**ПРИЛОЖЕНИЕ 1. Код файла Cargo.toml**

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**

**ПРИЛОЖЕНИЕ 2. Код для подключения к базе данных Postgres**

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**);  
 **}  
}**);

**ПРИЛОЖЕНИЕ 3. Код главного модуля**

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***(())  
**}**

**ПРИЛОЖЕНИЕ 4. Код модуля «Клиент»**

**customer.rs:**

use crate::models::models::Customer;  
use tokio\_postgres::**{** Error, Client **}**;  
  
  
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*))  
**}**pub async fn update\_customer(client: &Client, customer\_id: i32, updated\_customer: Customer) -> Result<String, Error> **{** let **result** = client.execute(  
 **"UPDATE customer SET name = $2, phone = $3, email = $4, budget = $5 WHERE id = $1"**,  
 &[&customer\_id, &updated\_customer.**name**, &updated\_customer.**phone**, &updated\_customer.**email**, &updated\_customer.**budget**],  
 ).await?;  
 if **result** > *0* **{  
 *Ok***(**"**\n\n\n**Изменения успешно сохранены"**.to\_string())  
 **}** else **{  
 *Ok***(**"**\n\n\n**Изменения не применены"**.to\_string())  
 **}  
}**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())  
 **}  
}**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***)  
 **}  
}**pub async fn get\_all\_customers(client: &Client) -> Result<Vec<Customer>, Error> **{** let **rows** = client.query(  
 **"SELECT id, name, phone, email, budget FROM customer"**,  
 &[],  
 ).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**)  
**}**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**)  
**}**

**add\_customer.rs:**

use std::io::**{**self, Write**}**;  
use tokio\_postgres::**{** Error, Client **}**;  
use crate::models::models::**{**Customer**}**;  
use crate::customer::customer::**{**create\_customer**}**;  
pub use crate::validation::valid\_email::valid\_email;  
  
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**)  
 **}  
 }  
}**

**choose\_customer.rs:**

use std::io::**{**self**}**;  
use tokio\_postgres::**{** Error, Client **}**;  
use crate::models::models::**{**Customer**}**;  
use crate::customer::customer::**{**get\_all\_customers, search\_customer**}**;  
use crate::prelude::prelude\_valid::\*;  
  
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***)  
 **}  
 }  
}**

**mod.rs:**

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

**ПРИЛОЖЕНИЕ 5. Код модуля «Сотрудник»**

**employee.rs:**

use crate::models::models::Employee;  
use tokio\_postgres::**{** Error, Client **}**;  
  
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*))  
**}**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())  
 **}  
}**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())  
 **}  
}**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***)  
 **}  
}**

**mod.rs:**

pub mod employee;

**ПРИЛОЖЕНИЕ 6. Код модуля «Автомобиль»**

**car.rs:**

use crate::models::models::Car;  
use tokio\_postgres::**{** Error, Client **}**;  
  
  
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*))  
**}**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())  
 **}  
}**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())  
 **}  
}**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***)  
 **}  
}**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**)  
**}**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**)  
**}**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**)  
**}**

**add\_car.rs:**

use tokio\_postgres::**{**Error, Client**}**;  
use std::io::**{**self, Write**}**;  
use crate::models::models::**{**Car**}**;  
use crate::car::car::**{**create\_car**}**;  
  
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**)  
 **}  
 }  
}**

**find\_car.rs:**

use tokio\_postgres::**{**Error, Client**}**;  
use std::io::**{**self, Write**}**;  
use std::option::Option;  
use crate::car::car::**{**budget\_car\_under, budget\_car\_upper, search\_cars**}**;  
  
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***(())  
**}**

**mod.rs:**

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

**ПРИЛОЖЕНИЕ 7. Код модуля «Заказ»**

**order.rs:**

use tokio\_postgres::**{**Client, Error**}**;  
use crate::models::models::Order;  
  
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*))  
**}**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())  
 **}  
}**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())  
 **}  
}**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***)  
 **}  
}**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())  
 **}  
}**

**close\_order.rs:**

use tokio\_postgres::**{**Error, Client**}**;  
use std::io::**{**self, Write**}**;  
use crate::order::order::cancel\_order;  
  
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***(())  
**}**

**new\_order.rs:**

use crate::validation::from\_str\_to\_int::**{**from\_str\_to\_int**}**;  
use std::io::**{**self, Write**}**;  
use tokio\_postgres::**{** Error, Client **}**;  
use crate::models::models::**{**Order**}**;  
use crate::order::order::**{**create\_order**}**;  
use chrono::Local;  
  
  
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**)  
 **}  
 }  
}**

**mod.rs:**

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

**ПРИЛОЖЕНИЕ 8. Код модуля «Модели»**

**models.rs:**

#[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,  
**}**

**mod.rs:**

pub mod models;

**ПРИЛОЖЕНИЕ 9. Описание модуля «Настройки»**

**settings.rs:**

use tokio\_postgres::**{**Error, Client**}**;  
use std::io::**{**self, Write**}**;  
use crate::settings::settings\_car::settings\_car;  
use crate::settings::settings\_cust::settings\_cust;  
use crate::settings::settings\_empl::settings\_empl;  
use crate::settings::settings\_order::settings\_order;  
use crate::validation::input\_error::input\_error;  
  
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***(())  
 **}  
 }  
}**

**settings\_cust.rs:**

use tokio\_postgres::**{**Error, Client**}**;  
use std::io::**{**self, Write**}**;  
use crate::car::car::**{**delete\_car, get\_car, update\_car**}**;  
use crate::models::models::Car;  
use crate::prelude::prelude\_valid::\*;  
  
pub async fn settings\_car(client: &Client) -> Result<(), Error> **{** let mut **input1** = String::*new*();  
  
 println!(**"**\n\n\n**1 - Редактировать машину"**);  
 println!(**"2 - Удалить машину"**);  
 println!(**"3 - Выход"**);  
  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input1**).unwrap();  
  
 match **input1**.as\_str().trim() **{  
 "1"** => **{** //Редактировать машину  
 let **car\_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***(**car\_id1**) => break **car\_id1**,  
 ***Err***(\_) => println!(**"**\n\n**Ошибка: id должен быть числом. Пожалуйста, повторите ввод.**\n\n**"**),  
 **}  
 }**;  
  
 let mut **brand1** = String::*new*();  
 let mut **model1** = String::*new*();  
 let mut **year1**: i32 = *0*;  
 let mut **price1**: i32 = *0*;  
 let mut **mileage1**: i32 = *0*;  
 let mut **is\_new1** = false;  
  
 match get\_car(&client, **car\_id**).await **{  
 *Ok***(***Some***(**car**)) => **{  
 brand1** = **car**.**brand**;  
 **model1** = **car**.**model**;  
 **year1** = **car**.**year**;  
 **price1** = **car**.**price**;  
 **mileage1** = **car**.**mileage**;  
 **is\_new1** = **car**.**is\_new**;  
 **}  
 *Ok***(***None***) => **{** println!(**"Машина с id {} не найден."**, **car\_id**);  
 **}  
 *Err***(**err**) => **{** eprintln!(**"Ошибка при получении машины: {:?}"**, **err**);  
 **}  
 }** println!(**"**\n**Введите новую марку автомобиля:"**);  
 old\_value(&**brand1**).await;  
 let mut **brand\_input** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **brand\_input**).unwrap();  
 let **brand** = if **brand\_input**.trim().is\_empty() **{  
 brand1**.clone()  
 **}** else **{  
 brand\_input**.trim().to\_string()  
 **}**;  
  
 println!(**"**\n**Введите новую модель автомобиля:"**);  
 old\_value(&**model1**).await;  
 let mut **model\_input** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **model\_input**).unwrap();  
 let **model** = if **model\_input**.trim().is\_empty() **{  
 model1**.clone()  
 **}** else **{  
 model\_input**.trim().to\_string()  
 **}**;  
  
 println!(**"**\n**Введите новый год автомобиля:"**);  
 old\_value(**year1**).await;  
 let mut **year\_str** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **year\_str**).unwrap();  
 let **year**: i32 = if **year\_str**.trim().is\_empty() **{  
 year1  
 }** else **{  
 year\_str** .trim()  
 .parse()  
 .unwrap\_or\_else(|\_| **{** eprintln!(**"Ошибка: бюджет должен быть числом. Используем старое значение."**);  
 **year1  
 }**)  
 **}**;  
  
 println!(**"**\n**Введите новую цену автомобиля:"**);  
 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: цена должна быть числом. Используем старое значение."**);  
 **price1  
 }**)  
 **}**;  
  
 println!(**"**\n**Введите новый пробег автомобиля:"**);  
 old\_value(**mileage1**).await;  
 let mut **mileage\_str** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **mileage\_str**).unwrap();  
 let **mileage**: i32 = if **mileage\_str**.trim().is\_empty() **{  
 mileage1  
 }** else **{  
 mileage\_str** .trim()  
 .parse()  
 .unwrap\_or\_else(|\_| **{** eprintln!(**"Ошибка: бюджет должен быть числом. Используем старое значение."**);  
 **mileage1  
 }**)  
 **}**;  
  
 println!(**"**\n**Введите новое состояние автомобиля:** \n\  
 **0 (Новый автомобиль) или 1 (Не новый автомобиль)"**);  
 old\_value(**is\_new1**).await;  
 let mut **is\_new\_input** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **is\_new\_input**).unwrap();  
 match **is\_new\_input**.trim() **{  
 "0"** => **is\_new1** = true,  
 **"1"** => **is\_new1** = false,  
 \_ => **{** println!(**"Неверный ввод. Введите 0 или 1."**);  
 **}  
 }** let **is\_new** = **is\_new1**;  
  
 let **updated\_car** = Car **{  
 id**: ***Some***(**car\_id**),  
 brand,  
 model,  
 year,  
 price,  
 mileage,  
 is\_new  
 **}**;  
  
 match update\_car(&client, **car\_id**, **updated\_car**).await **{  
 *Ok***(**message**) => **{** println!(**"{}"**, **message**);  
 ***Ok***(())  
 **}**,  
 ***Err***(**err**) => **{** eprintln!(**"Ошибка при обновлении машины: {:?}"**, **err**);  
 ***Err***(**err**)  
 **}  
 }  
 }**,  
 **"2"** => **{**//Удалить автомобиль  
 let mut **input\_str** = String::*new*();  
  
 println!(**"Введите id машины:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input\_str**).unwrap();  
  
 let **input2** = from\_str\_to\_int(**input\_str**.trim());  
  
 match delete\_car(&client, **input2**.await).await **{  
 *Ok***(**message**) => **{** println!(**"{}"**, **message**);  
 ***Ok***(())  
 **}**,  
 ***Err***(**err**) => **{** eprintln!(**"Ошибка при удалении автомобиля: {:?}"**, **err**);  
 ***Err***(**err**)  
 **}  
 }  
 }**,  
 **"3"** => ***Ok***(()),  
 \_ => **{** input\_error().await;  
 ***Ok***(())  
 **}  
 }  
}**

**settings\_empl.rs:**

use tokio\_postgres::**{**Error, Client**}**;  
use std::io::**{**self, Write**}**;  
use crate::employee::employee::**{**delete\_employee, get\_employee, update\_employee**}**;  
use crate::models::models::Employee;  
use crate::prelude::prelude\_valid::\*;  
  
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***(())  
 **}  
 }  
}**

**settings\_order.rs:**

use tokio\_postgres::**{**Error, Client**}**;  
use std::io::**{**self, Write**}**;  
use crate::models::models::Order;  
use crate::order::order::**{**get\_order, update\_order, delete\_order**}**;  
use crate::prelude::prelude\_valid::\*;  
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***(())  
 **}  
 }  
}**

**settings\_car.rs:**use tokio\_postgres::**{**Error, Client**}**;  
use std::io::**{**self, Write**}**;  
use crate::car::car::**{**delete\_car, get\_car, update\_car**}**;  
use crate::models::models::Car;  
use crate::prelude::prelude\_valid::\*;  
  
pub async fn settings\_car(client: &Client) -> Result<(), Error> **{** let mut **input1** = String::*new*();  
  
 println!(**"**\n\n\n**1 - Редактировать машину"**);  
 println!(**"2 - Удалить машину"**);  
 println!(**"3 - Выход"**);  
  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input1**).unwrap();  
  
 match **input1**.as\_str().trim() **{  
 "1"** => **{** //Редактировать машину  
 let **car\_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***(**car\_id1**) => break **car\_id1**,  
 ***Err***(\_) => println!(**"**\n\n**Ошибка: id должен быть числом. Пожалуйста, повторите ввод.**\n\n**"**),  
 **}  
 }**;  
  
 let mut **brand1** = String::*new*();  
 let mut **model1** = String::*new*();  
 let mut **year1**: i32 = *0*;  
 let mut **price1**: i32 = *0*;  
 let mut **mileage1**: i32 = *0*;  
 let mut **is\_new1** = false;  
  
 match get\_car(&client, **car\_id**).await **{  
 *Ok***(***Some***(**car**)) => **{  
 brand1** = **car**.**brand**;  
 **model1** = **car**.**model**;  
 **year1** = **car**.**year**;  
 **price1** = **car**.**price**;  
 **mileage1** = **car**.**mileage**;  
 **is\_new1** = **car**.**is\_new**;  
 **}  
 *Ok***(***None***) => **{** println!(**"Машина с id {} не найден."**, **car\_id**);  
 **}  
 *Err***(**err**) => **{** eprintln!(**"Ошибка при получении машины: {:?}"**, **err**);  
 **}  
 }** println!(**"**\n**Введите новую марку автомобиля:"**);  
 old\_value(&**brand1**).await;  
 let mut **brand\_input** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **brand\_input**).unwrap();  
 let **brand** = if **brand\_input**.trim().is\_empty() **{  
 brand1**.clone()  
 **}** else **{  
 brand\_input**.trim().to\_string()  
 **}**;  
  
 println!(**"**\n**Введите новую модель автомобиля:"**);  
 old\_value(&**model1**).await;  
 let mut **model\_input** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **model\_input**).unwrap();  
 let **model** = if **model\_input**.trim().is\_empty() **{  
 model1**.clone()  
 **}** else **{  
 model\_input**.trim().to\_string()  
 **}**;  
  
 println!(**"**\n**Введите новый год автомобиля:"**);  
 old\_value(**year1**).await;  
 let mut **year\_str** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **year\_str**).unwrap();  
 let **year**: i32 = if **year\_str**.trim().is\_empty() **{  
 year1  
 }** else **{  
 year\_str** .trim()  
 .parse()  
 .unwrap\_or\_else(|\_| **{** eprintln!(**"Ошибка: бюджет должен быть числом. Используем старое значение."**);  
 **year1  
 }**)  
 **}**;  
  
 println!(**"**\n**Введите новую цену автомобиля:"**);  
 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: цена должна быть числом. Используем старое значение."**);  
 **price1  
 }**)  
 **}**;  
  
 println!(**"**\n**Введите новый пробег автомобиля:"**);  
 old\_value(**mileage1**).await;  
 let mut **mileage\_str** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **mileage\_str**).unwrap();  
 let **mileage**: i32 = if **mileage\_str**.trim().is\_empty() **{  
 mileage1  
 }** else **{  
 mileage\_str** .trim()  
 .parse()  
 .unwrap\_or\_else(|\_| **{** eprintln!(**"Ошибка: бюджет должен быть числом. Используем старое значение."**);  
 **mileage1  
 }**)  
 **}**;  
  
 println!(**"**\n**Введите новое состояние автомобиля:** \n\  
 **0 (Новый автомобиль) или 1 (Не новый автомобиль)"**);  
 old\_value(**is\_new1**).await;  
 let mut **is\_new\_input** = String::*new*();  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **is\_new\_input**).unwrap();  
 match **is\_new\_input**.trim() **{  
 "0"** => **is\_new1** = true,  
 **"1"** => **is\_new1** = false,  
 \_ => **{** println!(**"Неверный ввод. Введите 0 или 1."**);  
 **}  
 }** let **is\_new** = **is\_new1**;  
  
 let **updated\_car** = Car **{  
 id**: ***Some***(**car\_id**),  
 brand,  
 model,  
 year,  
 price,  
 mileage,  
 is\_new  
 **}**;  
  
 match update\_car(&client, **car\_id**, **updated\_car**).await **{  
 *Ok***(**message**) => **{** println!(**"{}"**, **message**);  
 ***Ok***(())  
 **}**,  
 ***Err***(**err**) => **{** eprintln!(**"Ошибка при обновлении машины: {:?}"**, **err**);  
 ***Err***(**err**)  
 **}  
 }  
 }**,  
 **"2"** => **{**//Удалить автомобиль  
 let mut **input\_str** = String::*new*();  
  
 println!(**"Введите id машины:"**);  
 io::stdout().flush().unwrap();  
 io::stdin().read\_line(&mut **input\_str**).unwrap();  
  
 let **input2** = from\_str\_to\_int(**input\_str**.trim());  
  
 match delete\_car(&client, **input2**.await).await **{  
 *Ok***(**message**) => **{** println!(**"{}"**, **message**);  
 ***Ok***(())  
 **}**,  
 ***Err***(**err**) => **{** eprintln!(**"Ошибка при удалении автомобиля: {:?}"**, **err**);  
 ***Err***(**err**)  
 **}  
 }  
 }**,  
 **"3"** => ***Ok***(()),  
 \_ => **{** input\_error().await;  
 ***Ok***(())  
 **}  
 }  
}**

**mod.rs:**

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

**ПРИЛОЖЕНИЕ 10. Код модуля «Поиск»**

**search.rs:**

use tokio\_postgres::**{**Error, Client**}**;  
use std::io::**{**self**}**;  
  
use crate::customer::customer::**{**get\_customer**}**;  
use crate::employee::employee::**{**get\_employee**}**;  
use crate::order::order::**{**get\_order**}**;  
use crate::car::car::**{**get\_car**}**;  
use crate::prelude::prelude\_valid::\*;  
use crate::sleep::sleep\_700mil::sleep\_700mil;  
  
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***(())  
 **}  
 }  
}**

**mod.rs:**

pub mod search;

**ПРИЛОЖЕНИЕ 11. Код модуля «Перечисления»**

**enum\_email.rs:**

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"**,  
 **}  
 }  
}**

**mod.rs:**

pub mod enum\_email;

**ПРИЛОЖЕНИЕ 12. Код модуля «Валидация»**

**valid\_email.rs:**

use crate::enums::enum\_email::EmailDomain;  
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(**", "**)  
 ))  
 **}  
}**

**valid\_data.rs:**

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

**input\_error.rs:**

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

**from\_str\_to\_int.rs:**

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

**old\_value.rs:**

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

**mod.rs:**

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

**ПРИЛОЖЕНИЕ 13. Код модуля «prelude»**

**prelude\_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;

**prelude\_valid.rs:**

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;

**mod.rs:**

pub mod prelude\_valid;  
pub mod prelude\_main;

**ПРИЛОЖЕНИЕ 14. Код модуля приостановки программы**

**sleep\_700mil.rs:**

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

**mod.rs:**

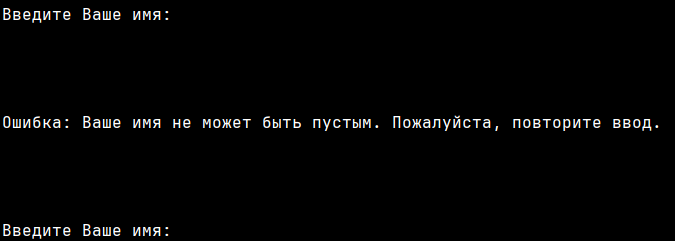
pub mod sleep\_700mil;

**ПРИЛОЖЕНИЕ 15. Код файла .env**

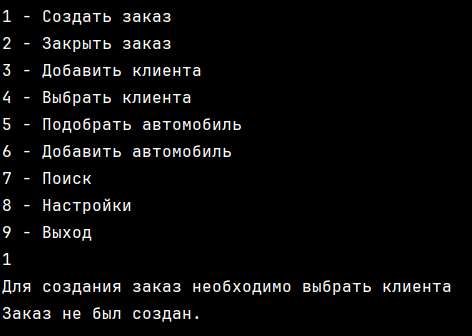
DATABASE\_URL=postgres://username:password@localhost/database\_name

**ПРИЛОЖЕНИЕ 16. Рисунки тестов**

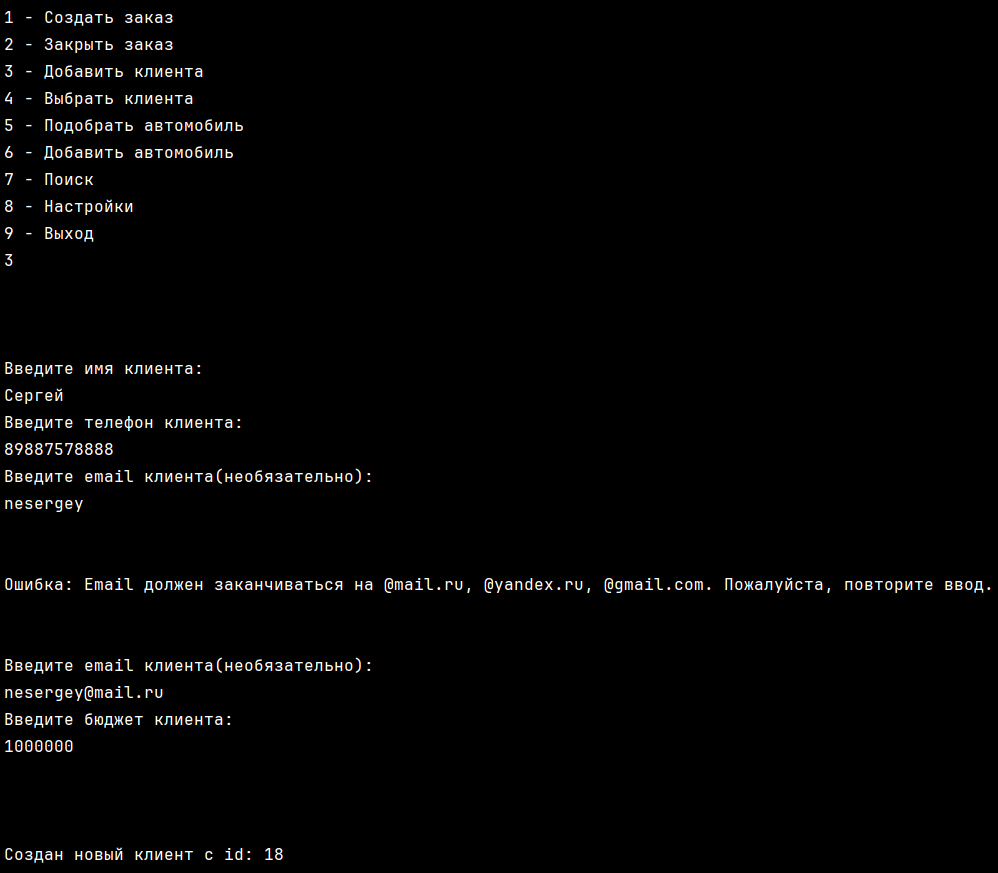
**Рисунок 1**



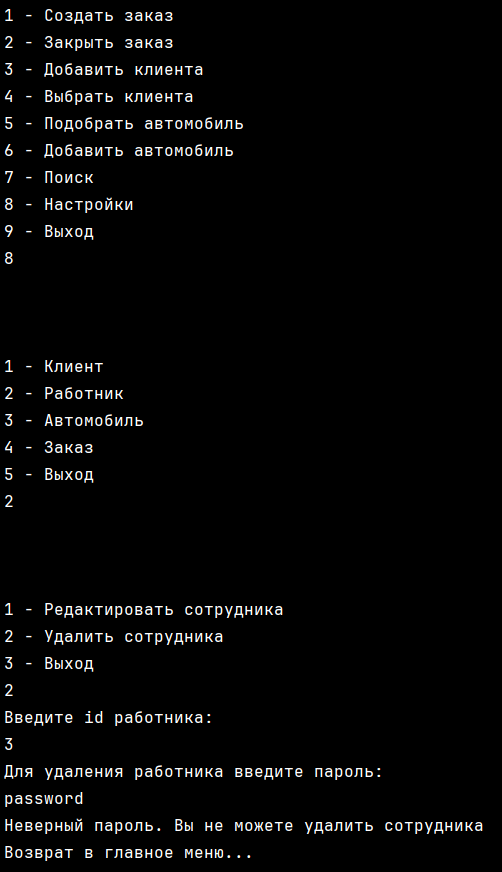
**Рисунок 2**



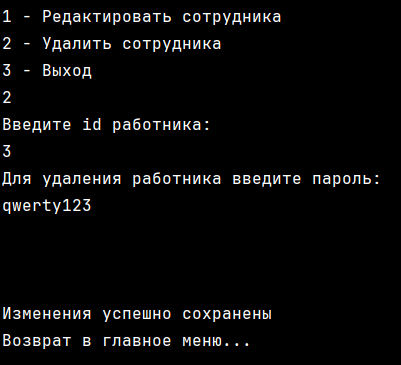
**Рисунок 3**



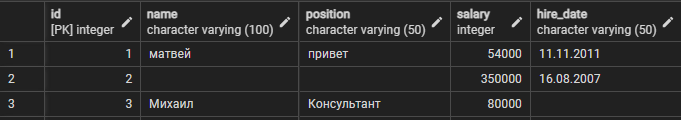
**Рисунок 4**



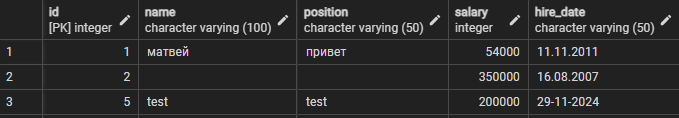
**Рисунок 5**



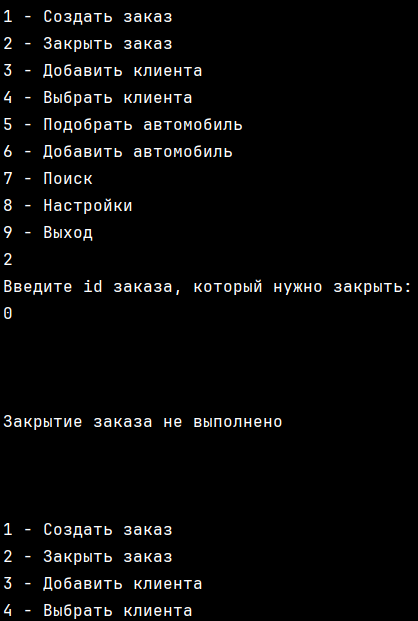
**Рисунок 6**



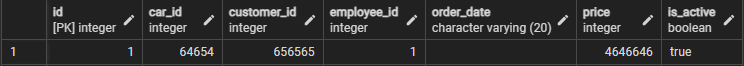
**Рисунок 7**



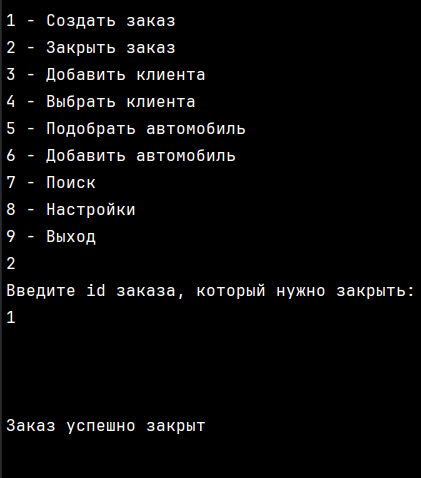
**Рисунок 8**



**Рисунок 9**



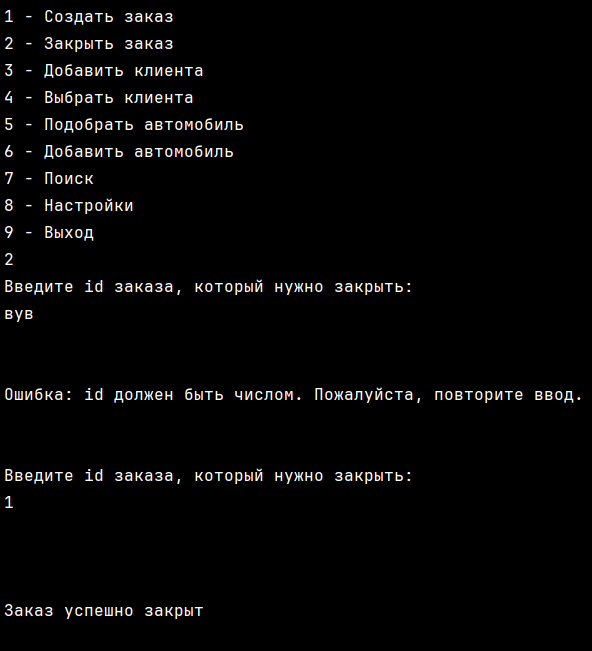
**Рисунок 10**



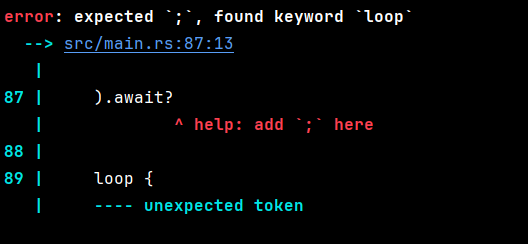
**Рисунок 11**



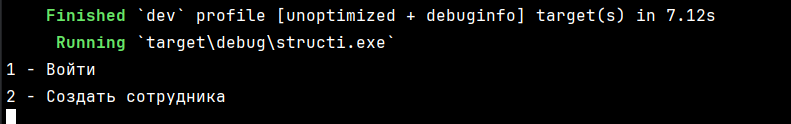
**Рисунок 12**



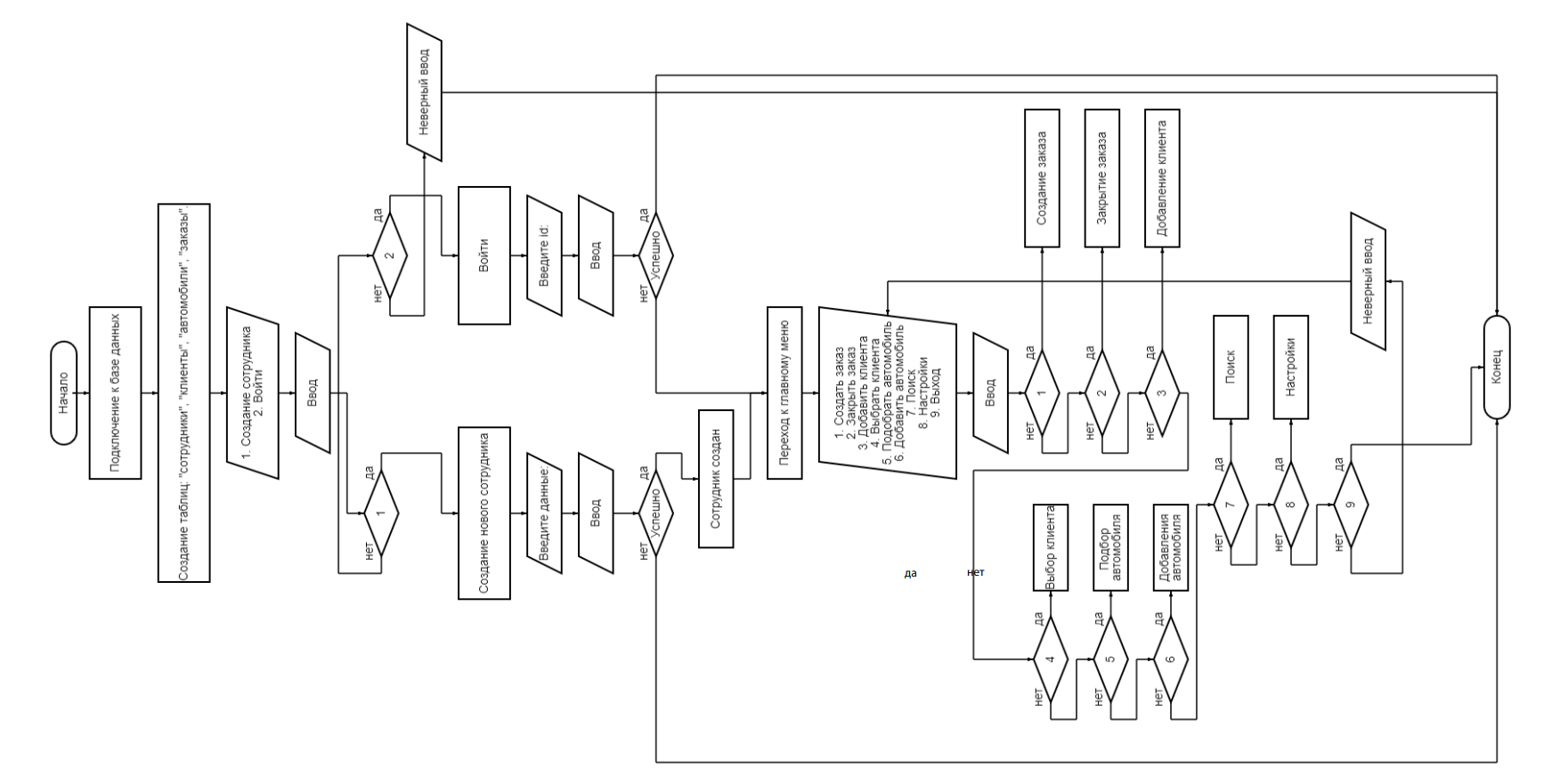
**Рисунок 13**



**Рисунок 14**

****

**ПРИЛОЖЕНИЕ 17. Блок схема 1**

****

**Диаграмма 1**

**Схема 1**

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

**Диаграмма 2**

добавляет

создает или выбирает

|  |
| --- |
| **Customer** |
| -id: int  -name: string  -phone: string  -email: string  -budget: int |
| +create\_customer()  +update\_customer ()  +delete\_customer()  +get\_customer()  +get\_all\_customers()  +search\_customer () |

|  |
| --- |
| **Employee** |
| -id: int  -name: string  -position: string  -salary: int  -hire\_date: string |
| +create\_employee()  +update\_employee()  +delete\_employee()  +get\_employee() |

|  |
| --- |
| **Car** |
| -id: in  -brand: string  -model: string  -year: int  -price: int  -mileage: int  -is\_new: bool |
| +create\_car()  +update\_car()  +delete\_car()  +get\_car()  +budget\_car\_under()  +budget\_car\_upper()  +search\_cars() |

|  |
| --- |
| **Order**  создает или закрывает |
| -id: in  -car\_id: string  -customer\_id: string  -employee\_id: int  -order\_date: int  -price: int  -is\_active: bool |
| +create\_order()  +update\_order()  +delete\_order()  +get\_order()  +cancel\_order() |