

**Федеральное государственное образовательное бюджетное
учреждение высшего образования
«Финансовый университет при Правительстве Российской Федерации»
(Финансовый университет)**

Колледж информатики и программирования

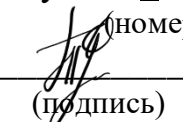
ОТЧЁТ
По учебной практике

Специальность 09.02.07 «Информационные системы и программирование»
(код) (наименование)

Профессиональный модуль ПМ.01 Разработка модулей программного обеспечения для компьютерных систем
(код) (наименование)

Междисциплинарный курс МДК.01.02 Поддержка и тестирование программных модулей
(код) (наименование)

Выполнил:

Студент 4 курса 4ИСИП-321 учебной группы
(номер) (номер)
 (подпись) Ф.А. Татарников
(инициалы, фамилия)

Проверил:

Руководитель практики от Колледжа
информатики и программирования
Преподаватель 1КК, к.п.н. Е.Л.Альшакова
(квалификационная категория или звание, (инициалы, фамилия)
должность)

Москва – 2024

Перечень работ, выполненных в ходе учебной практики

№ п/п	Виды работ	Оценка
1	Разработка кода программного продукта	ОТЛИЧНО
2	Рефакторинг кода программного продукта	ОТЛИЧНО
3	Разработка приложения	ОТЛИЧНО

СОДЕРЖАНИЕ

ОТЧЕТ О ВЫПОЛНЕНИИ ЗАДАНИЯ.....	4
---------------------------------	---

ОТЧЕТ О ВЫПОЛНЕНИИ ЗАДАНИЯ

Цель работы: разработка программного модуля для учета заявок на ремонт оборудования в соответствии с предоставленным техническим заданием. Разработанное приложение должно обеспечить удобный и эффективный процесс управления заявками, минимизировать простои оборудования, повысить качество обслуживания клиентов и сотрудников, а также предоставить возможность анализа и мониторинга процесса выполнения ремонтов. Приобрести практические навыки разработки приложения для учета заявок на ремонт оборудования, включая проектирование интерфейса на основе Tkinter, подключение базы данных, реализацию процесса авторизации пользователей и создание функциональных страниц приложения. Научиться организовывать логику работы с базой данных, добавлять, редактировать и удалять записи, а также анализировать статистику.

Перед началом разработки необходимо создать базу данных. Используя язык SQL создаем базу данных.

Диаграмма Базы данных представлена на рисунке 1.

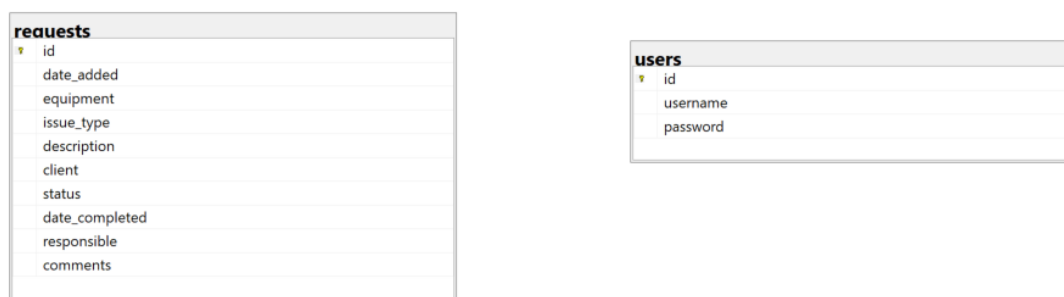


Рис. 1 Диаграмма базы данных.

Таблица requests (заявки):

- id: уникальный идентификатор заявки (INTEGER, основной ключ).
- date_added: дата и время добавления заявки (TEXT).
- equipment: оборудование, с которым возникла проблема (TEXT).

- issue_type: тип неисправности (TEXT).
- description: описание проблемы (TEXT).
- client: клиент, который сообщил о проблеме (TEXT).
- status: текущий статус заявки (TEXT, например, "Принята", "В работе", "Завершена").
- date_completed: дата завершения работы по заявке (TEXT, может быть NULL).
- responsible: ответственный за выполнение заявки (TEXT).
- comments: дополнительные комментарии (TEXT, может быть NULL).

Таблица users (пользователи):

- id: уникальный идентификатор пользователя (INTEGER, основной ключ).
- username: имя пользователя (TEXT, уникальное).
- password: пароль пользователя (TEXT).

База данных используется для хранения информации о заявках и пользователях системы. Для каждой заявки сохраняются данные, такие как оборудование, тип неисправности, описание проблемы, статус заявки и ответственный за выполнение. В таблице пользователей хранится информация о логинах и паролях для аутентификации. Созданная база данных показана на рисунках 2,3. Листинг скрипта для создания базы представлен в листинге 1.

requests			CREATE TABLE requests (id INTEGER PRIMARY
id	INTEGER	"id" INTEGER	
date_added	TEXT	"date_added" TEXT	
equipment	TEXT	"equipment" TEXT	
issue_type	TEXT	"issue_type" TEXT	
description	TEXT	"description" TEXT	
client	TEXT	"client" TEXT	
status	TEXT	"status" TEXT	
date_completed	TEXT	"date_completed" TEXT	
responsible	TEXT	"responsible" TEXT	
comments	TEXT	"comments" TEXT	

Рис. 2 Таблица requests базы данных.




users		CREATE TABLE users (id INTEGER PRIMARY KEY A
 id	INTEGER	"id" INTEGER
 username	TEXT	"username" TEXT UNIQUE
 password	TEXT	"password" TEXT

Рис. 3 Таблица requests базы данных.

Листинг 1 – Скрипт создания базы данных.

```
USE [master]
GO
CREATE DATABASE [repair_requests]
    CONTAINMENT = NONE
    ON PRIMARY
    ( NAME = N'repair_requests', FILENAME = N'C:\Program Files\Microsoft
SQL Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\repair_requests.mdf' , SIZE =
8192KB , MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB )
    LOG ON
    ( NAME = N'repair_requests_log', FILENAME = N'C:\Program
Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\repair_requests_log.ldf' , SIZE =
8192KB , MAXSIZE = 2048GB , FILEGROWTH = 65536KB )
    WITH CATALOG_COLLATION = DATABASE_DEFAULT, LEDGER = OFF
GO

IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))
begin
EXEC [repair_requests].[dbo].[sp_fulltext_database] @action = 'enable'
end
GO

ALTER DATABASE [repair_requests] SET ANSI_NULL_DEFAULT OFF
GO

ALTER DATABASE [repair_requests] SET ANSI_NULLS OFF
GO

ALTER DATABASE [repair_requests] SET ANSI_PADDING OFF
GO

ALTER DATABASE [repair_requests] SET ANSI_WARNINGS OFF
GO

ALTER DATABASE [repair_requests] SET ARITHABORT OFF
GO

ALTER DATABASE [repair_requests] SET AUTO_CLOSE OFF
GO

ALTER DATABASE [repair_requests] SET AUTO_SHRINK OFF
GO

ALTER DATABASE [repair_requests] SET AUTO_UPDATE_STATISTICS ON
GO

ALTER DATABASE [repair_requests] SET CURSOR_CLOSE_ON_COMMIT OFF
GO

ALTER DATABASE [repair_requests] SET CURSOR_DEFAULT GLOBAL
GO

ALTER DATABASE [repair_requests] SET CONCAT_NULL_YIELDS_NULL OFF
GO

ALTER DATABASE [repair_requests] SET NUMERIC_ROUNDABORT OFF
GO
```

```
ALTER DATABASE [repair_requests] SET QUOTED_IDENTIFIER OFF
GO

ALTER DATABASE [repair_requests] SET RECURSIVE_TRIGGERS OFF
GO

ALTER DATABASE [repair_requests] SET  DISABLE_BROKER
GO

ALTER DATABASE [repair_requests] SET AUTO_UPDATE_STATISTICS_ASYNC OFF
GO

ALTER DATABASE [repair_requests] SET DATE_CORRELATION_OPTIMIZATION OFF
GO

ALTER DATABASE [repair_requests] SET TRUSTWORTHY OFF
GO

ALTER DATABASE [repair_requests] SET ALLOW_SNAPSHOT_ISOLATION OFF
GO

ALTER DATABASE [repair_requests] SET PARAMETERIZATION SIMPLE
GO

ALTER DATABASE [repair_requests] SET READ_COMMITTED_SNAPSHOT OFF
GO

ALTER DATABASE [repair_requests] SET HONOR_BROKER_PRIORITY OFF
GO

ALTER DATABASE [repair_requests] SET RECOVERY FULL
GO

ALTER DATABASE [repair_requests] SET  MULTI_USER
GO

ALTER DATABASE [repair_requests] SET PAGE_VERIFY CHECKSUM
GO

ALTER DATABASE [repair_requests] SET DB_CHAINING OFF
GO

ALTER DATABASE [repair_requests] SET FILESTREAM( NON_TRANSACTED_ACCESS
= OFF )
GO

ALTER DATABASE [repair_requests] SET TARGET_RECOVERY_TIME = 60 SECONDS
GO

ALTER DATABASE [repair_requests] SET DELAYED_DURABILITY = DISABLED
GO

ALTER DATABASE [repair_requests] SET ACCELERATED_DATABASE_RECOVERY =
OFF
GO

ALTER DATABASE [repair_requests] SET QUERY_STORE = ON
GO

ALTER DATABASE [repair_requests] SET QUERY_STORE (OPERATION_MODE =
READ_WRITE, CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 30),
DATA_FLUSH_INTERVAL_SECONDS = 900, INTERVAL_LENGTH_MINUTES = 60,
MAX_STORAGE_SIZE_MB = 1000, QUERY_CAPTURE_MODE = AUTO,
```

```
SIZE_BASED_CLEANUP_MODE = AUTO, MAX_PLANS_PER_QUERY = 200,  
WAIT_STATS_CAPTURE_MODE = ON)  
GO  
  
ALTER DATABASE [repair_requests] SET READ_WRITE  
GO
```

Подключение базы данных SQL SM в коде показано в листинге 2.

Листинг 2 – Подключение базы данных.

```
def setup_database():  
    conn = sqlsm.connect('repair_requests.db')  
    cursor = conn.cursor()  
    cursor.execute('''  
        CREATE TABLE IF NOT EXISTS requests (  
            id INTEGER PRIMARY KEY AUTOINCREMENT,  
            date_added TEXT,  
            equipment TEXT,  
            issue_type TEXT,  
            description TEXT,  
            client TEXT,  
            status TEXT,  
            date_completed TEXT,  
            responsible TEXT,  
            comments TEXT  
        )  
    ''')  
    cursor.execute('''  
        CREATE TABLE IF NOT EXISTS users (  
            id INTEGER PRIMARY KEY AUTOINCREMENT,  
            username TEXT UNIQUE,  
            password TEXT  
        )  
    ''')
```

Первая страница приложения предназначена для авторизации пользователя в системе. На ней расположены два основных поля для ввода: логина и пароля, которые являются обязательными для входа в систему. Под полями для ввода могут находиться кнопка "Войти.. Внешний вид страницы прост и функционален, с ясным расположением элементов, чтобы пользователь мог легко и быстро выполнить авторизацию. Страница имеет минималистичный дизайн, что позволяет сосредоточиться на главных функциях — вводе данных и входе в систему. Визуальные элементы, такие как кнопки и поля ввода, оформлены в едином стиле, соответствующем общему дизайну приложения. Показано на рисунке 4.

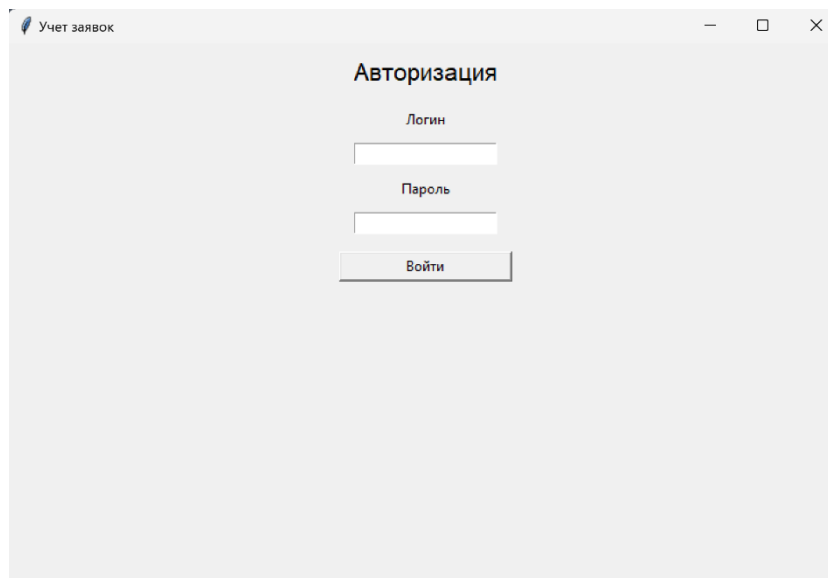


Рис.4 Страница авторизации.

В случае если пароль неправильный предусмотрена проверка и вывод ошибки. Показано на рисунке 5.

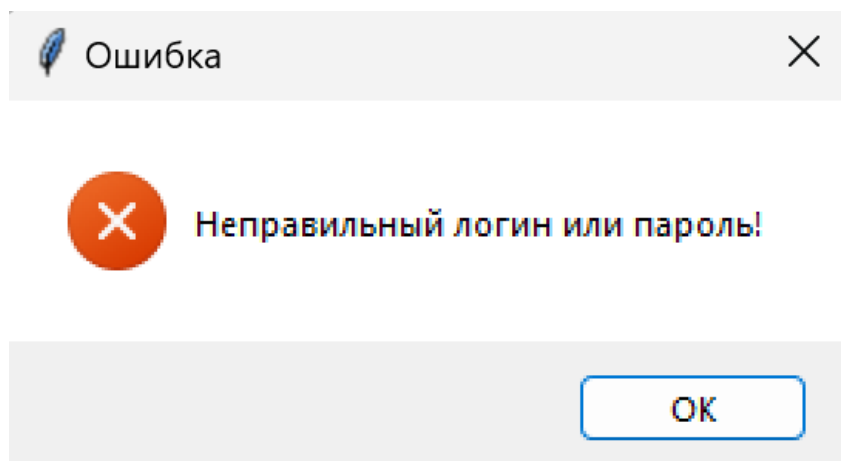


Рис. 5 Ошибка.

Код страницы авторизации представлен в листинге 3.

Листинг 3 – Код страницы авторизации.

```
class LoginPage(Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller

        Label(self, text="Авторизация", font=("Arial",
16)).pack(pady=10)

        Label(self, text="Логин").pack(pady=5)
        self.username_entry = Entry(self)
        self.username_entry.pack(pady=5)

        Label(self, text="Пароль").pack(pady=5)
        self.password_entry = Entry(self, show="*")
        self.password_entry.pack(pady=5)

        Button(self, text="Войти", command=self.login,
width=20).pack(pady=10)
```

```
def login(self):
    username = self.username_entry.get()
    password = self.password_entry.get()
    if verify_user(username, password):
        self.controller.show_frame("MainPage")
    else:
        messagebox.showerror("Ошибка", "Неправильный логин или
пароль!")
```

После авторизации у пользователя появляется главная страница, на которой выведены основные кнопки для управления приложением. Каждая кнопка ведет на определенную страницу управления. Показано на рисунке 6.

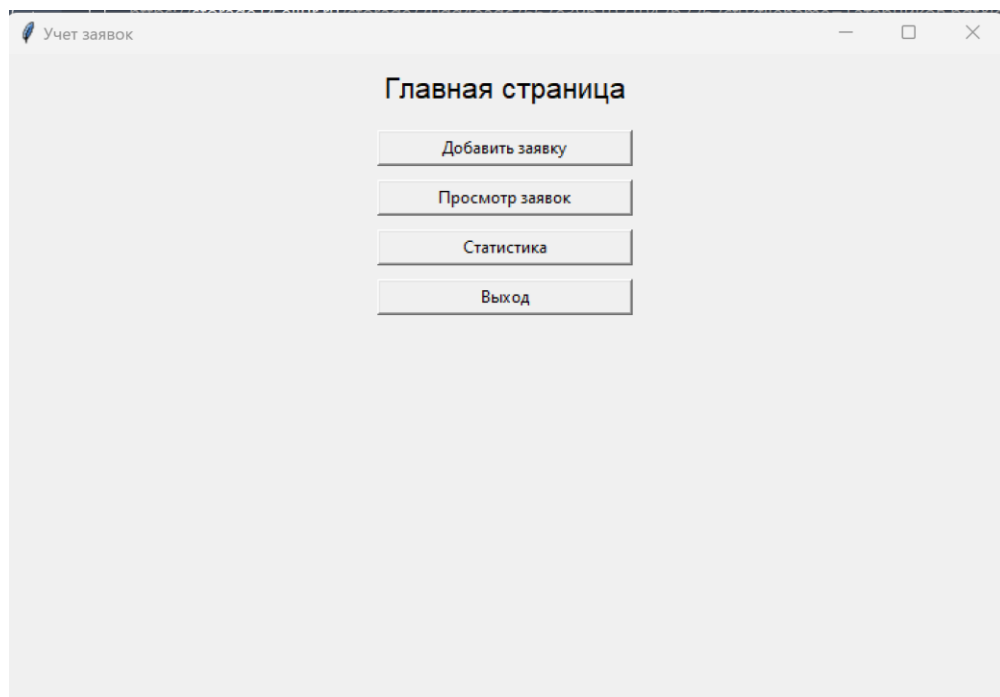


Рис. 6 Главная страница с кнопками управления.

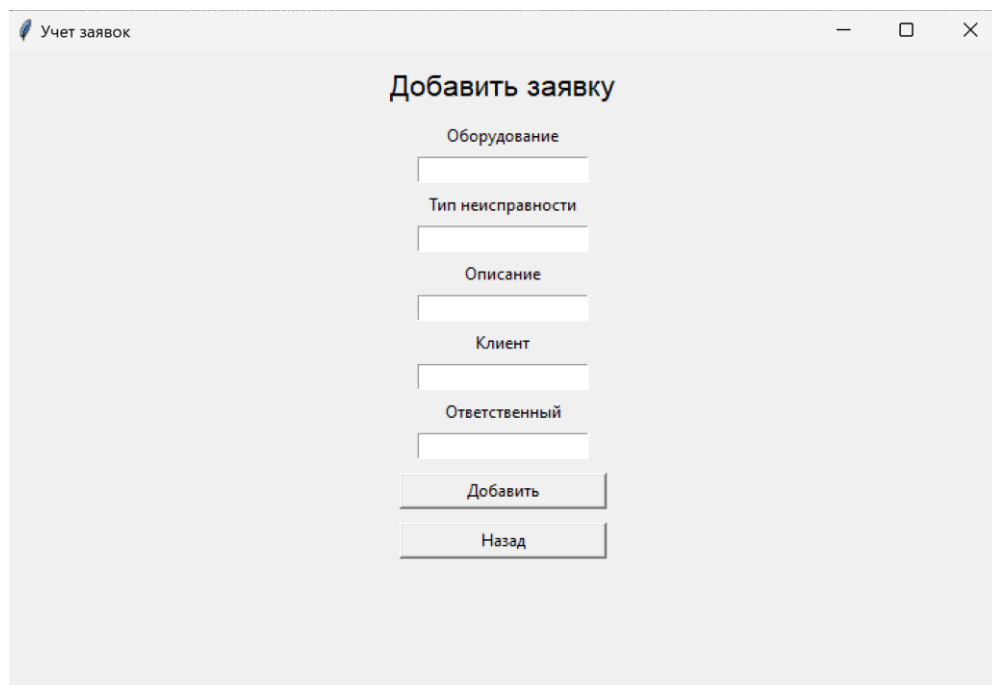
Код главной страницы представлен в листинге 4.

Листинг 4 – Код главной страницы.

```
class MainPage(Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller

        Label(self, text="Главная страница", font=("Arial",
16)).pack(pady=10)
        Button(self, text="Добавить заявку", command=lambda:
controller.show_frame("AddRequestPage"), width=25).pack(pady=5)
        Button(self, text="Просмотр заявок", command=lambda:
controller.show_frame("ViewRequestsPage"), width=25).pack(pady=5)
        Button(self, text="Статистика", command=lambda:
controller.show_frame("StatisticsPage"), width=25).pack(pady=5)
        Button(self, text="Выход", command=self.controller.quit,
width=25).pack(pady=5)
```

Кнопка Добавить заявку ведет на отдельную страницу с добавлениями заявки. Показано на рисунке 7. На ней выведены основные кнопки для управления приложением.



The screenshot shows a window titled 'Учет заявок' (Request Accounting) with a standard Windows title bar. The main content area is titled 'Добавить заявку' (Add Request). It contains a vertical stack of input fields with the following labels: 'Оборудование' (Equipment), 'Тип неисправности' (Type of fault), 'Описание' (Description), 'Клиент' (Client), and 'Ответственный' (Responsible). Below these fields are two buttons: 'Добавить' (Add) and 'Назад' (Back).

Рис 7. Страница добавления заявки.

Уведомление об успешном добавлении заявки показано на рисунке 8.

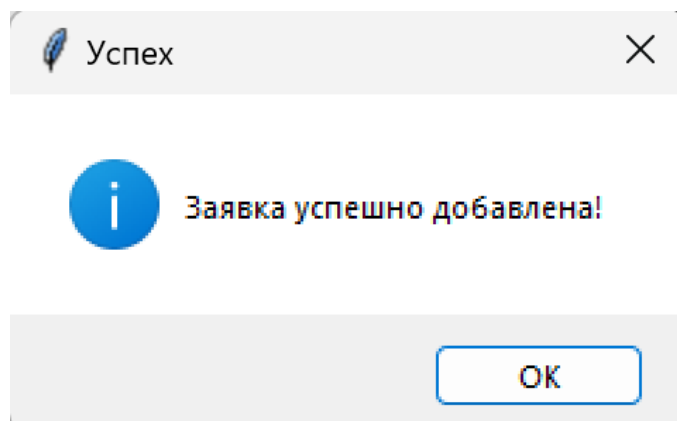


Рис. 8 Успешное добавление.

Если данные не заполнены выводится ошибка, обращающая внимание на то, что не все поля заполнены. Показано на рисунке 9.

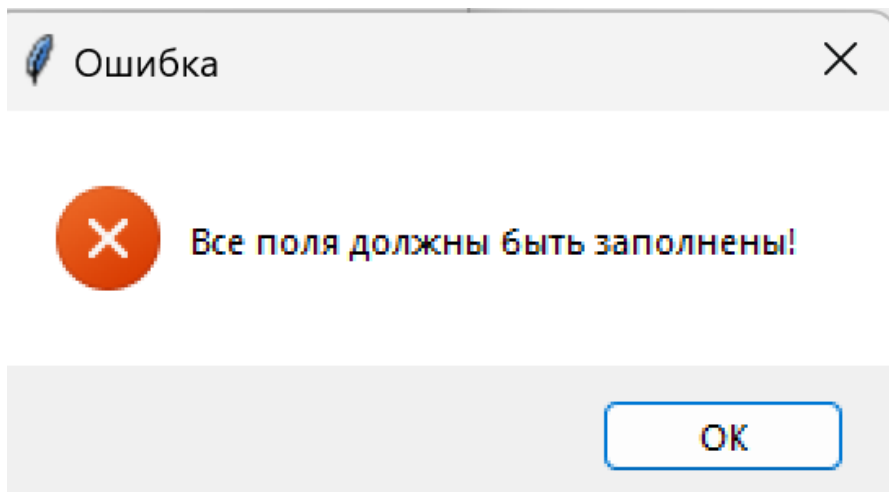


Рис. 9 Ошибка заполнения данных.

Кнопка назад отправляет пользователя на главную страницу.

Код страницы добавления заявки представлен в листинге 5.

Листинг 5 – Код страницы добавления заявок.

```
class AddRequestPage(Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller

        Label(self, text="Добавить заявку", font=("Arial",
16)).pack(pady=10)
        self.fields = {}
        labels = ["Оборудование", "Тип неисправности", "Описание",
"Клиент", "Ответственный"]
        for label_text in labels:
            Label(self, text=label_text).pack()
            entry = Entry(self)
            entry.pack(pady=5)
            self.fields[label_text] = entry

        Button(self, text="Добавить", command=self.add_request,
width=20).pack(pady=5)
        Button(self, text="Назад", command=lambda:
controller.show_frame("MainPage"), width=20).pack(pady=5)

    def add_request(self):
        data = {key: entry.get() for key, entry in self.fields.items()}
        if any(not value for value in data.values()):
            messagebox.showerror("Ошибка", "Все поля должны быть
заполнены!")
            return

        conn = sqlite3.connect('repair_requests.db')
        cursor = conn.cursor()
        date_added = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        cursor.execute('''
            INSERT INTO requests (date_added, equipment, issue_type,
description, client, status, date_completed, responsible, comments)
            VALUES (?, ?, ?, ?, ?, "Принята", NULL, ?, NULL)
            ''', (date_added, data["Оборудование"], data["Тип
неисправности"], data["Описание"], data["Клиент"],
data["Ответственный"])))
```

```
conn.commit()
conn.close()

messagebox.showinfo("Успех", "Заявка успешно добавлена!")
self.controller.show_frame("MainPage")
```

По нажатию на кнопку Просмотр заявок ведет на страницу, где пользователь может просматривать все заявки, найти по ID нужную заявку, а так же редактировать данные созданных заявок. Меню представлено на рисунке 10.

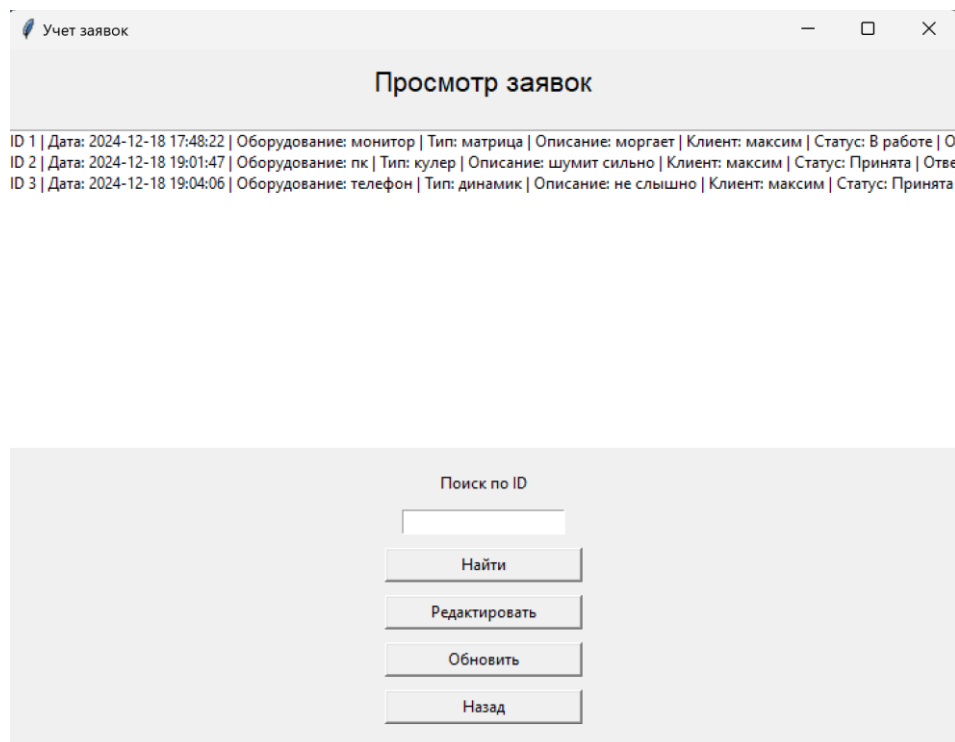


Рис. 10 Меню просмотра заявок.

При выборе заявки и по нажатию на кнопку "Редактировать" открывается окно с редактированием данных, где пользователь может изменить статус заявки. В этом окне отображаются все ключевые поля заявки. Пользователь может выбрать новый статус из выпадающего списка, что позволяет обновить информацию о заявке в соответствии с её текущим состоянием. Показано на рисунке 11.

Рис. 11 Редактирование заявки.

В случае если заявка не выбрана предусмотрена ошибка с просьбой выбрать заявку. Показано на рисунке 12.

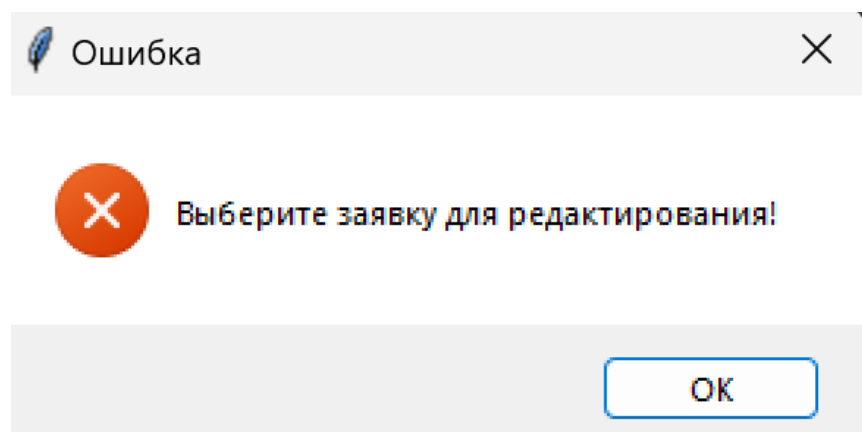


Рис. 12 Ошибка выбора заявки.

На рисунке 13 показана реализация поиска по ID заявки.

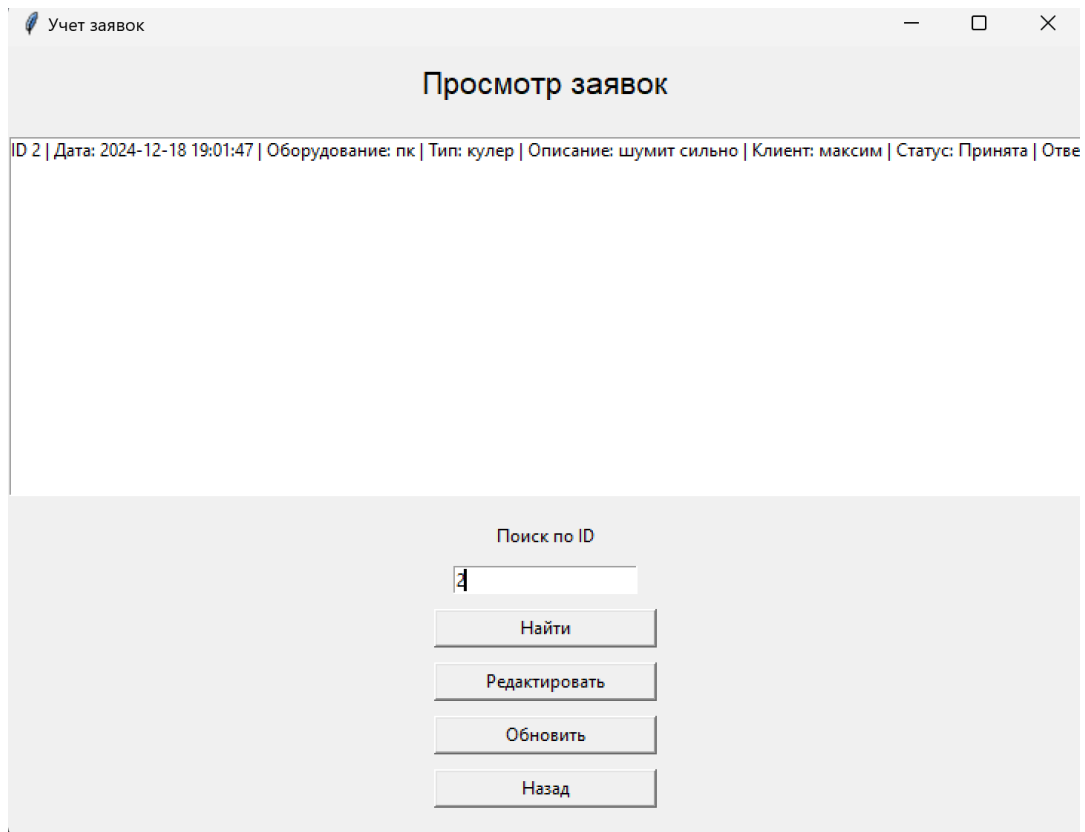


Рис 13. Поиск заявки по ID.

По кнопке назад пользователь возвращается на главную страницу приложения.

Код страницы представлен в листинге 6.

Листинг 6 – Код страницы просмотр заявок и редактирования.

```
class ViewRequestsPage(Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller

        Label(self, text="Просмотр заявок", font=("Arial",
16)).pack(pady=10)
        self.listbox = Listbox(self, width=120, height=15)
        self.listbox.pack(pady=10)

        Label(self, text="Поиск по ID").pack(pady=5)
        self.search_entry = Entry(self)
        self.search_entry.pack(pady=5)
        Button(self, text="Найти", command=self.search_request,
width=20).pack(pady=5)

        Button(self, text="Редактировать", command=self.edit_request,
width=20).pack(pady=5)
        Button(self, text="Обновить", command=self.load_requests,
width=20).pack(pady=5)
        Button(self, text="Назад", command=lambda:
controller.show_frame("MainPage"), width=20).pack(pady=5)

    def load_requests(self):
        self.listbox.delete(0, 'end')
```

```

        requests = list_requests()
        for request in requests:
            self.listbox.insert('end', f"ID {request[0]} | Дата:
{request[1]} | Оборудование: {request[2]} | Тип: {request[3]} | "
                                f"Описание: {request[4]} | "
Клиент: {request[5]} | Статус: {request[6]} | "
                                f"Ответственный: {request[8]}")

    def search_request(self):
        request_id = self.search_entry.get()
        if not request_id.isdigit():
            messagebox.showerror("Ошибка", "Введите корректный ID!")
            return

        request = get_request_by_id(int(request_id))
        if not request:
            messagebox.showerror("Ошибка", "Заявка не найдена!")
            return

        self.listbox.delete(0, 'end')
        self.listbox.insert('end', f"ID {request[0]} | Дата:
{request[1]} | Оборудование: {request[2]} | Тип: {request[3]} | "
                                f"Описание: {request[4]} | Клиент:
{request[5]} | Статус: {request[6]} | "
                                f"Ответственный: {request[8]}")

    def edit_request(self):
        selected = self.listbox.curselection()
        if not selected:
            messagebox.showerror("Ошибка", "Выберите заявку для
редактирования!")
            return

        request_id = int(self.listbox.get(selected[0]).split()[1])
        EditRequestWindow(self, request_id)

class EditRequestWindow(Toplevel):
    def __init__(self, parent, request_id):
        super().__init__(parent)
        self.request_id = request_id
        self.title("Редактировать заявку")
        self.geometry("400x500")

        self.fields = {}
        labels = ["Оборудование", "Тип неисправности", "Описание",
"Клиент", "Ответственный"]
        conn = sqlite3.connect('repair_requests.db')
        cursor = conn.cursor()
        cursor.execute("SELECT equipment, issue_type, description,
client, responsible, status FROM requests WHERE id = ?", (request_id,))
        data = cursor.fetchone()
        conn.close()

        for i, label_text in enumerate(labels):
            Label(self, text=label_text).pack()
            entry = Entry(self)
            entry.insert(0, data[i])
            entry.pack(pady=5)
            self.fields[label_text] = entry

        # Поле для редактирования статуса
        Label(self, text="Статус").pack()
        self.status_var = StringVar(value=data[5]) # Статус из базы
        данных

```



```

        statuses = ["Принята", "В работе", "Завершена"]
        OptionMenu(self, self.status_var, *statuses).pack(pady=5)

        Button(self, text="Сохранить", command=self.save_changes,
width=20).pack(pady=10)
        Button(self, text="Отмена", command=self.destroy,
width=20).pack(pady=10)

    def save_changes(self):
        data = {key: entry.get() for key, entry in self.fields.items()}
        if any(not value for value in data.values()):
            messagebox.showerror("Ошибка", "Все поля должны быть
заполнены!")
            return

        # Обновление данных заявки в базе данных
        update_request(
            self.request_id,
            data["Оборудование"],
            data["Тип неисправности"],
            data["Описание"],
            data["Клиент"],
            data["Ответственный"],
            self.status_var.get() # Новый статус
        )
        messagebox.showinfo("Успех", "Изменения сохранены!")
        self.destroy()

```

По кнопке Статистика пользователь может ознакомиться со статистикой заявок. Страница со статистикой представлена на рисунке 14.

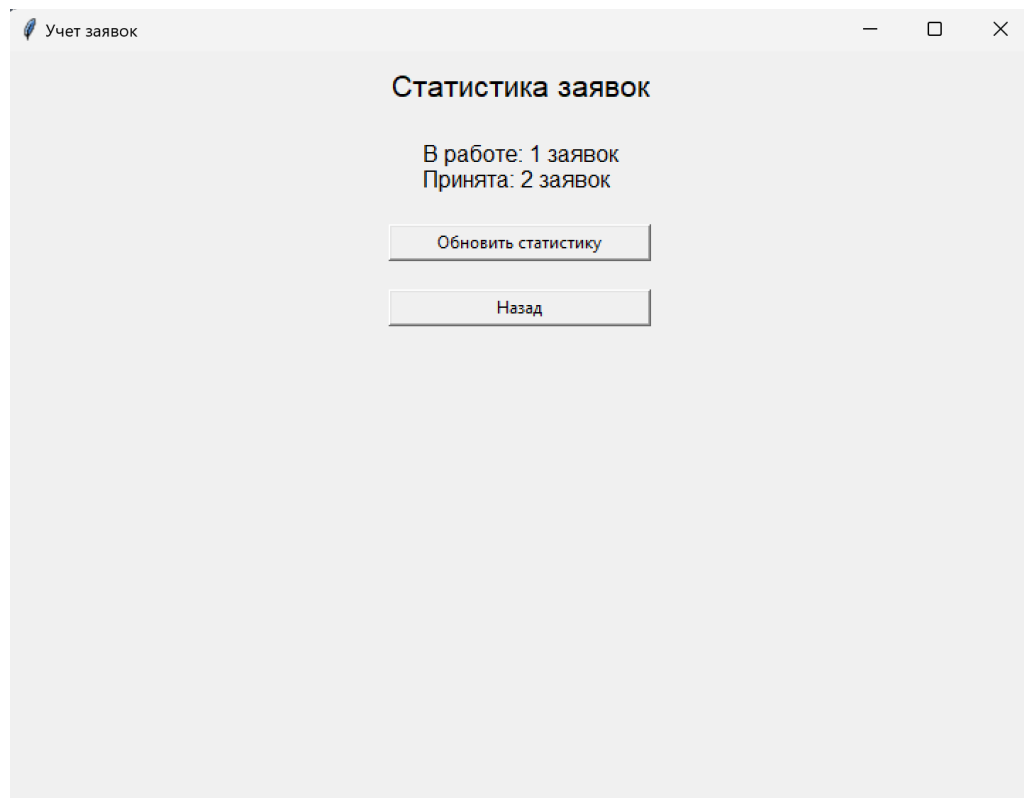


Рис. 14 Страница со статистикой.

По кнопке Обновить статистику происходит обновление и синхронизация статистики.

Код страницы со статистикой представлен в листинге 7.

Листинг 7 – Код страницы со статистикой.

```
class StatisticsPage(Frame):
    def __init__(self, parent, controller):
        super().__init__(parent)
        self.controller = controller

        Label(self, text="Статистика заявок", font=("Arial",
16)).pack(pady=10)
        self.stats_label = Label(self, text="", font=("Arial", 12),
justify="left")
        self.stats_label.pack(pady=10)

        Button(self, text="Обновить статистику",
command=self.load_statistics, width=25).pack(pady=10)
        Button(self, text="Назад", command=lambda:
controller.show_frame("MainPage"), width=25).pack(pady=10)

    def load_statistics(self):
        stats = count_requests_by_status()
        stats_text = "\n".join([f"{status}: {count} заявок" for status,
count in stats])
        self.stats_label.config(text=stats_text)
```

Код создания графического интерфейса для приложения представлен в листинге 8.

Листинг 9 – Код создания графического интерфейса

```
class Application(Tk):
    def __init__(self):
        super().__init__()
        self.title("Учет заявок")
        self.geometry("900x600")
        self.frames = {}

        for F in (LoginPage, MainPage, AddRequestPage,
ViewRequestsPage, StatisticsPage):
            page_name = F.__name__
            frame = F(parent=self, controller=self)
            self.frames[page_name] = frame
            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame("LoginPage")

    def show_frame(self, page_name):
        frame = self.frames[page_name]
        frame.tkraise()

if __name__ == "__main__":
    setup_database()
    app = Application()
    app.mainloop()
```