МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

ОТЧЁТ

По лабораторной работе №3

по дисциплине   
«Основы теории интеллектуальных вычислительных систем»

«Реализация проекта многопользовательской системы интеллектуальной поддержки с использованием технологии Клиент-сервер»

РУКОВОДИТЕЛЬ:

Мисевич П.В.

СТУДЕНТЫ:

Шатов Д.В.

Папанов Р.В.

19-ПО

Работа защищена

С оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_

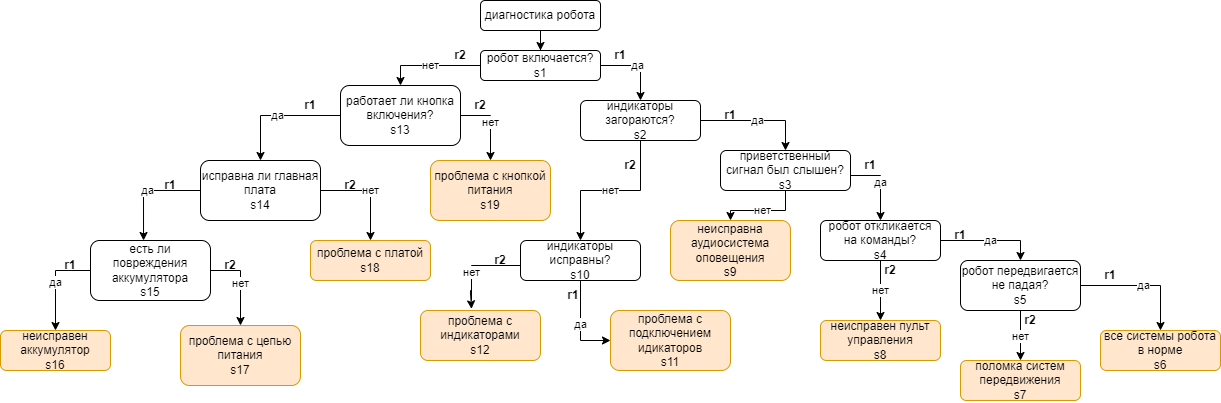
Нижний Новгород

2023

**Цель:** Реализовать проект многопользовательской системы интеллектуальной поддержки с использованием технологии Клиент-сервер.

**Тема:** Дерево проверки работоспособности систем робота.

**Дерево:**

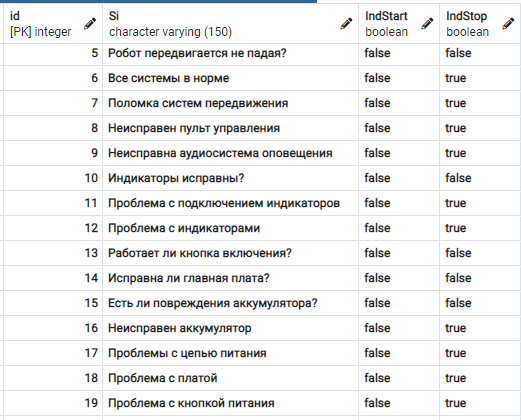


**Описание проекта:**

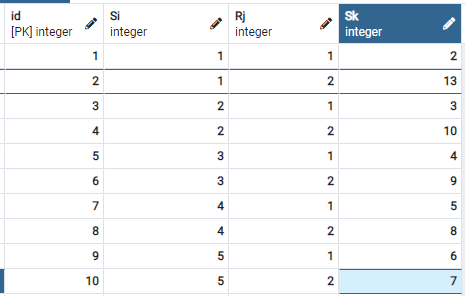
1) Серверное приложение – это Python-приложение, написанное с помощью микрофреймворка flask для организации сервисной архитектуры REST для доступа по http-запросам. База данных – PostgreSQL. Доступ к базе данных с серверной части происходит с помощью ORM SQLalchemy. Редактирование правил происходит в БД с помощью приложения администрирования PostgreSQL – pgAdmin4.

2) Клиентское приложение – это Python-приложение, написанное с помощью библиотеки графического интерфейса – tkinter. Доступ к серверной части происходит по средствам http-запросов, с помощью библиотеки requests.

**PgAdmin4. Базы данных.**

База данных событий:  


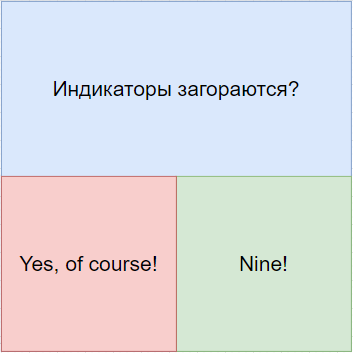
База данных правил:



**Этап запуска.**

1) Запускаем серверное приложение.

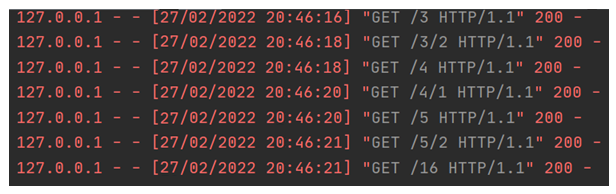
2) Запускаем клиентское приложение.





3) Так же видим сообщения в доказательство того, что это действительно клиент-сервер:



**Вывод:** в ходе выполнения данной лабораторной работы мы выяснили, что клиент-серверный подход наиболее гибкий для работы с экспертными системами и системами интеллектуальной поддержки. Многие пользователи могут пользоваться, при этом если необходимо добавить правил или подкорректировать – это делается довольно просто, достаточно администратору с помощью приложения администрирования БД быстро и качественно все поменять.

**Серверная часть:**

from flask import Flask, jsonify

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'postgres://postgres:12345@localhost:5432/intelligent\_support'

db = SQLAlchemy(app)

class Events(db.Model):

    \_\_tablename\_\_ = "Events"

    id = db.Column(db.Integer, primary\_key=True)

    Si = db.Column(db.String(150), nullable=False)

    IndStart = db.Column(db.Boolean, nullable=False)

    IndStop = db.Column(db.Boolean, nullable=False)

class Rules(db.Model):

    \_\_tablename\_\_ = "Rules"

    id = db.Column(db.Integer, primary\_key=True)

    Si = db.Column(db.Integer, nullable=False)

    Rj = db.Column(db.Integer, nullable=False)

    Sk = db.Column(db.Integer, nullable=False)

@app.route('/<int:si>/<int:rj>', methods=['GET'])

def get\_next\_state(si, rj):

    rule = db.session.query(Rules).filter(Rules.Si == si, Rules.Rj == rj).first()

    if rule is None:

        return jsonify("There is no such rule"), 400

    return jsonify({"id": rule.Sk}), 200

@app.route('/<int:id\_state>', methods=['GET'])

def get\_state\_info(id\_state):

    state = db.session.query(Events).filter(Events.id == id\_state).first()

    if state is None:

        return jsonify("There is no such condition"), 400

    return jsonify({"question": state.Si,

                    "stop": state.IndStop}), 200

@app.route('/start\_state', methods=['GET'])

def get\_start\_state():

    state = db.session.query(Events).filter(Events.IndStart).first()

    if state is None:

        return jsonify("There is no starting vertex"), 400

    return jsonify({"question": state.Si,

                    "id": state.id}), 200

if \_\_name\_\_ == "\_\_main\_\_":

    db.create\_all()

    app.run()

**Клиентская часть:**

from tkinter import \*

import requests

def main():

    log\_out = Logical\_output()

    windows = Tk()

    windows.geometry("500x500")

    windows.title("Диагностика систем робота")

    b\_yes = Button(text="Yes, of course!", font=('Arial',28),

                fg="black", bg="blue", command=log\_out.check\_yes)

    b\_yes.place(x=0,y=250,width=250,height=250)

    b\_no = Button(text="Nine", font=('Arial',28),

                fg="black", bg="brown", command=log\_out.check\_no)

    b\_no.place(x=250,y=250,width=250,height=250)

    global label\_question

    label\_question = Message(text=log\_out.question,

                             font=("Arial", 24), width=300)

    label\_question.place(x=100, y=0, width=300, height=200)

    windows.mainloop()

class Logical\_output(object):

    def \_\_init\_\_(self):

        response = requests.get('http://127.0.0.1:5000/start\_state')

        if not response:

            print(response.text)

            exit(1)

        self.current\_state = response.json()["id"]

        self.question = response.json()["question"]

        self.flag\_end = False

    def check(self, yes\_or\_no):

        if not self.flag\_end:

            response = requests.get(f'http://127.0.0.1:5000/{self.current\_state}/{yes\_or\_no}')

            if not response:

                print(response.text)

                exit(1)

            self.current\_state = response.json()["id"]

            data = requests.get(f'http://127.0.0.1:5000/{self.current\_state}')

            if not response:

                print(response.text)

                exit(1)

            data = data.json()

            if not data["stop"]:

                label\_question["text"] = data["question"]

            else:

                text = f" {data['question']}!"

                label\_question["text"] = text

                self.flag\_end = True

    def check\_yes(self):

        self.check(1)

    def check\_no(self):

        self.check(2)

if \_\_name\_\_ == "\_\_main\_\_":

    main()