Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение  
 высшего образования Московский авиационный институт

(национальный исследовательский университет)

Филиал «РКТ» МАИ в г. Химки Московской области

**ОТЧЁТ ПО ПРАКТИКЕ**

Специальность 09.02.07 — Информационные системы и программирование

**Выполнили:**

Студентка группы ИСП-41-19

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Зварич С.В./

**Проверил:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Гуров В.В./

Химки, 2023

**Оглавление**

[**ВВЕДЕНИЕ** 3](#_Toc127962057)

[**1. Разработка форм для приложения** 4](#_Toc127962058)

[**2. Проектирование базы данных** 6](#_Toc127962059)

[2.1 Концептуальная модель 6](#_Toc127962060)

[2.2 Логическая модель 7](#_Toc127962061)

[2.3 Физическая модель 8](#_Toc127962062)

[**3. Разработка и настройка базы данных** 9](#_Toc127962063)

[**4. Разработка приложения** 13](#_Toc127962064)

[4.1 Разработка окна авторизации 13](#_Toc127962065)

[4.2 Разработка функционала приложения 14](#_Toc127962066)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 19](#_Toc127962067)

[**ПРИЛОЖЕННИЕ А. Исходный код приложения** 20](#_Toc127962068)

# **ВВЕДЕНИЕ**

Учебная практика является важной составной частью учебного процесса подготовки высококвалифицированных специалистов. В период прохождения практики студенты конкретизируют и закрепляют теоретические знания, приобретенные в процессе обучения, овладевают навыками практической работы, стараются применить полученные знания на практике. Практика является подводящим этапом в процессе подготовки будущего специалиста к самостоятельной практической деятельности.

Учебная практика по профессиональному модулю ПМ.02 «Осуществление интеграции программных модулей» проходила в филиале «РКТ» МАИ в городе Химки Московской области с 9 февраля 2023 года по 23 февраля 2023 года. Работа велась в очном формате.

Целью данной практики является разработка desktop-приложения для точки проката горнолыжного курорта «Игора».

# **1. Разработка форм для приложения**

Для создания интерфейса необходимы заготовки – формы. Для их разработки использовалось приложение «Qt Designer». Результат разработки смотреть в приложении А.

При разработке следовало соблюдать такие условия, как:

1. Наличие заголовка окна, логотипа;
2. Присутствие возможности изменения размера окна с учетом изменения размера контентной части (например, таблицы);
3. Ограничение на минимальный размер окна;
4. Группировка элементов в категории;
5. Последовательность интерфейса с возможностью перемещения между существующими окнами;
6. Заголовок соответствует содержанию окна;
7. Использование шрифта Comic Sans MS;
8. Использование заданных RGB кодировкой цветов: основной фон (255, 255, 255), дополнительный фон (118, 227, 131), акценты (73, 140,81).

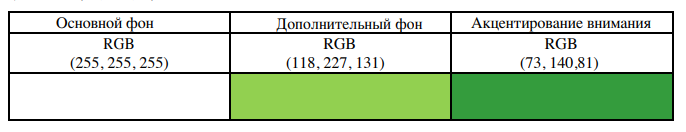


Рисунок 1 — Цветовая схема

Создание окон в «Qt Designer»:

1. Для создания главного окна ипользуется Main Window, для диалогового окна —Dialog without Buttons;
2. Для создания содержимого окна методом перетаскивания были использованы следующие виджеты:
   * Vertical Layout – вертикальная группировка виджетов;
   * Horizontal Layout – горизонтальная группировка виджетов;
   * Horizontal Spacer и Vertical Spacer нужны для заполнения пространство между виджетами, коррекции расположения;
   * Push Button – кнопка, при нажатии на которую посылается сигнал, который может быть обработан в коде;
   * QTableWidget – табличное представление элементов для отображения данных из базы данных;
   * ComboBox представляет собой всплывающий список;
   * LineEdit – это однострочный текстовый редактор, в который можно записать текст;
   * Label – вывод текста на экран;
   * GraphicsView предоставляет виджет для отображения содержимого QGraphicsScene.
3. Для группировки содержимого необходимо указать layout главного окна;
4. Для изменения названия виджета необходимо выбрать нужный виджет (лкм по виджету) и в «property editor» изменить поле «objectName»;
5. Для изменения параметров шрифта виджета необходимо нажать на нужный виджет пкм и выбрать «Change Style Sheet…»;
6. Для изменения минимального размера виджета необходимо выбрать нужный виджет и в «property editor» изменить параметры «minimumSize» (width, height);
7. Для установления изображения в label необходимо изменить поле «pixmap», а для кнопки поле «icon»;
8. Чтобы установить иконку окна нужно изменить поле «windowIcon», а для изменения заголовка окна нужно изменить поле «windowTitle».

# **2. Проектирование базы данных**

Для разработки базы данных, используемой приложением, необходимо правильно составить схему, отображающую сущности, атрибуты сущностей и их связи между собой. Для этого используются ER-диаграммы.

ER-диаграммы используют следующие элементы:

* сущность – подразумевает понятие или объект, несущий определенную информацию;
* атрибуты – характеризуют сущность;
* связь – обозначает взаимодействие между сущностями.

В данном случае предметной областью является горнолыжный курорт «Игора». Имеет следующие сущности и атрибуты:

* Сущность «Сотрудник». Атрибуты: идентификационный номер сотрудника, ФИО, должность, логин, пароль, последний вход, тип входа;
* Сущность «Клиент». Атрибуты: идентификационный номер клиента, фамилия, имя, отчество, серия паспорта, номер паспорта, дата рождения, адрес, e-mail;
* Сущность «Услуга». Атрибуты: идентификационный номер услуги, наименование услуги, код услуги, стоимость руб. за час;
* Сущность «Заказ». Атрибуты: идентификационный номер заказа, номер заказа, дата создания, время создания, клиент, услуга, статус заказа, дата закрытия, время проката, сотрудник.

2.1 Концептуальная модель

Концептуальная модель представляет собой абстрактную модель, определяющую структуру моделируемой системы. Отображает сущности, их атрибуты и связи между ними.

Для отображения сущности используется прямоугольник с её наименованием. Для указания атрибута указываются их наименования в овалах, связанных с прямоугольником линиями. Связь указывается ромбом, связывающим две сущности линиями.

Для отображения связей «один-к-одному», «один-к-многим» и «многие-к-многим» используется схематичное изображение, где перпендикулярная черта указывается на необходимость наличия только 1 элемента, 0 указывает на необязательность наличия у одной сущности данных из другой, а указание на использование данных несколько раз кисточка из 3 линий.

Для удобного построение ERD-диаграмм использован сервис ERDPlus.com. Для базы данных «Игора» создана диаграмма, указанная на рисунке 2.

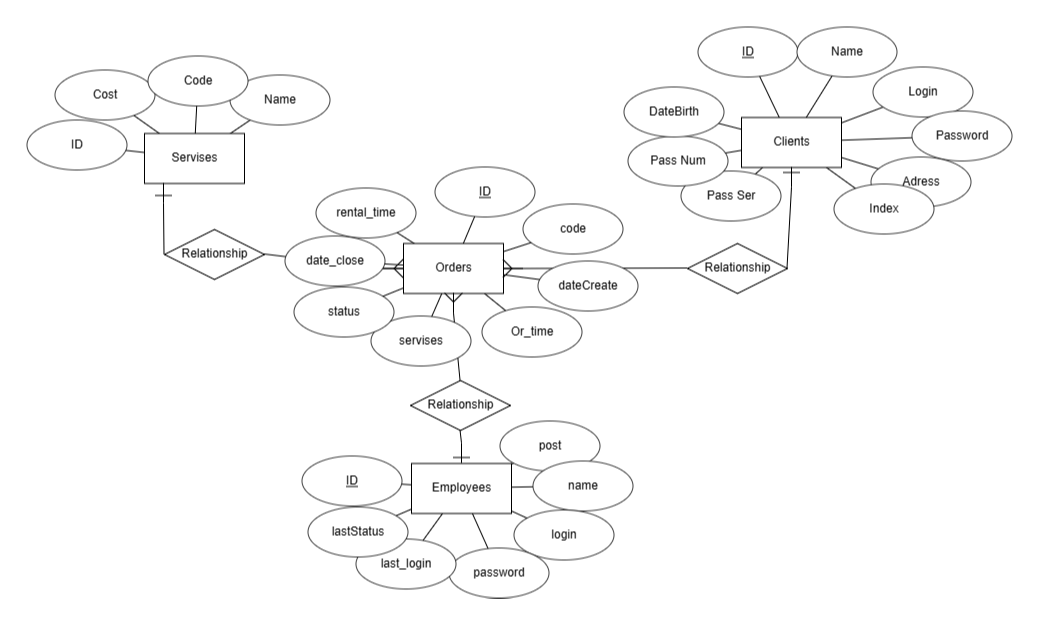


Рисунок 2 — ERD-диаграмма

2.2 Логическая модель

Логическая модель – это модель, указывающая графическое представление структуры базы данных с учетом всех сущностей, атрибутов, указанием всех связей, первичных и внешних ключей.

Логическая модель не привязана к конкретной СУБД.

Создание данной схемы так же возможна с помощью сервиса ERDPlus.com. Для этого в меню со схемами необходимо найти созданную диаграммы и с помощью контекстного меню (открывается нажатием на три точки) преобразовать в логическую модель с помощью “Convert to Relational Schema”.

На основе ERD-диаграммы создана логическая модель, указанная на рисунке 3.

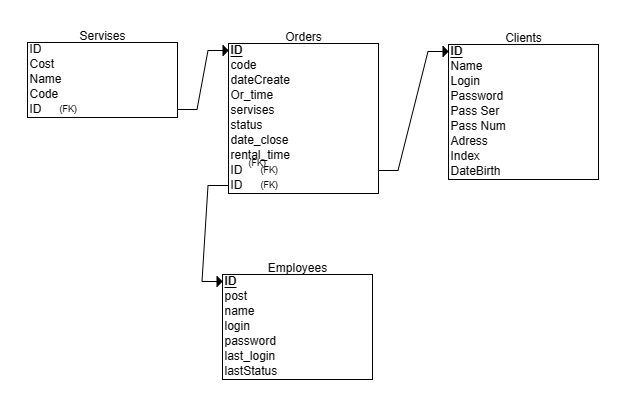


Рисунок 3 — Логическая модель.

2.3 Физическая модель

Физическая модель — это модель данных, содержащая все необходимые элементы для создания базы данных. Содержит все синтаксические особенности работы той или иной СУБД, указание типов данных.

Созданная физическая модель указана на рисунке 4.

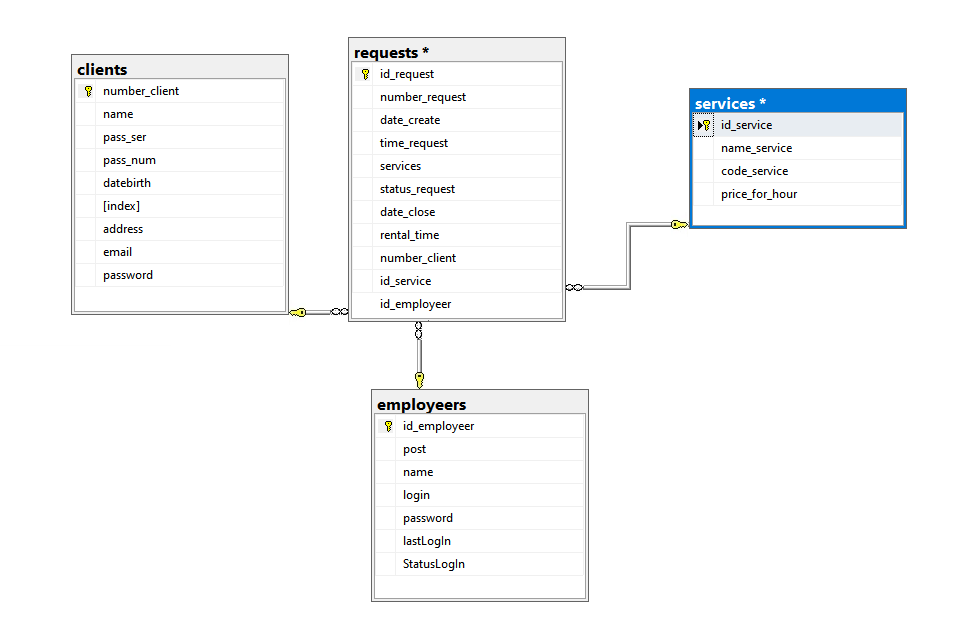


Рисунок 4 — Физическая модель.

# **3. Разработка и настройка базы данных**

В качестве СУБД выбрана Microsoft SQL Server Management.

SQL Server Management Studio (SSMS) — это интегрированная среда для управления любой инфраструктурой SQL. Используйте SSMS для доступа, настройки, администрирования и разработки всех компонентов SQL Server, Базы данных SQL Azure, Управляемого экземпляра SQL Azure, SQL Server на виртуальной машине Azure и Azure Synapse Analytics. Среда SSMS предоставляет единую комплексную служебную программу, которая сочетает в себе обширную группу графических инструментов с рядом многофункциональных редакторов скриптов для доступа к SQL Server для разработчиков и администраторов баз данных всех профессиональных уровней.

Настройка СУБД происходила следующим образом:

1. СУБД скачана с официального сайта в соответствии с инструкцией производителя: [Скачивание SQL Server Management Studio (SSMS) - SQL Server Management Studio (SSMS) | Microsoft Learn](https://learn.microsoft.com/ru-ru/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16)
2. При открытии СУБД появляется окно, указанное на рисунке 5.

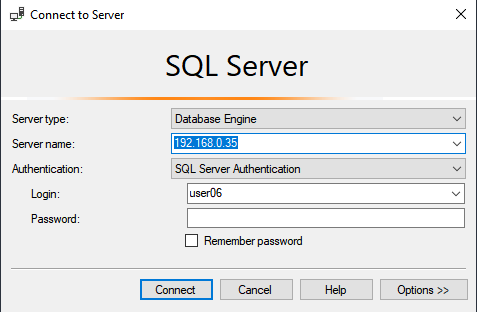


Рисунок 5 — Окно авторизации SSMS

По умолчанию наименование сервера и логин не указаны. В данном случае заранее создан сервер для создания локальной сети, предоставленной проходящим учебную практику. Указываются адрес сервера в графе названия, а логином и паролем является заранее созданный набор аккаунтов для подключения сети. В указанном примере это пользователь «user06», использующий логин «user06» и пароль «45358».

1. Создаем базу данных. Для этого использован следующий набор команд:

CREATE TABLE [dbo].[employeers](

[id\_employeer] [int] NOT NULL,

[post] [nvarchar](max) NOT NULL,

[name] [nvarchar](max) NOT NULL,

[login] [nvarchar](max) NOT NULL,

[password] [nvarchar](max) NOT NULL,

[lastLogIn] [nvarchar](max) NOT NULL,

[StatusLogIn] [nvarchar](max) NOT NULL,

PRIMARY KEY CLUSTERED

(

[id\_employeer] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

CREATE TABLE [dbo].[services](

[id\_service] [int] NOT NULL,

[name\_service] [nvarchar](max) NOT NULL,

[code\_service] [nvarchar](max) NOT NULL,

[price\_for\_hour] [int] NOT NULL,

PRIMARY KEY CLUSTERED

(

[id\_service] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

CREATE TABLE [dbo].[clients](

[number\_client] [int] NOT NULL,

[name] [nvarchar](max) NOT NULL,

[pass\_ser] [int] NOT NULL,

[pass\_num] [int] NOT NULL,

[datebirth] [nvarchar](max) NOT NULL,

[index] [int] NOT NULL,

[address] [nvarchar](max) NOT NULL,

[email] [nvarchar](max) NOT NULL,

[password] [nvarchar](max) NOT NULL,

PRIMARY KEY CLUSTERED

(

[number\_client] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

CREATE TABLE [dbo].[requests](

[id\_request] [int] NOT NULL,

[number\_request] [varchar](19) NOT NULL,

[date\_create] [varchar](10) NOT NULL,

[time\_request] [varchar](8) NOT NULL,

[services] [varchar](100) NOT NULL,

[status\_request] [varchar](50) NOT NULL,

[date\_close] [varchar](10) NOT NULL,

[rental\_time] [varchar](50) NOT NULL,

PRIMARY KEY CLUSTERED

(

[id\_request] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

1. Импортируем данные с помощью «Tasks» > «Import data…». Данные берем из excel документов, присланных вместе с техническим заданием. Результат выполнения импорта данных можно проверить с помощью выбора необходимой таблицы и вызова из контекстного меню запроса «Select top 1000 rows». Результат выполнения запроса представлен на рисунке 7.

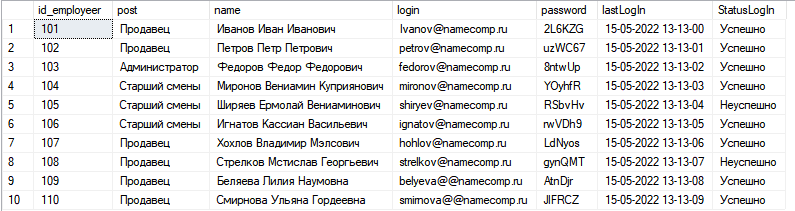


Рисунок 7 — Результат добавления данных

# **4. Разработка приложения**

4.1 Разработка окна авторизации

Для входа в систему сотруднику необходимо авторизоваться. Это создано для того, чтобы сотрудник попадал на окно работы, соответствующее его должности.

Авторизация созданна с помощью класса DialogAutorization. Изображение окна авторизации представлено на рисунке 8.

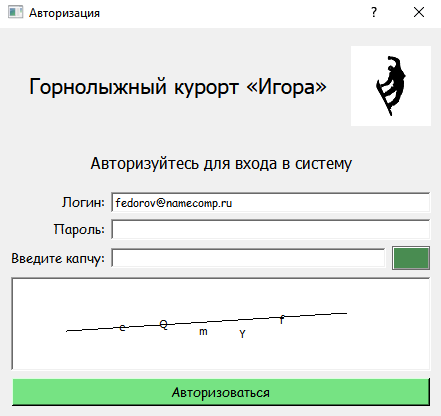


Рисунок 8 — окно авторизации.

Окно авторизации содержит следующий функционал:

1. Капча:
   * Появляется после 3 неудачных попыток ввода;
   * Содержит набор из нескольких символов, перечеркнутых линией;
   * Может быть перегенерирована кнопкой рядом с полем ввода капчи.
2. Разграничение уровня доступа:
   * Сотрудники с уровнем «Продавец» попадут на окно «Меню продавца»;
   * Сотрудники с уровнем «Старший смены» попадут на окно «Меню старшего смены»;
   * Сотрудники с уровнем «Администратор» попадут на окно «Меню администратора».
3. Сообщения о неверно введенном пароле, логине или капче.

Исходный код класса можно посмотреть в приложении А. Класс содержит следующие функции:

1. \_\_init\_\_ — инициация окна;
2. Autoriz — функция проверки введенных данных;
3. shif\_head\_open — открытие окна старшего смены;
4. admin\_open — открытие окна администратора;
5. seller\_open — открытие окна продавца;
6. gen\_captcha — генерация капчи;
7. empty\_pole, wrong\_captcha, wrong\_pass\_msg, wrong\_log\_msg — сообщения об ошибках.

4.2 Разработка функционала приложения

Приложение содержит следующие классы:

1. AdminMenu;
2. ShiftHeadMenu;
3. SellerMenu;
4. DialogAdd;
5. DialogAddEmp;
6. DialogAddServ;
7. Builder.

Классы DialogAdd, DialogAddEmp, DialogAddServ являются классами, содержащими окна добавления заказа, сотрудника и услуги соответственно. Каждый из них содержит функции инициализации окна (\_\_init\_\_) и функции add, обеспечивающие добавление данных в соответствующие таблицы. Интерфейсы окон представлены на рисунках 9, 10, 11.

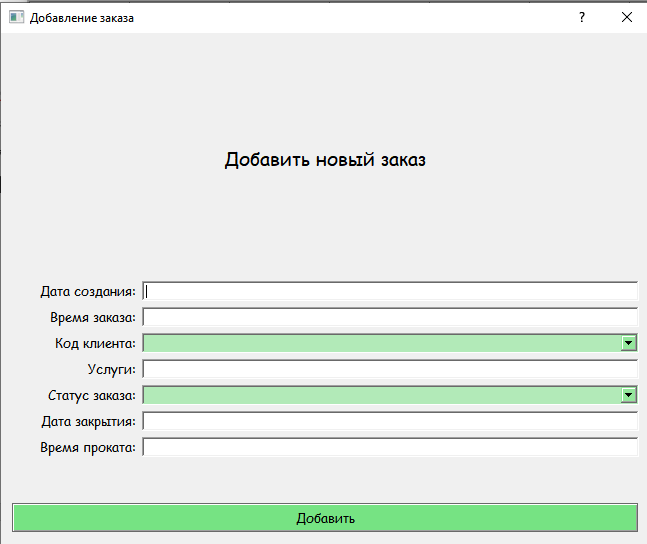


Рисунок 9 — Добавление заказа.

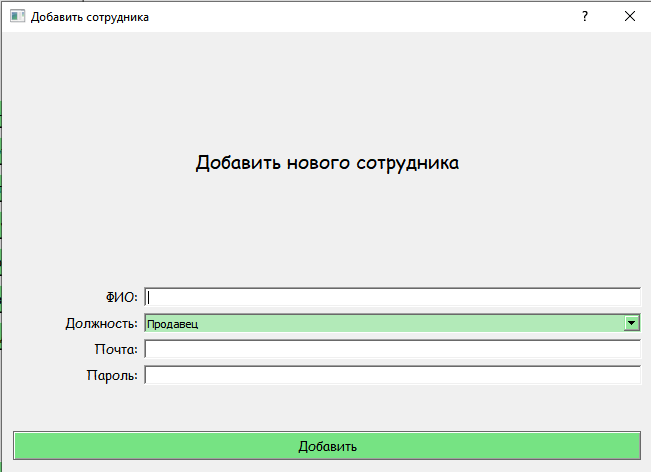


Рисунок 10 — Добавление сотрудника.

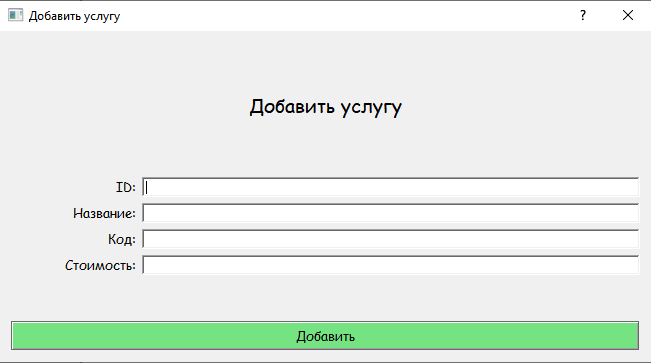


Рисунок 11 — Добавление услуги.

Классы AdminMenu, ShiftHeadMenu, SellerMenu являются специализированными окнами, содержащими возможности взаимодействия сотрудников с базой данных в соответствии с их уровнем доступа. Все содержат функцию инициализации окна (\_\_init\_\_).

Окно «Меню продавца» содержит только возможность добавления заказа (add) и просмотра таблицы заказов. Интерфейс представлен на рисунке 12.

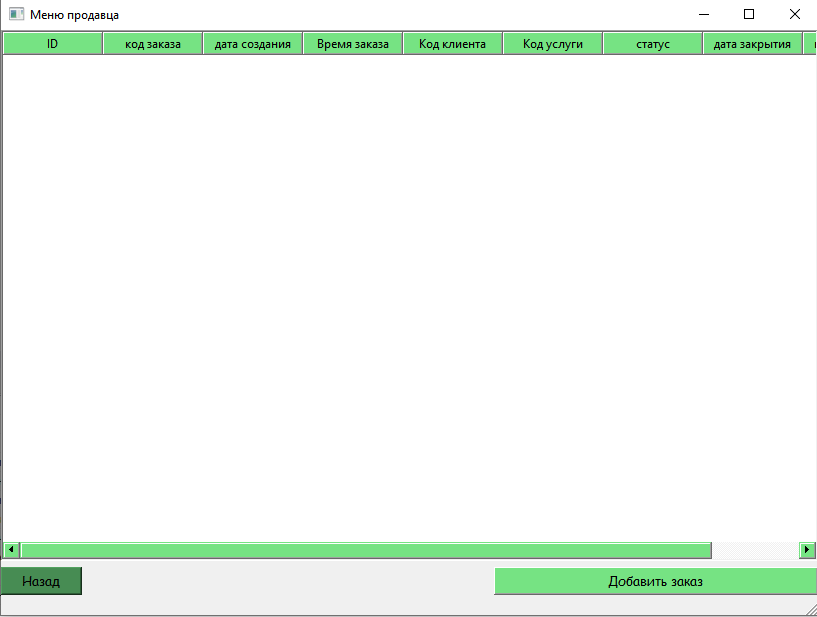


Рисунок 12 — Меню продавца.

Окно «Меню старшего смены» включает в себя возможность просмотра тиблицы заказов, добавления заказов (add), а так же изменения состояния заказов (принять оборудование). Интерфейс представлен на рисунке 13.

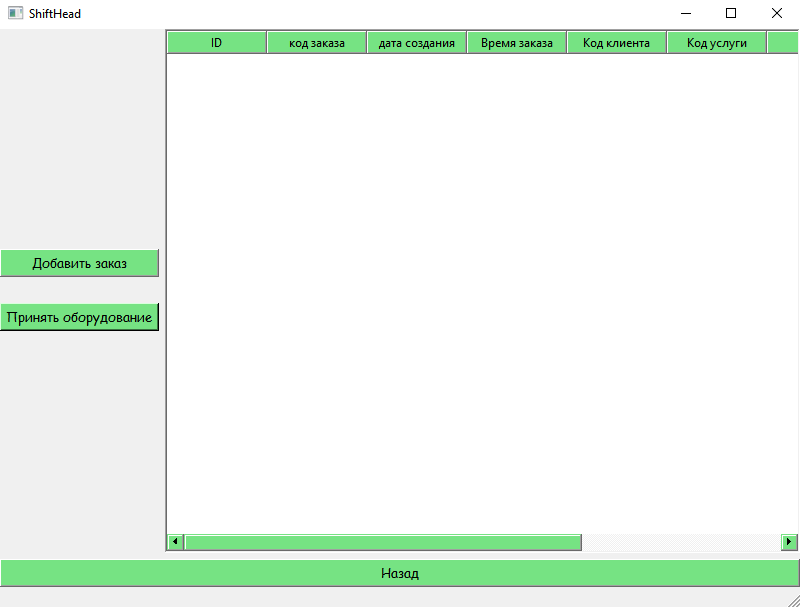


Рисунок 13 — Меню старшего смены.

Окно «Меню администратора» содержит следующий функционал:

1. employers — показывает таблицу сотрудников;
2. add\_emp — вызывает окно добавления сотрудника;
3. servises — показывает таблицу услуг;
4. add\_serv — вызывает окно добавления услуги;
5. orders — вызывает таблицу заказов;
6. history — вызывает таблицу с историей входа;
7. expenses — вызывает таблицу расходов;
8. exit — при нажатии на кнопку «назад» возвращает в окно авторизации.

Интерфейс меню администратора представлен на рисунке 14.

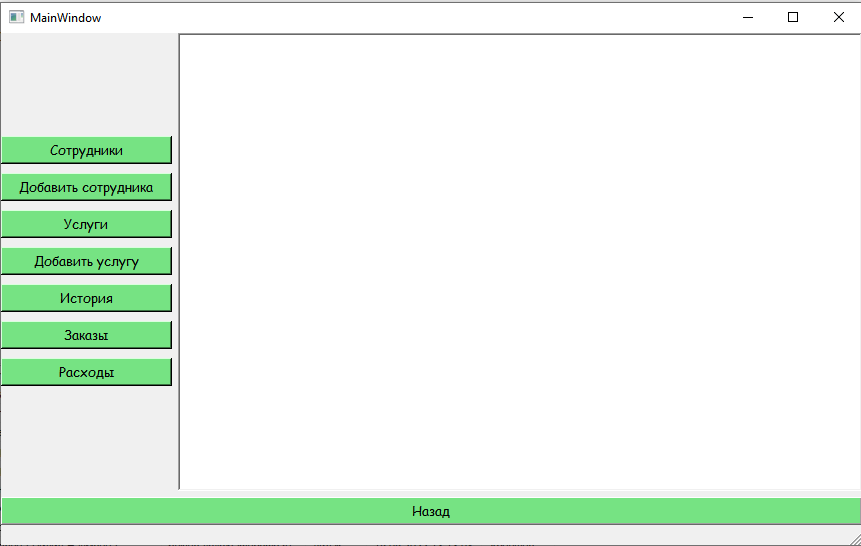


Рисунок 14 — Меню администратора.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. ГОСТ 7.32-2017 отчет о научно-исследовательской работе. М., Стандартинформ, 2017.
2. Прохоренок Н. А. Python 3 и PyQt. Разработка приложений. — СПб.: БХВ-Петербург, 2012. — 704 с.: ил.
3. Буйначев С. К. Основы программирования на языке python: учебное пособие / С. К. Буйначев, Н. Ю. Боклаг. – Екатеринбург: Издательствово Урал. ун-та, 2014. – 91, [1] c.
4. 3) 2 Лутц М. Изучаем Python, 3 е издание – Пер. с англ. – СПб.: Символ Плюс, 2009. – 848 с., ил.
5. Inno Setup Downloads. — URL: https://jrsoftware.org/isdl.php (дата обращения 2022-06-13).

**ПРИЛОЖЕННИЕ А. Исходный код приложения**

Файл bd.py

import pyodbc

class Database:

def \_\_init\_\_(self):

driver = 'DRIVER={SQL Server}'

server = 'SERVER=192.168.0.35'

port = 'PORT=3306'

db = 'DATABASE=user06'

user = 'UID=user06'

pw = 'PWD=45358'

conn\_str = '; '.join([driver, server, port, db, user, pw])

self.conn = pyodbc.connect(conn\_str)

def getCLients(self):

cursor = self.conn.cursor()

cursor.execute("SELECT \* FROM clients")

clients = cursor.fetchall()

return clients

def getRequests(self):

cursor = self.conn.cursor()

cursor.execute("SELECT \* FROM requests")

requests = cursor.fetchall()

cursor.close()

return requests

def insertRequests(self, number\_request, date\_create, time\_request, number\_client, services,

status\_request, rental\_time):

cursor = self.conn.cursor()

cursor.execute(

f"INSERT INTO requests"

f"(number\_request, date\_create, time\_request, number\_client, services, status\_request, rental\_time)"

f"VALUES ('{number\_request}', '{date\_create}', '{time\_request}', '{number\_client}', '{services}', '{status\_request}', '{rental\_time}')"

)

cursor.close()

self.conn.commit()

def getEmployeers(self):

cursor = self.conn.cursor()

cursor.execute("SELECT \* FROM employeers")

employeers = cursor.fetchall()

cursor.close()

return employeers

def insertEmp(self, id, name, position, login, pas, lastLogIn, StatusLogIn):

cursor = self.conn.cursor()

cursor.execute(

f"INSERT INTO employeers"

f"(id\_employeer,post,name, login, password,lastLogIn,StatusLogIn)"

f"VALUES ('{id}', N'{position}', N'{name}', N'{login}', N'{pas}', '{lastLogIn}', N'{StatusLogIn}')"

)

cursor.close()

self.conn.commit()

def getEmpIdList(self):

cursor = self.conn.cursor()

cursor.execute("SELECT id\_employeer FROM employeers")

empIdList = [str(i)[1:-3] for i in cursor.execute("SELECT id\_employeer FROM employeers")]

cursor.close()

return empIdList

def getServices(self):

cursor = self.conn.cursor()

cursor.execute("SELECT \* FROM services")

services = cursor.fetchall()

cursor.close()

return services

def insertServ(self, id, name, code, cost):

cursor = self.conn.cursor()

cursor.execute(

f"INSERT INTO services"

f"(id\_service,name\_service, code\_service, price\_for\_hour)"

f"VALUES ('{id}', N'{name}', '{code}', '{cost}')"

)

cursor.close()

self.conn.commit()

def getHistory(self):

cursor = self.conn.cursor()

cursor.execute("SELECT \* FROM History")

history = cursor.fetchall()

cursor.close()

return history

def check\_login(self):

log = []

cursor = self.conn.cursor()

cursor.execute(f"""SELECT login FROM employeers""")

rows = cursor.fetchall()

for i in rows:

for j in i:

log.append(j)

cursor.close()

return log

def get\_log(self, login):

log = []

cursor = self.conn.cursor()

cursor.execute(f"""SELECT password, post FROM employeers WHERE login = '{login}'""")

rows = cursor.fetchall()

for i in rows:

for j in i:

log.append(j)

cursor.close()

return log

Файл gui.py

"""

Впервые попав в лабиринт мышь долго выбирается из него,

но впоследствии попадая туда она безошибочно быстро находит выход.

Вот вам яркий пример того, как работает мышечная память.

"""

import sys

from bd import Database

import random

from PyQt5 import uic

from PyQt5.QtCore import Qt

from PyQt5.QtWidgets import QMainWindow, QMessageBox, QApplication, QDialog, QGraphicsScene, QTableWidgetItem

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super(MainWindow, self).\_\_init\_\_()

self.ui = uic.loadUi("forms/admin.ui", self)

self.setWindowTitle("ничто")

def authoriz(self, wnd):

dialog = DialogAutorization(wnd)

dialog.setWindowTitle("Авторизация")

dialog.show()

dialog.exec\_()

class DialogAutorization(QDialog):

def \_\_init\_\_(self, wnd, parent = None):

self.wnd = wnd

super(DialogAutorization, self).\_\_init\_\_(parent)

self.ui = uic.loadUi("forms/auth.ui", self)

self.scene = QGraphicsScene(0, 0, 350, 50)

self.scene.clear()

self.ui.autorization\_btn.clicked.connect(self.autoriz)

self.ui.captcha\_gen.setScene(self.scene)

self.ui.reboot\_btn.clicked.connect(self.gen\_captcha)

self.db = Database()

self.enter\_try = 0

self.cur\_captcha = None

def autoriz(self):

login = self.ui.line\_log.text()

password = self.ui.line\_pas.text()

if self.enter\_try >= 2:

self.gen\_captcha()

if login == '' or password == '':

self.empty\_pole()

if login not in self.db.check\_login():

self.wrong\_log\_msg()

self.enter\_try += 1

else:

aut = self.db.get\_log(login)

autpas = aut[0]

role = aut[1]

if self.enter\_try > 1 and self.ui.line\_cap.text() != self.cur\_captcha:

self.wrong\_captcha()

self.enter\_try += 1

if password != autpas:

self.enter\_try += 1

self.wrong\_pass\_msg()

else:

if role == 'Старший смены':

self.shif\_head\_open()

if role == 'Администратор':

self.admin\_open()

if role == 'Продавец':

self.seller\_open()

def shif\_head\_open(self):

self.ui.close()

self.ui = ShiftHeadMenu()

self.ui.show()

def admin\_open(self):

self.ui.close()

self.ui = AdminMenu()

self.ui.show()

def seller\_open(self):

self.ui.close()

self.ui = SellerMenu()

self.ui.show()

def gen\_captcha(self):

self.scene.clear()

symb = 'qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM1234567890'

s\_count = 5

cur\_symb = [1, 2, 3, 4, 5]

x, y = 30, 20

self.scene.addLine(20, random.randint(10, 40), 300, random.randint(10, 40))

for i in range(s\_count):

cur\_symb[i] = symb[random.randint(0, 61)]

text = self.scene.addText(f"{cur\_symb[i]}")

x += 40

text.moveBy(x, y+random.randint(-10, 10))

self.cur\_captcha=''.join(cur\_symb)

def empty\_pole(self):

self.mesbox = QMessageBox(self)

self.mesbox.setWindowTitle("Ошибка ввода")

self.mesbox.setText("Заполните все необходимые поля!")

self.mesbox.setStandardButtons(QMessageBox.Ok)

self.mesbox.show()

def wrong\_captcha(self):

self.mesbox = QMessageBox(self)

self.mesbox.setWindowTitle("Ошибка ввода")

self.mesbox.setText("Неверно введена капча!")

self.mesbox.setStandardButtons(QMessageBox.Ok)

self.mesbox.show()

def wrong\_pass\_msg(self):

self.mesbox = QMessageBox(self)

self.mesbox.setWindowTitle("Ошибка ввода")

self.mesbox.setText("Неверно введен пароль!")

self.mesbox.setStandardButtons(QMessageBox.Ok)

self.mesbox.show()

def wrong\_log\_msg(self):

self.mesbox = QMessageBox(self)

self.mesbox.setWindowTitle("Ошибка ввода")

self.mesbox.setText("Неверно введен логин.")

self.mesbox.setStandardButtons(QMessageBox.Ok)

self.mesbox.show()

class ShiftHeadMenu(QMainWindow):

def \_\_init\_\_(self):

super(ShiftHeadMenu, self).\_\_init\_\_()

self.ui = uic.loadUi("forms/shift\_head.ui", self)

self.window().setWindowTitle("ShiftHead")

self.ui.back\_btn.clicked.connect(self.exit)

self.table = self.ui.tableWidget

self.db = Database()

self.ui.add\_order\_btn.clicked.connect(self.add\_order)

self.ui.equipment\_btn.clicked.connect(self.equip)

self.orders()

def add\_order(self):

dialog = DialogAdd()

dialog.setWindowTitle("Добавить заказ")

dialog.show()

dialog.exec\_()

def orders(self):

self.table.clear()

out = self.db.getRequests()

self.table.setColumnCount(9) # кол-во столбцов

self.table.setRowCount(len(out)) # кол-во строк

self.table.setHorizontalHeaderLabels(['ID', 'код заказа', 'дата создания','Время заказа','Код клиента','Код услуги','статус', 'дата закрытия','время аренды'])

for i, order in enumerate(out):

for x, field in enumerate(order): # i, x - координаты ячейки, в которую будем записывать текст

item = QTableWidgetItem()

item.setText(str(field)) # записываем текст в ячейку

item.setFlags(Qt.ItemIsEnabled)

self.table.setItem(i, x, item)

def equip(self):

self.mesbox = QMessageBox(self)

self.mesbox.setWindowTitle("Ошибка")

self.mesbox.setText("По техническим причинам данная функция недоступна сейчас!")

self.mesbox.setStandardButtons(QMessageBox.Ok)

self.mesbox.show()

def exit(self):

dialog = DialogAutorization(self.window)

self.ui.close()

dialog.setWindowTitle("Авторизация")

dialog.show()

dialog.exec\_()

class AdminMenu(QMainWindow):

def \_\_init\_\_(self):

super(AdminMenu, self).\_\_init\_\_()

self.ui = uic.loadUi("forms/admin.ui", self)

self.db = Database()

self.ui.emp\_btn.clicked.connect(self.employers)

self.ui.add\_emp\_btn.clicked.connect(self.add\_emp)

self.ui.servises\_btn.clicked.connect(self.servises)

self.ui.add\_servises\_btn.clicked.connect(self.add\_serv)

self.ui.orders\_btn.clicked.connect(self.orders)

self.ui.history\_btn.clicked.connect(self.history)

self.ui.expenses\_btn.clicked.connect(self.expenses)

self.ui.back\_btn.clicked.connect(self.exit)

self.table = self.ui.tableWidget

def employers(self):

self.table.clear()

out = self.db.getEmployeers()

self.table.setColumnCount(7) # кол-во столбцов

self.table.setRowCount(len(out)) # кол-во строк

self.table.setHorizontalHeaderLabels(

['ID', 'Должность', 'ФИО', 'Логин', 'Пароль', 'Последний вход', 'Статус авторизации'])

for i, order in enumerate(out):

for x, field in enumerate(order): # i, x - координаты ячейки, в которую будем записывать текст

item = QTableWidgetItem()

item.setText(str(field)) # записываем текст в ячейку

if x == 0: # для id делаем некликабельные ячейки

item.setFlags(Qt.ItemIsEnabled)

self.table.setItem(i, x, item)

def add\_emp(self):

dialog = DialogAddEmp()

dialog.setWindowTitle("Добавить сотрудника")

dialog.show()

dialog.exec\_()

def servises(self):

self.table.clear()

out = self.db.getServices()

self.table.setColumnCount(4) # кол-во столбцов

self.table.setRowCount(len(out)) # кол-во строк

self.table.setHorizontalHeaderLabels(

['ID', 'Название', 'Код', 'Цена'])

for i, order in enumerate(out):

for x, field in enumerate(order): # i, x - координаты ячейки, в которую будем записывать текст

item = QTableWidgetItem()

item.setText(str(field)) # записываем текст в ячейку

if x == 0: # для id делаем некликабельные ячейки

item.setFlags(Qt.ItemIsEnabled)

self.table.setItem(i, x, item)

def add\_serv(self):

dialog = DialogAddServ()

dialog.setWindowTitle("Добавить услугу")

dialog.show()

dialog.exec\_()

def orders(self):

self.table.clear()

out = self.db.getRequests()

self.table.setColumnCount(9) # кол-во столбцов

self.table.setRowCount(len(out)) # кол-во строк

self.table.setHorizontalHeaderLabels(['ID', 'код заказа', 'дата создания','Время заказа','Код клиента','Код услуги','статус', 'дата закрытия','время аренды'])

for i, order in enumerate(out):

for x, field in enumerate(order): # i, x - координаты ячейки, в которую будем записывать текст

item = QTableWidgetItem()

item.setText(str(field)) # записываем текст в ячейку

if x == 0: # для id делаем некликабельные ячейки

item.setFlags(Qt.ItemIsEnabled)

self.table.setItem(i, x, item)

def history(self):

self.mesbox = QMessageBox(self)

self.mesbox.setWindowTitle("Ошибка")

self.mesbox.setText("По техническим причинам данная функция недоступна сейчас!")

self.mesbox.setStandardButtons(QMessageBox.Ok)

self.mesbox.show()

def expenses(self):

self.mesbox = QMessageBox(self)

self.mesbox.setWindowTitle("Ошибка")

self.mesbox.setText("По техническим причинам данная функция недоступна сейчас!")

self.mesbox.setStandardButtons(QMessageBox.Ok)

self.mesbox.show()

def exit(self):

dialog = DialogAutorization(self.window)

self.ui.close()

dialog.show()

dialog.exec\_()

class SellerMenu(QMainWindow):

def \_\_init\_\_(self):

super(SellerMenu, self).\_\_init\_\_()

self.ui = uic.loadUi("forms/seller.ui", self)

self.table = self.ui.order\_table

self.db = Database()

self.ui.order\_add\_btn.clicked.connect(self.add\_order)

self.table = self.ui.order\_table

self.ui.back\_btn.clicked.connect(self.exit)

self.orders()

def exit(self):

dialog = DialogAutorization(self.window)

self.ui.close()

dialog.show()

dialog.exec\_()

def orders(self):

self.table.clear()

out = self.db.getRequests()

self.table.setColumnCount(9) # кол-во столбцов

self.table.setRowCount(len(out)) # кол-во строк

self.table.setHorizontalHeaderLabels(['ID', 'код заказа', 'дата создания','Время заказа','Код клиента','Код услуги','статус', 'дата закрытия','время аренды'])

for i, order in enumerate(out):

for x, field in enumerate(order): # i, x - координаты ячейки, в которую будем записывать текст

item = QTableWidgetItem()

item.setText(str(field)) # записываем текст в ячейку

item.setFlags(Qt.ItemIsEnabled)

self.table.setItem(i, x, item)

def add\_order(self):

dialog = DialogAdd()

dialog.show()

dialog.exec\_()

class DialogAdd(QDialog):

def \_\_init\_\_(self):

super(DialogAdd, self).\_\_init\_\_()

self.ui = uic.loadUi("forms/add\_order.ui", self)

self.db = Database()

self.ui.add\_btn\_2.clicked.connect(self.add\_err)

def add\_err(self):

self.mesbox = QMessageBox(self)

self.mesbox.setWindowTitle("Ошибка")

self.mesbox.setText("Добавление недоступно")

self.mesbox.setStandardButtons(QMessageBox.Ok)

self.mesbox.show()

def add(self):

create\_date = self.ui.create\_date.text()

order\_code = self.ui.order\_code.text()

order\_time = self.ui.order\_time.text()

client\_code = self.ui.client\_code.text()

services = self.ui.services.text()

order\_status = self.ui.order\_status.text()

use\_time = self.ui.use\_time.text()

self.db.insertRequests(order\_code, create\_date,order\_time, client\_code,services, order\_status,use\_time)

self.ui.close()

class DialogAddEmp(QDialog):

def \_\_init\_\_(self):

super(DialogAddEmp, self).\_\_init\_\_()

self.ui = uic.loadUi("forms/add\_emp.ui", self)

self.db = Database()

self.ui.add\_btn\_2.clicked.connect(self.add)

self.ui.comboBox.addItem("Продавец")

self.ui.comboBox.addItem("Старший смены")

self.ui.comboBox.addItem("Администратор")

def add(self):

idList = self.db.getEmpIdList()

idList.reverse()

id = int(idList[0]) + 1

name = self.ui.name.text()

position = self.ui.comboBox.currentText()

login = self.ui.login.text()

pas = self.ui.pas.text()

lastLogIn = "15-05-2022 13-13-00"

StatusLogIn = "Успешно"

self.db.insertEmp(id, name, position, login, pas, lastLogIn, StatusLogIn)

self.ui.close()

class DialogAddServ(QDialog):

def \_\_init\_\_(self):

super(DialogAddServ, self).\_\_init\_\_()

self.ui = uic.loadUi("forms/add\_serv.ui", self)

self.db = Database()

self.ui.add\_btn\_2.clicked.connect(self.add)

def add(self):

id = int(self.ui.id.text())

name = self.ui.name.text()

code = self.ui.code.text()

cost = int(self.ui.cost.text())

self.db.insertServ(id, name, code, cost)

self.ui.close()

class Builder:

def \_\_init\_\_(self):

self.app = QApplication(sys.argv)

self.wnd = MainWindow()

self.auth()

def auth(self):

self.wnd.authoriz(self.wnd)

self.app.exec()

if \_\_name\_\_ == '\_\_main\_\_':

B = Builder()