ГОУ ЯО «Лицей № 86 г. Ярославль»

Творческий проект

Чат-бот для организации заявок на питание

Секция: информационно-телекоммуникационные технологии

Выполнил ученик 8 н класса

ГОУ ЯО Лицей №86

Мутовкин Антон Максимович

Научный руководитель:

Козлова Анна Александровна

Ярославль 2023

Содержание

1. Введение 3
2. Выбор и обоснование проекта 4
3. Составление алгоритма чат-бота 4
4. Обзор и выбор ПО для создания бота 5
5. Результаты тестирования чат-бота 6
6. Вывод 9
Приложение 10

1. Введение

В настоящее время интернет является неотъемлемой частью жизни практически каждого человека. Он дал людям возможность гораздо быстрее и удобнее обмениваться информацией. Процесс развития интернет-технологий является толчком для поиска инновационных решений, с помощью которых появляется возможность повышения удобства и эффективности в том числе и организационного процесса. Это обусловлено возможностью внедрения технологичных решений, настроенных на упрощение некоторых процессов, связанных со сбором и последующим применением информации.

С недавнего времени классные руководители стали вести журналы питания для дальнейшего заказа порций еды на каждого человека из класса, за который они отвечают, в системе “Ладошки”. Это повлекло за собой следующие проблемы:

1. Нехватка времени на собственноручную запись каждого ученика (что в некоторых случаях ведёт к потере времени от урока);

2. Необходимость заменять классных руководителей, которые отсутствуют.

Актуальность темы проектной работы заключается в решении ранее приведённых проблем посредством создания онлайн-сервиса на популярной платформе-мессенджере “Telegram” для самостоятельной записи каждого ученика и просмотра полученных данных учителем.

Цель проекта: создать и ввести в эксплуатацию чат-бот в мессенджере “Telegram”, который позволит обучающимся самостоятельно заказать порцию питания, а классным руководителям получить список питающихся в удобном формате для ввода в систему “Ладошки”.

Задачи проекта:

- Изучить способы и средства для создания “Telegram-ботов”;
- Выявить наиболее подходящие средства для создания “ботов”;
- Составить алгоритм работы “бота”;
- Подготовка необходимого программного обеспечения для создания “бота” и разработки программного кода;
- Провести тестирование готового продукта на наличие ошибок и добавить новые удобные функции.

2. Выбор и обоснование проекта

Данную тему я выбрал в связи с неудобством использования бумажных журналов, желанием сэкономить время классных руководителей, а также моей увлечённостью программированием с раннего возраста.

3. Составление алгоритма чат-бота

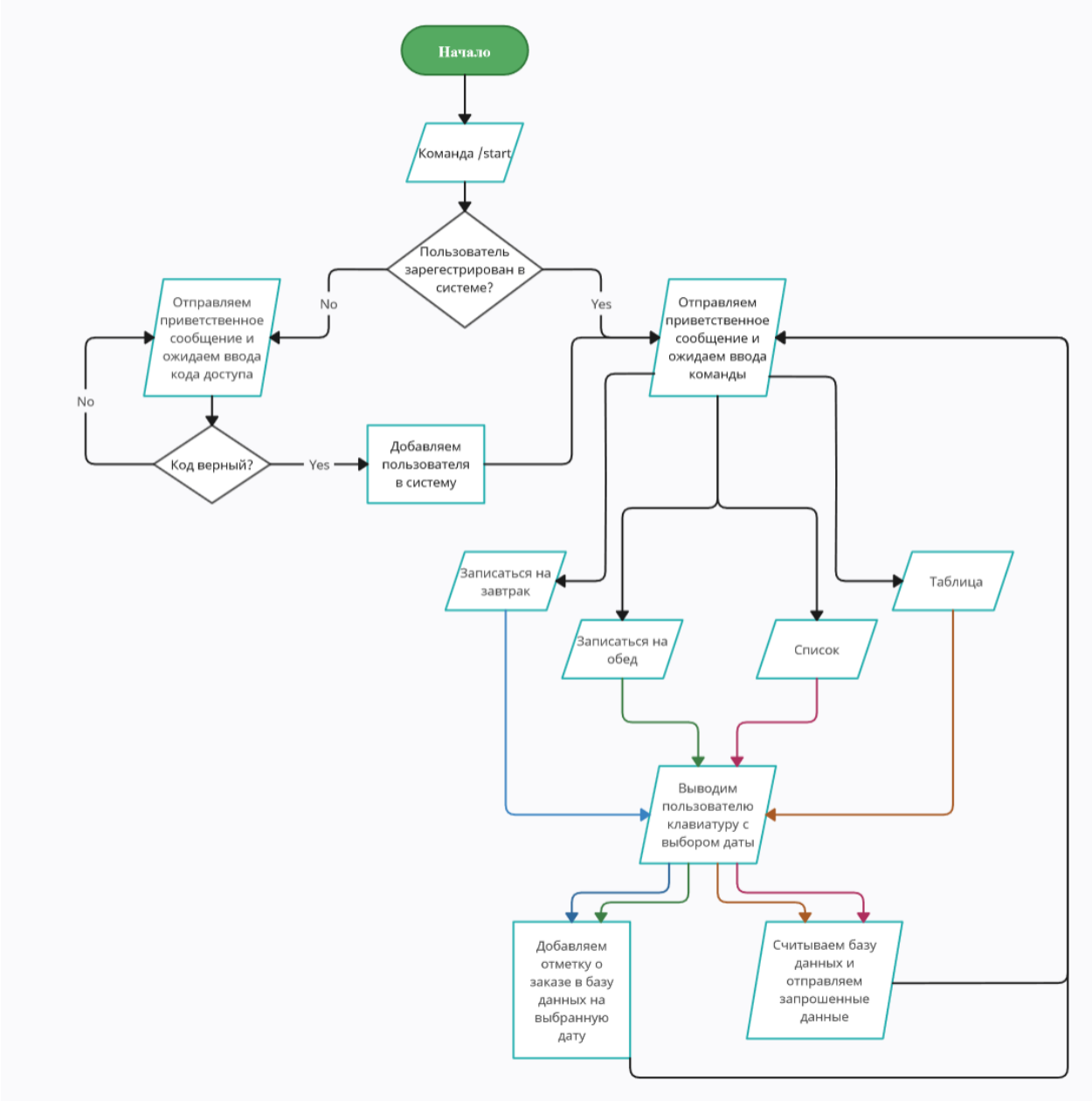


Рисунок 1 – Блок-Схема

Чтобы начать работу с ботом, пользователю необходимо нажать на кнопку “Запустить” (что отправит команду /start за пользователя). Сразу после этого чат-бот проверит, есть ли пользователь в базе данных. Если нет, то программа отправит приветственное сообщение и попросит ввести код доступа (его расскажет ученикам классный руководитель). В случае ввода неверного кода процесс повторится. Если же код верный, то бот попросит ввести фамилию ученика и добавит её в базу данных. После этого ввод кода больше не потребуется.

Если пользователь есть в базе, то бот отправит приветственное сообщение и предложит на выбор 4 команды: 1. Записаться на завтрак; 2. Записаться на обед; 3. Список; 4. Таблица. При выборе любой из команд, пользователю будет предложено выбрать дату.

1. Записаться на завтрак: при вводе этой команды на выбранную дату в базе данных будет создана отметка завтрака.
2. Записаться на обед: идентично первой команде, но отметка создаётся на обед.
3. Список: выведет все отметки с выбранной даты в текстовом формате.
4. Таблица: идентично третьей команде, но информация выводится в графическом формате с помощью таблицы.

4. Обзор и выбор ПО для создания бота

Первым делом необходимо определить язык программирования, на котором будет создан чат-бот. Самые популярные языки программирования (далее ЯП) для создания чат-ботов - это Python и PHP. Я остановился на ЯП Python, из-за его легкого в понимании синтаксиса и наличия огромной базы пользователей, которая сможет помочь с любой проблемой. Также в 2022 году Python являлся самым популярным языком среди программистов и до сих пор остаётся на этом почётном месте.

Следующим шагом следует выбрать основную библиотеку для написания бота. Самыми известными являются aiogram и PyTelegramBotAPI. У каждого из этих вариантов есть преимущества и недостатки, но я выбрал библиотеку aiogram из-за удобных функций, которые сокращают написание кода, а также поддержки асинхронности, что является очень важным аспектом для чат-ботов, ориентированных на большую аудиторию.

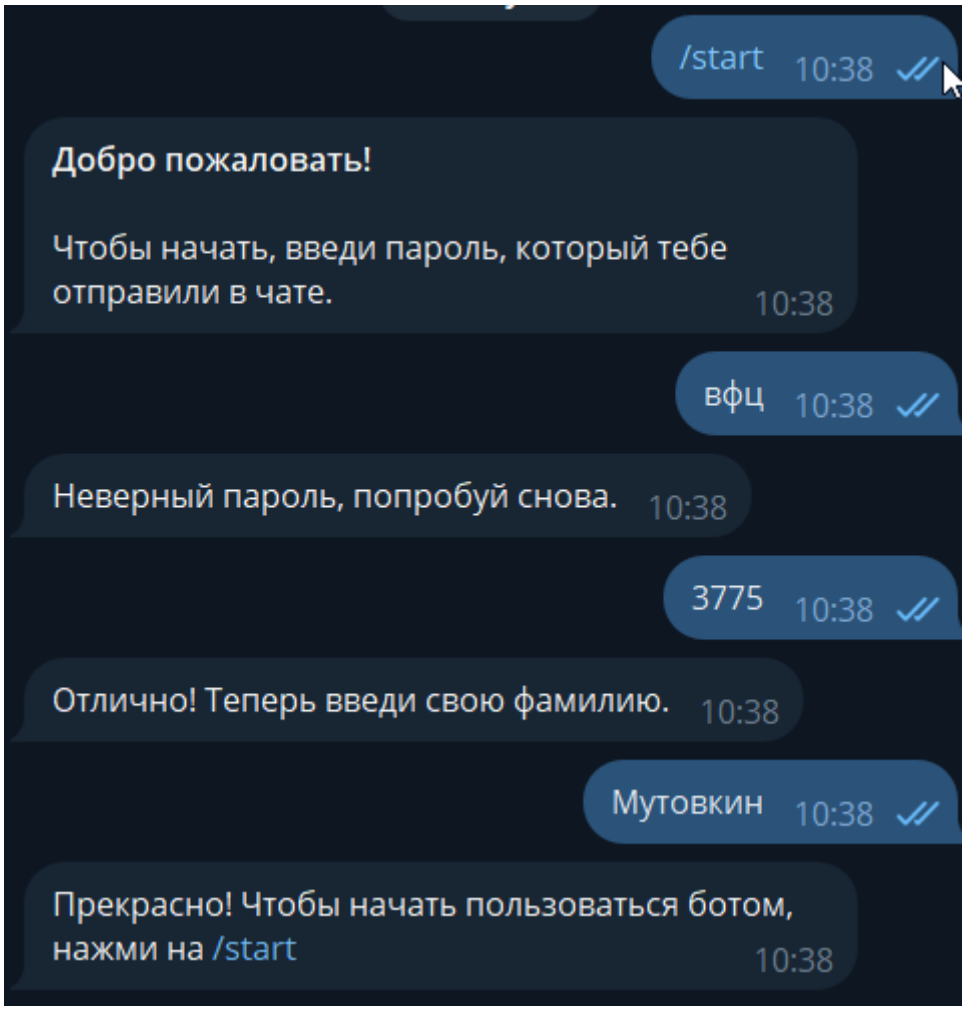
И последним делом необходимо выбрать среду разработки (далее IDE) программного кода. Самые используемые IDE – это PyCharm и VSCode. Мой выбор пал на последний вариант, потому что он использует мало ресурсов по сравнению с PyCharm, хотя и обладает идентичным функционалом.

5. Результаты тестирования чат-бота

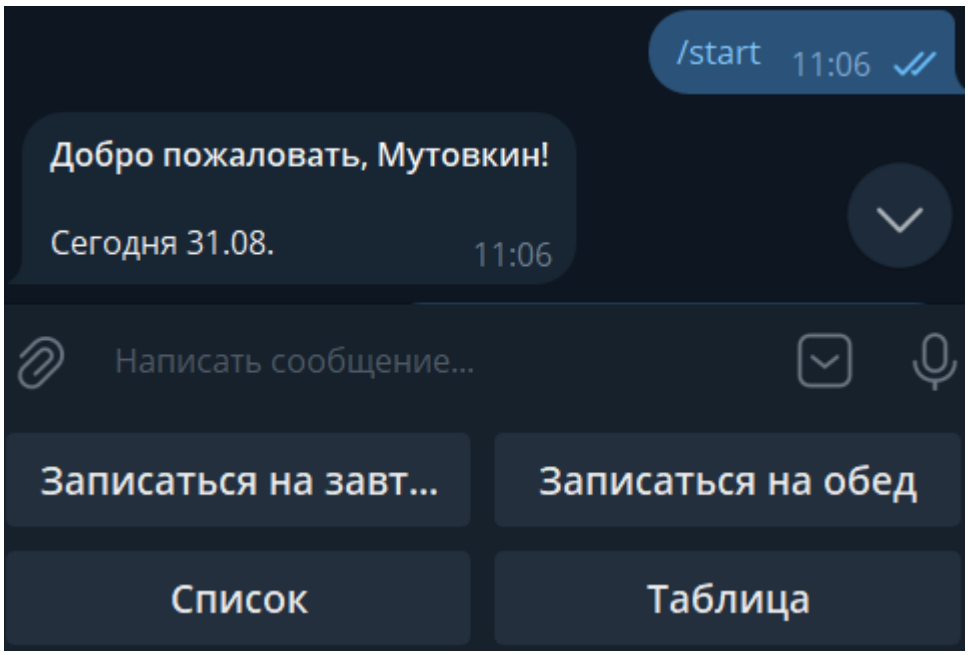
Готовый чат-бот был введен в эксплуатацию в конце августа. В качестве хостинга (сервера для запуска программы) был выбран сервис Timeweb Cloud, в котором можно за небольшие деньги арендовать удалённый сервер.

Тестирование проводилось в течение месяца, и за это время не было обнаружено ни одной ошибки. Также была введена новая функция, а именно команда для отмены выбора даты, потому что иногда происходят случайные нажатия и выйти из этого меню нельзя.

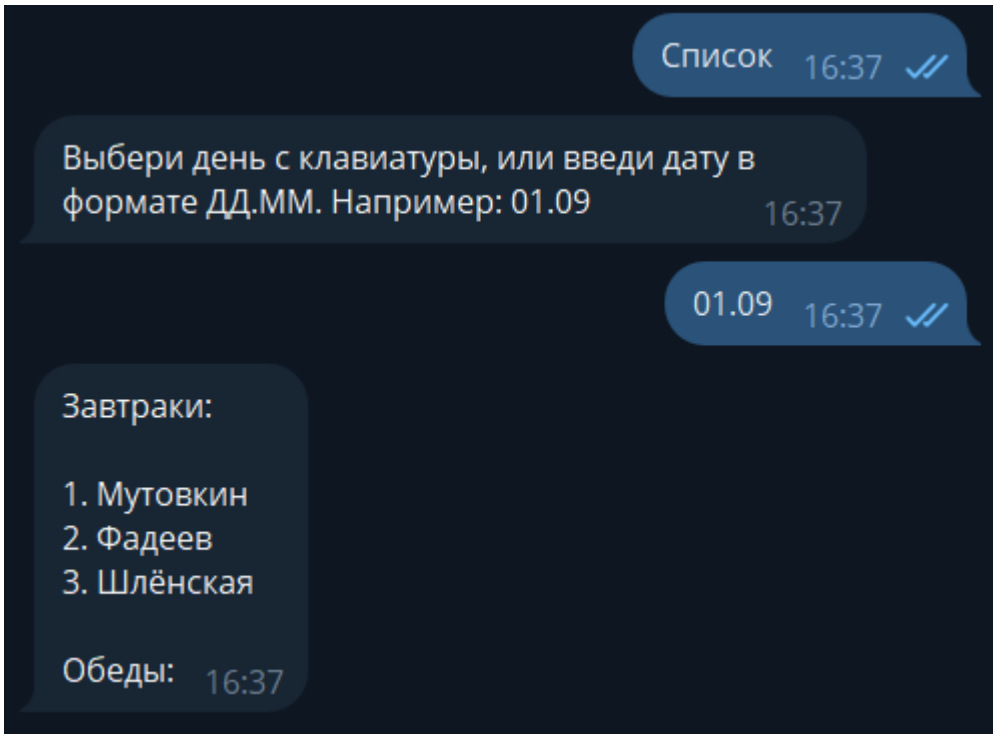
Пример работы бота:



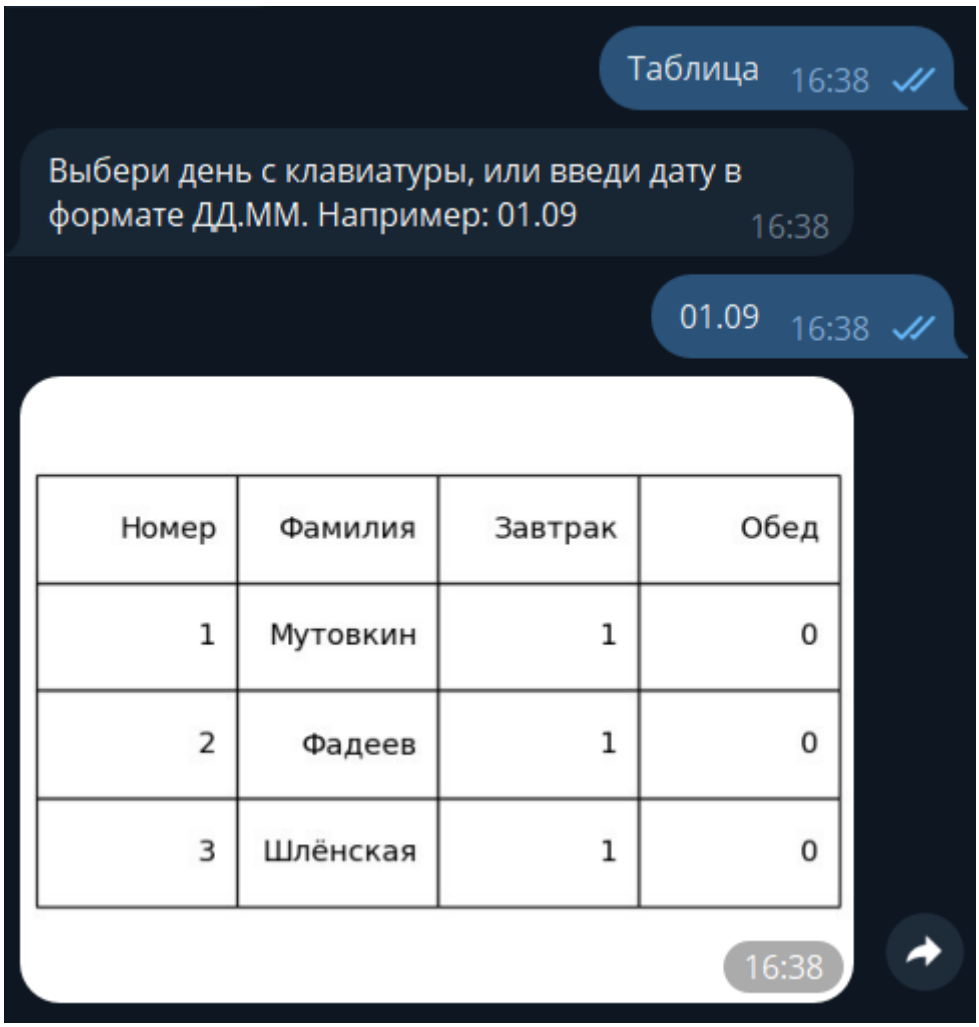
Начало работы с ботом



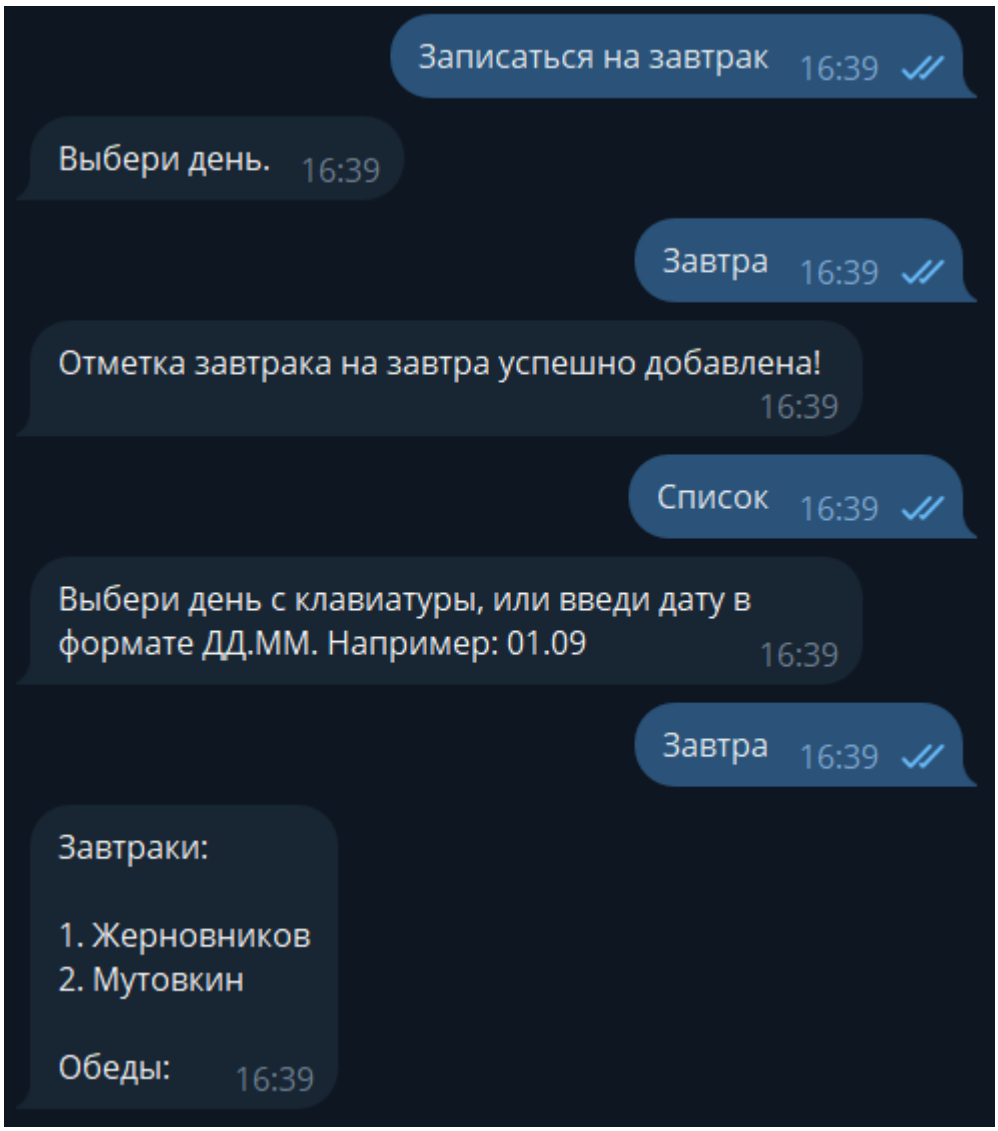
Команда /start после авторизации



Вывод данных списком



Вывод данных таблицей



Пример работы команды записи на завтрак (идентичен с обедом)

6. Вывод

В результате работы был создан и введён в эксплуатацию чат-бот в мессенджере “Telegram” на языке программирования Python. На протяжении месяца проводилось тестирование продукта, которое не выявило проблем. В дальнейшем планируется расширение функционала. Например, возможность просматривать домашнее задание и расписание уроков/звонков.

Приложение

```python
from aiogram import Bot, Dispatcher, executor, types
import stimp as st
import table
from datetime import datetime, timedelta
from aiogram.contrib.fsm_storage.memory import MemoryStorage
from aiogram.dispatcher import FSMContext
from aiogram.dispatcher.filters.state import State, StatesGroup
import re
import os

TOKEN = st.load("settings.json", "token")
allowed: list = st.load("settings.json", "allowed")
dates: dict = st.load("settings.json", "dates")
lasts: dict = st.load("settings.json", "lasts")

storage = MemoryStorage()
bot = Bot(TOKEN)
dp = Dispatcher(bot, storage=storage)
```

Импортирование нужных библиотек и задание переменных

```python
class States(StatesGroup):
    select_date = State()

    enter_pass = State()
    enter_last = State()
```

Объявление класса для работы с состояниями

```python
def get_date(param):
    if param == "n":
        day = str(datetime.now().day)
        month = str(datetime.now().month)

        if len(day) == 1:
            strday = f"0{day}"
        else:
            strday = day

        if len(month) == 1:
            strmonth = f"0{month}"
        else:
            strmonth = month

        return f"{strday}.{strmonth}"
    elif param == "t":
        day = str((datetime.now()+timedelta(days=1)).day)
        month = str((datetime.now()+timedelta(days=1)).month)

        if len(day) == 1:
            strday = f"0{day}"
        else:
            strday = day

        if len(month) == 1:
            strmonth = f"0{month}"
        else:
            strmonth = month

        return f"{strday}.{strmonth}"
    elif param == "y":
        day = str((datetime.now()-timedelta(days=1)).day)
        month = str((datetime.now()-timedelta(days=1)).month)

        if len(day) == 1:
            strday = f"0{day}"
        else:
            strday = day

        if len(month) == 1:
            strmonth = f"0{month}"
        else:
            strmonth = month

        return f"{strday}.{strmonth}"
```

Реализация функции для получения даты

```python
def breakfast(us_id, date):
    try:
        if dates[date] != None:
            try:
                if dates[date][str(us_id)] != None:
                    if dates[date][str(us_id)][0] != 1:
                        dates[date][str(us_id)][0] = 1
                        st.dump_value("settings.json", "dates", dates)
                        return True
                    else:
                        return False
            except:
                dates[date][str(us_id)] = [1, 0]
                st.dump_value("settings.json", "dates", dates)
                return True
    except:
        dates[date] = {str(us_id): [1, 0]}
        st.dump_value("settings.json", "dates", dates)
        return True

def lunch(us_id, date):
    try:
        if dates[date] != None:
            try:
                if dates[date][str(us_id)] != None:
                    if dates[date][str(us_id)][1] != 1:
                        dates[date][str(us_id)][1] = 1
                        st.dump_value("settings.json", "dates", dates)
                        return True
                    else:
                        return False
            except:
                dates[date][str(us_id)] = [0, 1]
                st.dump_value("settings.json", "dates", dates)
                return True
    except:
        dates[date] = {str(us_id): [0, 1]}
        st.dump_value("settings.json", "dates", dates)
        return True
```

Реализация функций для записи на завтрак и обед

```python
def create_list(date):
    breakfasts = ""
    lunches = ""
    try:
        if dates[date] != None:
            ids = 1
            for us in dates[date]:
                if dates[date][us][0] == 1:
                    breakfasts += f"{ids}. {lasts[us]}\n"
                if dates[date][us][1] == 1:
                    lunches += f"{ids}. {lasts[us]}\n"

                ids += 1

            return f"Завтраки:\n\n{breakfasts}\nОбеды:\n\n{lunches}"
    except:
        return "Никто не записался на эту дату."

def create_table(date):
    tab = [["Номер", "Фамилия", "Завтрак", "Обед"]]
    try:
        if dates[date] != None:
            ids = 1
            for us in dates[date]:
                t = [ids, lasts[us], dates[date][us][0], dates[date][us][1]]
                tab.append(t)
                ids += 1
            table.create_tab(tab)
            return True
    except:
        return "Никто не записался на эту дату."
```

Реализация функций для получения данных списком и таблицей

```python
@dp.message_handler(commands="start")
async def start(m: types.Message, state: FSMContext):
    if m.from_id not in allowed:
        await m.answer("*Добро пожаловать!*\n\nЧтобы начать, введи пароль, который тебе отправили в чате.", parse_mode="Markdown")
        await state.set_state(States.enter_pass.state)
    else:
        await m.answer(f"*Добро пожаловать, {lasts[str(m.from_id)]}!*\n\nСегодня {get_date('n')}.", reply_markup=menu_kb, parse_mode="Markdown")

@dp.message_handler(state=States.enter_pass)
async def enter_pass(m: types.Message, state: FSMContext):
    if m.text == "3775":
        await m.answer("Отлично! Теперь введи свою фамилию.")
        await state.set_state(States.enter_last.state)
    else:
        await m.answer("Неверный пароль, попробуй снова.")

@dp.message_handler(state=States.enter_last)
async def last_name(m: types.Message, state: FSMContext):
    await state.finish()
    lasts[str(m.from_id)] = m.text
    st.dump_value("settings.json", "lasts", lasts)
    allowed.append(m.from_id)
    st.dump_value("settings.json", "allowed", allowed)

    await m.answer("Прекрасно! Чтобы начать пользоваться ботом, нажми на /start")
```

Функции для авторизации пользователя

```python
@dp.message_handler()
async def buttons(m: types.Message, state: FSMContext):
    if m.text == "Записаться на завтрак":
        date_kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
        date_kb.add(types.KeyboardButton("Сегодня"), types.KeyboardButton("Завтра"))

        await m.answer("Выбери день.", reply_markup=date_kb)
        await state.set_state(States.select_date.state)
        await state.update_data(target="breakfast")

    elif m.text == "Записаться на обед":
        date_kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
        date_kb.add(types.KeyboardButton("Сегодня"), types.KeyboardButton("Завтра"))

        await m.answer("Выбери день.", reply_markup=date_kb)
        await state.set_state(States.select_date.state)
        await state.update_data(target="lunch")

    elif m.text == "Список":
        date_kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
        date_kb.add(types.KeyboardButton("Вчера"), types.KeyboardButton("Сегодня"), types.KeyboardButton("Завтра"))

        await m.answer("Выбери день с клавиатуры, или введи дату в формате ДД.ММ. Например: 01.09", reply_markup=date_kb)
        await state.set_state(States.select_date.state)
        await state.update_data(target="list")

    elif m.text == "Таблица":
        date_kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
        date_kb.add(types.KeyboardButton("Вчера"), types.KeyboardButton("Сегодня"), types.KeyboardButton("Завтра"))

        await m.answer("Выбери день с клавиатуры, или введи дату в формате ДД.ММ. Например: 01.09", reply_markup=date_kb)
        await state.set_state(States.select_date.state)
        await state.update_data(target="table")
```

Функция для распознания нажатий кнопок пользователем

```python
@dp.message_handler(state=States.select_date)
async def date(m: types.Message, state: FSMContext):
    data = await state.get_data()
    target = data['target']

    if target == "breakfast":
        if m.text == "Сегодня":
            res = breakfast(m.from_id, get_date('n'))
            if res:
                await m.answer("Отметка завтрака на сегодня успешно добавлена!", reply_markup=menu_kb)
                await state.finish()
            else:
                await m.answer("Ты уже записался на сегодняшний завтрак, не балуй!", reply_markup=menu_kb)
                await state.finish()
        elif m.text == "Завтра":
            res = breakfast(m.from_id, get_date('t'))
            if res:
                await m.answer("Отметка завтрака на завтра успешно добавлена!", reply_markup=menu_kb)
                await state.finish()
            else:
                await m.answer("Ты уже записался на завтрашний завтрак, не балуй!", reply_markup=menu_kb)
                await state.finish()
        else:
            await m.answer("Не понял тебя! Нажми на кнопку.")

    elif target == "lunch":
        if m.text == "Сегодня":
            res = lunch(m.from_id, get_date('n'))
            if res:
                await m.answer("Отметка обеда на сегодня успешно добавлена!", reply_markup=menu_kb)
                await state.finish()
            else:
                await m.answer("Ты уже записался на сегодняшний обед, не балуй!", reply_markup=menu_kb)
                await state.finish()
        elif m.text == "Завтра":
            res = lunch(m.from_id, get_date('t'))
            if res:
                await m.answer("Отметка обеда на завтра успешно добавлена!", reply_markup=menu_kb)
                await state.finish()
            else:
                await m.answer("Ты уже записался на завтрашний обед, не балуй!", reply_markup=menu_kb)
                await state.finish()
        else:
            await m.answer("Не понял тебя! Нажми на кнопку.")
```

Функция для записи пользователя на завтрак/обед через интерфейс бота

```
254     elif target == "list":
255         if m.text == "Вчера":
256             res = create_list(get_date("y"))
257             await m.answer(res, reply_markup=menu_kb)
258             await state.finish()
259         elif m.text == "Сегодня":
260             res = create_list(get_date("n"))
261             await m.answer(res, reply_markup=menu_kb)
262             await state.finish()
263         elif m.text == "Завтра":
264             res = create_list(get_date("t"))
265             await m.answer(res, reply_markup=menu_kb)
266             await state.finish()
267         else:
268             dat = re.search(r"\d{2}[.]\d{2}", m.text)
269             if dat:
270                 d = dat[0]
271             else:
272                 await m.answer("Дата введена неправильно! Попробуй снова.")
273                 return
274 
275             res = create_list(d)
276             await m.answer(res, reply_markup=menu_kb)
277             await state.finish()
```

Функция для вывода данных списком

```
279     elif target == "table":
280         if m.text == "Вчера":
281             res = create_table(get_date("y"))
282             if res == True:
283                 with open("table.png", 'rb') as plot:
284                     await m.answer_photo(plot, reply_markup=menu_kb)
285                 os.remove("table.png")
286             else:
287                 await m.answer(res, reply_markup=menu_kb)
288             await state.finish()
289         elif m.text == "Сегодня":
290             res = create_table(get_date("n"))
291             if res == True:
292                 with open("table.png", 'rb') as plot:
293                     await m.answer_photo(plot, reply_markup=menu_kb)
294                 os.remove("table.png")
295             else:
296                 await m.answer(res, reply_markup=menu_kb)
297             await state.finish()
298         elif m.text == "Завтра":
299             res = create_table(get_date("t"))
300             if res == True:
301                 with open("table.png", 'rb') as plot:
302                     await m.answer_photo(plot, reply_markup=menu_kb)
303                 os.remove("table.png")
304             else:
305                 await m.answer(res, reply_markup=menu_kb)
306             await state.finish()
307         else:
308             dat = re.search(r"\d{2}[.]\d{2}", m.text)
309             if dat:
310                 d = dat[0]
311             else:
312                 await m.answer("Дата введена неправильно! Попробуй снова.")
313                 return
314 
315             res = create_table(d)
316             if res == True:
317                 with open("table.png", 'rb') as plot:
318                     await m.answer_photo(plot, reply_markup=menu_kb)
319                 os.remove("table.png")
320             else:
321                 await m.answer(res, reply_markup=menu_kb)
322             await state.finish()
```

Функция для вывода данных таблицей

```
324 if __name__ == "__main__":
325     executor.start_polling(dp, skip_updates=True)
```

Запуск бота