

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ
РУТ (МИИТ)

Кафедра «Автоматизированные системы управления»

КУРСОВАЯ РАБОТА

По дисциплине «Технологии программирования»

**По теме: «Экспертная система диагностики заболевания
пациента»**

Группа: УВВ-211

Студент: Иванушкин Иван

Александрович

Преподаватель: Давыдовский М.А.

Москва

2021 г.

Задание по курсовой работе для дисциплины «Технологии программирования»

Язык программирования - Python.

Описание проекта разрабатываемой информационной системы выполнено на языке UML в среде UML Designer.

Документация по заданию содержит:

1. Задание по курсовой работе.
2. Диаграммы на языке UML.
3. Краткое описание каждого класса и метода.
4. Руководство пользователя со скриншотами интерфейса программы.
5. Набор тестов и результатов для отладки отдельных методов, модулей и комплексной отладки задания.
6. Тексты программ с комментариями

Диаграммы языка UML:

- Диаграмма вариантов использования (Use Case Diagram)
- Диаграмма классов (Class Diagram)
- Диаграмма деятельности (Activity Diagram)
- Диаграмма состояний (State Machine Diagram)
- Диаграмма последовательности (Sequence Diagram)
- Диаграмма компонентов (Component Diagram)
- Диаграмма развертывания (размещения) (Deployment Diagram)

Общие требования по выполнению задания:

- Каждый пользователь информационной системы должен регистрироваться в системе сам или это делает администратор
- При входе в систему каждый пользователь вводит свой логин и пароль
- Данные в системе хранятся в файлах
- При запуске системы все данные из файлов считываются в оперативную память и хранятся в форме таблиц.
- По завершению работы с системой данные из таблиц записываются обратно в файлы. Файлы при этом не обновляются, а создаются заново на основании данных из таблиц.

Предметная область – медицинское учреждение. Диагностика заболеваний.

Варианты использования для актеров:

Администратор	Составление списков экспертов, пациентов, врачей.
Эксперт	Подготовка перечня вопросов для пациента, на основе которых определяется некоторое заболевание. Ответом на вопрос может быть да, нет. Вопросы разрабатываются группой экспертов. Каждый эксперт разрабатывает список вопросов и ответов отдельно, а затем списки вопросов объединяются в общий список.
Врач	Выбор вопросов из списка, на которые должен ответить пациент. Просмотр результатов ответов и диагноза
Пациент	Отвечает на вопросы. Выбирает варианты ответов.
Система	Ставит диагноз пациенту на основе сравнения ответов пациента с ответом экспертов.

Поток событий системы

Основной поток событий	
Действия исполнителя	Отклик системы
1. Авторизуется.	2. Предоставляет окно входа.
1.1. Регистрируется.	2.1 Заполняет БД.
3. Редактирует тест.	4. Обновляет таблицу БД.
5. Проходит тест.	6. Выносит диагноз.
7. Смотрит результат.	8. Отображает результат.

Авторизация	
1. Ввод логина и пароля.	2. Проверка связки логин-пароль.
	3. Идентификация уровня доступа пользователя.
	4. Отображение меню действий на дисплей.

Регистрация нового пользователя	
	1. Отображение списка типов пользователей.
2. Выбор типа нового пользователя.	
2.1. Выбор эксперта.	
2.2. Выбор врача.	
2.3 Выбор пациента.	
	3. Обработка нажатия.
	4. Отображение новой анкеты.
5. Заполнение анкеты пользователя.	6. Сохранение данных в файл.

Добавление вопросов	
	1. Отображение кнопки добавления.

3. Нажатие кнопки.	4. Обработка нажатия.
	5. Вывод формы ввода вопроса.
6. Заполнение формы вопроса.	
7. Расстановка баллов ответов.	8. Сохранение данных в файл.

Редактирование вопроса	
	1. Отображение всех вопросов.
2. Нажатие кнопки редактирования.	3. Обработка нажатия.
	4. Вывод меню редактирования.
5. Редактирование.	6. Обновление БД.

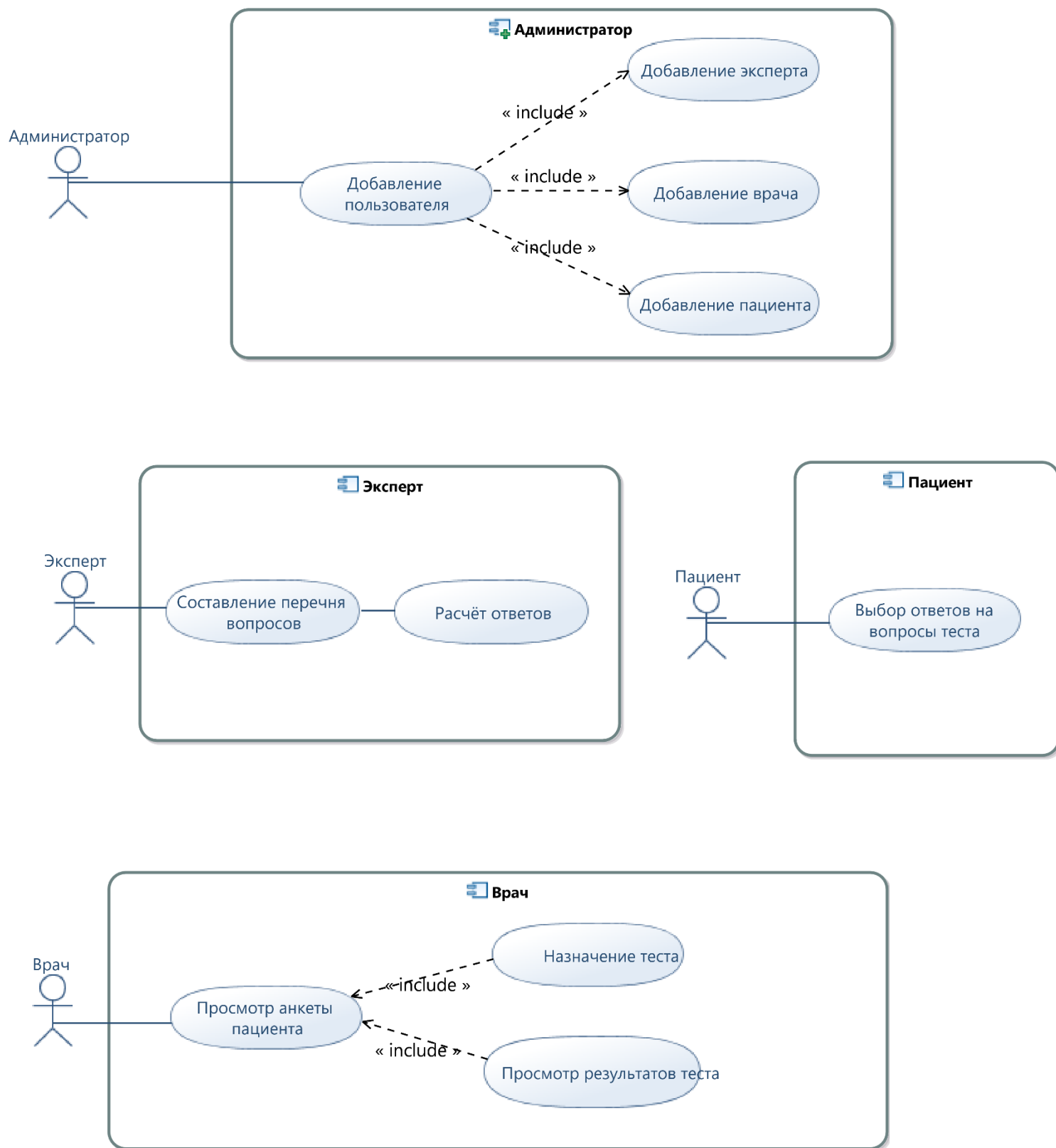
Удаление вопроса	
	1. Отображение всех вопросов.
2. Нажатие кнопки удаления.	3. Обработка нажатия.
	4. Вывод меню подтверждения.
5. Удаление.	6. Обновление БД.

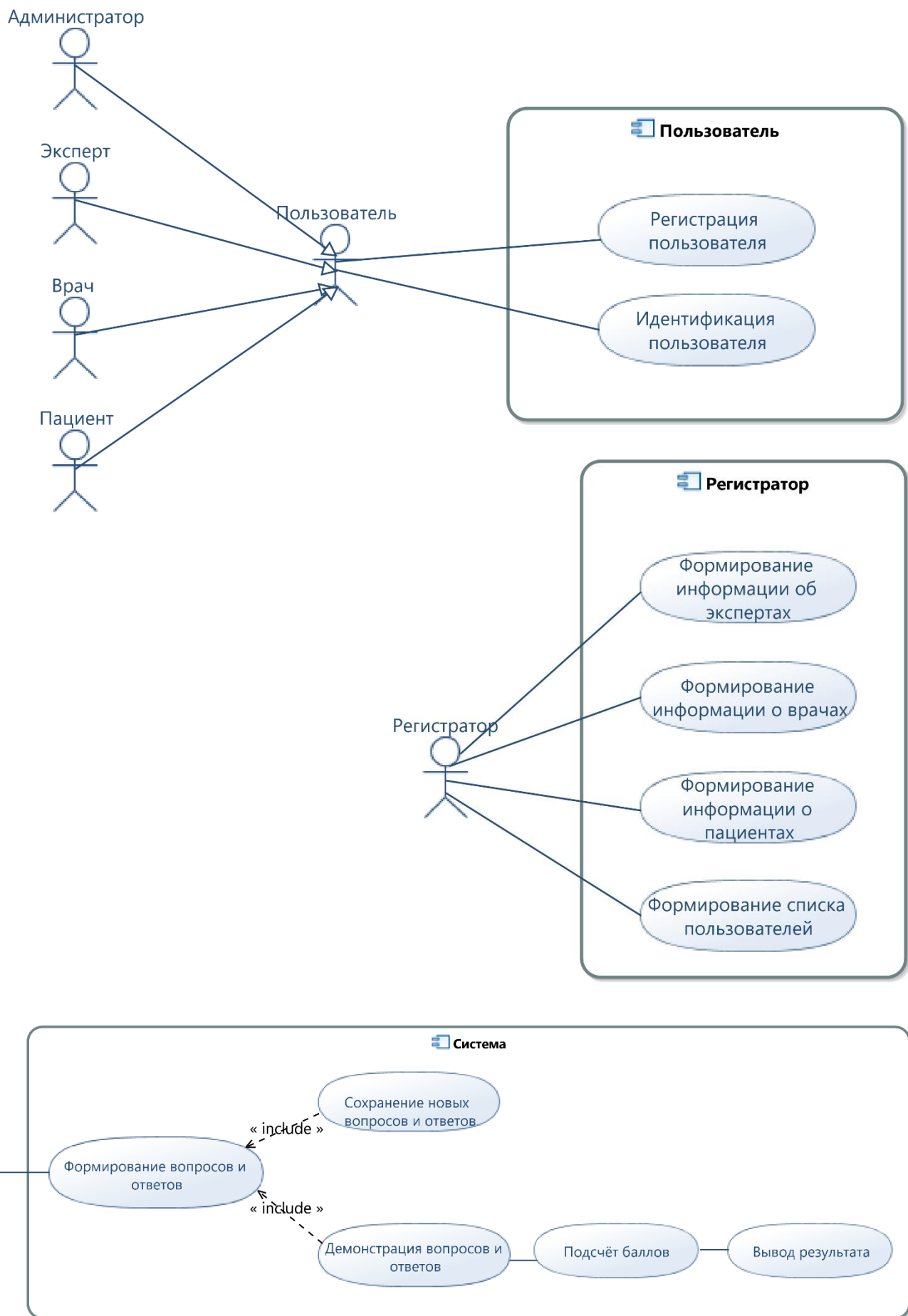
Прохождение теста	
	1. Вывод вопроса и вариантов ответа на дисплей.
2. Выбор варианта ответа.	3. Сохранение варианта ответа.
	4. Подсчёт баллов.
5. Переход к следующему вопросу.	6 Вывод результата на дисплей.
	7. Сохранение данных в файл.

Под-поток «Ошибка авторизации»	
1. Ввод логина и пароля.	2. Проверка связки логин-пароль.
	3. Вывод на экран сообщения об ошибке.

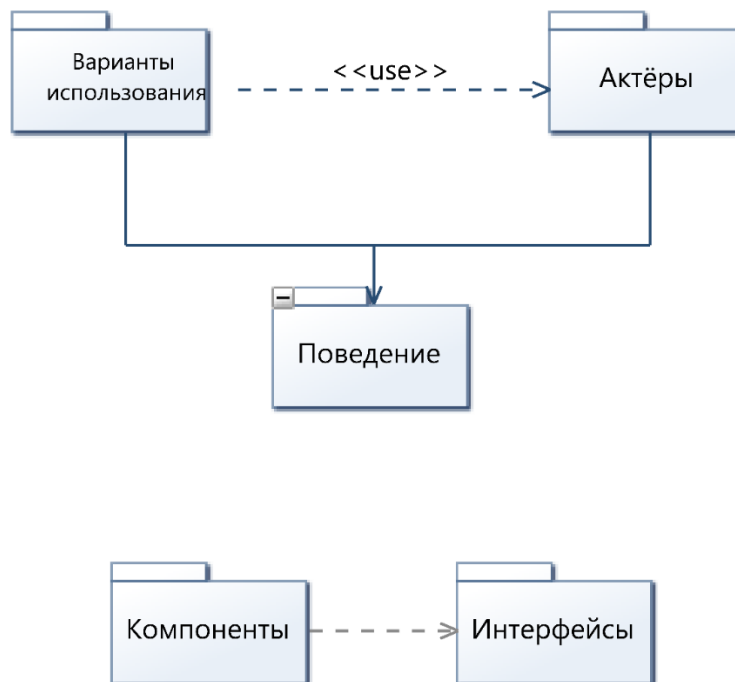
Диаграммы на языке UML

1. Диаграмма вариантов использования (Use Case Diagram)



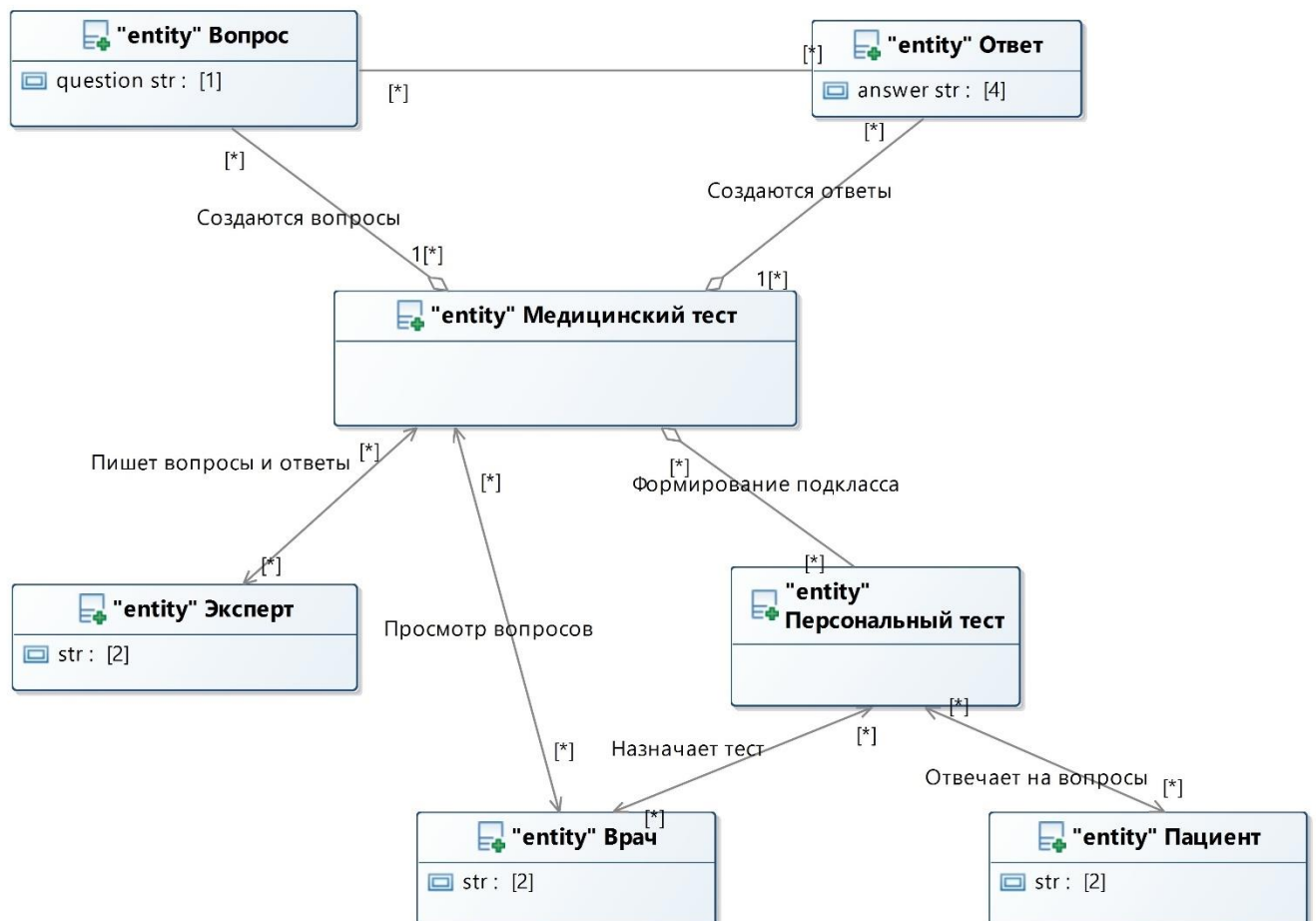


1.1 Диаграмма пакетов (Package Hierarchy Diagram)

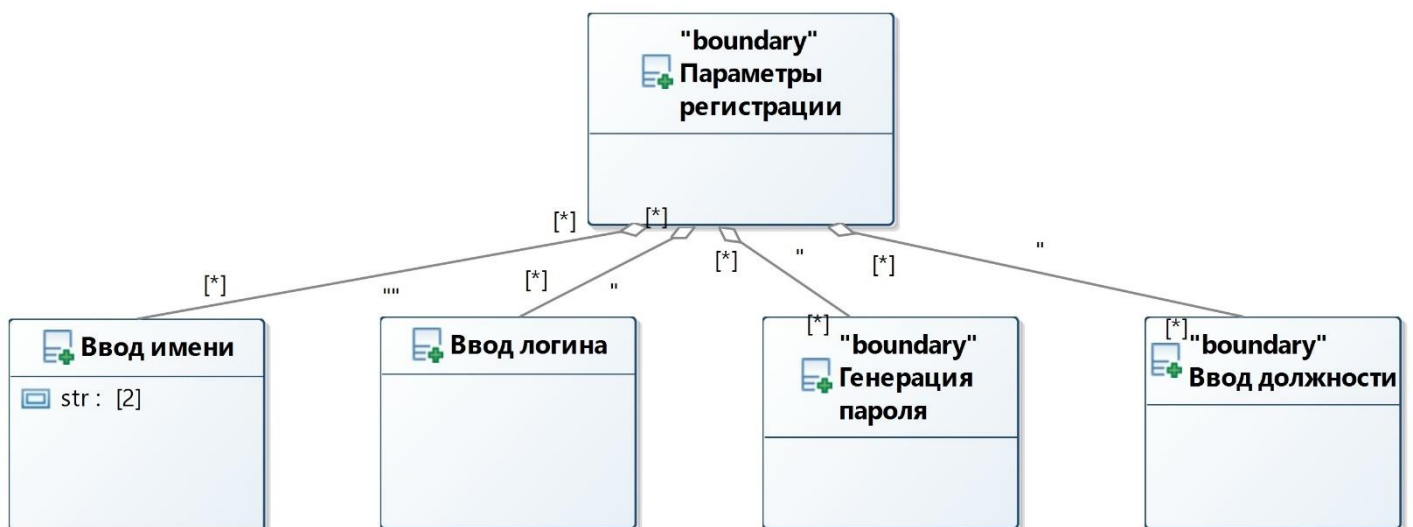
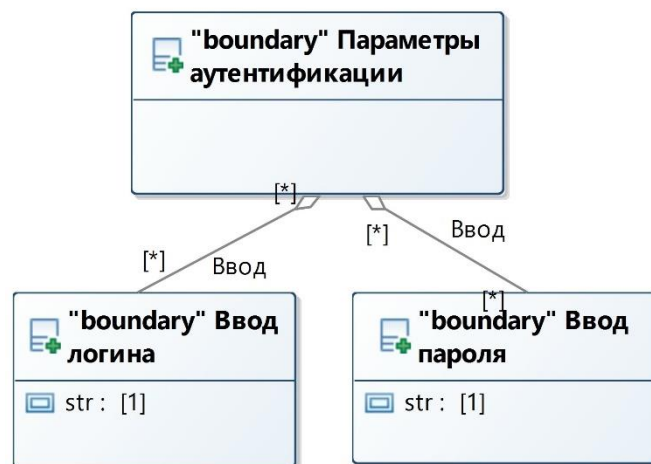
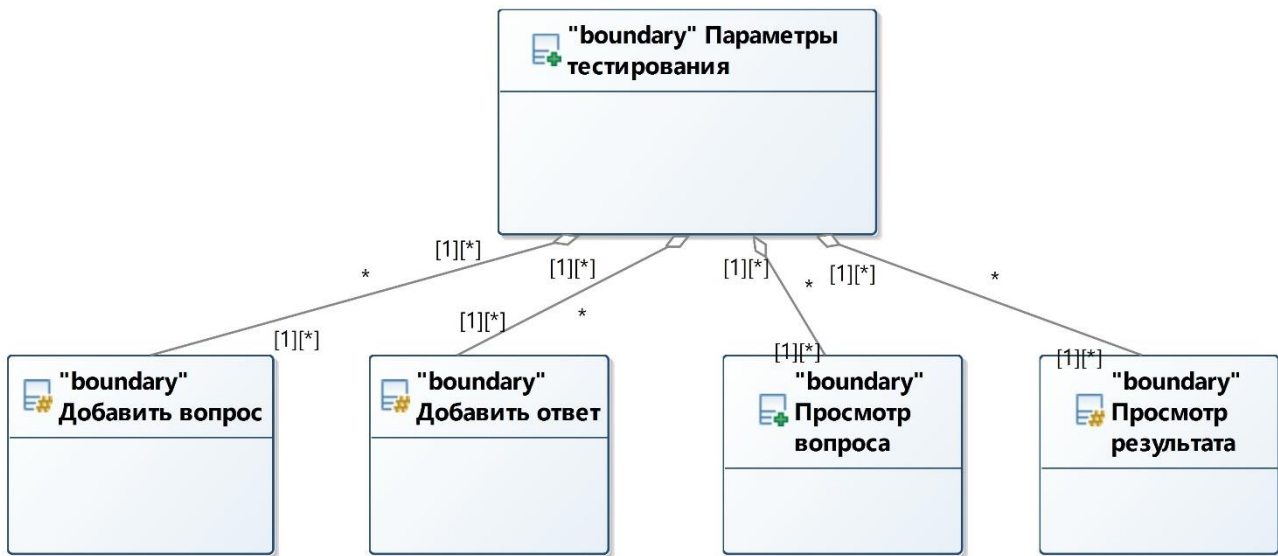


2. Диаграмма классов (Class Diagram)

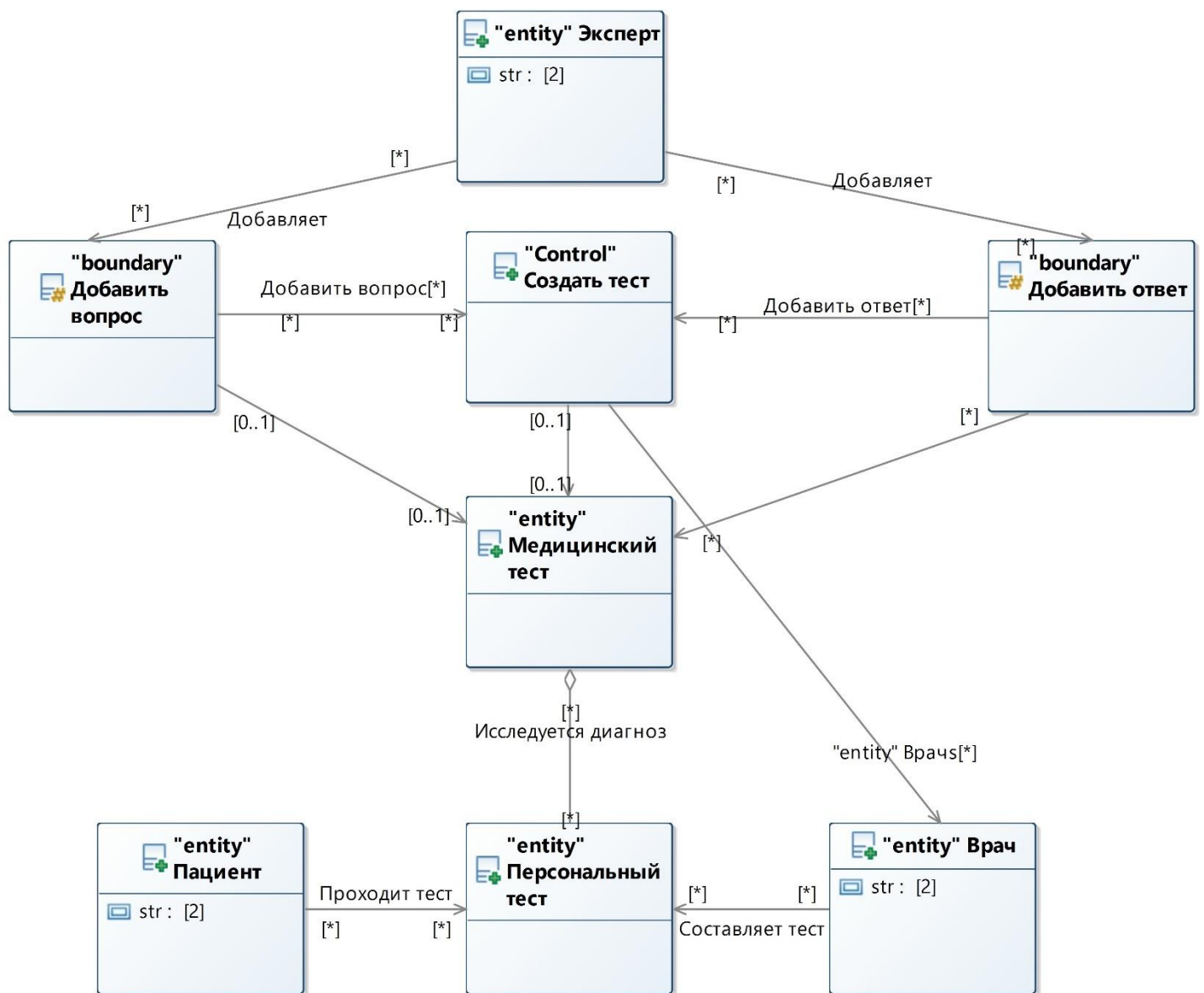
Entity class - класс, используемый для моделирования объектов реального мира.



Boundary class - класс, используемый в качестве посредника между актерами, взаимодействующими с системой, и самой системой.

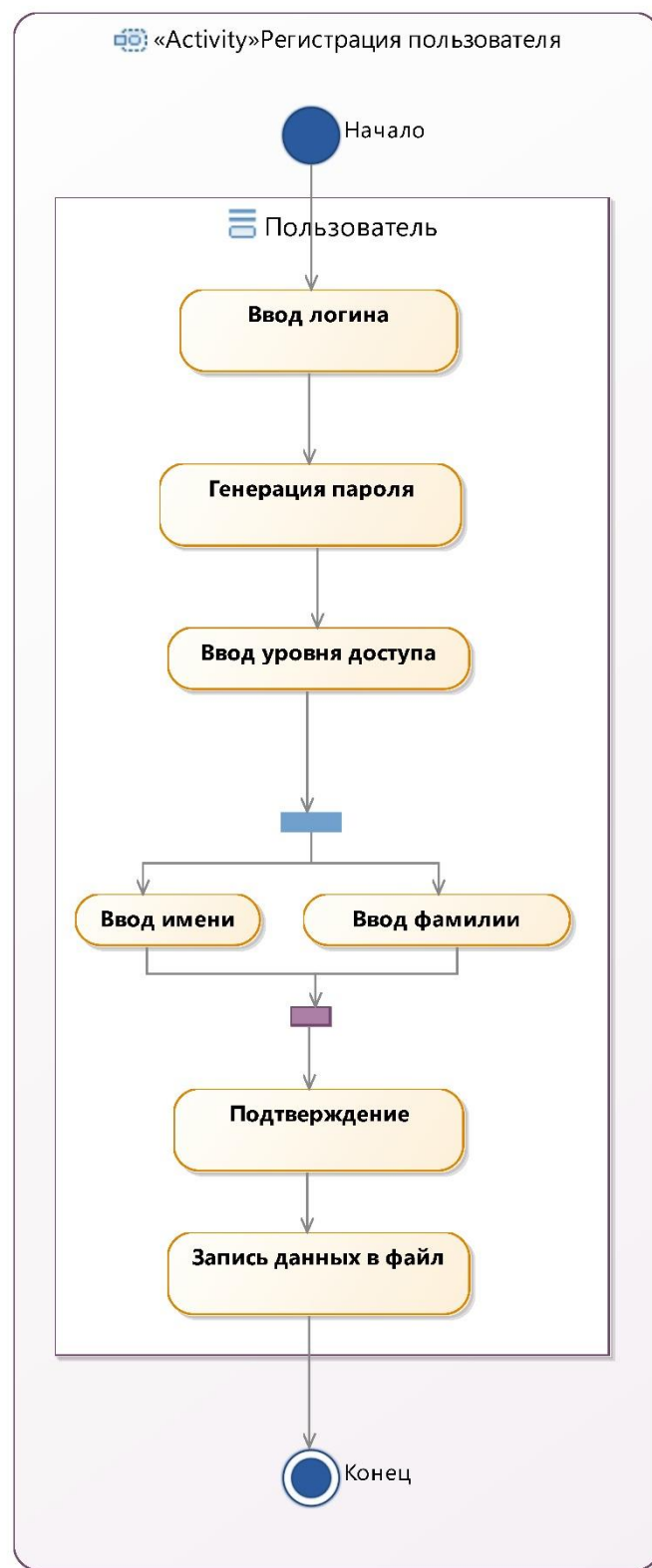


Control - класс, реализующий поведение варианта использования

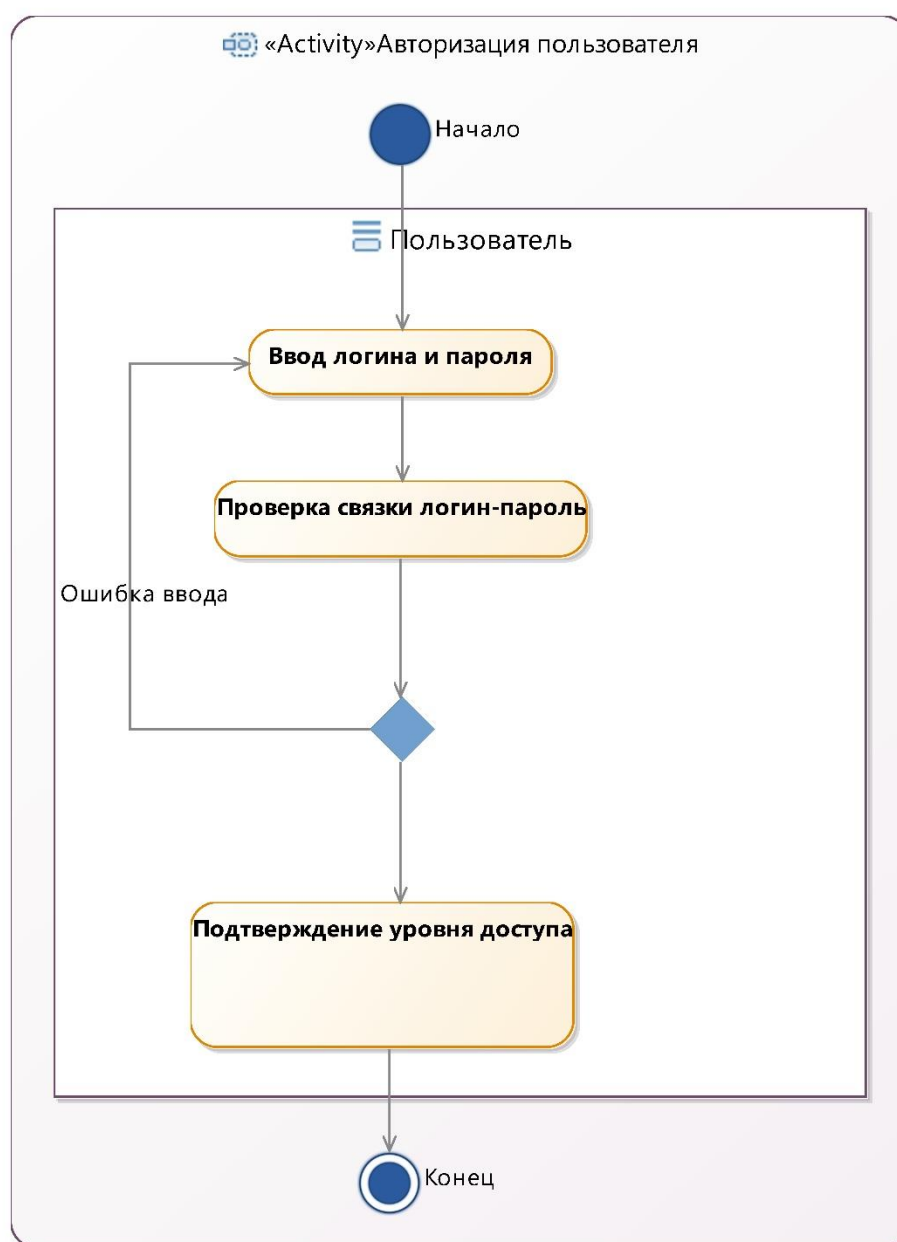


3. Диаграмма деятельности (Activity Diagram)

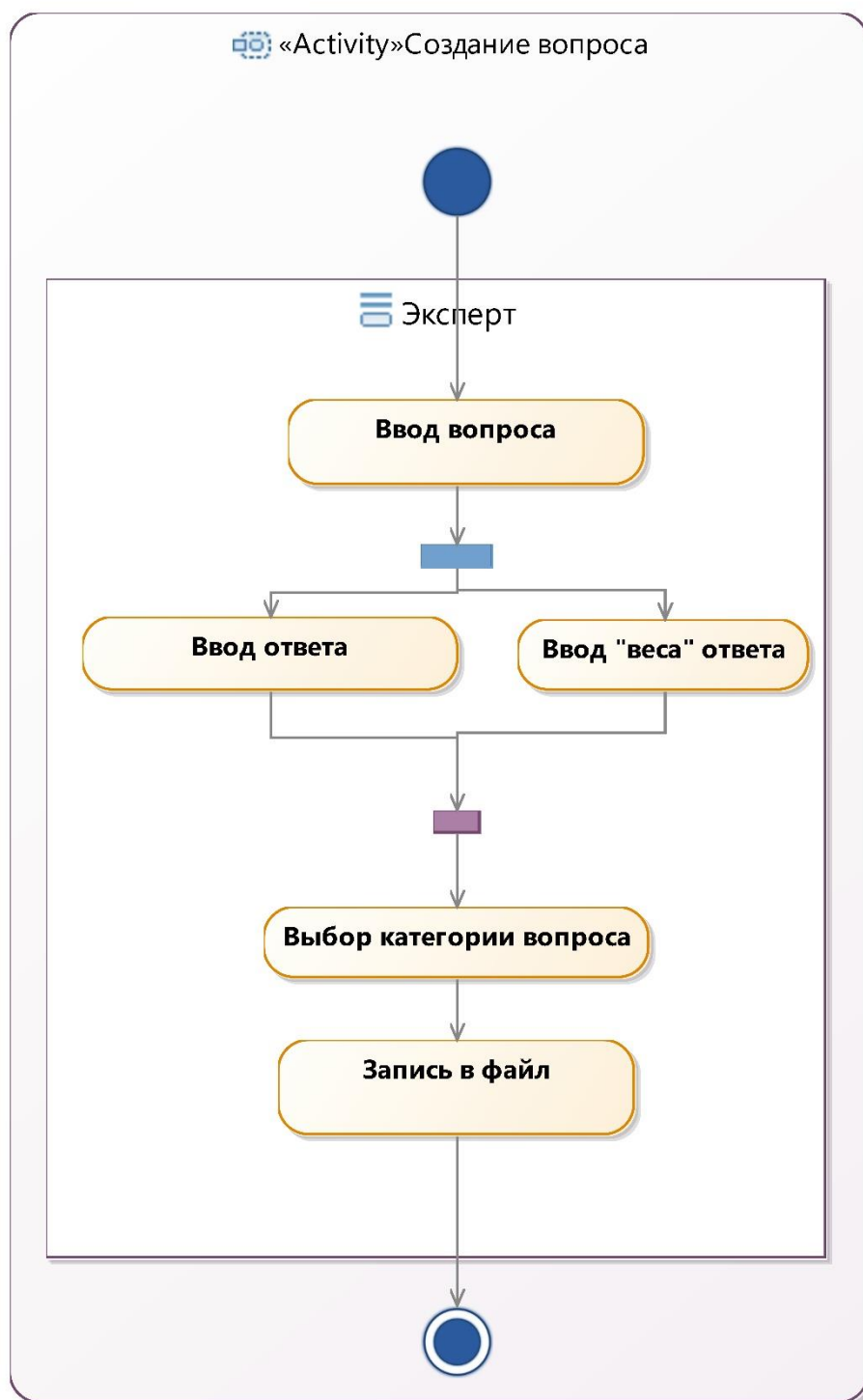
3.1 Регистрация пользователя.



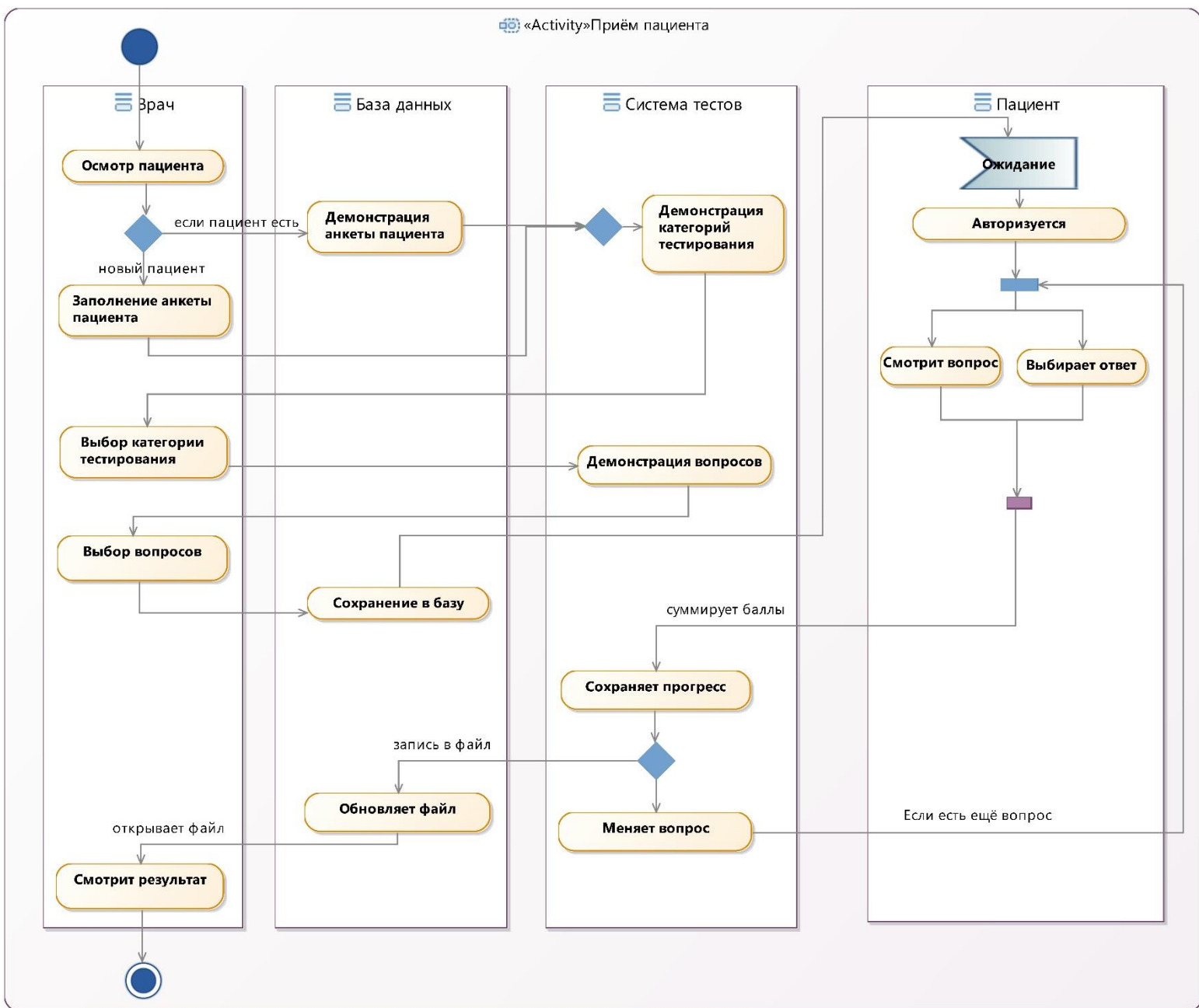
3.2 Авторизация пользователя.



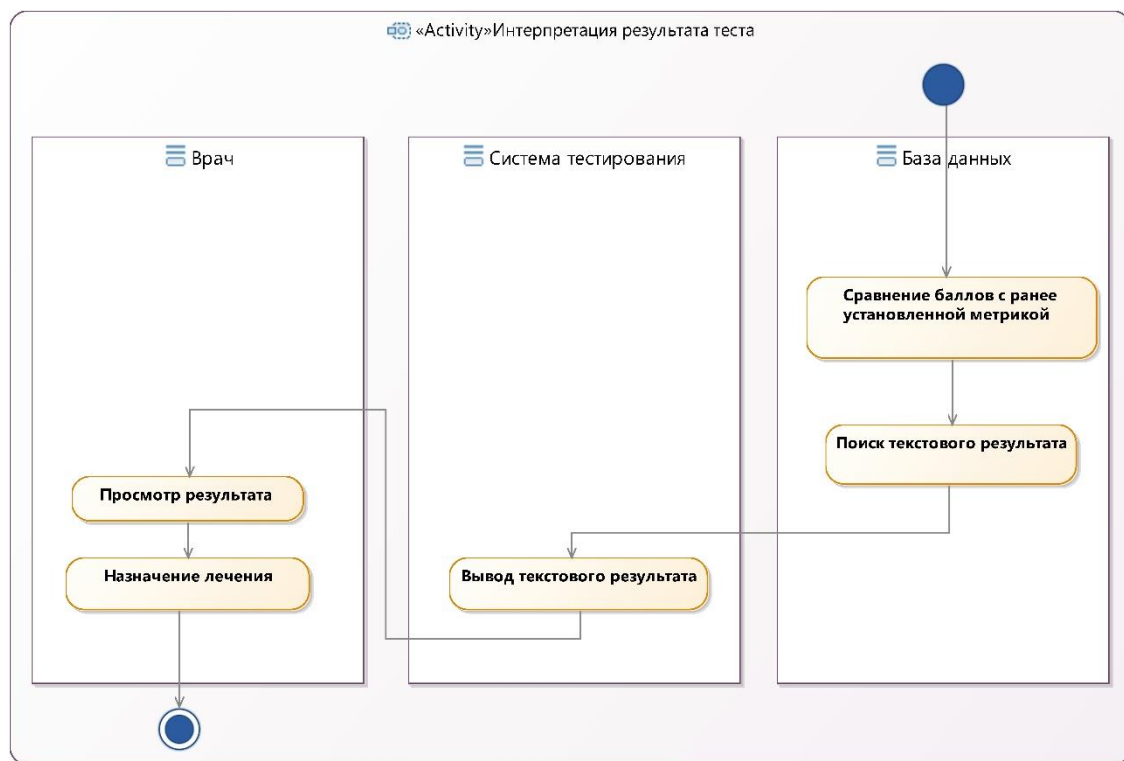
3.2 Создание вопроса.



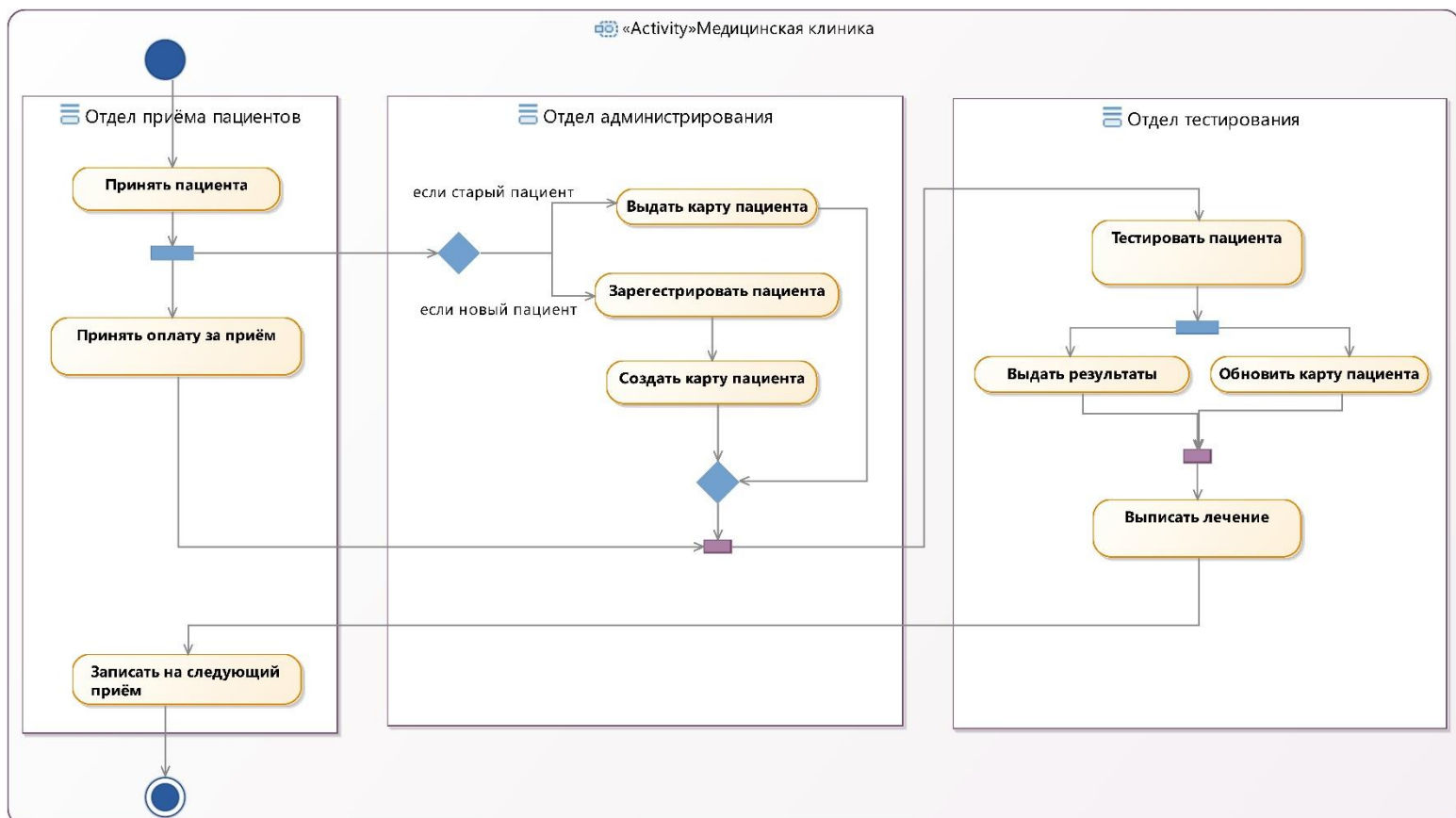
3.3 Приём пациента.



3.4 Интерпретация результата теста.

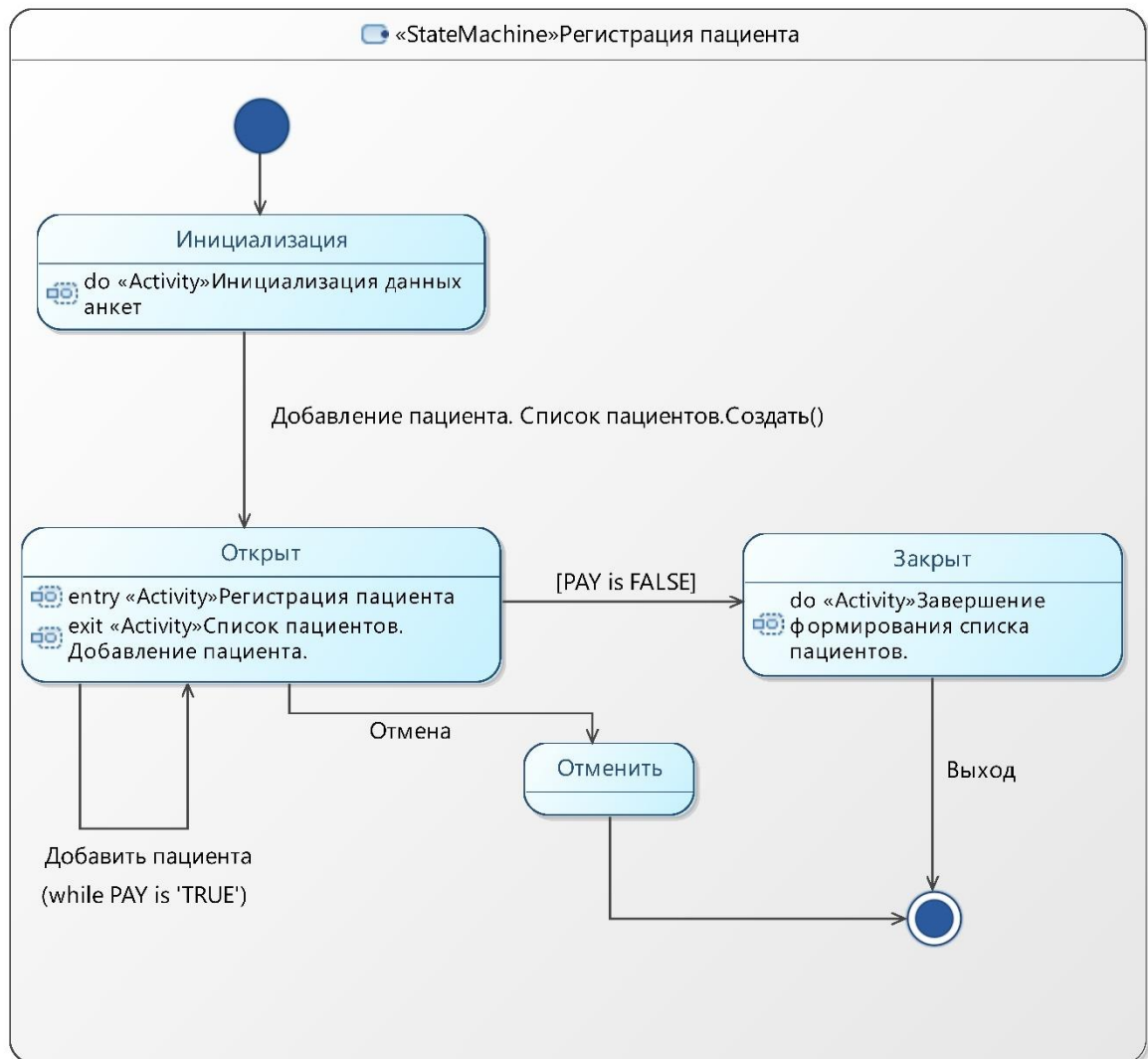


3.5 Бизнес-процесс медицинской клиники.

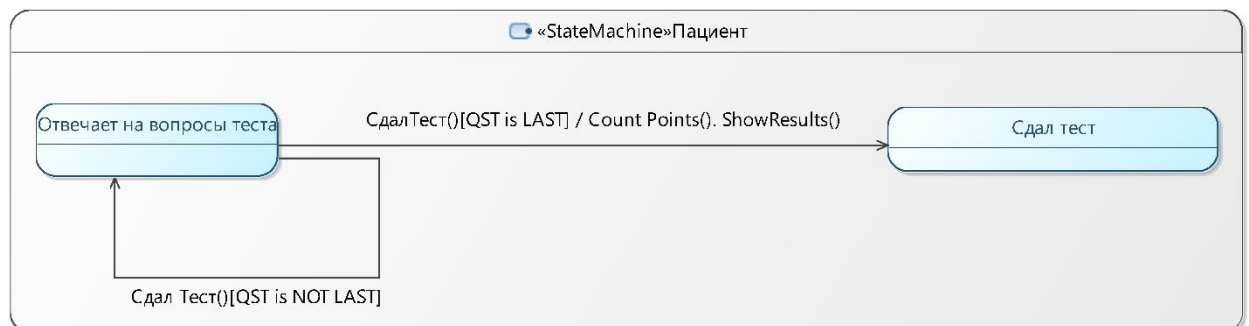


4. Диаграмма состояний (State Machine Diagram)

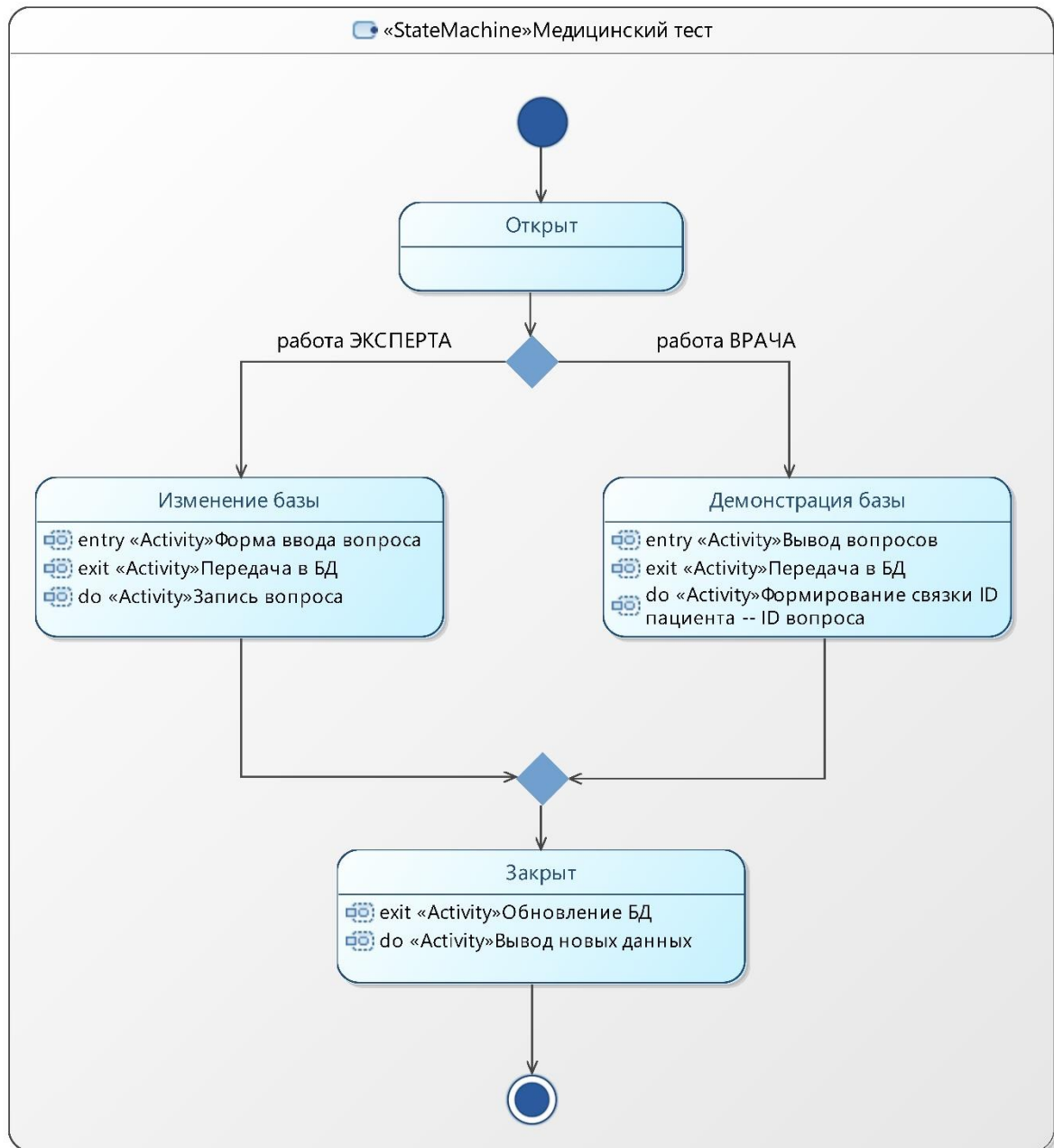
4.1. Регистрация пациентов для прохождения теста после оплаты.



4.2. Пациент проходит тест.

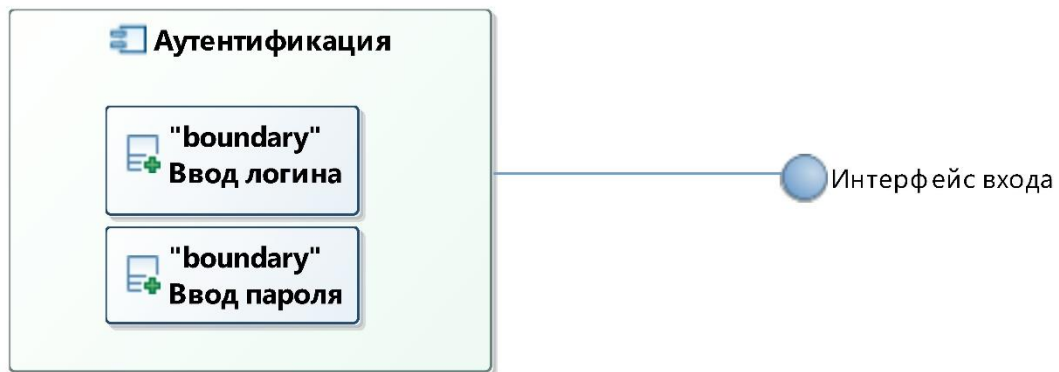


4.3. **Медицинский тест** редактирую два актёра: **эксперт**, который добавляет новые вопросы и **врач**, который назначает тест из вопросов для анкеты пациента.

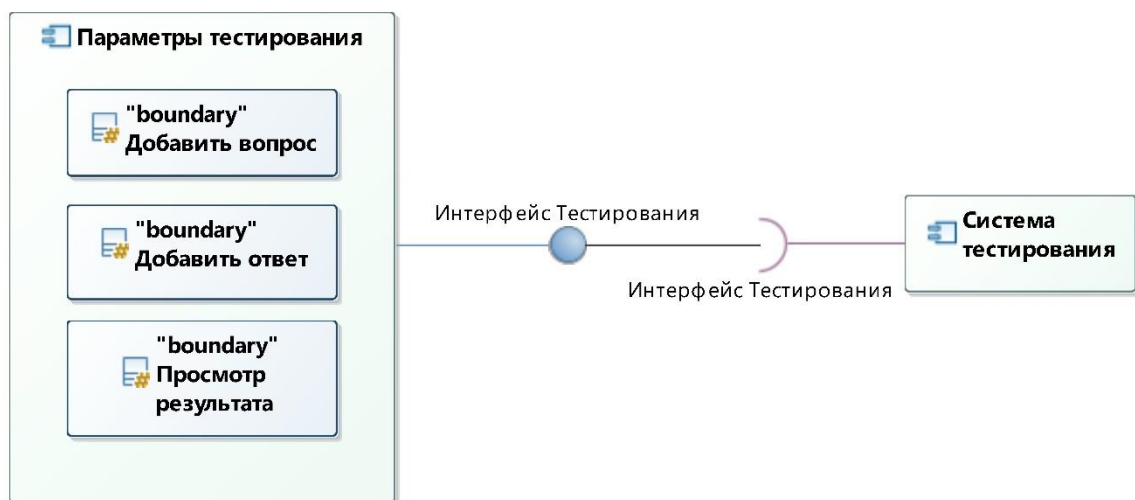


5. Диаграмма компонентов (Component Diagram)

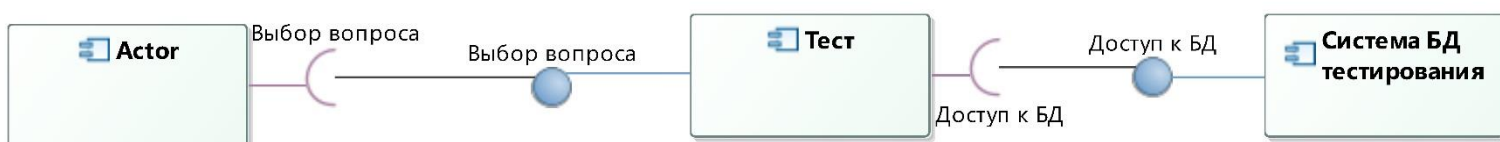
5.1 Аутентификация актёра.



5.2 Добавление вопросов в тест от эксперта.

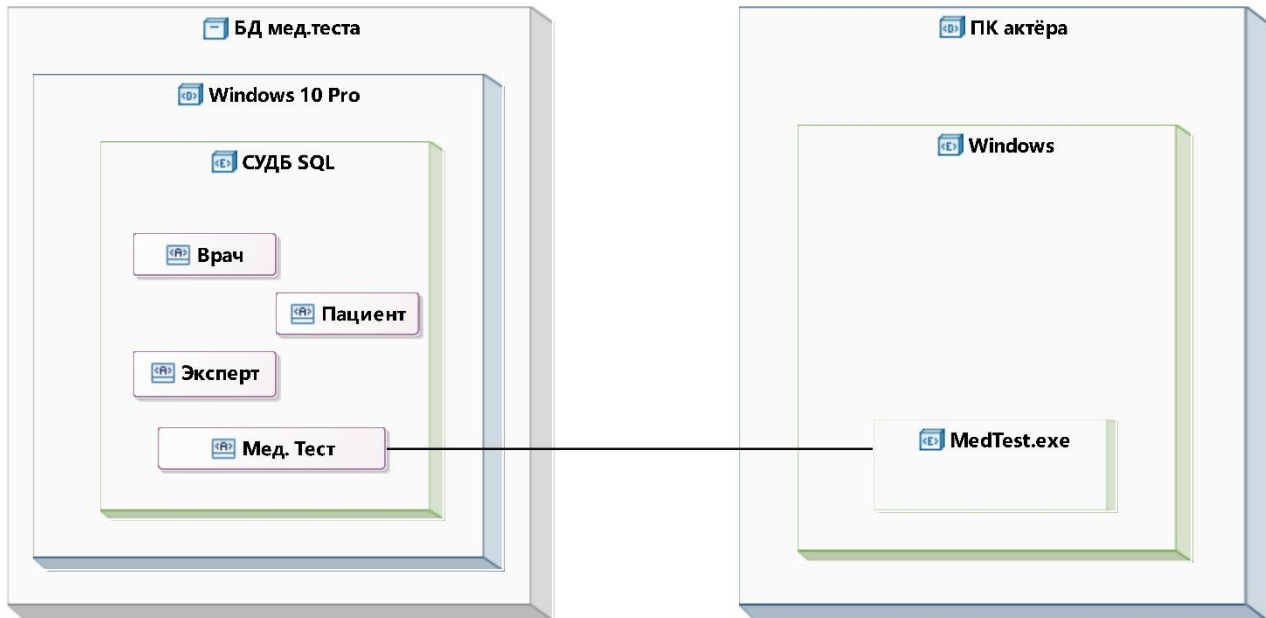


5.3 Прохождение пациента теста // Выбор теста врачом

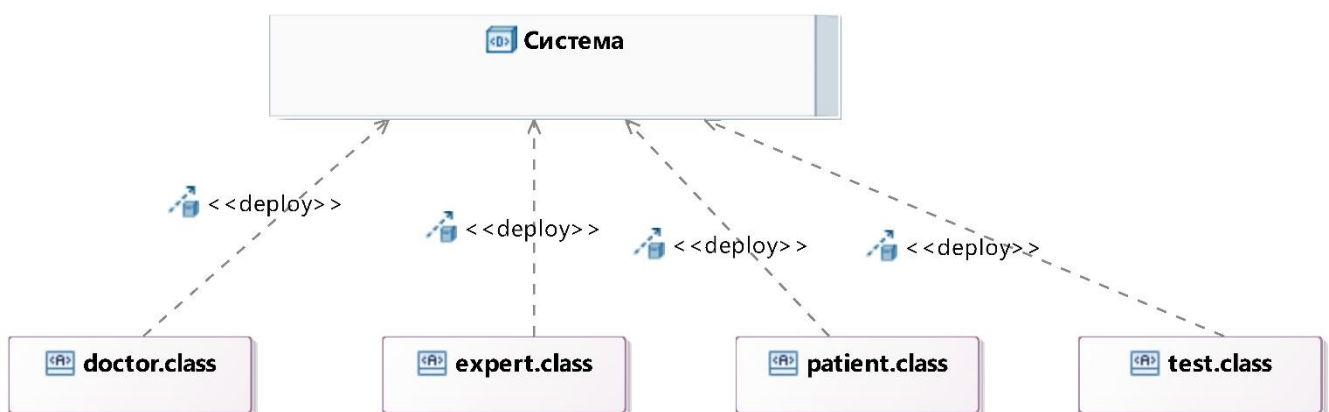


6. Диаграмма развертывания (размещения) (Deployment Diagram)

На диаграмме представлено физическое размещение системы на ПК и обращение актёра к медицинскому тесту.

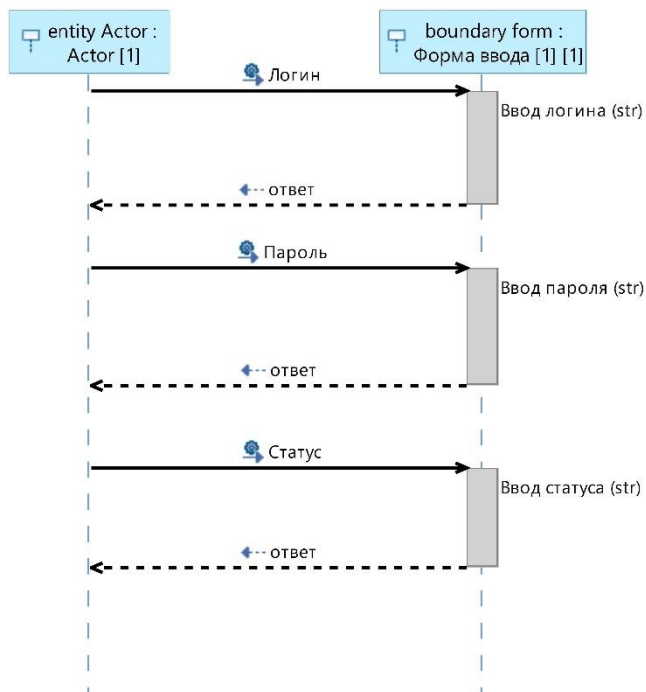


На диаграмме представлено расположение артефактов внутри узла "система" с помощью отношения зависимости "deploy".

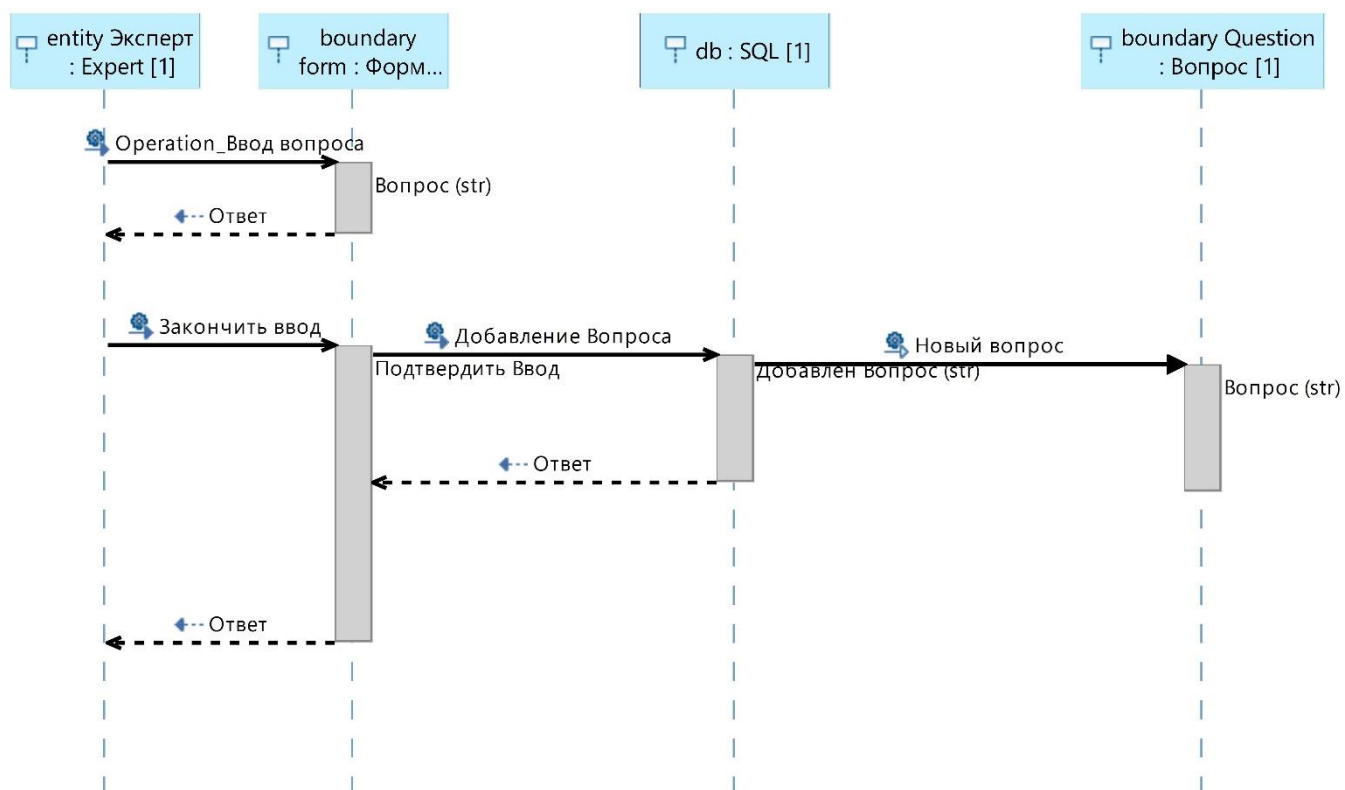


7. Диаграмма последовательности (Sequence Diagram)

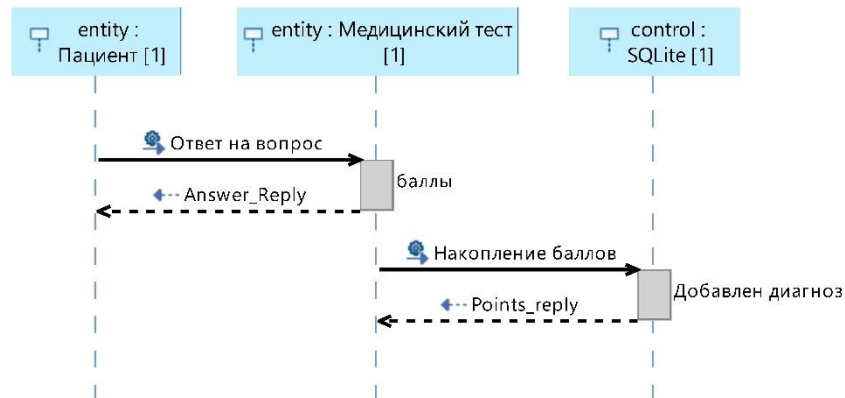
7.1 Аутентификация пользователя.



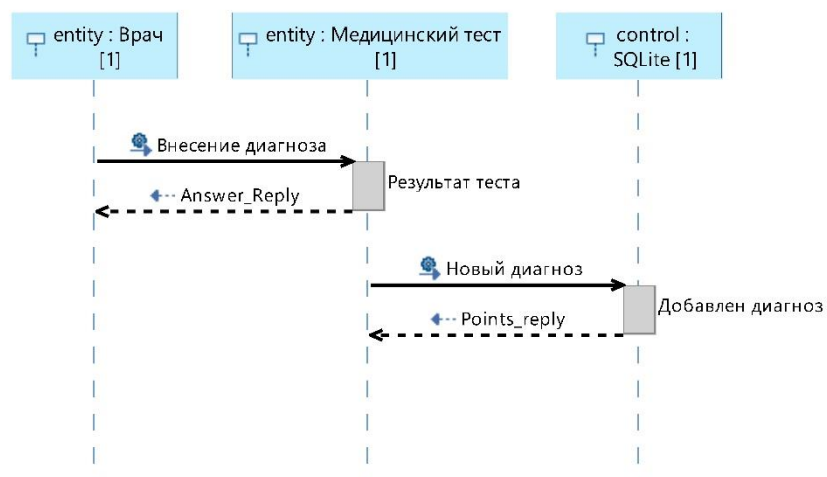
7.2 Добавление вопроса.



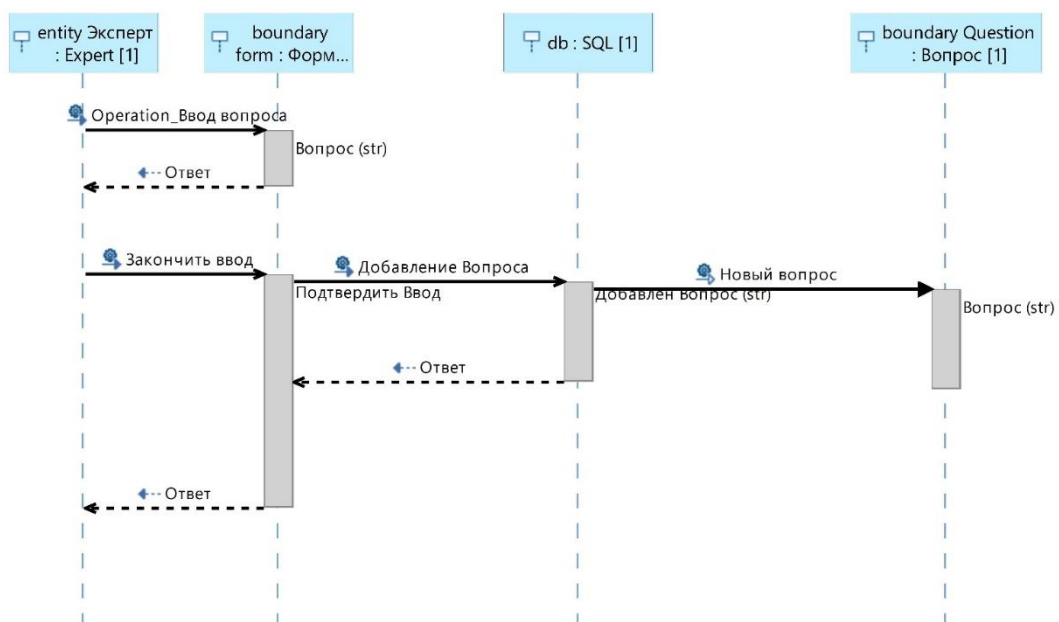
7.3 Прохождение теста.



7.4 Подтверждение диагноза.



7.5 Редактирование вопроса.



Руководство пользователя

1. Введение.

1.1. Область применения

Требования настоящего документа применяются при:

- Эксплуатации в медицинских учреждениях;
- Опросно-тестовых испытаний;

1.2. Краткое описание возможностей

Система предназначена для оптимизации стратегических решений конечными бизнес-пользователями на основе всех данных о проявлении симптомов заболеваний. Так же система предназначена для осуществления контроля над вынесением диагноза на основе анализа и рекомендаций системы.

1.3. Уровень подготовки пользователя

Пользователь системы должен иметь опыт работы с ОС Windows (XP/7/8/10).

1.4. Перечень эксплуатационной документации, с которой необходимо ознакомиться пользователю

- Руководство пользователя ОС Windows.

2. Назначения и условия применения.

1. Система предназначена для автоматизированного вынесения предварительного диагноза на основе результатов тестирования с участием третьего лица – врача.
2. Работа с программой возможна всегда, когда есть необходимость в медицинских учреждениях в целях экономии времени врача и пациента составить предварительный тест на риск развития заболевания.

3. Подготовка к работе.

3.1 Состав и содержание дистрибутивного носителя

1. ОС Windows (XP/7/8/10).

3.2 Порядок загрузки данных и программ

1. Вставить USB накопитель в компьютер.
2. Скопировать файлы папки MedTest.
3. Запустить программу main.py.
4. Если хотите посмотреть файл test.db, то скачайте DB Browser (SQLite) с официального сайта, перенесите файл в окно программы «Структура БД» и во вкладке «Данные» Вы увидите базу данных программы.

3.3 Порядок проверки работоспособности

1. Запустить программу.
2. Пройти регистрацию (Эксперт/Врач/Пациент).
3. Убедиться, что открылось окно (Эксперт/Пациент/Пациент).

4. Описание операций. (Эксперт)

4.1. Выполняемые функции и задачи

Функции: Добавление, редактирование или удаление вопросов к тесту.

Задачи:

- визуализация таблицы вопросов (при выполнении данной задачи эксперту будут доступны 3 окна: номер вопроса, его описание, баллы)

4.2. Описание операций процесса обработки данных, необходимых для выполнения задач

Задача: «Визуализация таблицы вопросов»

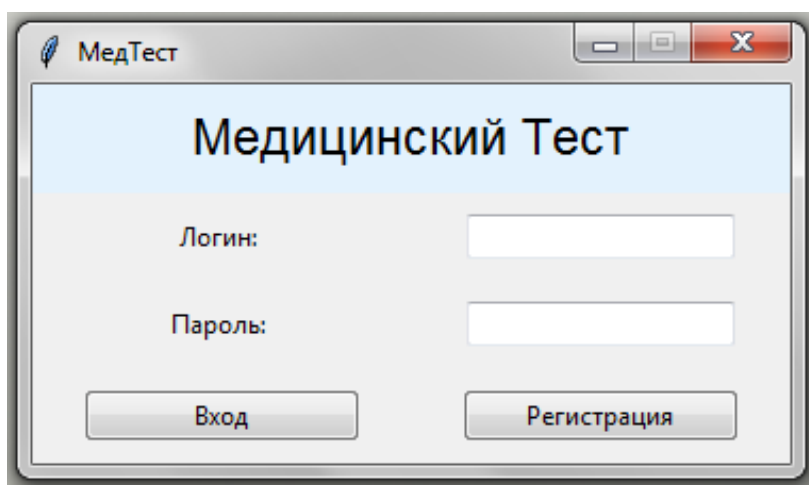
Операция 1: Регистрация в программе/вход в программу.

Условия, при соблюдении которых возможно выполнение операции:

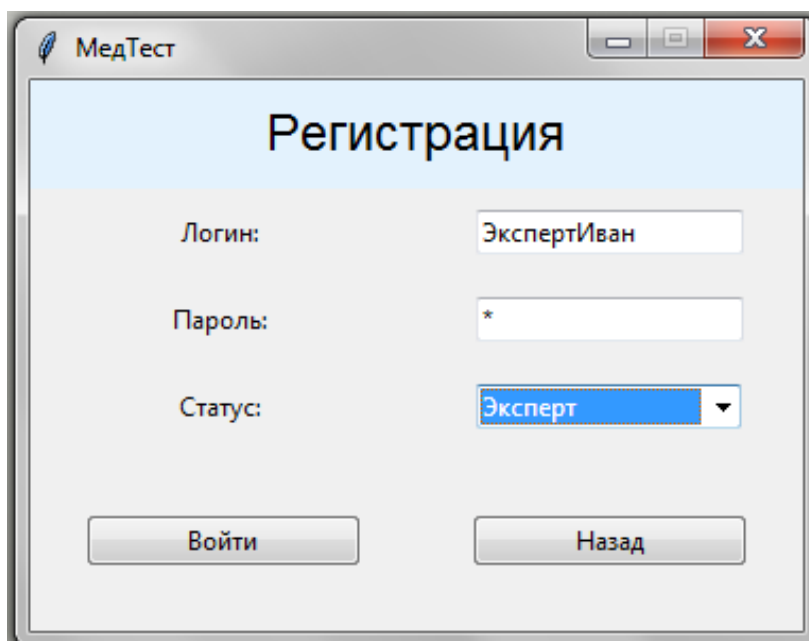
1. Компьютер пользователя подключен к сети.
2. Программа работает корректно.

Основные действия в требуемой последовательности:

1. Введите логин/пароль в окне входа в систему (для зарегистрированных пользователей) или нажмите кнопку «Регистрация», если Вы являетесь новым пользователем.



2. В окне регистрации введите Ваш логин, выберите должность эксперта и введите пароль. Далее нажмите Войти.



Заключительные действия:

Не требуются.

Ресурсы, расходуемые на операцию:

10-20 секунд.

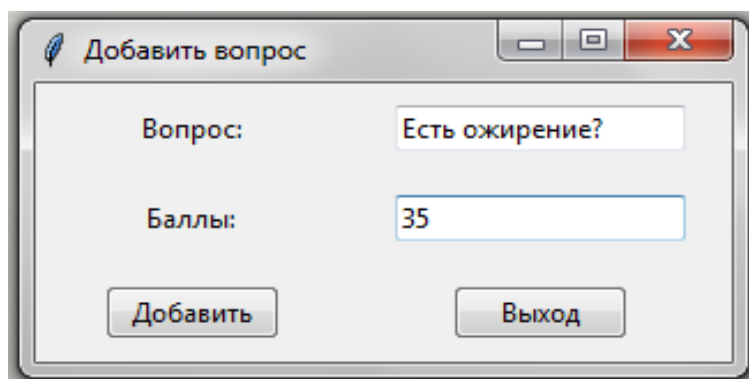
Операция 2: Добавление/удаление/редактирование вопроса.

Условия, при соблюдении которых возможно выполнение операции:

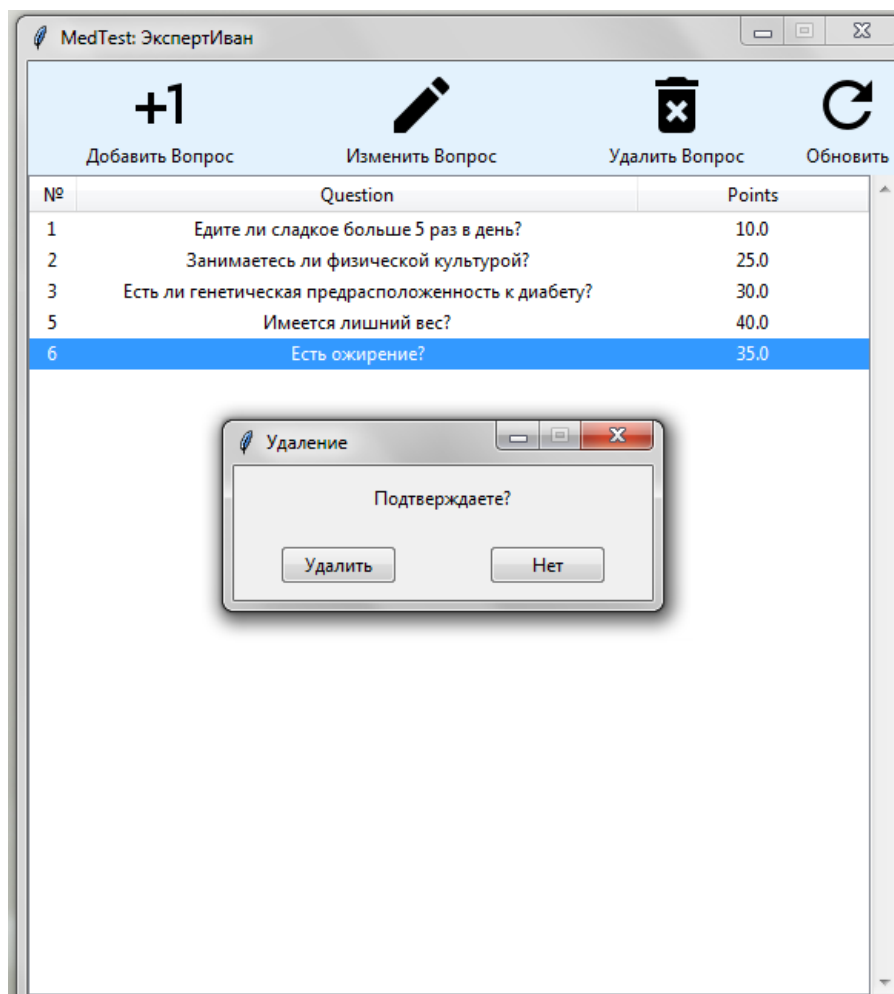
1. Успешно пройденная регистрация в программе, наличие логина/пароля для входа в систему.

Основные действия в требуемой последовательности:

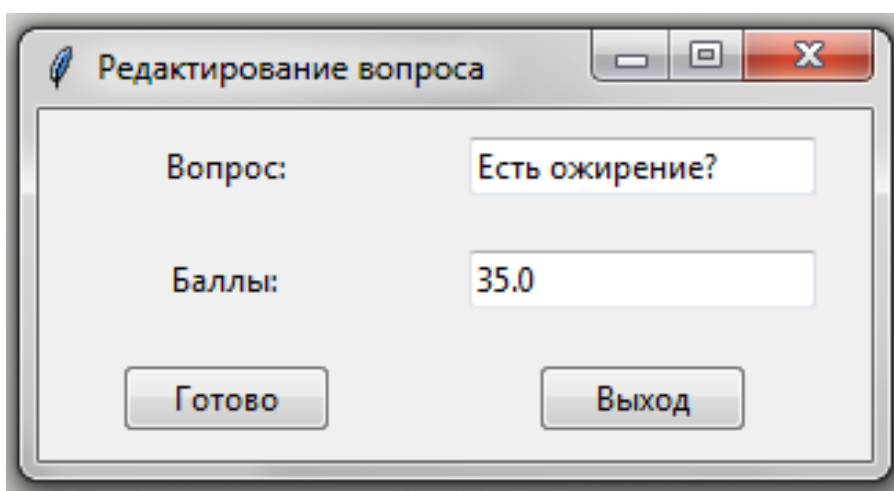
1. Войдите в систему с аккаунта, имеющего права эксперта.
2. Если Вы хотите добавить новый вопрос, нажмите кнопку “Добавить Вопрос”, введите свой вопрос и баллы за него.



3. Если Вы хотите удалить вопрос, выберите его из списка всех вопросов и нажмите кнопку «Удалить вопрос».



4. Если Вы хотите редактировать вопрос, выберите его из списка всех вопросов и нажмите кнопку «Изменить вопрос».



Заключительные действия:

Не требуются.

Ресурсы, расходуемые на операцию:

20-40 секунд.

5. Описание операций. (Пациент)

5.1. Выполняемые функции и задачи

Функции: Ответ на вопросы теста, просмотр результата теста.

Задачи:

- визуализация таблицы вопросов (при выполнении данной задачи, пациенту будут доступны окна: номер вопроса, описание и баллы за него)

5.2. Описание операций процесса обработки данных, необходимых для выполнения задач

Операция 1: Регистрация в программе/вход в программу.

Условия, при соблюдении которых возможно выполнение операции:

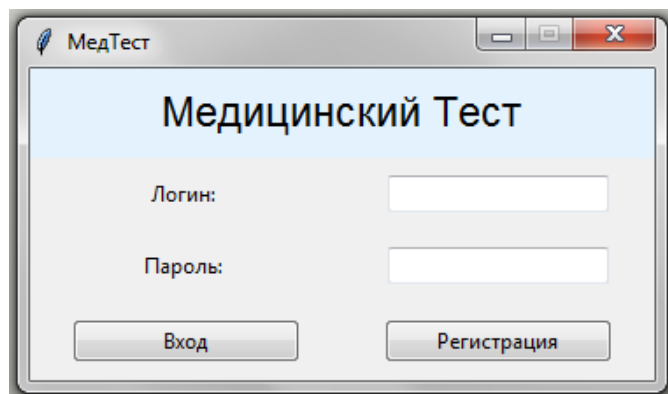
1. Компьютер пользователя подключен к сети.
2. Программа работает корректно.

Подготовительные действия:

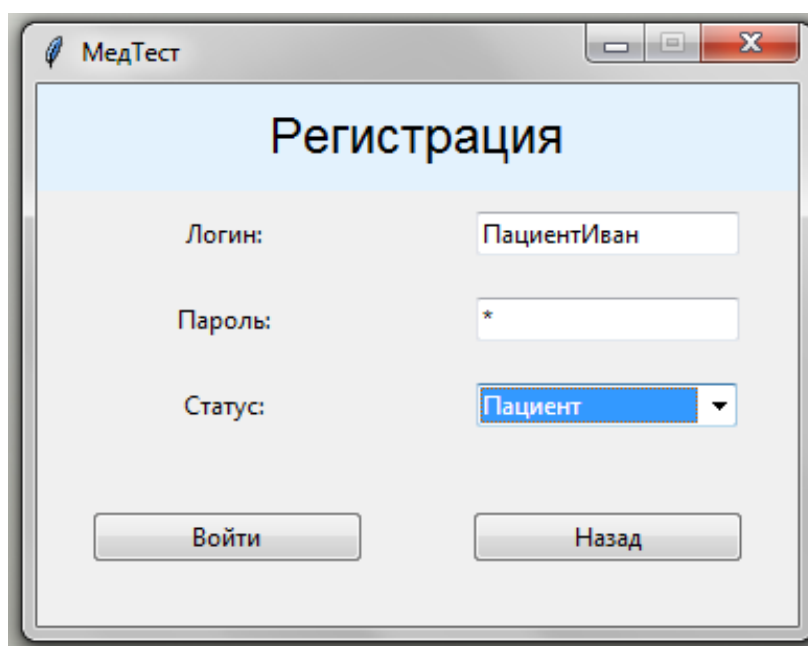
На компьютере пользователя необходимо выполнить дополнительные настройки, приведенные в п. 3.2 настоящего документа.

Основные действия в требуемой последовательности:

1. Ввести логин/пароль в окне входа в систему (для зарегистрированных пользователей) или нажать кнопку «зарегистрироваться», если вы являетесь новым пользователем.



2. В окне регистрации введите Ваш логин и пароль и выберите должность пациента.



МедТест

Регистрация

Логин: ПациентИван

Пароль: *

Статус: Пациент ▼

Войти Назад

Заключительные действия:

Не требуются.

Ресурсы, расходуемые на операцию:

10-20 секунд.

Операция 2: Прохождение теста.

Условия, при соблюдении которых возможно выполнение операции:

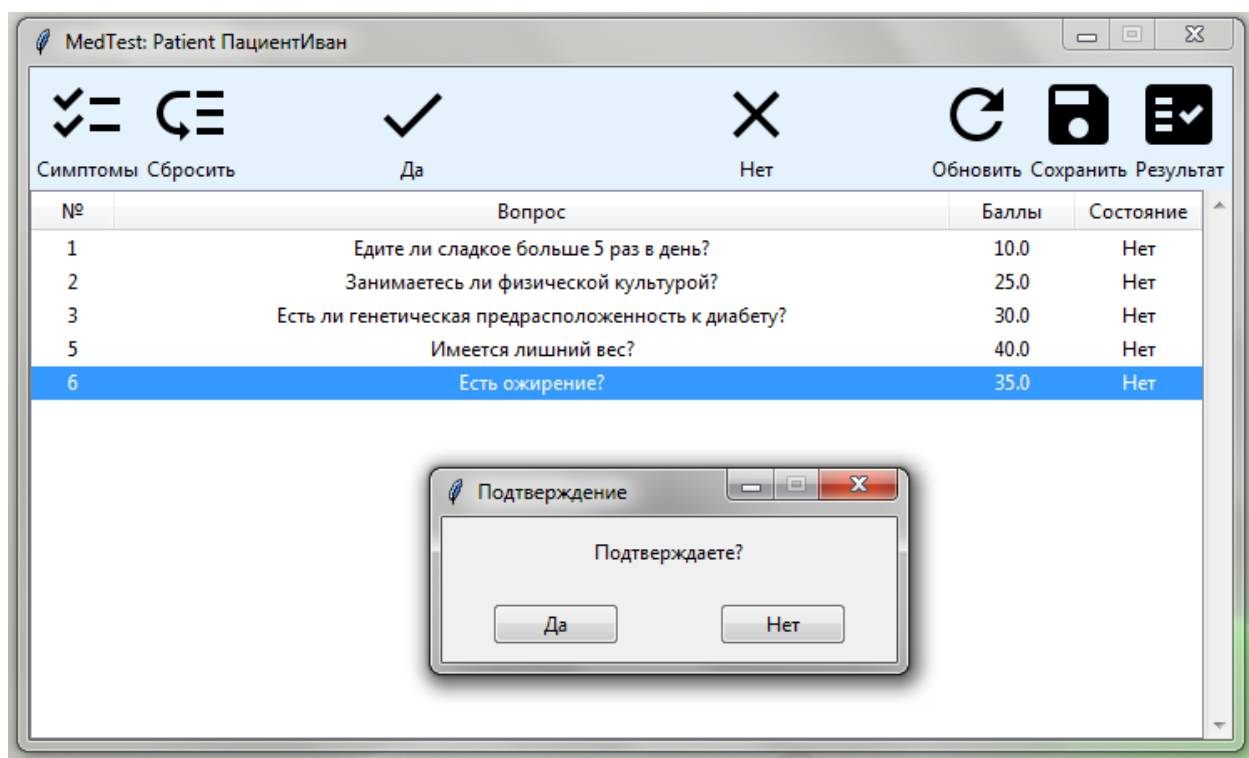
1. Успешно пройденная регистрация в программе, наличие логина/пароля для входа в систему.

Подготовительные действия:

Не требуются.

Основные действия в требуемой последовательности:

1. Войдите в систему с аккаунта, имеющего права пациента.
2. Читая вопросы, необходимо выбрать вариант «Да» или «Нет» на соответствующей кнопке сверху.



Заключительные действия:

Не требуются.

Ресурсы, расходуемые на операцию:

20-30 секунд.

Операция 3: Сброс прогресса / Просмотр симптомов / Просмотр результата

Условия, при соблюдении которых возможно выполнение операции:

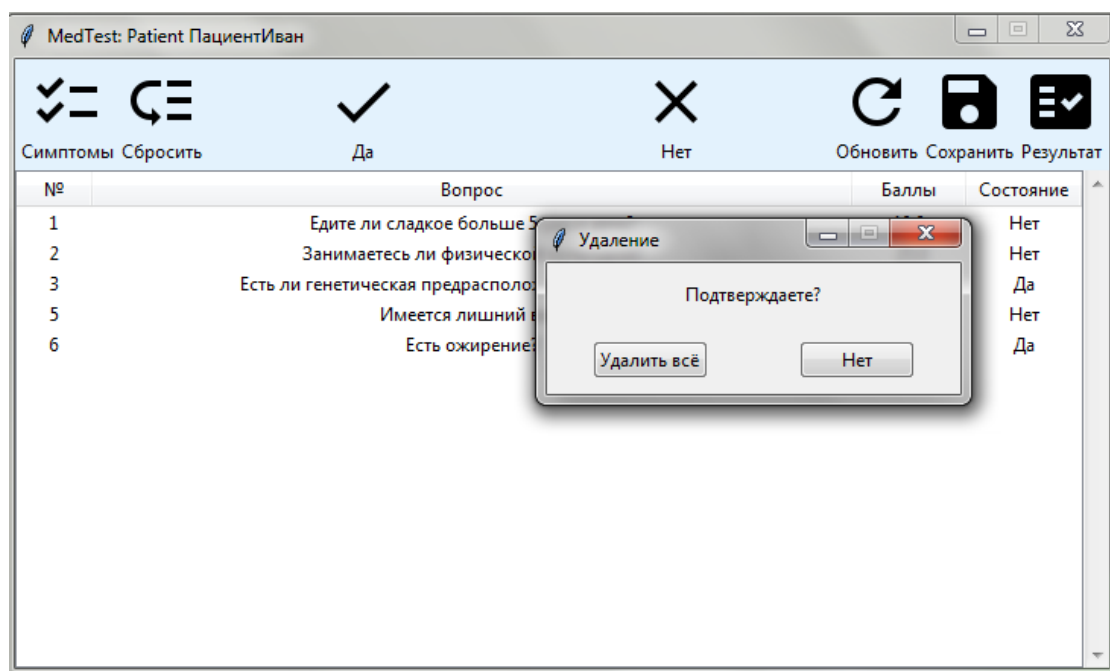
1. Успешно пройденная регистрация в программе, наличие логина/пароля для входа в систему, накопленный прогресс.

Подготовительные действия:

Не требуются.

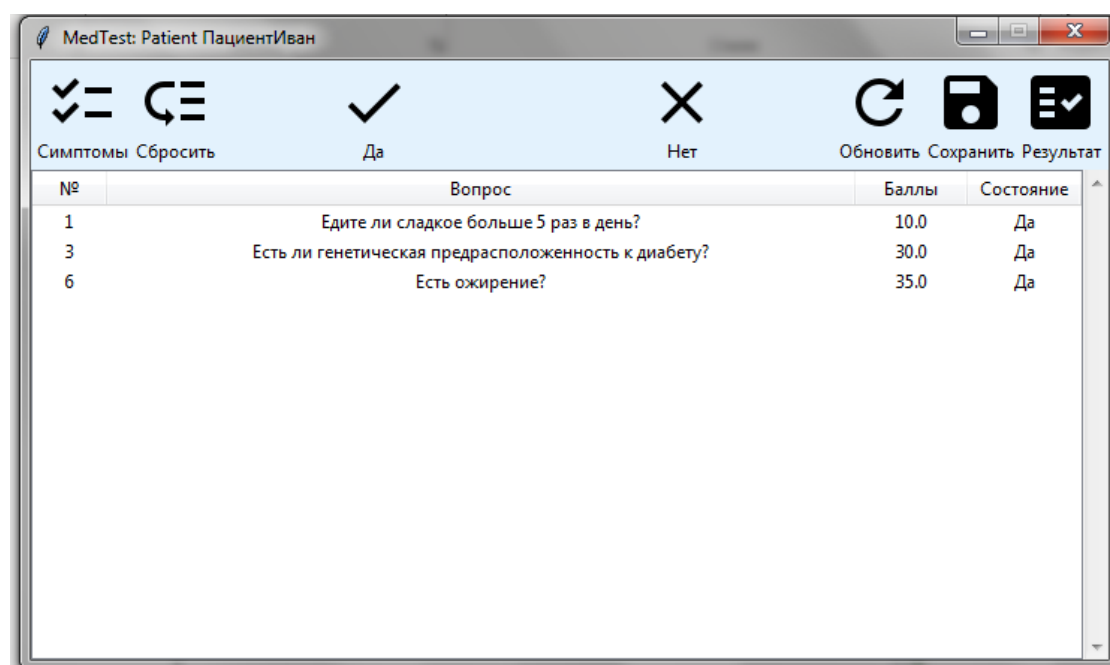
При выборе действия сброс прогресса Ваши основные действия в требуемой последовательности:

1. Войдите в систему с аккаунта, имеющего права пациента.
2. Для сброса накопленного прогресса необходимо нажать соответствующую кнопку «Сбросить» сверху.



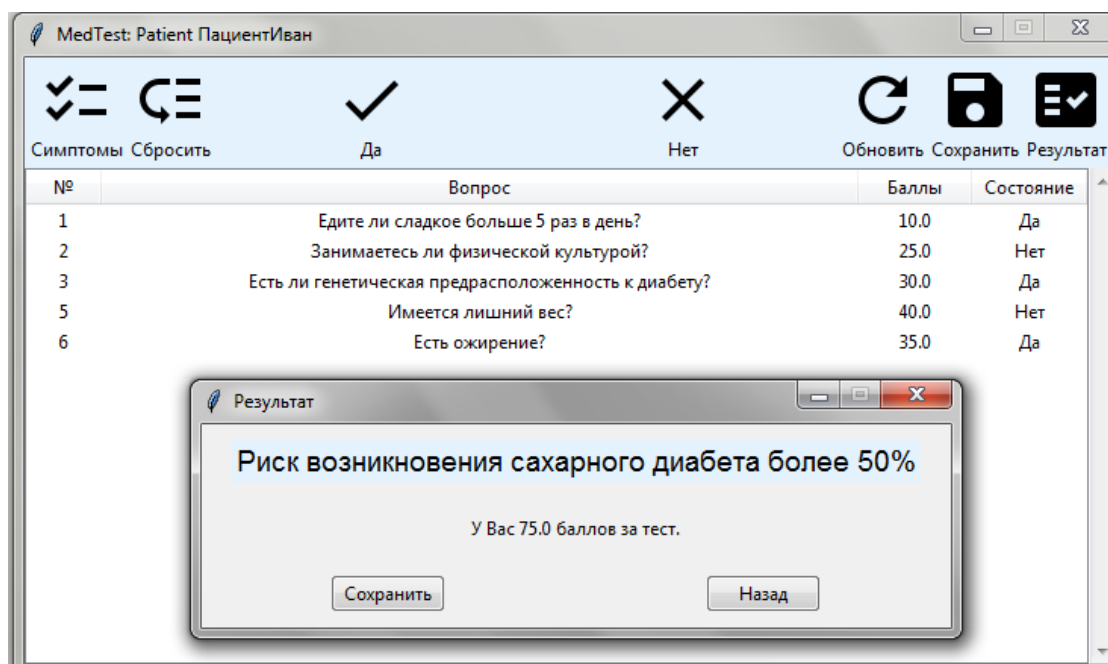
При выборе действия просмотр симптомов Ваши основные действия в требуемой последовательности:

1. Для просмотра накопленного прогресса необходимо нажать соответствующую кнопку «Симптомы» сверху. Вы получите список отмеченных положительно симптомов.



При выборе действия результат Ваши основные действия в требуемой последовательности:

1. Для просмотра результата нажмите соответствующую кнопку «Результат» сверху.



2. Сохранить результат можно нажав на кнопку «Сохранить», которая располагается в нижней левой части активного окна (см. скриншот выше).

Заключительные действия:

Не требуются.

Ресурсы, расходуемые на операцию:

60-120 секунд.

4. Описание операций. (Врач)

4.1. Выполняемые функции и задачи

Функции: Одобрение/отклонение диагноза, просмотр результата.

Задачи:

- визуализация результатов тестирования (при выполнении данной задачи врачу будет доступно 1 окно: с подробной информацией о диагнозе, баллах и рекомендациях по заболеванию)

4.2. Описание операций процесса обработки данных, необходимых для выполнения задач

Задача: «Визуализация результатов тестирования»

Операция 1: Регистрация в программе/вход в программу.

Условия, при соблюдении которых возможно выполнение операции:

1. Компьютер пользователя подключен к сети.

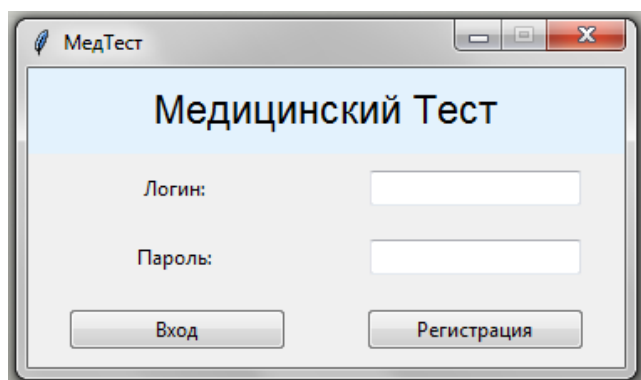
2. Программа работает корректно.

Подготовительные действия:

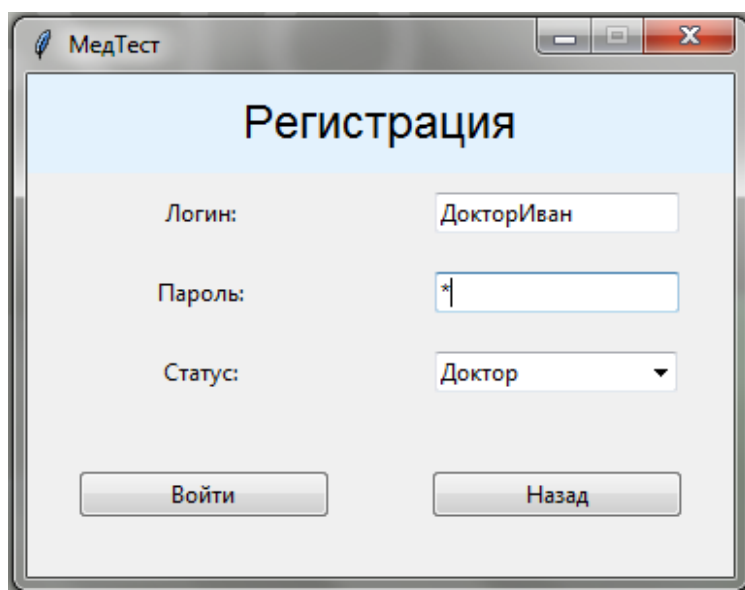
На компьютере пользователя необходимо выполнить дополнительные настройки, приведенные в п. 3.2 настоящего документа.

Основные действия в требуемой последовательности:

1. Ввести логин/пароль в окне входа в систему (для зарегистрированных пользователей) или нажать кнопку «Регистрация», если вы являетесь новым пользователем.



2. В окне регистрации выберите должность Доктора и введите логин/пароль



Заключительные действия:

Не требуются.

Ресурсы, расходуемые на операцию:

20-30 секунд.

Операция 2: Принятие/отклонение результата.

Условия, при соблюдении которых возможно выполнение операции:

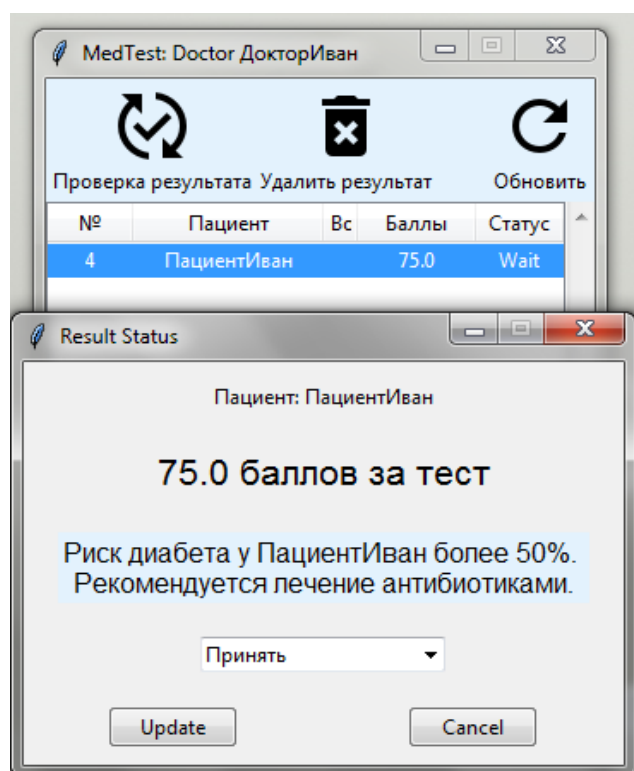
1. Успешно пройденная регистрация в программе, наличие логина/пароля для входа в систему.

Подготовительные действия:

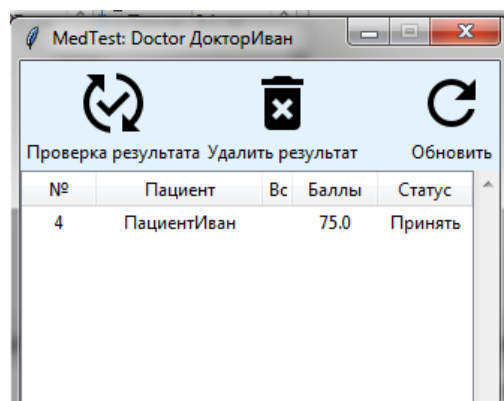
Не требуются.

Основные действия в требуемой последовательности:

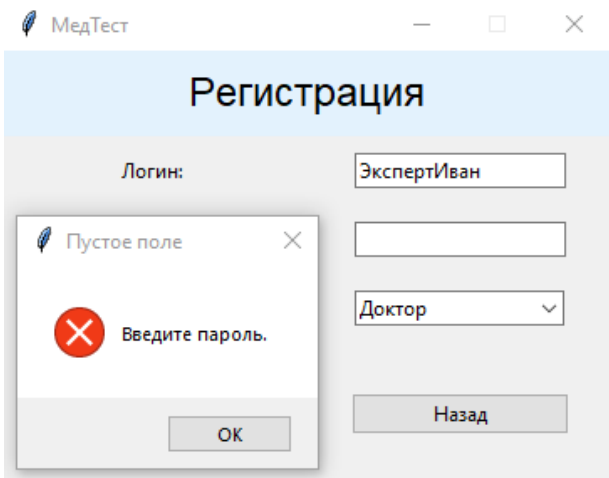
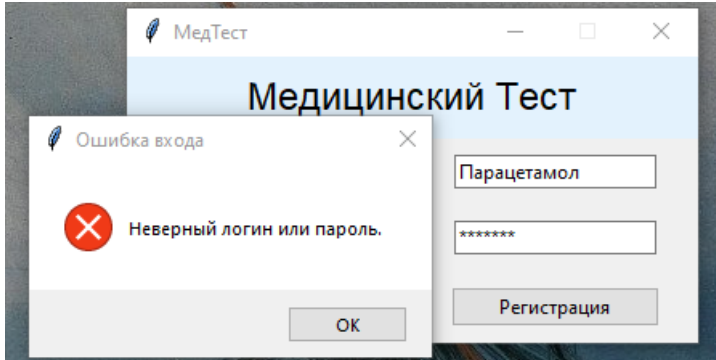
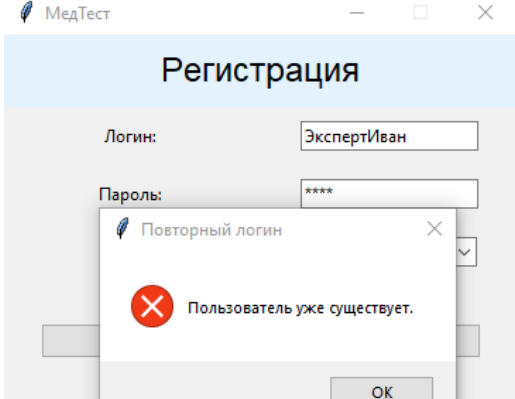
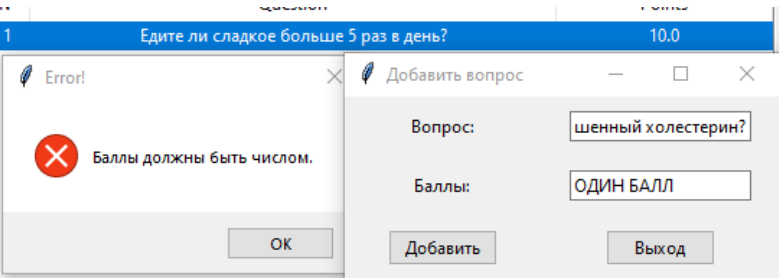
1. Войдите в систему с аккаунта, имеющего права доктора
2. Если необходимо проверить результат, необходимо нажать соответствующую кнопку «Проверка Результата».

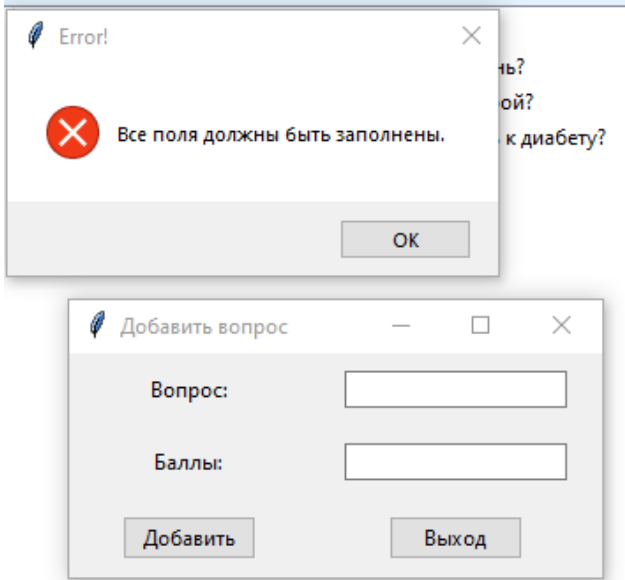


3. Если Вы ознакомлены и согласны с диагнозом системы, выберите кнопку «Принято» и нажмите на кнопку Update. Статус вопроса поменяется на «Принято».



Тестирование крайних случаев:

ПРОБЛЕМА	СКРИНШОТ	РЕШЕНИЕ
1. Не заполнен логин или пароль при регистрации:		Требуется ввести пароль.
2. Неправильный логин или пароль:		Требуется зарегистрироваться.
3. Регистрация уже существующего пользователя:		Требуется войти в аккаунт.
4. Ввод данных типа string вместо int:		Надо ввести баллы одним числом.

<p>5. Пустые поля в форме заполнения вопросов:</p>		<p>Надо заполнить поля для вопроса.</p>
--	--	---

Краткое описание каждого класса и метода:

class WelcomeFrame: Класс главного первого окна входа и регистрации.

def login_frame: метод определения виджетов окна входа в аккаунт или регистрации нового аккаунта.

def login: метод идентификации пользователя

def register_frame: метод определения виджетов регистрации юзера

def register: метод регистрации пользователя

create_user_db: метод создания записи юзера в бд sql

Start(username): метод-мост для запуска следующего класса

class Start: Класс вызова интерфейсов по статусу юзера.

def start_test: метод определения принадлежности уникального класса для уникального вызова.

def set_usermode: метод получения статуса юзера из бд sql

class MainFrame: Класс дефолтного окна интерфейса.

def view_data: метод передачи данных из бд sql в дерево отображения.

def get_user_id: метод получения выбранной записи

class ExpertFrame: Класс юзеринтерфейса эксперта

def init_user: метод построения основного окна работы эксперта.

class PatientFrame: Класс юзеринтерфейса врача

def init_user: метод построения основного окна работы врача.

`def show_questions:` метод передачи записей из бд вопросов в дерево.

class DoctorFrame: Класс юзеринтерфейса врача

`def init_user:` метод построения основного окна работы врача.

class AddFrame: Класс добавления вопроса.

`def widgets:` метод определения виджетов окна добавления вопроса.

`def safe_set:` метод безопасной передачи вопросов в бд.

class AddToTestFrame: Класс подтверждения добавления вопроса.

`def ok_button:` метод добавления вопроса в бд

class EditFrame: Класс редактирования вопроса.

`def default_data:` метод предоставления старых данных из бд

`def try_set:` метод записи нового вопроса в бд.

class DelFrame: Класс удаления вопроса.

`def ok_button:` метод удаления вопроса из бд

class DelFromTestFrame: Класс подтверждения удаления вопроса.

`def ok_button:` метод удаления вопроса из бд

class CleanProgress: Класс удаления прогресса тестирования.

`def ok_button:` метод удаления прогресса из бд

class AnamnesisFrame: Класс окна подтверждения завершения теста.

`def questions_info:` метод получения вопросов из бд

`def widgets:` метод виджетов результатов теста

`def safe_res:` метод безопасной передачи результата в бд

`def widgets:` метод виджетов результатов теста

class ChangeStatusFrame: Класс окна смены статуса диагноза.

`def diagnosis:` метод установления статуса диагнозу

`def default_data:` метод получения информации о результате из бд

class ShowAnswersFrame: Класс просмотра ответов юзера.

`def get_data_by_user:` передача данных из бд в отображение

class DefaultButton: Класс дефолтного конструктора кнопок.

class AddButton: Класс кнопки добавления вопроса.

```

class DelButton: Класс кнопки удаления вопроса.
    def safe_del: метод срабатывания при выбранном вопросе

class EditButton: Класс кнопки редактирования вопроса.
    def safe_edit: метод срабатывания при выбранном вопросе

class YesAnswerButton: Класс кнопки ответа Да.
    def safe_Yes: метод срабатывания при выбранном вопросе

class NoAnswerButton: Класс кнопки ответа Да.
    def safe_Yes: метод срабатывания при выбранном вопросе

class ShowSymptomsButton: Класс просмотра симптомов.
class DelSymptomsButton: Класс кнопки сброса прогресса.
class ShowAnswersButton: Класс кнопки сохранения.
class ShowResultButton: Класс кнопки проверки результата.
class ResResultButton: Класс кнопки удаления результата.

```

Код программы:

Main.py

```

from connector import root_frame
from authorization import EnterFrame

if __name__ == "__main__":
    EnterFrame(root_frame)

```

Connector.py

```

import tkinter as tk
from db import QuestionsDB, UsersDB, TestDB
"""Подключение к таблицам бд"""
qdb = QuestionsDB()
udb = UsersDB()
tdb = TestDB()

root_frame = tk.Tk() # Базовый пустой фрейм
root_frame.resizable(False, False)

```

Authorization.py

```
import tkinter as tk
from tkinter import ttk
from connector import udb
from tkinter import messagebox as ms
from navigator import Start

class EnterFrame:
    """Окно входа/регистрации"""
    def __init__(self, frame):
        self.frame = frame
        self.frame.title("МедТест")
        self.frame.geometry("+650+350")
        self.regfr = tk.Frame(self.frame)
        self.logfr = tk.Frame(self.frame)
        self.db = udb
        self.login_frame()
        self.frame.mainloop()

    def login_frame(self):
        """Виджеты логина"""
        head = tk.Label(self.logfr, text='Медицинский Тест', bg="#e3f2fd", width="25", font=(, 18), pady=10)
        head.grid(row=0, column=0, columnspan=2)
        label_username = tk.Label(self.logfr, text='Логин: ')
        label_username.grid(row=1, column=0, padx=20, pady=10)
        label_password = tk.Label(self.logfr, text='Пароль: ')
        label_password.grid(row=2, column=0, padx=20, pady=10)
        self.entry_username = ttk.Entry(self.logfr)
        self.entry_username.grid(row=1, column=1, padx=20, pady=10)
        self.entry_password = ttk.Entry(self.logfr, show='*')
        self.entry_password.grid(row=2, column=1, padx=20, pady=10)
        button_login = ttk.Button(self.logfr, text='Вход', width = '20',
                                command=lambda: self.login(self.entry_username.get(), self.entry_password.get()))
        button_login.grid(row=3, column=0, padx=20, pady=10)
        button_register = ttk.Button(self.logfr, text='Регистрация', width = '20', command=self.log_to_reg)
        button_register.grid(row=3, column=1, padx=20, pady=10)

        self.logfr.pack()

    def login(self, username, password):
        self.db.find_user_db(username, password)
        if self.db.c.fetchall():
            self.logfr.pack_forget()
            Start(username)
        else:
            ms.showerror('Ошибка входа', "Неверный логин или пароль.")

    def log_to_reg(self):
        self.logfr.pack_forget()
        self.register_frame()

    def register_frame(self):
        """Виджеты регистрации"""
        head = tk.Label(self.regfr, text='Регистрация', bg="#e3f2fd", width="25", font=(, 18), padx=5, pady=10)
        head.grid(row=0, column=0, columnspan=2)
        lbl_unm = tk.Label(self.regfr, text='Логин: ')
        lbl_unm.grid(row=1, column=0, padx=20, pady=10)
        lbl_pwd = tk.Label(self.regfr, text='Пароль: ')
        lbl_pwd.grid(row=2, column=0, padx=20, pady=10)
```

```

lbl_umode = tk.Label(self.regfr, text='Статус: ')
lbl_umode.grid(row=3, column=0, padx=20, pady=10)
self.entry_username = ttk.Entry(self.regfr)
self.entry_username.grid(row=1, column=1, padx=20, pady=10)
self.entry_password = ttk.Entry(self.regfr, show='*')
self.entry_password.grid(row=2, column=1, padx=20, pady=10)
self.cbox_umode = ttk.Combobox(self.regfr,
                               state="readonly",
                               width = '17',
                               values=[u'Администратор', u'Эксперт', u'Доктор', u'Пациент'])
self.cbox_umode.grid(row=3, column=1, padx=20, pady=10)
self.cbox_umode.current(2)
button_register = ttk.Button(self.regfr, text='Войти', width = '20',
                              command=lambda:
                                self.register(self.entry_username.get(),
                                                self.entry_password.get(),
                                                self.cbox_umode.get()))
button_register.grid(row=4, column=0, padx=20, pady=30)
button_login = ttk.Button(self.regfr, text='Назад', width = '20', command=self.reg_to_log)
button_login.grid(row=4, column=1, padx=20, pady=30)
self.regfr.pack()

def register(self, username, password, umode):
    self.db.find_username_db(username)
    if self.db.c.fetchall():
        ms.showerror('Повторный логин', 'Пользователь уже существует.')
    elif password == "":
        ms.showerror('Пустое поле', 'Введите пароль.')
    elif username == "":
        ms.showerror('Пустое поле', 'Введите логин.')
    else:
        self.db.create_user_db(username, password, umode)
        self.regfr.pack_forget()
        Start(username)

def reg_to_log(self):
    self.regfr.pack_forget()
    self.login_frame()

```

Navigator.py

```

from interface import PatientFrame, ExpertFrame, DoctorFrame
from connector import root_frame, udb

class Start:
    """Класс вызова визуальных интерфейсов по ролям"""
    def __init__(self, username):
        self.db = udb
        self.username = username
        self.set_usermode()
        self.start_test()

    def start_test(self):
        if self.usermode == 'Эксперт' or self.usermode == "Администратор":
            root_frame.title(f"MedTest: {self.username}")
            app = ExpertFrame(root_frame)
            app.pack()

            elif self.usermode == 'Пациент':

```

```

root_frame.title(f"MedTest: Patient {self.username}")
app = PatientFrame(root_frame, self.username)
app.pack()

elif self.usermode == 'Доктор':
    root_frame.title(f"MedTest: Doctor {self.username}")
    app = DoctorFrame(root_frame)
    app.pack()

def set_usermode(self):
    """Получение юзермода по юзернейму из дб"""
    self.db.find_usermode_db(self.username)
    self.usermode = self.db.c.fetchone()[0]

```

Interface.py

```

from buttons import *

class MainFrame(tk.Frame):
    """Дефолтное окно интерфейса"""
    def __init__(self, root_frame):
        super().__init__(root_frame)
        self.root_frame = root_frame
        self.root_frame.bind("<FocusIn>", self.handle_focus_user)
        self.tree = ttk.Treeview(self) #дерево отображения записей из бд
        self.db = qdb
        self.toolbar = tk.Frame(bg="#e3f2fd", bd=2)
        self.toolbar.pack(side=tk.TOP, fill=tk.X)

    def handle_focus_user(self, event):
        """Обновление отображаемых данных при попадании фрейма в фокус"""
        if event.widget == self.root_frame:
            self.view_data()

    def view_data(self):
        """Метод передачи данных из бд sql в дерево отображения"""
        self.db.view_data_db()
        [self.tree.delete(i) for i in self.tree.get_children()]
        [self.tree.insert("", 'end', values=row) for row in self.db.c.fetchall()]

    def get_user_id(self):
        """Получение id выбранной записи"""
        selected = self.tree.focus()
        return self.tree.item(selected, 'values')[0]

class ExpertFrame(MainFrame):
    """Юзеринтерфейс ЭКСПЕРТА"""
    def __init__(self, root_frame):
        super().__init__(root_frame)
        self.tree = ttk.Treeview(self, columns=('№', 'Question', 'costs'),
                                height=25, show='headings', selectmode="browse")
        self.scroll = tk.Scrollbar(self, command=self.tree.yview)
        self.init_user()

    def init_user(self):
        self.view_data()
        update_button = UpdateButton(self)
        add_button = AddButton(self)

```

```

edit_button = EditQuestButton(self)
del_button = DelButton(self)

self.tree.column('№', width=30, anchor=tk.CENTER)
self.tree.column('Question', width=365, anchor=tk.CENTER)
self.tree.column('costs', width=150, anchor=tk.CENTER)

self.tree.heading('№', text='№')
self.tree.heading('Question', text='Question')
self.tree.heading('costs', text='Points')

self.tree.pack(side=tk.LEFT)
self.scroll.pack(side=tk.LEFT, fill=tk.Y)
self.tree.configure(yscrollcommand=self.scroll.set)

class PatientFrame(MainFrame):
    """Юзеринтерфейс ПАЦИЕНТА"""
    def __init__(self, root_frame, username):
        super().__init__(root_frame)
        self.session_username = username
        self.tree = ttk.Treeview(self, columns=('№', 'Вопрос', 'Баллы', 'Состояние'),
                                height=15, show='headings', selectmode="browse")
        self.scroll = tk.Scrollbar(self, command=self.tree.yview)
        self.init_user()

    def init_user(self):
        self.view_data()
        cart_button = ShowSymptomsButton(self)
        del_cart_button = DelSymptomsButton(self)
        add_to_cart_button = YesAnswerButton(self)
        rm_from_cart_button = NoAnswerButton(self)
        order_button = ResultButton(self)
        show_orders = ShowAnswersButton(self)
        update_button = UpdateButton(self)

        self.tree.column('№', width=50, anchor=tk.CENTER)
        self.tree.column('Вопрос', width=500, anchor=tk.CENTER)
        self.tree.column('Баллы', width=75, anchor=tk.CENTER)
        self.tree.column('Состояние', width=75, anchor=tk.CENTER)

        self.tree.heading('№', text='№')
        self.tree.heading('Вопрос', text='Вопрос')
        self.tree.heading('Баллы', text='Баллы')
        self.tree.heading('Состояние', text='Состояние')

        self.tree.pack(side=tk.LEFT)

        self.scroll.pack(side=tk.LEFT, fill=tk.Y)
        self.tree.configure(yscrollcommand=self.scroll.set)

    def show_questions(self):
        """метод передачи записей из бд вопросов в дерево"""
        self.db.show_cart_db()
        [self.tree.delete(i) for i in self.tree.get_children()]
        [self.tree.insert("", 'end', values=row) for row in self.db.c.fetchall()]

class DoctorFrame(MainFrame):
    """Юзеринтерфейс ВРАЧА"""
    def __init__(self, root_frame):

```



```

super().__init__(root_frame)
self.db = tdb
self.tree = ttk.Treeview(self, columns=('№', 'Пациент', 'Вопросы', 'Баллы', 'Статус'),
                        height=15, show='headings', selectmode="browse")
self.scroll = tk.Scrollbar(self, command=self.tree.yview)
self.init_user()

def init_user(self):
    self.view_data()
    change_state_button = ShowResultButton(self)
    delete_order_button = DelResButton(self)
    update_button = UpdateButton(self)

    self.tree.column('№', width=50, anchor=tk.CENTER)
    self.tree.column('Пациент', width=50, anchor=tk.CENTER)
    self.tree.column('Вопросы', width=100, anchor=tk.CENTER)
    self.tree.column('Баллы', width=50, anchor=tk.CENTER)
    self.tree.column('Статус', width=50, anchor=tk.CENTER)

    self.tree.heading('№', text='№')
    self.tree.heading('Пациент', text='Пациент')
    self.tree.heading('Вопросы', text='Вопросы')
    self.tree.heading('Баллы', text='Баллы')
    self.tree.heading('Статус', text='Статус')
    self.tree.pack(side=tk.LEFT)
    self.scroll.pack(side=tk.LEFT, fill=tk.Y)
    self.tree.configure(yscrollcommand=self.scroll.set)

```

Test.py

```

from tkinter import ttk
from connector import qdb, tk, tdb
from tkinter import messagebox as ms

class AddFrame(tk.Toplevel):
    """Окно добавления вопроса"""
    def __init__(self):
        super().__init__()
        self.db = qdb
        self.init_child()
        self.geometry('+900+500')
        self.widgets()
        self.width = "90"
        self.grab_set()
        self.focus_set()

    def init_child(self):
        self.title('Добавить вопрос')
        self.btn_ok = ttk.Button(self, text='Добавить')

    def widgets(self):
        self.label_sum = tk.Label(self, text='Баллы:')
        self.label_description = tk.Label(self, text='Вопрос:')
        self.btn_cancel = ttk.Button(self, text='Выход', command=self.destroy)
        self.entry_money = ttk.Entry(self)
        self.entry_description = ttk.Entry(self)

```

```

self.label_description.grid(row=0, column=0, padx=20, pady=10)
self.label_sum.grid(row=1, column=0, padx=20, pady=10)
self.entry_description.grid(row=0, column=1, padx=20, pady=10)
self.entry_money.grid(row=1, column=1, padx=20, pady=10)
self.btn_cancel.grid(row=2, column=1, padx=30, pady=10)
self.btn_ok.grid(row=2, column=0, padx=30, pady=10)
self.btn_ok.bind('<Button-1>', self.safe_set)

def safe_set(self, event):
    """метод безопасной передачи вопросов в бд"""
    descr = self.entry_description.get()
    pnts = self.entry_money.get()
    if descr == " " or pnts == "":
        ms.showerror('Error!', 'Все поля должны быть заполнены.')
    else:
        try:
            self.try_set(descr, pnts)
            self.destroy()
        except:
            ms.showerror('Error!', 'Баллы должны быть числом.')

def try_set(self, descr, price):
    self.db.insert_data_db(descr, float(price))

class EditFrame(AddFrame):
    """Окно редактирования вопроса"""
    def __init__(self, sel_id):
        super().__init__()
        self.sel_id = sel_id
        self.default_data()

    def init_child(self):
        self.title('Редактирование вопроса')
        self.btn_ok = ttk.Button(self, text='Готово')

    def try_set(self, descr, price):
        self.db.edit_record_db(descr, float(price), self.sel_id)

    def default_data(self):
        """Подставление старых данных из бд"""
        self.db.default_data_db(self.sel_id)
        row = self.db.c.fetchone()
        self.entry_description.insert(0, row[1])
        self.entry_money.insert(0, row[2])

class DelFrame(tk.Toplevel):
    """Окно подтверждения удаления вопроса"""
    def __init__(self, sel_id):
        super().__init__()
        self.sel_id = sel_id
        self.btn_cancel = ttk.Button(self, text='Нет', command=self.destroy)
        self.label_description = tk.Label(self, text='Подтверждаете?')
        self.db = qdb
        self.init_child()

    def init_child(self):
        self.title('Удаление')
        self.geometry('+400+300')
        self.resizable(False, False)

```

```

self.ok_button()
self.label_description.grid(row=0, column=0, columnspan=2, padx=20, pady=10)
self.btn_cancel.grid(row=1, column=1, padx=30, pady=10)
self.btn_ok.grid(row=1, column=0, padx=30, pady=10)
self.grab_set()
self.focus_set()

def ok_button(self):
    self.btn_ok = ttk.Button(self, text='Удалить')
    self.btn_ok.bind('<Button-1>', lambda event: [self.db.del_record_db(self.sel_id),
                                                self.destroy()])

class AddToTestFrame(DelFrame):
    """Окно подтверждения добавления вопроса"""
    def __init__(self, sel_id):
        super().__init__(sel_id)
        self.title('Подтверждение')

    def ok_button(self):
        self.btn_ok = ttk.Button(self, text='Да')
        self.btn_ok.bind('<Button-1>', lambda event: [self.db.add_to_cart_db(self.sel_id),
                                                    self.destroy()])

class DelFromTestFrame(DelFrame):
    """Окно подтверждения удаления вопроса"""
    def __init__(self, sel_id):
        super().__init__(sel_id)
        self.title('Удаление')

    def ok_button(self):
        self.btn_ok = ttk.Button(self, text='Готово')
        self.btn_ok.bind('<Button-1>', lambda event: [self.db.rm_from_cart_db(self.sel_id),
                                                    self.destroy()])

class CleanProgress(DelFrame):
    """Окно подтверждения очистки прогресса"""
    def __init__(self):
        super().__init__(0)
        self.title('Удаление')

    def ok_button(self):
        self.btn_ok = ttk.Button(self, text='Удалить всё')
        self.btn_ok.bind('<Button-1>', lambda event: [self.db.clean_cart_db(),
                                                    self.destroy()])

class AnamnesisFrame(tk.Toplevel):
    """Окно подтверждения прохождения теста"""
    def __init__(self, username):
        super().__init__()
        self.db = qdb
        self.odb = tdb
        self.init_child()
        self.username = username

        self.res_ids = ""
        self.total = 0

```

```

self.questions_info()
self.widgets()

self.grab_set()
self.focus_set()

def questions_info(self):
    """Получение вопросов"""
    self.db.show_cart_db()
    rows = self.db.c.fetchall()
    names, ids = [], []
    for row in rows:
        ids.append(row[0])
        names.append(row[1])
        self.total += row[2]

def init_child(self):
    self.title('Результат')
    self.geometry('+900+500')
    self.resizable(False, False)
    self.btn_ok = ttk.Button(self, text='Сохранить')

def widgets(self):

    self.label_total_value = tk.Label(self, text=f'У Вас {self.total} баллов за тест.')
    if self.total > 100:
        self.label_total = tk.Label(self,
                                     text='Риск возникновения сахарного диабета более 80%',
                                     font=("", 14),
                                     bg="#e3f2fd")
        res_ids = 'Риск возникновения сахарного диабета более 80%'

    elif self.total > 50 and self.total < 100:
        self.label_total = tk.Label(self,
                                     text='Риск возникновения сахарного диабета более 50%',
                                     font=("", 14),
                                     bg="#e3f2fd")
        res_ids = 'Риск возникновения сахарного диабета более 50%'

    elif self.total > 20 and self.total < 50:
        self.label_total = tk.Label(self,
                                     text='Риск возникновения сахарного диабета более 20%',
                                     font=("", 14),
                                     bg="#e3f2fd")
        res_ids = 'Риск возникновения сахарного диабета более 20%'

    elif self.total < 20:
        self.label_total = tk.Label(self,
                                     text='Риск возникновения сахарного диабета менее 10%',
                                     font=("", 14),
                                     bg="#e3f2fd")
        res_ids = 'Риск возникновения сахарного диабета менее 10%'

    self.btn_cancel = ttk.Button(self, text='Назад', command=self.destroy)
    self.label_total.grid(row=0, column=0, padx=20, pady=10, columnspan=2)
    self.label_total_value.grid(row=1, column=0, padx=20, pady=10, columnspan=2)
    self.btn_cancel.grid(row=3, column=1, padx=30, pady=10)
    self.btn_ok.grid(row=3, column=0, padx=30, pady=10)
    self.btn_ok.bind('<Button-1>', self.safe_res)

```

```

def safe_res(self, event):
    """Безопасная передача результата"""
    if self.label_total == "":
        ms.showerror('Error!', 'Результат не может быть пустым.')
    else:
        try:
            self.odbc.create_order_db(self.username, self.res_ids, self.total)
            self.db.clean_cart_db()
            self.destroy()
        except:
            ms.showerror('Error!', 'Что-то не так...')

class ChangeStatusFrame(tk.Toplevel):
    """Окно смены статуса диагноза"""
    def __init__(self, sel_id):
        super().__init__()
        self.sel_id = sel_id
        self.db = tdb

        self.default_data()
        self.diagnosis()
        self.title('Result Status')
        self.widgets()
        self.geometry('+300+500')
        self.resizable(False, False)
        self.grab_set()
        self.focus_set()

    def diagnosis(self):
        self.label_uid = tk.Label(self, text=f'Пациент: {self.client_id}')
        self.cbox_state = ttk.Combobox(self, state="readonly", values=[u'Принять', u'Отклонить'])
        self.label_state = tk.Label(self, text='Статус диагноза:')
        self.label_rate = tk.Label(self, text=f'{self.total} баллов за тест', font=("", 16))
        if self.total > 100:
            self.label_total = tk.Label(self,
                                         text=f'Критический риск диабета у {self.client_id} - более 80%. '
                                              f'\nРекомендуется лечение в стационаре.',
                                         font=("", 12),
                                         bg="#e3f2fd")

        elif self.total > 50 and self.total < 100:
            self.label_total = tk.Label(self,
                                         text=f'Риск диабета у {self.client_id} более 50%. '
                                              f'\nРекомендуется лечение антибиотиками.',
                                         font=("", 12),
                                         bg="#e3f2fd")

        elif self.total > 20 and self.total < 50:
            self.label_total = tk.Label(self,
                                         text=f'Риск диабета у {self.client_id} более 20%. '
                                              f'\nРекомендуется строгая диета.',
                                         font=("", 12),
                                         bg="#e3f2fd")

        elif self.total < 20:
            self.label_total = tk.Label(self,
                                         text=f'Минимальный риск диабета у {self.client_id} - менее 10%. '
                                              f'\nРекомендуется умеренная диета.',
                                         font=("", 12),
                                         bg="#e3f2fd")

        self.btn_cancel = ttk.Button(self, text='Cancel', command=self.destroy)

```

```

        self.btn_ok = ttk.Button(self, text='Update')

def default_data(self):
    """Получение информации о результате из бд"""
    self.db.default_data_db(self.sel_id)
    row = self.db.c.fetchone()
    self.client_id = row[1]
    self.goods = row[2]
    self.total = row[3]
    self.state = row[4]

def widgets(self):
    self.cbox_state.current(0)
    self.label_uid.grid(row=0, column=0, padx=20, pady=10, columnspan=2)
    self.label_rate.grid(row=1, column=0, padx=20, pady=10, columnspan=2)
    self.label_total.grid(row=2, column=0, padx=20, pady=10, columnspan=2)
    self.cbox_state.grid(row=3, column=0, padx=20, pady=10, columnspan=2)
    self.btn_ok.grid(row=4, column=0, padx=20, pady=10)
    self.btn_cancel.grid(row=4, column=1, padx=20, pady=10)
    self.btn_ok.bind('<Button-1>', lambda event: [self.db.update_order_state_db(self.sel_id, self.cbox_state.get()),
                                                self.destroy()])

class RmFrame(DelFrame):
    """Окно подтверждения удаления"""
    def __init__(self, sel_id):
        super().__init__(sel_id)
        self.db = tdb
        self.title('Удаление')

    def ok_button(self):
        self.btn_ok = ttk.Button(self, text='Готово')
        self.btn_ok.bind('<Button-1>', lambda event: [self.db.delete_order_db(self.sel_id),
                                                    self.destroy()])

class ShowAnswersFrame(tk.Toplevel):
    """Окно просмотра ответов"""
    def __init__(self, uid):
        super().__init__()
        self.uid = uid
        self.db = tdb
        self.tree = ttk.Treeview(self, columns=('№', 'Баллы', 'Состояние'), height=15, show='headings',
                                selectmode="browse") # Дерево отображения заказов юзера
        self.scroll = tk.Scrollbar(self, command=self.tree.yview)
        self.init_child()
        self.title("Мои ответы")
        self.geometry('225x150+300+300')
        self.grab_set()
        self.focus_set()

    def init_child(self):
        self.get_data_by_user()

        self.tree.column('№', width=75, anchor=tk.CENTER)
        self.tree.column('Баллы', width=77, anchor=tk.CENTER)
        self.tree.column('Состояние', width=50, anchor=tk.CENTER)

        self.tree.heading('№', text='№')
        self.tree.heading('Баллы', text='Баллы')

```

```

self.tree.heading('Состояние', text='Состояние')
self.tree.pack(side=tk.LEFT)
self.scroll.pack(side=tk.LEFT, fill=tk.Y)
self.tree.configure(yscrollcommand=self.scroll.set)

def get_data_by_user(self):
    """Передача данных об ответах из бд в дерево отображения"""
    self.db.get_orders_by_username_db(self.uid)
    [self.tree.delete(i) for i in self.tree.get_children()]
    [self.tree.insert("", 'end', values=row) for row in self.db.c.fetchall()]

```

Db.py

```

import sqlite3

class MedicineDB:
    """Создание бд медтестов"""
    def __init__(self):
        self.conn = sqlite3.connect('shop.db')
        self.c = self.conn.cursor()
        self._create_goods_db()
        self._create_users_db()
        self._create_orders_db()

    def _create_goods_db(self):
        """Создание таблицы вопросов"""
        self.c.execute(
            "CREATE TABLE IF NOT EXISTS questions (id INTEGER PRIMARY KEY, description TEXT NOT NULL, costs REAL NOT NULL, cart TEXT)"
        )
        self.conn.commit()

    def _create_users_db(self):
        """Создание таблицы юзеров"""
        self.c.execute(
            "CREATE TABLE IF NOT EXISTS users (username TEXT NOT NULL PRIMARY KEY, password TEXT NOT NULL, role TEXT NOT NULL)"
        )
        self.conn.commit()

    def _create_orders_db(self):
        """Создание таблицы заказов"""
        self.c.execute(
            "CREATE TABLE IF NOT EXISTS orders (id INTEGER PRIMARY KEY, username TEXT NOT NULL, goods_ids TEXT NOT NULL, score REAL NOT NULL, state TEXT NOT NULL)"
        )
        self.conn.commit()

class QuestionsDB(MedicineDB):
    """Таблица вопросов"""
    def __init__(self):
        super().__init__()
        self.clean_cart_db() # При первой инициации в сессии очистка состояний корзины

    def insert_data_db(self, description, cost):
        """Добавление вопроса в бд"""
        self.c.execute(
            "INSERT INTO questions(description, costs, cart) VALUES (?, ?, ?)",
            (description, cost, 'Her')
        )
        self.conn.commit()

```

```

def edit_record_db(self, description, cost, sel_id):
    """Обновление информации о вопросе в бд"""
    self.c.execute("UPDATE questions SET description=?, costs=? WHERE ID=?", (description, cost, sel_id))
    self.conn.commit()

def add_to_cart_db(self, sel_id):
    """Смена статуса на \ "Да\ " в бд"""
    self.c.execute("UPDATE questions SET cart=? WHERE ID=?", ('Да', sel_id))
    self.conn.commit()

def rm_from_cart_db(self, sel_id):
    """Смена статуса на \ "Нет\ " в бд"""
    self.c.execute("UPDATE questions SET cart=? WHERE ID=?", ('Нет', sel_id))
    self.conn.commit()

def view_data_db(self):
    """Получение всех записей из таблицы вопросов"""
    self.c.execute("SELECT * FROM questions")

def del_record_db(self, sel_id):
    """Удаление вопроса по id"""
    self.c.execute("DELETE FROM questions WHERE id=?", (sel_id,))
    self.conn.commit()

def show_cart_db(self):
    """Получение всех вопросов с состоянием \ "Да\ """
    description = ('%Да%',)
    self.c.execute("SELECT * FROM questions WHERE cart LIKE ?", description)

def default_data_db(self, sel_id):
    """Получение записи вопроса по id"""
    self.c.execute("SELECT * FROM questions WHERE id=?", (sel_id,))

def clean_cart_db(self):
    """Смена состояния всех вопросов на \ "Нет\ """
    self.c.execute("UPDATE questions SET cart = 'Нет'")
    self.conn.commit()

class UsersDB(MedicineDB):
    """Таблица юзеров"""
    def __init__(self):
        super().__init__()

    def find_user_db(self, username, password):
        """Получение юзера по юзернейму и паролю"""
        self.c.execute("SELECT * FROM users WHERE username = ? and password = ?", (username, password))

    def create_user_db(self, username, password, role):
        """Запись юзера в бд"""
        self.c.execute("INSERT INTO users(username,password,role) VALUES(?,?,?)", (username, password, role))
        self.conn.commit()

    def find_username_db(self, username):
        """Проверка наличия юзернейма в бд"""
        self.c.execute("SELECT username FROM users WHERE username = ?", (username,))

    def find_usermode_db(self, username):
        """Получение роли по юзернейму из бд"""

```



```

self.c.execute("SELECT role FROM users WHERE username = ?", (username,))

class TestDB(MedicineDB):
    """Таблица вопросов"""
    def __init__(self):
        super().__init__()

    def view_data_db(self):
        """Получение всех записей из таблицы вопросов"""
        self.c.execute("SELECT * FROM orders")

    def get_orders_by_username_db(self, username):
        """Получение всех записей о вопросах от юзернейма"""
        self.c.execute("SELECT ID, SCORE, STATE FROM orders WHERE username = ? ", (username,))

    def get_total_by_username_db(self, username):
        """Получение total юзернейма"""
        self.c.execute("SELECT SCORE FROM orders WHERE username = ? ", (username,))

    def default_data_db(self, sel_id):
        """Получение записи о вопросе по id заказа"""
        self.c.execute("SELECT * FROM orders WHERE id=?", (sel_id,))

    def create_order_db(self, username, goods_ids, score):
        """Создание записи о вопросе в бд"""
        self.c.execute("INSERT INTO orders(username, goods_ids, score, state) VALUES(?,?,?,?)",
            (username, goods_ids, score, 'Wait'))
        self.conn.commit()

    def update_order_state_db(self, order_id, state):
        """Обновление статуса вопроса в бд"""
        self.c.execute("UPDATE orders SET state=? WHERE ID=?", (state, order_id))
        self.conn.commit()

    def delete_order_db(self, order_id):
        """Удаление записи вопроса из бд"""
        self.c.execute("DELETE FROM orders WHERE id=?", (order_id,))
        self.conn.commit()
        print(order_id, ' order deld')

```

Buttons.py

```

import tkinter

from test import *

class DefaultButton(tk.Frame):
    """Дефолтный конструктор кнопок"""
    def __init__(self, toolbar, text, command, image, side='l'):
        super().__init__(toolbar)
        btn_open_dialog = tk.Button(toolbar, text=text, command=command, bg='#e3f2fd', bd=0,
            compound=tk.TOP, image=image)

        if side == 'r':
            btn_open_dialog.pack(side=tk.RIGHT)
        elif side == 'c':
            btn_open_dialog.pack(side=tk.LEFT, expand = tk.YES)
        else:

```

```

btn_open_dialog.pack(side=tk.LEFT)

class AddButton(DefaultButton):
    """Кнопка добавления вопроса"""
    def __init__(self, frame):
        self.img = tk.PhotoImage(file='icons/add.png')
        super().__init__(toolbar=frame.toolbar,
                         text='Добавить Вопрос',
                         command=AddFrame,
                         image=self.img,
                         side = 'c')

class DelButton(DefaultButton):
    """Кнопка удаления вопроса"""
    def __init__(self, frame):
        self.img = tk.PhotoImage(file='icons/delete.png')
        self.frame = frame
        super().__init__(frame.toolbar,
                         text='Удалить Вопрос',
                         command=lambda: self.safe_del(),
                         image=self.img,
                         side = 'c')

    def safe_del(self):
        """Срабатывание только при выбранном вопросе"""
        try:
            DelFrame(self.frame.get_user_id())
        except:
            pass

class EditQuestButton(DefaultButton):
    """Кнопка редактирования вопроса"""
    def __init__(self, frame):
        self.img = tk.PhotoImage(file='icons/edit.png')
        self.frame = frame
        super().__init__(frame.toolbar,
                         text='Изменить Вопрос',
                         command=lambda: self.safe_edit(),
                         image=self.img,
                         side = 'c')

    def safe_edit(self):
        """Срабатывание только при выбранном вопроса"""
        try:
            EditFrame(self.frame.get_user_id())
        except:
            pass

class YesAnswerButton(DefaultButton):
    """Кнопка ответа ДА"""
    def __init__(self, frame):
        self.img = tk.PhotoImage(file='icons/yes.png')
        self.frame = frame
        super().__init__(frame.toolbar,
                         text='Да',
                         command=lambda: self.safe_Yes(),
                         image=self.img,

```

```

        side='c')

def safe_Yes(self):
    """Срабатывание только при выбранном товаре"""
    try:
        AddToTestFrame(self.frame.get_user_id())
    except:
        pass

class NoAnswerButton(DefaultButton):
    def __init__(self, frame):
        self.img = tk.PhotoImage(file='icons/no.png')
        self.frame = frame
        super().__init__(frame.toolbar,
                        text='Нет',
                        command=lambda: self.safe_No(),
                        image=self.img,
                        side='c')

def safe_No(self):
    """Срабатывание только при выбранном вопросе"""
    try:
        DelFromTestFrame(self.frame.get_user_id())
    except:
        pass

class ShowSymptomsButton(DefaultButton):
    """Кнопка просмотр симптомов"""
    def __init__(self, frame):
        self.img = tk.PhotoImage(file='icons/list.png')
        self.frame = frame
        super().__init__(frame.toolbar,
                        text='Симптомы',
                        command=lambda: frame.show_questions(),
                        image=self.img)

class DelSymptomsButton(DefaultButton):
    """Кнопка сброса"""
    def __init__(self, frame):
        self.img = tk.PhotoImage(file='icons/again.png')
        self.frame = frame
        super().__init__(frame.toolbar,
                        text='Сбросить',
                        command=lambda: CleanProgress(),
                        image=self.img)

class ResultButton(DefaultButton):
    """Кнопка просмотра результата"""
    def __init__(self, frame):
        self.img = tk.PhotoImage(file='icons/result.png')
        super().__init__(frame.toolbar,
                        text='Результат',
                        command=lambda: AnamnesisFrame(frame.session_username),
                        image=self.img,
                        side='r')

```

```

class ShowAnswersButton(DefaultButton):
    """Кнопка сохранения ответов пациента"""
    def __init__(self, frame):
        self.img = tk.PhotoImage(file='icons/save.png')
        super().__init__(frame.toolbar,
                         text='Сохранить',
                         command=lambda: ShowAnswersFrame(frame.session_username),
                         image=self.img,
                         side='r')

class ShowResultButton(DefaultButton):
    """Кнопка проверки результата"""
    def __init__(self, frame):
        self.img = tk.PhotoImage(file='icons/check.png')
        self.frame = frame
        super().__init__(frame.toolbar,
                         text='Проверка результата',
                         command=lambda: self.safe_change(),
                         image=self.img)

    def safe_change(self):
        """Срабатывание только при выборе"""
        try:
            ChangeStatusFrame(self.frame.get_user_id())
        except:
            pass

class DelResButton(DefaultButton):
    """Кнопка удаления результата"""
    def __init__(self, frame):
        self.img = tk.PhotoImage(file='icons/delete.png')
        self.frame = frame
        super().__init__(frame.toolbar,
                         text='Удалить результат',
                         command=lambda: self.safe_del_order(),
                         image=self.img)

    def safe_del_order(self):
        """Срабатывание только при выбранном заказе"""
        try:
            RmFrame(self.frame.get_user_id())
        except:
            pass

class UpdateButton(DefaultButton):
    """Кнопка перезагрузки дерева отображения"""
    def __init__(self, frame):
        self.img = tk.PhotoImage(file='icons/updating.png')
        super().__init__(frame.toolbar,
                         text='Обновить',
                         command=lambda: frame.view_data(),
                         image=self.img,
                         side='r')

```