**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

**Отчет по практической работе №5**

по дисциплине «Введение в информационные технологии» на тему:

Создать телеграм-бота с расписанием и графический интерфейс для редактирования расписания.

Выполнил: студент группы БВТ2104

Кравченко Николай Алексеевич

Проверил:

Мкртчян Грач Маратович

Москва

2021

**1 Постановка задачи**

1. Создать телеграм-бота с расписанием для Вашей группы, используя базу данных postgres и библеотеку telebot.
2. Создать графический интерфейс для редактирования расписания, используя библеотеку PyQt5.

**Задание**

1)

* Бот должен иметь ник-нейм формата <номер группы>\_<фамилия разработчика>\_bot.
* Бот должен иметь имя формата <номер группы>\_<фамилия разработчика>.
* Манера общения бота – "на Вы".
* Использование базы данных PostgreSQL.
* Использование pyTelegramBotAPI.
* Использование адаптера psycopg2.
* Формат вывода на каждый день недели:
* <День недели>

\_\_\_\_\_\_\_\_\_\_\_\_

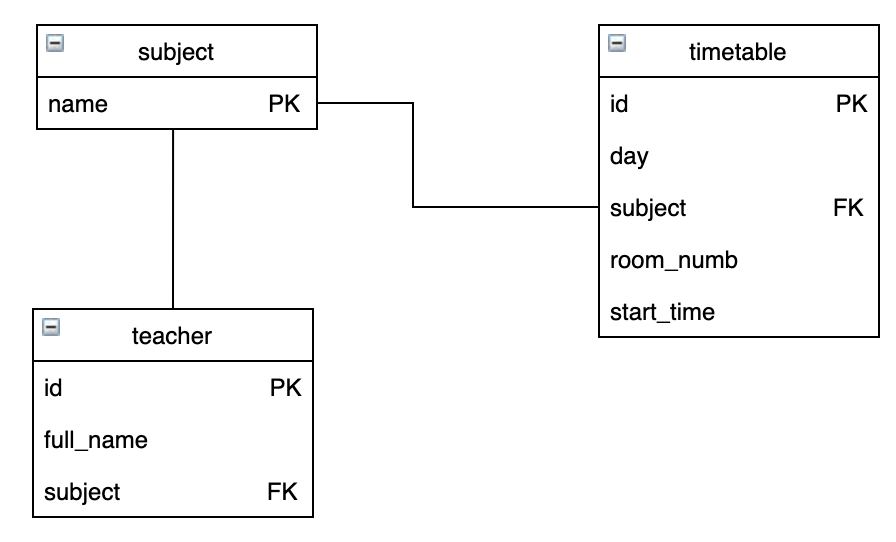
<Предмет> <Кабинет> <Время> <Преподаватель>

…

<Предмет> <Кабинет> <Время> <Преподаватель>

\_\_\_\_\_\_\_\_\_\_\_\_

* Во время работы бота должны быть отображены следующие графические кнопки:
  + Понедельник
  + Вторник
  + Среда
  + Четверг
  + Пятница
  + Расписание на текущую неделю
  + Расписание на следующую неделю
* При нажатии на кнопку с днем недели бот должен выводить информацию из базы данных с расписанием на выбранный день текущей недели.
* При нажатии на кнопку "Расписание на текущую неделю" бот должен выводить информацию из базы данных с расписанием на всю текущую неделю.
* При нажатии на кнопку "Расписание на следующую неделю" бот должен выводить информацию из базы данных с расписанием на всю следующую неделю неделю.
* При использовании команды /week бот должен выводить какая на данный момент неделя – верхняя/нижняя.
* При использовании команды /mtuci бот должен выводить ссылку на официальный сайт МТУСИ – https://mtuci.ru/
* При использовании команды /help бот должен выводить краткую информацию о себе, краткую документацию и список команд с их пояснениями.
* При вводе неизвестной команды или неизвестного боту сообщения, бот должен отправлять пользователю сообщение – "Извините, я Вас не понял".
* Структура базы данных может дополняться полями в таблицах, но при этом должна иметь следующую структуру:



2) Создать оконное приложение позволяющее редактировать базу данных с расписанием Вашей группы.

2 Результат выполненной работы

Код bot.py:

import telebot  
from telebot import types  
from config import \*  
import db  
  
bot = telebot.TeleBot(token)  
  
  
@bot.message\_handler(commands=["start"])  
@log  
def cmd\_start(message):  
 markup = types.ReplyKeyboardMarkup(row\_width=1)  
 markup.add(\*[types.KeyboardButton(s) for s in  
 days\_in\_week + ("Расписание на текущую неделю", "Расписание на следующую неделю")])  
 bot.send\_message(message.chat.id, "Здраствуй!", reply\_markup=markup)  
  
  
  
  
  
@bot.message\_handler(commands=["week"])  
@log  
def cmd\_week(message):  
 bot.send\_message(message.chat.id, f"Текущая неделя: {'верхняя' if get\_week\_id() else 'нижняя'}")  
  
  
@bot.message\_handler(commands=["help"])  
@log  
def cmd\_help(message):  
 bot.send\_message(message.chat.id, "Бот расписания группы БВТ2104 МТУСИ")  
 bot.send\_message(message.chat.id,  
 "При нажатии на кнопку с днем недели бот должен выводить информацию из базы данных с расписанием на выбранный день текущей недели\n"  
 "При нажатии на кнопку 'Расписание на текущую неделю' бот должен выводить информацию из базы данных с расписанием на всю текущую неделю\n"  
 "При нажатии на кнопку 'Расписание на следующую неделю' бот должен выводить информацию из базы данных с расписанием на всю следующую неделю неделю\n"  
 "При использовании команды /week бот должен выводить какая на данный момент неделя – верхняя/нижняя\n"  
 "При использовании команды /mtuci бот должен выводить ссылку на официальный сайт МТУСИ – https://mtuci.ru/\n"  
 "При использовании команды /help бот должен выводить краткую информацию о себе, краткую документацию и список команд с их пояснениями.")  
  
  
@bot.message\_handler(commands=["mtuci"])  
@log  
def cmd\_url(message):  
 bot.send\_message(message.chat.id, "https://mtuci.ru/")  
  
  
def send\_week(user\_id, week\_delta):  
 for i, day in enumerate(db.get\_schedule\_for\_week(get\_week\_id() ^ week\_delta)):  
 bot.send\_message(user\_id, days\_in\_week[i])  
 bot.send\_message(user\_id, '\n'.join(day) if day else "Выходной")  
  
  
@bot.message\_handler(regexp=' \*Расписание на текущую неделю \*')  
@log  
def week0(message):  
 send\_week(message.chat.id, 0)  
  
  
@bot.message\_handler(regexp=' \*Расписание на следующую неделю \*')  
@log  
def week1(message):  
 send\_week(message.chat.id, 1)  
  
  
@bot.message\_handler(func=lambda message: message.text.strip().lower() in days\_in\_week\_lower)  
@log  
def get\_day(message):  
 data = db.get\_schedule\_for\_day(get\_week\_id(), days\_in\_week\_lower.index(message.text.strip().lower()))  
 bot.send\_message(message.chat.id, '\n'.join(data) if data else "Отдых")  
  
  
@bot.message\_handler()  
@log  
def unrecognized\_text(message):  
 bot.send\_message(message.chat.id, "Извините, я Вас не понял")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 bot.infinity\_polling()

Код config.py:

from os import environ  
import logging  
from functools import partial  
from datetime import datetime, date, timezone, timedelta  
  
moscow = timezone(timedelta(hours=3))  
now = partial(datetime.now, moscow)  
  
if environ["DEBUG"] == "YES":  
 logging.basicConfig(level=logging.DEBUG,  
 format="%(asctime)s - %(levelname)s : %(message)s")  
else:  
 logging.basicConfig(level=logging.INFO,  
 format="%(asctime)s - %(levelname)s : %(message)s")  
  
  
def debug(f, logger\_normal=logging.debug):  
 def dec(\*args, \*\*kwargs):  
 s = f"Run {f.\_\_name\_\_} with args( {args}{' | ' + str(kwargs) if kwargs else ''} )"  
 try:  
 logger\_normal(s)  
 return f(\*args, \*\*kwargs)  
 except Exception as e:  
 logging.error(f"Error while {s} with message {e}")  
  
 return dec  
  
  
log = partial(debug, logger\_normal=logging.info)  
  
token = environ["TELEGRAM\_BOT\_TOKEN"]  
  
  
def get\_week\_id():  
 time = now().date()  
 return ((time - date(time.year, 9, 6)).days // 7) % 2  
  
  
days\_in\_week = ("Понедельник", "Вторник", "Среда", "Четверг", "Пятница")  
  
days\_in\_week\_lower = tuple(x.lower() for x in days\_in\_week)

Код подключения к базе данных:

from config import \*  
from sqlalchemy import \*  
from sqlalchemy.orm import declarative\_base, Session, relationship  
  
Base = declarative\_base()  
engine = create\_engine(  
 f'postgresql+psycopg2://{environ["POSTGRES\_USER"]}:{environ["POSTGRES\_PASSWORD"]}@{environ["URL\_DB"]}:{environ["PORT\_DB"]}/{environ["POSTGRES\_DB"]}')  
session = Session(engine, future=True)  
session.flush()  
  
  
class Subject(Base):  
 \_\_tablename\_\_ = 'subjects'  
 name = Column(String, primary\_key=True)  
 teachers = relationship('Teacher', cascade='all,delete', back\_populates='subjects')  
 timetables = relationship('TimeTable', cascade='all,delete', back\_populates='subjects')  
  
  
class Teacher(Base):  
 \_\_tablename\_\_ = 'teachers'  
 id = Column(Integer, primary\_key=True)  
 full\_name = Column(String, nullable=False)  
 subject = Column(String, ForeignKey('subjects.name'), nullable=False)  
 subjects = relationship('Subject', cascade='all,delete', back\_populates='teachers')  
  
  
class TimeTable(Base):  
 \_\_tablename\_\_ = 'timetable'  
 id = Column(Integer, primary\_key=True)  
 week = Column(Integer, nullable=False)  
 day = Column(Integer, nullable=False)  
 subject = Column(String, ForeignKey('subjects.name'), nullable=False)  
 room = Column(String, nullable=True)  
 start\_time = Column(Integer, nullable=False)  
 subjects = relationship('Subject', back\_populates='timetables')  
  
 def \_\_str\_\_(*self*):  
 t = str(*self*.start\_time)  
 if len(t) < 4:  
 t = f"0{t[0]}:{t[1:]}"  
 else:  
 t = f"{t[:2]}:{t[2:]}"  
 return f"{*self*.subject} {*self*.room} {t} "  
  
  
def get\_schedule\_for\_week(week\_id):  
 data = session.query(TimeTable, Teacher.full\_name).filter(TimeTable.week == week\_id).join(  
 TimeTable.subjects).join(Subject.teachers).order\_by(TimeTable.day).order\_by(  
 TimeTable.start\_time).all()  
 answer = [[] for i in range(5)]  
 for lesson, teacher\_name in data:  
 answer[lesson.day].append(str(lesson) + teacher\_name)  
 return answer  
  
  
def get\_schedule\_for\_day(week\_id, day\_id):  
 data = session.query(TimeTable, Teacher.full\_name).filter(TimeTable.week == week\_id).filter(  
 TimeTable.day == day\_id).join(TimeTable.subjects).join(Subject.teachers).order\_by(TimeTable.day).order\_by(  
 TimeTable.start\_time).all()  
 return [str(lesson) + teacher\_name for lesson, teacher\_name in data]

2)Код графического интерфейса на PyQt5:

from PyQt5 import QtCore, QtGui  
from PyQt5.QtCore import Qt, QModelIndex  
from PyQt5.QtSql import QSqlRelationalTableModel, QSqlTableModel, QSqlRelation, QSqlRelationalDelegate, QSqlDatabase  
from PyQt5.QtWidgets import (  
 QMessageBox,  
 QApplication,  
 QWidget,  
 QHBoxLayout,  
 QVBoxLayout,  
 QPushButton,  
 QTableView  
)  
import sys  
from os import environ  
from typing import List  
  
  
  
class ImportSqlTableModel(QSqlRelationalTableModel):  
 def \_\_init\_\_(*self*, tablename, hide, default, sort\_id, filter\_s, \*args, \*\*kwargs):  
 super(ImportSqlTableModel, *self*).\_\_init\_\_(\*args, \*\*kwargs)  
 *self*.setTable(tablename)  
 *self*.setEditStrategy(QSqlTableModel.EditStrategy.OnFieldChange)  
 *self*.setFilter(filter\_s)  
  
 *self*.sort\_id = sort\_id  
 *self*.hide = hide  
 *self*.default = default  
 *self*.newRow = None  
 *self*.clearRow()  
 *self*.select()  
  
 def clearRow(*self*):  
 *self*.newRow = [''] \* *self*.columnCount()  
 if *self*.default:  
 *self*.newRow[*self*.default[0]] = *self*.default[1]  
  
 def data(*self*, index, role=Qt.DisplayRole):  
 if role == Qt.DisplayRole or role == Qt.EditRole:  
 if index.row() + 1 == *self*.rowCount(index.parent()):  
 return *self*.newRow[index.column()]  
 return QSqlTableModel.data(*self*, index, role)  
  
 def setData(*self*, index, value, role=Qt.EditRole):  
 if not index.isValid():  
 return False  
 print(role, index.column(), index.row(), value)  
 if role == Qt.EditRole or role == Qt.DisplayRole:  
 if index.row() + 1 == *self*.rowCount(index.parent()):  
 *self*.newRow[index.column()] = value  
 return True  
 return QSqlTableModel.setData(*self*, index, value, role)  
  
 def flags(*self*, index):  
 return Qt.ItemIsSelectable | Qt.ItemIsEnabled | Qt.ItemIsEditable  
  
 def rowCount(*self*, index):  
 return QSqlTableModel.rowCount(*self*, index) + 1  
  
 def addRow(*self*, \*args):  
 r = *self*.record()  
 r.setGenerated('id', False)  
 for i in range(r.count()):  
 r.setValue(r.fieldName(i), *self*.newRow[i])  
 *self*.insertRecord(0, r)  
 *self*.clearRow()  
 *self*.select()  
 *self*.layoutChanged.emit()  
  
 def delRows(*self*, rows: List[QModelIndex]):  
 for row in set(x.row() for x in rows):  
 print("delete", row)  
 *self*.removeRow(row)  
 *self*.select()  
 *self*.layoutChanged.emit()  
  
  
class TimeTable(QWidget):  
 stylebutton = "background-color : palette(window); border: 1px solid palette(dark); border-radius: 0; padding: 5px"  
 stylefocus = "background-color : palette(light); border: 1px solid palette(dark); border-radius: 0; padding: 5px"  
  
 def \_\_init\_\_(*self*, parent=None):  
 super().\_\_init\_\_(parent)  
 *self*.setWindowTitle("TimeTable Editor")  
 *self*.resize(600, 600)  
 *self*.palette().light()  
 *self*.vbox = QVBoxLayout(*self*)  
 *self*.hbox = QHBoxLayout()  
  
 *self*.table\_name = ("Предметы", "Преподаватели", "Понедельник", "Вторник", "Среда", "Четверг", "Пятница")  
 *self*.tables = [("subjects", (), (), 0, ""), ("teachers", (0,), (), 1, "")] + \  
 [("timetable", (0, 2), (2, i), 5, f"day = {i}") for i in range(5)]  
 *self*.tab = 0  
  
 *self*.btn\_tab = [QPushButton(x, *self*) for x in *self*.table\_name]  
 for i, btn in enumerate(*self*.btn\_tab):  
 btn.setStyleSheet(*self*.stylebutton)  
 *self*.hbox.addWidget(btn)  
 btn.clicked.connect(lambda \*args, x=i: *self*.setTab(x))  
 *self*.hbox.setSpacing(0)  
 *self*.btn\_tab[*self*.tab].setStyleSheet(*self*.stylefocus)  
  
 *self*.vbox.addLayout(*self*.hbox)  
 *self*.view = [QTableView() for x, \_, \_, \_, \_ in *self*.tables]  
 *self*.model = [ImportSqlTableModel(\*x) for x in *self*.tables]  
  
 *self*.model[1].setRelation(2, QSqlRelation('subjects', 'name', 'name'))  
 *self*.model[2].setRelation(3, QSqlRelation('subjects', 'name', 'name'))  
  
 for view, model in zip(*self*.view, *self*.model):  
 view.setModel(model)  
 for x in model.hide:  
 view.hideColumn(x)  
 view.sortByColumn(model.sort\_id, Qt.SortOrder.AscendingOrder)  
 view.resizeColumnsToContents()  
 view.setItemDelegate(QSqlRelationalDelegate(view))  
 view.hide()  
 model.select()  
  
 *self*.vbox.addWidget(*self*.view[*self*.tab])  
 *self*.view[*self*.tab].show()  
  
 *self*.btn\_add = QPushButton("+", *self*)  
 *self*.vbox.addWidget(*self*.btn\_add)  
 *self*.btn\_add.clicked.connect(lambda: *self*.model[*self*.tab].addRow())  
  
 *self*.btn\_del = QPushButton("-", *self*)  
 *self*.vbox.addWidget(*self*.btn\_del)  
 *self*.btn\_del.clicked.connect(lambda: *self*.model[*self*.tab].delRows(*self*.view[*self*.tab].selectedIndexes()))  
  
 def setTab(*self*, id):  
 if id != *self*.tab:  
 *self*.btn\_tab[*self*.tab].setStyleSheet(*self*.stylebutton)  
 *self*.view[*self*.tab].hide()  
 *self*.vbox.replaceWidget(*self*.view[*self*.tab], *self*.view[id])  
 *self*.btn\_tab[id].setStyleSheet(*self*.stylefocus)  
 *self*.view[id].show()  
 *self*.tab = id  
  
  
def createConnection():  
 con = QSqlDatabase.addDatabase("QPSQL")  
  
 con.setDatabaseName(environ["POSTGRES\_DB"])  
 con.setHostName(environ["URL\_DB"])  
 con.setPort(int(environ["PORT\_DB"]))  
 con.setUserName(environ["POSTGRES\_USER"])  
 con.setPassword(environ["POSTGRES\_PASSWORD"])  
  
 if not con.open():  
 QMessageBox.critical(  
 None,  
 "QTableView Example - Error!",  
 "Database Error: %s" % con.lastError().databaseText(),  
 )  
 return False  
 return True  
  
  
app = QApplication(sys.argv)  
if not createConnection():  
 sys.exit(1)  
win = TimeTable()  
win.show()  
sys.exit(app.exec\_())

Dokerfile для TimeTable:

FROM ubuntu:20.04  
  
ENV DEBIAN\_FRONTEND=noninteractive  
  
  
RUN *adduser* --quiet --disabled-password qtuser && *usermod* -a -G audio qtuser  
  
  
ENV LIBGL\_ALWAYS\_INDIRECT=1  
  
  
RUN *apt-get* update && *apt-get* install -y python3-pyqt5  
  
RUN *apt-get* install -y python3-pyqt5.qtsql libqt5sql5-psql  
  
WORKDIR /usr/src/app  
  
ENV PYTHONDONTWRITEBYTECODE 1  
ENV PYTHONUNBUFFERED 1  
  
COPY . /usr/src/app/  
  
CMD ["python3", "main.py"]

docker-compose.yml:

version: '3'  
services:  
 database:  
 build: postgres  
 image: postgres\_init  
 env\_file:  
 - .env  
 - secret.env  
 volumes:  
 - database-data:/var/lib/postgresql/data/  
 ports:  
 - ${PORT\_DB}:${PORT\_DB}  
 restart: unless-stopped  
 healthcheck:  
 test: ["CMD-SHELL", "pg\_isready -U ${POSTGRES\_USER} -d ${POSTGRES\_DB}"]  
 interval: 5s  
 timeout: 3s  
 retries: 5  
 start\_period: 10s  
 bot:  
 build: bot  
 image: bot  
 env\_file:  
 - .env  
 - secret.env  
 depends\_on:  
 database:  
 condition: service\_healthy  
 restart: unless-stopped  
 editor:  
 build: TimeTableEditor  
 image: time\_table\_editor  
 env\_file:  
 - .env  
 - secret.env  
 volumes:  
 - /tmp/.X11-unix:/tmp/.X11-unix  
 environment:  
 - DISPLAY=$DISPLAY  
 user:  
 qtuser  
 depends\_on:  
 database:  
 condition: service\_healthy  
volumes:  
 database-data:

**3 Вывод**

В результате проделанной работы мы изучили принципы создания Telegram-ботов, баз-данных, графических интерфейсов.