str** = String::*new*();  
 println!(**"Введите телефон:"**);  
 io::stdin().read\_line(&mut **model\_str**).unwrap();  
 **model1** = ***Some***(**model\_str**.trim().to\_string());  
 **}**,  
 **"4"** => **{** let mut **year\_str** = String::*new*();  
 println!(**"Введите год:"**);  
 io::stdin().read\_line(&mut **year\_str**).unwrap();  
 **year1** = **year\_str**.trim().parse::<i32>().ok();  
 **}**,  
 **"5"** => **{** let mut **price\_str** = String::*new*();  
 println!(**"Введите бюджет:"**);  
 io::stdin().read\_line(&mut **price\_str**).unwrap();  
 **price1** = **price\_str**.trim().parse::<i32>().ok();  
 **}**,  
 **"6"** => **{** let mut **mileage\_str** = String::*new*();  
 println!(**"Введите пробег"**);  
 io::stdin().read\_line(&mut **mileage\_str**).unwrap();  
 **mileage1** = **mileage\_str**.trim().parse::<i32>().ok();  
 **}**,  
 **"7"** => **{** let mut **is\_new\_str** = String::*new*();  
 println!(**"Введите 0 (Новый автомобиль) или 1 (Не новый автомобиль):"**);  
 io::stdin().read\_line(&mut **is\_new\_str**).unwrap();  
  
 match **is\_new\_str**.trim() **{  
 "0"** => **is\_new** = ***Some***(true),  
 **"1"** => **is\_new** = ***Some***(false),  
 \_ => **{** println!(**"Неверный ввод. Введите 0 или 1."**);  
 **is\_new** = ***None***;  
 **}  
 }  
 }**,  
 **"8"** => break,  
 \_ => println!(**"Неверный ввод"**)  
 **}  
 }** match search\_cars(&client, **id1**, **brand1**, **model1**, **year1**, **price1**, **mileage1**, **is\_new**).await **{  
 *Ok***(**cars**) => **{** if **cars**.is\_empty() **{** println!(**"Автомобили не найдены"**)  
 **}** else **{** for **car** in **cars {** println!(**"{}"**, **car**.pretty\_print\_car());  
 **}  
 }  
 }  
 *Err***(**err**) => eprintln!(**"Ошибка при поиске автомобиля: {:?}"**, **err**)  
 **}  
 }**,  
 **"3"** => **{** println!(**"Выход из поиска автомобилей."**);  
 return ***Ok***(());  
 **}** \_ => println!(**"Некорректный выбор. Попробуйте снова."**),  
 **}  
 *Ok***(())  
**}**

*Листинг 10. mod.rs для экспорта модуля*

pub mod car;  
pub mod find\_car;  
pub mod add\_car;

**3.7 Описание модуля «Заказ»**

В этом модуле описаны функции, которые относятся к заказу.

*Листинг 1. Функция создания*

pub async fn create\_order(client: &Client, order: Order) -> Result<i32, Error> **{** let **row** = client.query\_one(  
 **"INSERT INTO orders (car\_id, customer\_id, employee\_id, order\_date, price, is\_active)**\  
 **VALUES ($1, $2, $3, $4, $5, $6)**\  
 **RETURNING id"**,  
 &[&order.**car\_id**, &order.**customer\_id**, &order.**employee\_id**, &order.**order\_date**.to\_string(), &order.**price**, &order.**is\_active**],  
 ).await?;  
 ***Ok***(**row**.get(*0*))  
**}**

*Листинг 2. Функция редактирования*

pub async fn update\_order(client: &Client, order\_id: i32, updated\_order: Order) -> Result<String, Error> **{** let **result** = client.execute(  
 **"UPDATE orders SET car\_id = $2, customer\_id = $3, employee\_id = $4,** \  
 **order\_date = $5, price = $6, is\_active = $7 WHERE id = $1"**,  
 &[  
 &order\_id,  
 &updated\_order.**car\_id**,  
 &updated\_order.**customer\_id**,  
 &updated\_order.**employee\_id**,  
 &updated\_order.**order\_date**.to\_string(),  
 &updated\_order.**price**,  
 &updated\_order.**is\_active** ],  
 ).await?;  
  
 if **result** > *0* **{  
 *Ok***(**"**\n\n\n**Изменения успешно сохранены"**.to\_string())  
 **}** else **{  
 *Ok***(**"**\n\n\n**Изменения не применены"**.to\_string())  
 **}  
}**

*Листинг 3. Функция удаления*

pub async fn delete\_order(client: &Client, order\_id: i32) -> Result<String, Error> **{** let **result** = client.execute(  
 **"DELETE FROM orders WHERE id = $1"**,  
 &[&order\_id],  
 ).await?;  
  
 if **result** > *0* **{  
 *Ok***(**"**\n\n\n**Изменения успешно сохранены"**.to\_string())  
 **}** else **{  
 *Ok***(**"**\n\n\n**Удаление не выполнено"**.to\_string())  
 **}  
}**

*Листинг 4. Функция поиска по id*

pub async fn get\_order(client: &Client, order\_id: i32) -> Result<Option<Order>, Error> **{** let **row** = client.query\_opt(  
 **"SELECT id, car\_id, customer\_id, employee\_id, order\_date, price, is\_active FROM orders WHERE id = $1"**,  
 &[&order\_id],  
 ).await?;  
  
 if let ***Some***(**row**) = **row {  
 *Ok***(***Some***(Order **{  
 id**: **row**.get(*0*),  
 **car\_id**: **row**.get(*1*),  
 **customer\_id**: **row**.get(*2*),  
 **employee\_id**: **row**.get(*3*),  
 **order\_date**: **row**.get(*4*),  
 **price**: **row**.get(*5*),  
 **is\_active**: **row**.get(*6*),  
 **}**))  
 **}** else **{  
 *Ok***(***None***)  
 **}  
}**

*Листинг 5. Функция закрытия*

pub async fn cancel\_order(client: &Client, order\_id: i32) -> Result<String, Error> **{** let **result** = client.execute(  
 **"UPDATE orders SET is\_active = false WHERE id = $1"**,  
 &[&order\_id],  
 ).await?;  
  
 if **result** > *0* **{  
 *Ok***(**"**\n\n\n**Заказ успешно закрыт"**.to\_string())  
 **}** else **{  
 *Ok***(**"**\n\n\n**Закрытие заказа не выполнено"**.to\_string())  
 **}  
}**

*Листинг 6. Функция создания с интерфейсом*

pub async fn new\_order(current\_id\_cust: i32, current\_id\_empl: i32, client: &Client) -> Result<i32, Error> **{** if current\_id\_cust == *0* **{** println!(**"Для создания заказ необходимо выбрать клиента"**);  
 return ***Ok***(*0*);  
 **}** let mut **price** = String::*new*();  
  
 let **car\_id** = loop **{** let mut **input** = String::*new*();  
 println!(**"**\n\n\n**Введите id машины:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 match **input**.trim().parse::<i32>() **{  
 *Ok***(**car1**) => break **car1**,  
 ***Err***(\_) => println!(**"**\n\n**Ошибка: id машины должен быть числом. Пожалуйста, повторите ввод.**\n\n**"**),  
 **}  
 }**;  
  
 println!(**"Введите цену:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **price**).unwrap();  
 let **price** = from\_str\_to\_int(&**price**).await;  
  
 let **order\_date** = Local::*now*().format(**"%d.%m.%Y"**).to\_string();  
  
 let **new\_order** = Order **{  
 id**: ***None***,  
 car\_id,  
 **customer\_id**: current\_id\_cust,  
 **employee\_id**: current\_id\_empl,  
 order\_date,  
 price,  
 **is\_active**: true,  
 **}**;  
  
 match create\_order(&client, **new\_order**).await **{  
 *Ok***(**order\_id**) => **{** println!(**"**\n\n**Новый заказ создан c id: {}"**, **order\_id**);  
 ***Ok***(**order\_id**)  
 **}**,  
 ***Err***(**err**) => **{** eprintln!(**"**\n\n**Ошибка при создании нового заказа: {}"**, **err**);  
 ***Err***(**err**)  
 **}  
 }  
}**

*Листинг 7. Функция закрытия с интерфейсом*

pub async fn close\_order(client: &Client) -> Result<(), Error> **{** let **order\_id** = loop **{** let mut **input** = String::*new*();  
 println!(**"Введите id заказа, который нужно закрыть:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 match **input**.trim().parse::<i32>() **{  
 *Ok***(**order\_id1**) => break **order\_id1**,  
 ***Err***(\_) => println!(**"**\n\n**Ошибка: id должен быть числом. Пожалуйста, повторите ввод.**\n\n**"**)  
 **}  
 }**;  
  
 match cancel\_order(client, **order\_id**).await **{  
 *Ok***(**message**) => println!(**"{}"**, **message**),  
 ***Err***(**err**) => eprintln!(**"Ошибка при закрытии заказа: {:?}"**, **err**),  
 **}  
  
 *Ok***(())  
**}**

*Листинг 8. mod.rs для экспорта модуля*

pub mod order;  
pub mod new\_order;  
pub mod close\_order;

**3.8 Описание модуля «Модели»**

В этом модуле представлены модели клиента, сотрудника, машины и заказа, а также имплементации для понятного вывода объкта при его выводе.

*Листинг 1. Модули*

#[derive(Debug)]  
pub struct Car **{** pub **id**: Option<i32>,  
 pub **brand**: String,  
 pub **model**: String,  
 pub **year**: i32,  
 pub **price**: i32,  
 pub **mileage**: i32,  
 pub **is\_new**: bool,  
**}**impl Car **{** pub fn pretty\_print\_car(&self) -> String **{** format!(  
 **"**\n\n\n**id: {}**\n**brand: {}**\n**model: {}**\n**year: {}**\n**price: {}**\n**mileage: {}**\n**is\_new: {}"**,  
 self.**id**.unwrap\_or\_default(),  
 self.**brand**,  
 self.**model**,  
 self.**year**,  
 self.**price**,  
 self.**mileage**,  
 if self.**is\_new { "Новый" }** else **{ "Не новый" }** )  
 **}  
}**#[derive(Debug)]  
pub struct Customer **{** pub **id**: Option<i32>,  
 pub **name**: String,  
 pub **phone**: String,  
 pub **email**: Option<String>,  
 pub **budget**: i32,  
**}**impl Customer **{** pub fn pretty\_print\_customers(&self) -> String **{** format!(  
 **"**\n\n\n**id: {}**\n**name: {}**\n**phone: {}**\n**email: {}**\n**budget: {}"**,  
 self.**id**.unwrap\_or\_default(),  
 self.**name**,  
 self.**phone**,  
 self.**email**.clone().unwrap\_or\_else(|| String::*from*(**"Не указан"**)),  
 self.**budget** )  
 **}  
}**#[derive(Debug)]  
pub struct Employee **{** pub **id**: Option<i32>,  
 pub **name**: String,  
 pub **position**: String,  
 pub **salary**: i32,  
 pub **hire\_date**: String,  
**}**#[derive(Debug)]  
pub struct Order **{** pub **id**: Option<i32>,  
 pub **car\_id**: i32,  
 pub **customer\_id**: i32,  
 pub **employee\_id**: i32,  
 pub **order\_date**: String,  
 pub **price**: i32,  
 pub **is\_active**: bool,  
**}**

*Листинг 2. mod.rs для экспотра модуля*

pub mod models;

**3.9 Описание модуля «Настройки»**

В этом модуле представлены функции для манипуляций с клиентом, сотрудником, машиной и заказом.

*Листинг 1. Функция настройки*

pub async fn settings(client: &Client, employee\_id: i32) -> Result<(), Error> **{** let mut **input** = String::*new*();  
  
 println!(**"**\n\n\n**1 - Клиент"**);  
 println!(**"2 - Работник"**);  
 println!(**"3 - Автомобиль"**);  
 println!(**"4 - Заказ"**);  
 println!(**"5 - Выход"**);  
  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
  
 match **input**.as\_str().trim() **{  
 "1"** => **{** //Клиент  
 settings\_cust(&client).await  
 **}**,  
 **"2"** => **{**//Работник  
 settings\_empl(&client, employee\_id).await  
 **}**,  
 **"3"** => **{** //Автомобиль  
 settings\_car(&client).await  
 **}**,  
 **"4"** => **{** //Заказ  
 settings\_order(&client).await  
 **}**,  
 **"5"** => **{** //Выход  
 ***Ok***(())  
 **}**,  
 \_ => **{** input\_error().await;  
 ***Ok***(())  
 **}  
 }  
}**

*Листинг 2. Функция настройки клиента*

pub async fn settings\_cust(client: &Client) -> Result<(), Error> **{** let mut **input1** = String::*new*();  
  
 println!(**"**\n\n\n**1 - Редактировать покупателя"**);  
 println!(**"2 - Удалить покупателя"**);  
 println!(**"3 - Выход"**);  
  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input1**).unwrap();  
  
 match **input1**.as\_str().trim() **{  
 "1"** => **{** //редактировать покупателя  
 let **customer\_id** = loop **{** let mut **input** = String::*new*();  
 println!(**"Введите id клиента, данные которой хотите обновить:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 match **input**.trim().parse::<i32>() **{  
 *Ok***(**customer\_id1**) => break **customer\_id1**,  
 ***Err***(\_) => println!(**"**\n\n**Ошибка: id должен быть числом. Пожалуйста, повторите ввод.**\n\n**"**),  
 **}  
 }**;  
  
 let mut **name1** = String::*new*();  
 let mut **phone1** = String::*new*();  
 let mut **email1**: Option<String> = Option::*from*(String::*new*());  
 let mut **budget1**: i32 = *0*;  
  
 match get\_customer(&client, **customer\_id**).await **{  
 *Ok***(***Some***(**customer**)) => **{  
 name1** = **customer**.**name**;  
 **phone1** = **customer**.**phone**;  
 **email1** = **customer**.**email**;  
 **budget1** = **customer**.**budget**;  
 **}  
 *Ok***(***None***) => **{** println!(**"Клиент с ID {} не найден."**, **customer\_id**);  
 **}  
 *Err***(**err**) => **{** eprintln!(**"Ошибка при получении клиента: {:?}"**, **err**);  
 **}  
 }** println!(**"**\n**Введите новое имя клиента:"**);  
 old\_value(&**name1**).await;  
 let mut **name\_input** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **name\_input**).unwrap();  
 let **name** = if **name\_input**.trim().is\_empty() **{  
 name1**.clone()  
 **}** else **{  
 name\_input**.trim().to\_string()  
 **}**;  
  
  
 println!(**"**\n**Введите новый номер телефона клиента:"**);  
 old\_value(&**phone1**).await;  
 let mut **phone\_input** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **phone\_input** ).unwrap();  
 let **phone** = if **phone\_input**.trim().is\_empty() **{  
 phone1**.clone()  
 **}** else **{  
 phone\_input**.trim().to\_string()  
 **}**;  
  
 println!(**"**\n**Введите новый email клиента:"**);  
 old\_value(**email1**.clone().unwrap\_or\_else(|| **"Не указан"**.to\_string())).await;  
 let mut **email\_input** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **email\_input** ).unwrap();  
 let **email** = if **email\_input**.trim().is\_empty() **{  
 email1**.clone()  
 **}** else **{** let **email\_input\_trimmed** = **email\_input**.trim().to\_string();  
  
 match valid\_email(**email\_input\_trimmed**.clone()).await **{  
 *Ok***(\_) => ***Some***(**email\_input\_trimmed**),  
 ***Err***(**err**) => **{** println!(**"Ошибка: {}"**, **err**);  
 ***None* }  
 }  
 }**;  
  
 println!(**"**\n**Введите новый бюджет клиента:"**);  
 old\_value(**budget1**).await;  
 let mut **budget\_input** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **budget\_input**).unwrap();  
 let **budget**: i32 = if **budget\_input**.trim().is\_empty() **{  
 budget1  
 }** else **{  
 budget\_input** .trim()  
 .parse()  
 .unwrap\_or\_else(|\_| **{** eprintln!(**"Ошибка: бюджет должен быть числом. Используем старое значение."**);  
 **budget1  
 }**)  
 **}**;  
  
 let **updated\_customer** = Customer **{  
 id**: ***Some***(**customer\_id**),  
 name,  
 phone,  
 email,  
 budget,  
 **}**;  
  
 match update\_customer(&client, **customer\_id**, **updated\_customer**).await **{  
 *Ok***(**message**) => **{** println!(**"{}"**, **message**);  
 ***Ok***(())  
 **}**,  
 ***Err***(**err**) => **{** eprintln!(**"Ошибка при обновлении клиента: {:?}"**, **err**);  
 ***Err***(**err**)  
 **}**,  
 **}  
 }**,  
 **"2"** => **{** //удалить покупателя  
 let mut **input\_str** = String::*new*();  
  
 println!(**"Введите id покупателя:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input\_str**).unwrap();  
  
 let **input2** = from\_str\_to\_int(**input\_str**.trim()).await;  
  
 match delete\_customer(&client, **input2**).await **{  
 *Ok***(**message**) => **{** println!(**"{}"**, **message**);  
 ***Ok***(())  
 **}**,  
 ***Err***(**err**) => **{** eprintln!(**"Ошибка при удалении клиента: {:?}"**, **err**);  
 ***Err***(**err**)  
 **}  
 }  
 }**,  
 **"3"** => ***Ok***(()),  
 \_ => **{** input\_error().await;  
 ***Ok***(())  
 **}  
 }  
}**

*Листинг 3. Функция настройки сотрудника*

pub async fn settings\_empl(client: &Client, employee\_id: i32) -> Result<(), Error> **{** let mut **input1** = String::*new*();  
  
 println!(**"**\n\n\n**1 - Редактировать сотрудника"**);  
 println!(**"2 - Удалить сотрудника"**);  
 println!(**"3 - Выход"**);  
  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input1**).unwrap();  
  
 match **input1**.as\_str().trim() **{  
 "1"** =>**{** //Редактировать работника  
 let **employee\_id** = loop **{** let mut **input** = String::*new*();  
 println!(**"Введите id сотрудника, данные которой хотите обновить:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 match **input**.trim().parse::<i32>() **{  
 *Ok***(**employee\_id1**) => break **employee\_id1**,  
 ***Err***(\_) => println!(**"**\n\n**Ошибка: id должен быть числом. Пожалуйста, повторите ввод.**\n\n**"**),  
 **}  
 }**;  
  
 let mut **name1** = String::*new*();  
 let mut **position1** = String::*new*();  
 let mut **salary1**: i32 = *0*;  
 let mut **hire\_date1** = String::*new*();  
  
 match get\_employee(&client, **employee\_id**).await **{  
 *Ok***(***Some***(**employee**)) => **{  
 name1** = **employee**.**name**;  
 **position1** = **employee**.**position**;  
 **salary1** = **employee**.**salary**;  
 **hire\_date1** = **employee**.**hire\_date**;  
 **}  
 *Ok***(***None***) => **{** println!(**"Работник с id {} не найден."**, **employee\_id**);  
 **}  
 *Err***(**err**) => **{** eprintln!(**"Ошибка при получении работника: {:?}"**, **err**);  
 **}  
 }** println!(**"**\n**Введите новое имя сотрудника:"**);  
 old\_value(&**name1**).await;  
 let mut **name\_input** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **name\_input**).unwrap();  
 let **name** = if **name\_input**.trim().is\_empty() **{  
 name1**.clone()  
 **}** else **{  
 name\_input**.trim().to\_string()  
 **}**;  
  
 println!(**"**\n**Введите новую должность сотрудника:"**);  
 old\_value(&**position1**).await;  
 let mut **position\_input** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **position\_input**).unwrap();  
 let **position** = if **position\_input**.trim().is\_empty() **{  
 position1**.clone()  
 **}** else **{  
 position\_input**.trim().to\_string()  
 **}**;  
  
 println!(**"**\n**Введите новую зарплату сотрудника:"**);  
 old\_value(**salary1**).await;  
 let mut **salary\_str** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **salary\_str**).unwrap();  
 let **salary**: i32 = if **salary\_str**.trim().is\_empty() **{  
 salary1  
 }** else **{  
 salary\_str** .trim()  
 .parse()  
 .unwrap\_or\_else(|\_| **{** eprintln!(**"Ошибка: зарплата должна быть числом. Используем старое значение."**);  
 **salary1  
 }**)  
 **}**;  
  
 println!(**"**\n**Введите новую дату, когда устроился на работу:"**);  
 old\_value(&**hire\_date1**).await;  
 let mut **hire\_date** = String::*new*();  
 loop **{** io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **hire\_date**).unwrap();  
 **hire\_date** = **hire\_date**.trim().to\_string();  
  
 if **hire\_date**.is\_empty() **{  
 hire\_date** = **hire\_date1**.clone();  
 break;  
 **}** match valid\_date(**hire\_date**.clone()).await **{  
 *Ok***(\_) => break,  
 ***Err***(**err**) => **{** println!(**"Ошибка: {}"**, **err**);  
 println!(**"Попробуйте снова. Введите дату в формате дд.мм.гггг:"**);  
 **hire\_date**.clear();  
 **}  
 }  
 }** let **updated\_employee** = Employee **{  
 id**: ***Some***(**employee\_id**),  
 name,  
 position,  
 salary,  
 hire\_date,  
 **}**;  
  
 match update\_employee(&client, **employee\_id**, **updated\_employee**).await **{  
 *Ok***(**message**) => **{** println!(**"{}"**, **message**);  
 ***Ok***(())  
 **}**,  
 ***Err***(**err**) => **{** eprintln!(**"Ошибка при обновлении клиента: {:?}"**, **err**);  
 ***Err***(**err**)  
 **}  
 }  
 }**,  
 **"2"** => **{** //Удалить работника  
 let mut **input\_str** = String::*new*();  
  
 println!(**"Введите id работника:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input\_str**).unwrap();  
  
 let **input2** = from\_str\_to\_int(**input\_str**.trim()).await;  
  
 if **input2** == employee\_id **{** println!(**"Вы не можете удалить этого работника, т.к. сейчас находитесь на этой учетной записи"**);  
 ***Ok***(())  
 **}** else **{** let mut **password** = String::*new*();  
 println!(**"Для удаления работника введите пароль:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **password**).unwrap();  
 let **password** = **password**.trim();  
 if **password** == **"qwerty123" {** match delete\_employee(&client, **input2**).await **{  
 *Ok***(**message**) => **{** println!(**"{}"**, **message**);  
 ***Ok***(())  
 **}**,  
 ***Err***(**err**) => **{** eprintln!(**"Ошибка при удалении работника: {:?}"**, **err**);  
 ***Err***(**err**)  
 **}  
 }  
 }** else **{** println!(**"Неверный пароль. Вы не можете удалить сотрудника"**);  
 ***Ok***(())  
 **}  
 }**  
 **}**,  
 **"3"** => ***Ok***(()),  
 \_ => **{** input\_error().await;  
 ***Ok***(())  
 **}  
 }  
}**

*Листинг 4. Функция настройки заказа*

pub async fn settings\_order(client: &Client) -> Result<(), Error> **{** let mut **input1** = String::*new*();  
  
 println!(**"**\n\n\n**1 - Редактировать заказ"**);  
 println!(**"2 - Удалить заказ"**);  
 println!(**"3 - Выход"**);  
  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input1**).unwrap();  
  
 match **input1**.as\_str().trim() **{  
 "1"** => **{** //Редактировать заказ  
 let **order\_id** = loop **{** let mut **input** = String::*new*();  
 println!(**"Введите id заказа, данные которой хотите обновить:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input**).unwrap();  
 match **input**.trim().parse::<i32>() **{  
 *Ok***(**order\_id1**) => break **order\_id1**,  
 ***Err***(\_) => println!(**"**\n\n**Ошибка: id должен быть числом. Пожалуйста, повторите ввод.**\n\n**"**),  
 **}  
 }**;  
  
 let mut **car\_id1**: i32 = *0*;  
 let mut **customer\_id1**: i32 = *0*;  
 let mut **employee\_id1**: i32 = *0*;  
 let mut **order\_date1** = String::*new*();  
 let mut **price1**: i32 = *0*;  
 let mut **is\_active1** = false;  
  
 match get\_order(&client, **order\_id**).await **{  
 *Ok***(***Some***(**order**)) => **{  
 car\_id1** = **order**.**car\_id**;  
 **customer\_id1** = **order**.**customer\_id**;  
 **employee\_id1** = **order**.**employee\_id**;  
 **order\_date1** = **order**.**order\_date**;  
 **price1** = **order**.**price**;  
 **is\_active1** = **order**.**is\_active**;  
 **}  
 *Ok***(***None***) => **{** println!(**"Заказ с id {} не найден."**, **order\_id**);  
 **}  
 *Err***(**err**) => **{** eprintln!(**"Ошибка при получении заказа: {:?}"**, **err**);  
 **}  
 }** println!(**"**\n**Введите car\_id:"**);  
 old\_value(**car\_id1**).await;  
 let mut **car\_id\_str** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **car\_id\_str**).unwrap();  
 let **car\_id**: i32 = if **car\_id\_str**.trim().is\_empty() **{  
 car\_id1  
 }** else **{  
 car\_id\_str** .trim()  
 .parse()  
 .unwrap\_or\_else(|\_| **{** eprintln!(**"Ошибка: car\_id должен быть числом. Используем старое значение."**);  
 **car\_id1  
 }**)  
 **}**;  
  
 println!(**"**\n**Введите customer\_id:"**);  
 old\_value(**customer\_id1**).await;  
 let mut **customer\_id\_str** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **customer\_id\_str**).unwrap();  
 let **customer\_id**: i32 = if **customer\_id\_str**.trim().is\_empty() **{  
 customer\_id1  
 }** else **{  
 customer\_id\_str** .trim()  
 .parse()  
 .unwrap\_or\_else(|\_| **{** eprintln!(**"Ошибка: customer\_id должен быть числом. Используем старое значение."**);  
 **customer\_id1  
 }**)  
 **}**;  
  
 println!(**"**\n**Введите employee\_id:"**);  
 old\_value(**employee\_id1**).await;  
 let mut **employee\_id\_str** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **employee\_id\_str**).unwrap();  
 let **employee\_id**: i32 = if **employee\_id\_str**.trim().is\_empty() **{  
 employee\_id1  
 }** else **{  
 employee\_id\_str** .trim()  
 .parse()  
 .unwrap\_or\_else(|\_| **{** eprintln!(**"Ошибка: customer\_id должен быть числом. Используем старое значение."**);  
 **employee\_id1  
 }**)  
 **}**;  
  
 println!(**"**\n**Введите новую дату, когда создали заказ:"**);  
 old\_value(&**order\_date1**).await;  
 let mut **order\_date** = String::*new*();  
 loop **{** io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **order\_date**).unwrap();  
 **order\_date** = **order\_date**.trim().to\_string();  
  
 if **order\_date**.is\_empty() **{  
 order\_date** = **order\_date1**.clone();  
 break;  
 **}** match valid\_date(**order\_date**.clone()).await **{  
 *Ok***(\_) => break,  
 ***Err***(**err**) => **{** println!(**"Ошибка: {}"**, **err**);  
 println!(**"Попробуйте снова. Введите дату в формате дд.мм.гггг:"**);  
 **order\_date**.clear();  
 **}  
 }  
 }** println!(**"**\n**Введите новую сумму заказа:"**);  
 old\_value(**price1**).await;  
 let mut **price\_str** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **price\_str**).unwrap();  
 let **price**: i32 = if **price\_str**.trim().is\_empty() **{  
 price1  
 }** else **{  
 price\_str** .trim()  
 .parse()  
 .unwrap\_or\_else(|\_| **{** eprintln!(**"Ошибка: сумма должна быть числом. Используем старое значение."**);  
 **price1  
 }**)  
 **}**;  
  
 println!(**"**\n**Введите новое состояние заказа:** \n\  
 **0 (Актуальный) или 1 (Неактуальный)"**);  
 old\_value(**is\_active1**).await;  
 let mut **is\_active\_input** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **is\_active\_input**).unwrap();  
 match **is\_active\_input**.trim() **{  
 "0"** => **is\_active1** = true,  
 **"1"** => **is\_active1** = false,  
 \_ => **{** println!(**"Неверный ввод. Введите 0 или 1."**);  
 **}  
 }** let **is\_active** = **is\_active1**;  
  
 let **updated\_order** = Order **{  
 id**: ***Some***(**order\_id**),  
 car\_id,  
 customer\_id,  
 employee\_id,  
 order\_date,  
 price,  
 is\_active  
 **}**;  
  
 match update\_order(&client, **order\_id**, **updated\_order**).await **{  
 *Ok***(**message**) => **{** println!(**"{}"**, **message**);  
 ***Ok***(())  
 **}**,  
 ***Err***(**err**) => **{** eprintln!(**"Ошибка при обновлении заказа: {:?}"**, **err**);  
 ***Err***(**err**)  
 **}  
 }  
 }**,  
 **"2"** => **{**//Удалить заказ  
 let mut **input\_str** = String::*new*();  
  
 println!(**"Введите id заказа:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input\_str**).unwrap();  
  
 let **input2** = from\_str\_to\_int(**input\_str**.trim());  
  
 match delete\_order(&client, **input2**.await).await **{  
 *Ok***(**message**) => **{** println!(**"{}"**, **message**);  
 ***Ok***(())  
 **}**,  
 ***Err***(**err**) => **{** eprintln!(**"Ошибка при удалении заказа: {:?}"**, **err**);  
 ***Err***(**err**)  
 **}  
 }  
 }**,  
 **"3"** => ***Ok***(()),  
 \_ => **{** input\_error().await;  
 ***Ok***(())  
 **}  
 }  
}**

*Листинг 5. mod.rs для экспорта модуля*

pub mod settings;  
mod settings\_cust;  
mod settings\_empl;  
mod settings\_car;  
mod settings\_order;

**3.10 Описание модуля «Поиск»**

В этом модуле представлены функции поиска покупателей, сотрудников, машин и заказов по id.

*Листинг 1. Поиск по id*

pub async fn search(client: &Client) -> Result<(), Error> **{** let mut **input** = String::*new*();  
  
 println!(**"**\n\n\n**1 - Поиск покупателя по id"**);  
 println!(**"2 - Поиск работника по id"**);  
 println!(**"3 - Поиск автомобиля по id"**);  
 println!(**"4 - Поиск заказа по id"**);  
 println!(**"5 - Выход"**);  
  
 io::stdin().read\_line(&mut **input**).expect(**"Ошибка"**);  
  
 match **input**.as\_str().trim() **{  
 "1"** => **{**//Поиск покупателя по id  
 let mut **id** = String::*new*();  
 println!(**"Введите id покупателя:"**);  
 io::stdin().read\_line(&mut **id**).unwrap();  
 let **id** = from\_str\_to\_int(&**id**);  
  
 match get\_customer(&client, **id**.await).await **{  
 *Ok***(***Some***(**customer**)) => **{** match **customer**.**id {  
 *Some***(**id**) => println!(**"**\n\n\n**id: {}"**, **id**),  
 ***None*** => println!(**"**\n\n\n**id: Не найдено"**)  
 **}** println!(**"Имя: {}"**, **customer**.**name**);  
 println!(**"Телефон: {}"**, **customer**.**phone**);  
 match **customer**.**email {  
 *Some***(**email**) => println!(**"Email: {}"**, **email**),  
 ***None*** => println!(**"Email: Не указано"**)  
 **}** println!(**"Бюджет: {}"**, **customer**.**budget**);  
 sleep\_700mil().await;  
 ***Ok***(())  
 **}  
 *Ok***(***None***) => **{** println!(**"Клиент не найден"**);  
 sleep\_700mil().await;  
 ***Ok***(())  
 **}  
 *Err***(**err**) => **{** eprintln!(**"Ошибка при получении клиента: {}"**, **err**);  
 sleep\_700mil().await;  
 ***Err***(**err**)  
 **}  
 }  
 }**,  
 **"2"** => **{**//Поиск работника по id  
 let mut **id** = String::*new*();  
 println!(**"Введите id работника:"**);  
 io::stdin().read\_line(&mut **id**).unwrap();  
 let **id** = from\_str\_to\_int(&**id**);  
  
 match get\_employee(&client, **id**.await).await **{  
 *Ok***(***Some***(**employee**)) => **{** match **employee**.**id {  
 *Some***(**id**) => println!(**"**\n\n\n**id: {}"**, **id**),  
 ***None*** => println!(**"**\n\n\n**id: Не найдено"**)  
 **}** println!(**"Имя: {}"**, **employee**.**name**);  
 println!(**"Должность: {}"**, **employee**.**position**);  
 println!(**"Зарплата: {}"**, **employee**.**salary**);  
 println!(**"Работает с {}"**, **employee**.**hire\_date**);  
 sleep\_700mil().await;  
 ***Ok***(())  
 **}  
 *Ok***(***None***) => **{** println!(**"Работник не найден"**);  
 sleep\_700mil().await;  
 ***Ok***(())  
 **}  
 *Err***(**err**) => **{** eprintln!(**"Ошибка при получении работника: {}"**, **err**);  
 sleep\_700mil().await;  
 ***Err***(**err**)  
 **}  
 }  
 }**,  
 **"3"** => **{**//Поиск автомобиля по id  
 let mut **id** = String::*new*();  
 println!(**"Введите id автомобиля:"**);  
 io::stdin().read\_line(&mut **id**).unwrap();  
 let **id** = from\_str\_to\_int(&**id**);  
  
 match get\_car(&client, **id**.await).await **{  
 *Ok***(***Some***(**car**)) => **{** match **car**.**id {  
 *Some***(**id**) => println!(**"**\n\n\n**id: {}"**, **id**),  
 ***None*** => println!(**"**\n\n\n**id: Не найдено"**)  
 **}** println!(**"Марка: {}"**, **car**.**brand**);  
 println!(**"Модель: {}"**, **car**.**model**);  
 println!(**"Год: {}"**, **car**.**year**);  
 println!(**"Цена: {}"**, **car**.**price**);  
 println!(**"Пробег: {}"**, **car**.**mileage**);  
 println!(**"Новый: {}"**, **car**.**is\_new**);  
 sleep\_700mil().await;  
 ***Ok***(())  
 **}  
 *Ok***(***None***) => **{** println!(**"Автомобиль не найдена"**);  
 sleep\_700mil().await;  
 ***Ok***(())  
 **}  
 *Err***(**err**) => **{** eprintln!(**"Ошибка при получении автомобиля: {}"**, **err**);  
 sleep\_700mil().await;  
 ***Err***(**err**)  
 **}  
 }  
 }**,  
 **"4"** => **{** let mut **id** = String::*new*();  
 println!(**"Введите id автомобиля:"**);  
 io::stdin().read\_line(&mut **id**).unwrap();  
 let **id** = from\_str\_to\_int(&**id**);  
  
 match get\_order(&client, **id**.await).await **{  
 *Ok***(***Some***(**order**)) => **{** match **order**.**id {  
 *Some***(**id**) => println!(**"**\n\n\n**id: {}"**, **id**),  
 ***None*** => println!(**"**\n\n\n**id: Не найдено"**)  
 **}** match **order**.**id {  
 *Some***(**id**) => println!(**"**\n\n\n**id: {}"**, **id**),  
 ***None*** => println!(**"**\n\n\n**id: Не найдено"**)  
 **}** println!(**"id Клинта: {}"**, **order**.**customer\_id**);  
 println!(**"id Работника: {}"**, **order**.**employee\_id**);  
 println!(**"Дата заказа: {}"**, **order**.**order\_date**);  
 println!(**"Цена: {}"**, **order**.**price**);  
 println!(**"Активен: {}"**, **order**.**is\_active**);  
 sleep\_700mil().await;  
 ***Ok***(())  
 **}  
 *Ok***(***None***) => **{** println!(**"Заказ не найден"**);  
 sleep\_700mil().await;  
 ***Ok***(())  
 **}**,  
 ***Err***(**err**) => **{** eprintln!(**"Ошибка при получении заказа: {}"**, **err**);  
 sleep\_700mil().await;  
 ***Err***(**err**)  
 **}  
 }  
 }**,  
 **"5"** => **{** println!(**"Выход из поиска."**);  
 sleep\_700mil().await;  
 ***Ok***(())  
 **}** \_ => **{** input\_error().await;  
 sleep\_700mil().await;  
 ***Ok***(())  
 **}  
 }  
}**

*Листинг 2. mod.rs для экспорта модуля*

pub mod search;

**3.11 Описание модуля «Перечисления»**

В этом модуле представлено перечисление возможных доменов электронной почты.

*Листинг 1. Перечисление доменов*

pub enum EmailDomain **{  
 *MailRu***,  
 ***YandexRu***,  
 ***GmailCom***,  
**}**impl EmailDomain **{** pub fn as\_str(&self) -> &*'static* str **{** match self **{** EmailDomain::***MailRu*** => **"@mail.ru"**,  
 EmailDomain::***YandexRu*** => **"@yandex.ru"**,  
 EmailDomain::***GmailCom*** => **"@gmail.com"**,  
 **}  
 }  
}**

*Листинг 2. mod.rs для экспорта модуля*

pub mod enum\_email;

**3.12 Описание модуля «Валидация»**

В этом модуле представлены функции валидации, вывода текста, преобразование типов и вывода старых значений при редактировании объекта.

*Листинг 1. Функция валидации email*

pub async fn valid\_email(email: String) -> Result<(), String> **{** let **allowed\_domains** = [  
 EmailDomain::***MailRu***,  
 EmailDomain::***YandexRu***,  
 EmailDomain::***GmailCom***,  
 ];  
  
 if **allowed\_domains**.iter().any(|domain| email.ends\_with(domain.as\_str())) **{  
 *Ok***(())  
 **}** else **{  
 *Err***(format!(  
 **"Email должен заканчиваться на {}"**,  
 **allowed\_domains** .iter()  
 .map(|domain| domain.as\_str())  
 .collect::<Vec<&str>>()  
 .join(**", "**)  
 ))  
 **}  
}**

*Листинг 2. Функция валидация даты*

pub async fn valid\_date(date: String) -> Result<(), String> **{** let **parts**: Vec<&str> = date.split(**'.'**).collect();  
  
 if **parts**.len() != *3* **{** return ***Err***(**"Дата должна быть в формате ДД.ММ.ГГГГ"**.to\_string());  
 **}** let **day**: u32 = match **parts**[*0*].parse() **{  
 *Ok***(**d**) => **d**,  
 ***Err***(\_) => return ***Err***(**"День должен быть числом"**.to\_string()),  
 **}**;  
  
 let **month**: u32 = match **parts**[*1*].parse() **{  
 *Ok***(**m**) => **m**,  
 ***Err***(\_) => return ***Err***(**"Месяц должен быть числом"**.to\_string()),  
 **}**;  
  
 let **year**: u32 = match **parts**[*2*].parse() **{  
 *Ok***(**y**) => **y**,  
 ***Err***(\_) => return ***Err***(**"Год должен быть числом"**.to\_string()),  
 **}**;  
  
 if **day** < *1* || **day** > *31* **{** return ***Err***(**"День должен быть в диапазоне от 1 до 31"**.to\_string());  
 **}** if **month** < *1* || **month** > *12* **{** return ***Err***(**"Месяц должен быть в диапазоне от 1 до 12"**.to\_string());  
 **}** if **year** > *2025* **{** return ***Err***(**"Год не может быть больше 2025"**.to\_string());  
 **}** if **month** == *2* && **day** > *29* **{** return ***Err***(**"Февраль не может содержать более 29 дней"**.to\_string());  
 **}** let **days\_in\_month** = match **month {** *4* | *6* | *9* | *11* => *30*,  
 *2* => **{** if is\_leap\_year(**year**) **{** *29* **}** else **{** *28* **}  
 }** \_ => *31*,  
 **}**;  
  
 if **day** > **days\_in\_month {** return ***Err***(format!(  
 **"Месяц {} не может содержать более {} дней"**,  
 **month**, **days\_in\_month** ));  
 **}  
 *Ok***(())  
**}**

*Листинг 2.1. Функция определения високосного года*

fn is\_leap\_year(year: u32) -> bool **{** (year % *4* == *0* && year % *100* != *0*) || (year % *400* == *0*)  
**}**

*Листинг 3. Функция вывода текста*

pub async fn input\_error() **{** println!(**"Неверный ввод"**)  
**}**

*Листинг 4. Функция преобразования типа их строки в число*

pub async fn from\_str\_to\_int(input: &str) -> i32 **{** input.trim().parse::<i32>().unwrap\_or\_else(|\_| **{** println!(**"Ошибка: введите корректное число"**);  
 *0* **}**)  
**}**

*Листинг 5. Вывод старых значений при редактировании*

pub async fn old\_value<T: Display>(old\_value: T) **{** println!(**"Старое значение: {}"**, old\_value);  
**}**

*Листинг 6. mod.rs для экспорта модуля*

pub mod from\_str\_to\_int;  
pub mod valid\_date;  
pub mod input\_error;  
pub mod old\_value;  
pub mod valid\_email;

**3.13 Описание модуля prelude (для удобного импорта модулей)**

В этом модуле представлены организация и упрощения импорта модулей и функций в файлы для лучшей читаемости и понимания кода.

*Листинг 1. Экспорт для main.rs*

pub use crate::order::new\_order::new\_order;  
pub use crate::customer::add\_customer::add\_customer;  
pub use crate::customer::choose\_customer::choose\_customer;  
pub use crate::car::find\_car::find\_car;  
pub use crate::car::add\_car::add\_car;  
pub use crate::prelude::prelude\_valid::\*;  
pub use crate::settings::settings::settings;  
pub use crate::employee::employee::**{**create\_employee, get\_employee**}**;  
pub use crate::models::models::**{**Employee**}**;  
pub use crate::order::close\_order::close\_order;  
pub use crate::sleep::sleep\_700mil::sleep\_700mil;  
pub use crate::search::search::search;

*Листинг 2. Экспорт валидации*

pub use crate::validation::from\_str\_to\_int::from\_str\_to\_int;  
pub use crate::validation::input\_error::input\_error;  
pub use crate::validation::old\_value::old\_value;  
pub use crate::validation::valid\_date::valid\_date;

**3.14 Описание модуля приостановки программы**

В этом модуле представлены функции для приостановления выполнения кода программы

*Листинг 1. Функция приостановления программы на 700 миллисекунд*

pub async fn sleep\_700mil() **{** task::sleep(Duration::*from\_millis*(*700*)).await;  
**}**

*Листинг 2. Для экспорта модуля*

pub mod sleep\_700mil;

**РАЗДЕЛ 4**

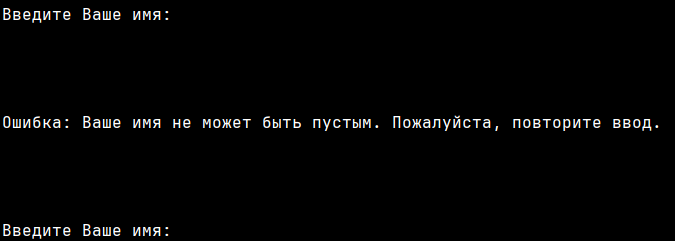
**Тестирование и отладка разработанной консольной программы**

**4.1 Тестирование**

**Тест 1.** Проверка ввода имени при создании работника

Действия: При вводе имени я нажму клавишу «Enter», тем самым пропущу это поле.

Ожидаемый результат: программа попросит повторить ввод.

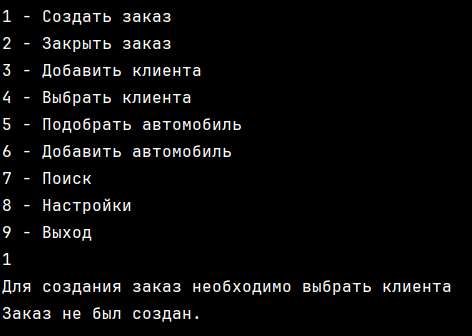
Результат:  


Итог: Ожидаемый результат совпал с фактическим результатом

**Тест 2.** Проверка на создание заказа

Действия: Я создаю заказ, заранее не выбрав покупателя.

Ожидаемый результат: Ошибка, т.к. не был выбрал покупатель.

Результат:  


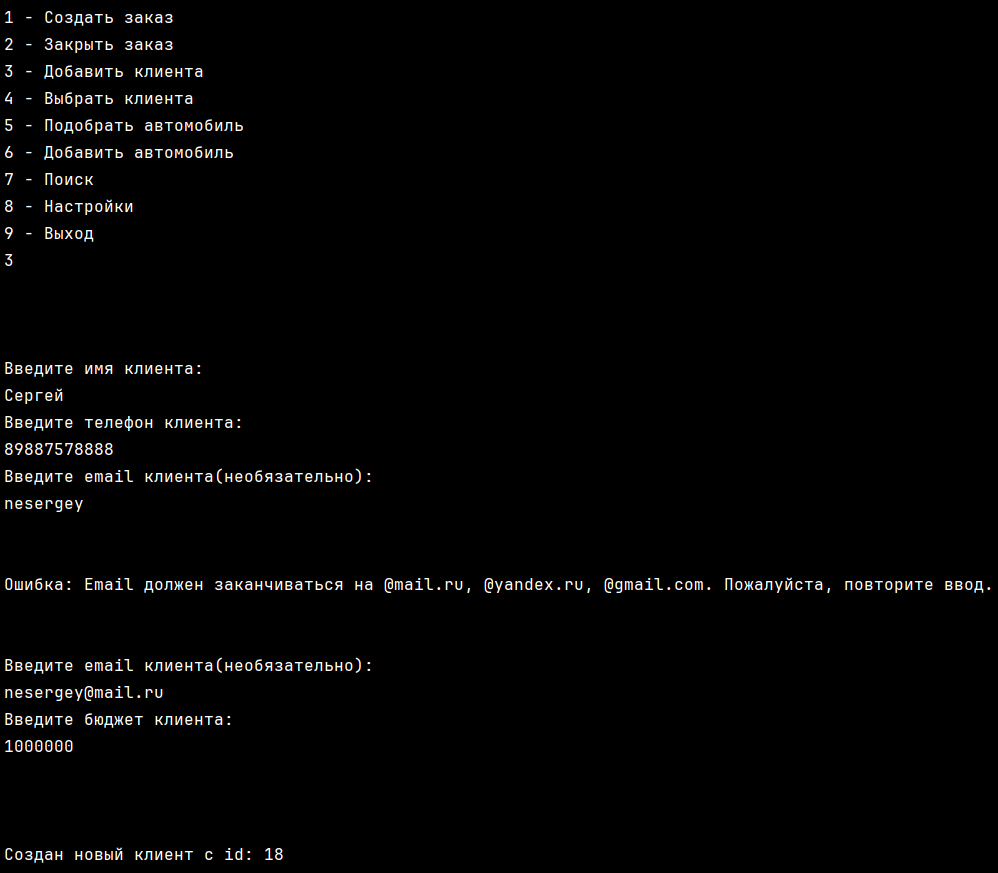
Итог: Ожидаемый результат совпал с фактическим результатом

**Тест 3.** Проверка неверного ввода email при создании клиента

Действия: При создании нового клиента я введу неправильный email.

Ожидаемый результат: Ошибка, программа попросит ввести корректный email.

Результат:



Итог: Ожидаемый результат совпал с фактическим результатом

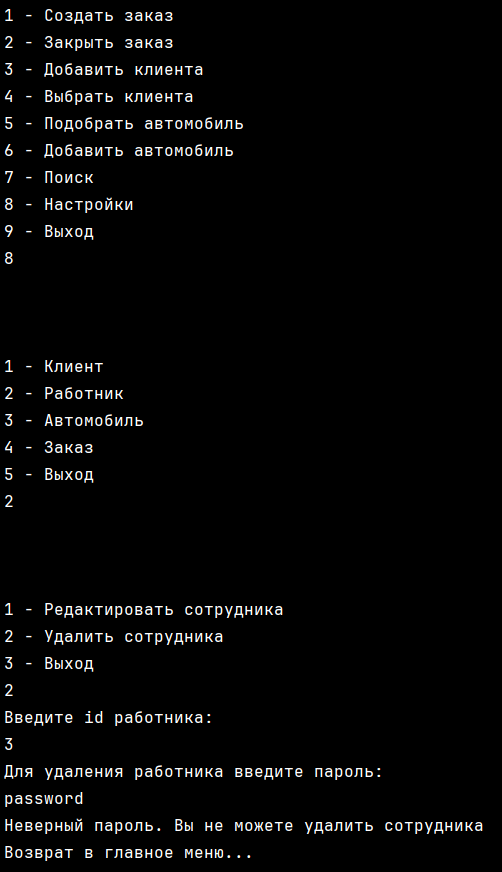
**Тест 4.** Удаление сотрудника

Действия: Я удаляю сотрудника

Ожидаемый результат: Программа попросит пароль:

* 1 тест. Пароль введен неверно, программа вернет меня в главное меню и сотрудник не будет удален.
* 2 тест. Пароль введен верно, сотрудник удален.

**1 тест.** Результат при неверном вводе пароля:



**2 тест**. Результат при правильном вводе пароля:

Таблица «Сотрудники» до удаления:

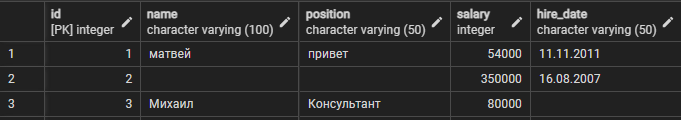
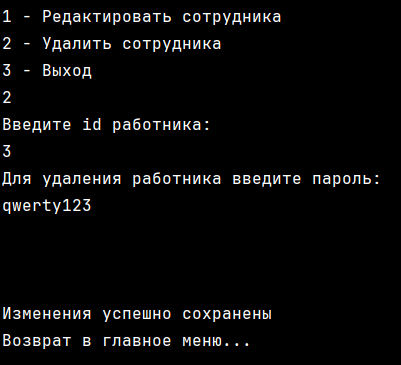
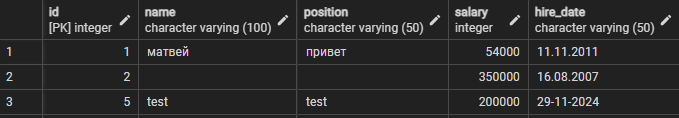


Таблица «Сотрудники» после удаления:



Итог: Ожидаемый результат совпал с фактическим результатом

**Тест 5.** Закрытие заказа

Действия: Заказ выполнили – его нужно закрыть (т.к. он больше неактуальный).

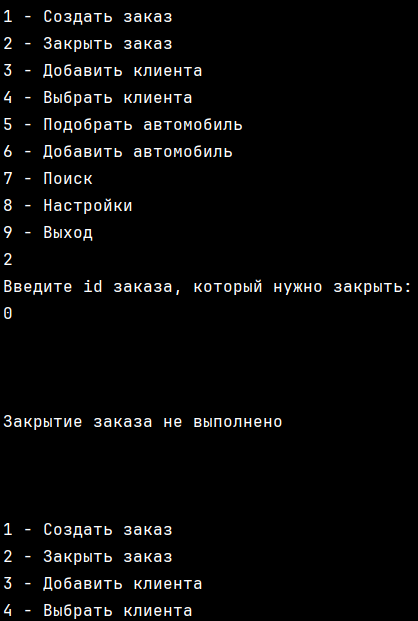
* 1 тест. Я введу несуществующий id.
* 2 тест. Я введу существующий id.
* 3 тест. Я введу буквы вместо цифр.

Ожидаемый результат:

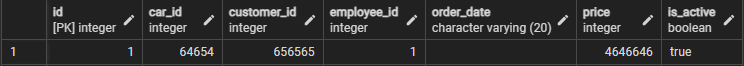
* 1 тест. Ошибка, программа вернет в главное меню.
* 2 тест. Программа выполнит успешное изменения заказа и сделает его неактуальным.
* 3 тест. Ошибка, программа попросит ввести корректный id.

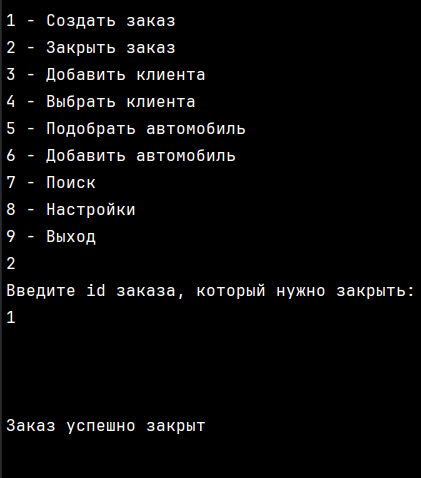
Результат:

1 тест.



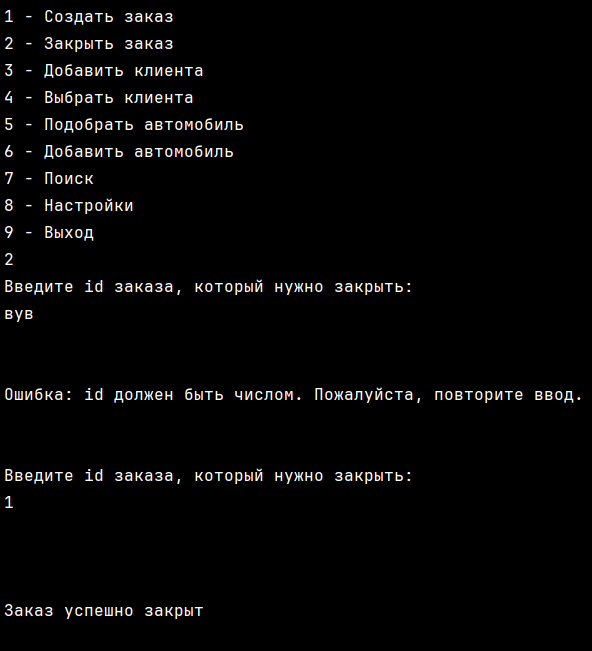
2 тест.







3 тест.



Итог: Ожидаемый результат совпал с фактическим результатом

Для тестирования программы я провёл 5 ручных тестов, тщательно проверяя ключевые функциональные модули. Каждый тест охватывал различные сценарии использования, включая обработку корректных и некорректных данных. Все тесты завершились успешно — ожидаемый результат совпал с фактическим во всех случаях.

В ходе тестирования была также проверена работа основных функций, таких как добавление данных, их выборка, поиск, а также корректность взаимодействия между модулями. Кроме того, были проведены проверки на устойчивость программы к вводу некорректных данных и стабильность при выполнении последовательности операций. Это подтвердило надёжность и корректность работы программы.

**4.2 Отладка программы**

В ходе тестирования программы ошибок и багов не выявлено, тем самым отладка не требуется.

**5 ГЛАВА**

**ЭКСПЛУАТАЦИОННАЯ ЧАСТЬ**

**5.1 Руководство оператора**

**АННОТАЦИЯ**

В данном программном документе приведено руководство оператора по применению и эксплуатации консольной программы «Автосалон», предназначенной для облегчения работы автосалонов.

В данном программном документе, в разделе «Назначение программы» указаны сведения о назначении программы и информация, достаточная для понимания функций программы и её эксплуатации.

В разделе «Условия выполнения программы» указаны условия, необходимые для выполнения программы (минимальный состав аппаратных и программных средств и т.п.).

В данном программном документе, в разделе «Выполнение программы» указана последовательность действий оператора, обеспечивающих загрузку, запуск, выполнение и завершение программы, приведено описание функций, формата и возможных вариантов команд, с помощью которых оператор осуществляет загрузку и управляет выполнением программы, а также ответы программы на эти команды.

**5.1.1 Назначение программы**

Специальное программное обеспечение «Автосалон» предназначено для автоматизации и упрощения ключевых процессов управления в автосалонах. Оно предоставляет удобные инструменты для работы с базами данных сотрудников, клиентов, автомобилей и заказов, что значительно облегчает организацию и контроль всех этапов работы.

С помощью программы можно быстро добавлять и редактировать данные о клиентах, формировать и отслеживать заказы, управлять информацией о сотрудниках, а также вести учёт автомобилей, включая их поиск и обновление данных.

Программное обеспечение также включает модули для анализа данных и формирования отчётов, что позволяет эффективно оценивать текущие показатели работы автосалона и принимать управленческие решения на основе актуальной информации.

Благодаря простому интерфейсу и логичной структуре, «Автосалон» подходит как для опытных пользователей, так и для начинающих, обеспечивая высокий уровень удобства и надёжности.

**5.1.2 Назначение функций**

1. «Войти» – войти в свою учетную запись сотрудника.
2. «Создать сотрудника» – создание нового сотрудника.
3. «Создать заказ» – создание нового заказа (сначала необходимо создать клиента, если он первый раз или выбрать уже существующего).
4. «Закрыть заказ» – закрытие заказа.
5. «Добавить клиента» - добавление клиента.
6. «Выбрать клиента» - выбор клиента.
   1. «Вывести всех клиентов» – вывод всех клиентов.
   2. «Найти клиента по фильтрам» - ввод определенных фильтров и вывод клиентов.
7. «Подобрать автомобиль» - подбор автомобиля.
   1. «По бюджету» - Выбор автомобиля по бюджету.
      1. «От» - ввод от какой суммы будут выведены машины.
      2. «До» - ввод от какой суммы будут выведены машины.

7.2 «По определенным фильтрам» - ввод определенных фильтров для поиска автомобилей.

1. «Добавить автомобиль» - создание нового автомобиля.
2. «Поиск» - поиск клиента, сотрудника, автомобиля и заказа по id.
3. «Настройки» - редактирование или удаление клиента, сотрудника, автомобиля или заказа.
4. «Выход» - программа прекращает свою работу.

**5.1.3 Условия выполнения программы**

Операционная система: Windows 98/XP/7/8/10 /macOS/Linux.

Процессор (CPU): Любой.

Оперативная память (RAM): 512 МБ.

Видеоадаптер: Любой.

Свободное место на жёстком диске: ~ 50Мб.

Установленный на компьютер язык программирования Rust.

Установленный на компьютер Cargo.

**5.1.4 Выполнение программы**

Программу можно запустить 2 способами:

1 способ через консоль

* 1. Создать файл .env
  2. Вставить туда .env DATABASE\_URL=postgres://username:password@localhost/database\_name

Вместо username, password и database\_name ввести свои значения

* 1. В консоль написать cargo build
  2. В консоль написать cargo run

2 способ использовать файл .exe

2.1) Создать файл .env

2.2) Вставить туда .env DATABASE\_URL=postgres://username:password@localhost/database\_name Вместо username, password и database\_name ввести свои значения

* 1. В консоль написать cargo build –release
  2. Перейти по следуюзему пути:

название\_проекта/target/release/название\_проекта.exe

**5.2 To-Do List**

1. При входе в учетную запись сотрудника добавить ввод пароля.

2. Добавить программу скидок для покупателей.

3. Улучшить систему подсчета зарплаты сотрудников.

4. Добавить новые ограничения для создания заказов.

**Заключение**

В рамках курсового проекта была разработана программа «Автосалон», предназначенная для автоматизации ключевых процессов управления в автосалоне. Целью данного проекта являлось создание консольного приложения, обеспечивающего удобное управление базами данных сотрудников, клиентов, автомобилей и заказов. В процессе реализации программы были достигнуты все поставленные задачи, включая:

1. Управление базой данных работников, включая добавление, удаление и обновление.
2. Поиск и фильтрация данных по заданным критериям.
3. Учет автомобилей, включая добавление новых моделей, редактирование и удаление.

**Основные практические результаты**

1. **Функциональность управления базами данных:**

* Программа позволяет эффективно работать с данными сотрудников, клиентов и автомобилей.
* Реализована система управления заказами, включая их создание, редактирование и закрытие.

1. **Удобство использования:**

* Программа имеет простой и интуитивно понятный интерфейс, подходящий как для опытных пользователей, так и для начинающих.
* Встроенная система поиска и валидации данных позволяет минимизировать ошибки при работе с программой.

1. **Надёжность и безопасность:**

* Обеспечена обработка некорректного ввода для повышения устойчивости программы.

**Предложения по улучшению:**

1. **Интеграция графического интерфейса:**

* Добавление графического интерфейса для повышения удобства работы пользователей и расширения возможностей программы.

1. **Расширение функциональности:**

* Внедрение аналитических модулей для автоматического анализа данных о продажах, клиентах и заказах.

**Пути внедрения**

Разработанное приложение «Автосалон» может быть внедрено в рабочую среду автосалонов без значительных затрат. Для начала эксплуатации достаточно установить программу на сервер или рабочие станции и провести обучение сотрудников. Программа не требует значительных вычислительных ресурсов, что делает её доступной для большинства компаний.

Итогом работы над проектом стало создание функционального, безопасного и удобного консольного приложения, которое соответствует современным требованиям управления бизнес-процессами в автосалонах. Реализация предложенных доработок и дальнейшая модернизация программы позволят ещё больше расширить её возможности и повысить эффективность работы автосалонов.

**Список литературы и интернет-источников**

<https://www.infoconnector.ru/devices/glossary/c-terms/id-identifikator>

https://www.jetbrains.com/rust/

https://www.jetbrains.com/rust/whatsnew/2024-3/

<https://doc.rust-lang.ru/book/>

<https://metanit.com/rust/tutorial/>

<https://github.com/madmattheww/casino-Rust>

https://github.com/madmattheww/rust\_toDo\_postgres