

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
им. Н.Э. Баумана

Факультет “Информатика и системы управления”  
Кафедра ИУ5 “Системы обработки информации и управления”



Дисциплина “Парадигмы и конструкции языков программирования”

Отчет по домашней работе  
“Телеграмм-бот”

**Выполнил:**  
Студент группы ИУ5-33Б  
Просвирякова Д. С.

**Преподаватель:**  
Гапанюк Ю. Е.

Москва 2025

## Задание:

### Задание:

1. Выберите язык программирования (который Вы ранее не изучали) и (1) напишите по нему реферат с примерами кода или (2) реализуйте на нем небольшой проект (с детальным текстовым описанием).
2. Реферат (проект) может быть посвящен отдельному аспекту (аспектам) языка или содержать решение какой-либо задачи на этом языке.
3. Необходимо установить на свой компьютер компилятор (интерпретатор, транспилятор) этого языка и произвольную среду разработки.
4. В случае написания реферата необходимо разработать и откомпилировать примеры кода (или модифицировать стандартные примеры).
5. В случае создания проекта необходимо детально комментировать код.
6. При написании реферата (создании проекта) необходимо изучить и корректно использовать особенности парадигмы языка и основных конструкций данного языка.
7. Приветствуется написание черновика статьи по результатам выполнения ДЗ. Черновик статьи может быть подготовлен группой студентов, которые исследовали один и тот же аспект в нескольких языках или решили одинаковую задачу на нескольких языках.

## Листинг кода

```
# токен api для телеграм бота
API_TOKEN = "8201918414:AAEAYeft2fkt9s2aYTWeYmrtQ0c_XZfZUJM"

# словарь с тематическим словарем
# каждая тема содержит английские слова с переводом и примерами
VOCABULARY_BY_TOPIC = {
    # тема "еда"
    "Еда": {
        # слово apple с переводом "яблоко" и примерами
        "apple": {"ru": "яблоко", "examples": ["I eat an apple", "Red apple"]},
        # слово bread с переводом "хлеб" и примерами
        "bread": {"ru": "хлеб", "examples": ["Fresh bread", "Slice of bread"]},
        # слово water с переводом "вода" и примерами
        "water": {"ru": "вода", "examples": ["Glass of water", "Drink water"]},
        # слово milk с переводом "молоко" и примерами
        "milk": {"ru": "молоко", "examples": ["Glass of milk", "Milk is white"]},
        # слово cheese с переводом "сыр" и примерами
        "cheese": {"ru": "сыр", "examples": ["Piece of cheese", "Yellow cheese"]},
    },
    # тема "путешествия"
    "Путешествия": {
        # слово airport с переводом "аэропорт" и примерами
        "airport": {"ru": "аэропорт", "examples": ["Go to airport", "Airport terminal"]},
        # слово hotel с переводом "отель" и примерами
        "hotel": {"ru": "отель", "examples": ["Book a hotel", "Hotel room"]},
        # слово ticket с переводом "билет" и примерами
        "ticket": {"ru": "билет", "examples": ["Buy ticket", "Ticket price"]},
        # слово passport с переводом "паспорт" и примерами
        "passport": {"ru": "паспорт", "examples": ["My passport", "Show passport"]},
        # слово luggage с переводом "багаж" и примерами
        "luggage": {"ru": "багаж", "examples": ["Check luggage", "Heavy luggage"]},
    },
}
```

```

# тема "дом"
"Дом": {
    # слово house с переводом "дом" и примерами
    "house": {"ru": "дом", "examples": ["Big house", "My house"]},
    # слово room с переводом "комната" и примерами
    "room": {"ru": "комната", "examples": ["Clean room", "Living room"]},
    # слово kitchen с переводом "кухня" и примерами
    "kitchen": {"ru": "кухня", "examples": ["Kitchen table", "Cook in kitchen"]},
    # слово bathroom с переводом "ванная" и примерами
    "bathroom": {"ru": "ванная", "examples": ["Bathroom mirror", "Go to
bathroom"]},
    # слово bedroom с переводом "спальня" и примерами
    "bedroom": {"ru": "спальня", "examples": ["Bedroom window", "Sleep in
bedroom"]},
},
# тема "работа"
"Работа": {
    # слово office с переводом "офис" и примерами
    "office": {"ru": "офис", "examples": ["Office building", "Work in office"]},
    # слово meeting с переводом "встреча" и примерами
    "meeting": {"ru": "встреча", "examples": ["Business meeting", "Schedule a
meeting"]},
    # слово computer с переводом "компьютер" и примерами
    "computer": {"ru": "компьютер", "examples": ["Computer screen", "Use
computer"]},
    # слово salary с переводом "зарплата" и примерами
    "salary": {"ru": "зарплата", "examples": ["Monthly salary", "Good salary"]},
    # слово colleague с переводом "коллега" и примерами
    "colleague": {"ru": "коллега", "examples": ["Friendly colleague", "Talk to
colleague"]},
},
# тема "образование"
"Образование": {
    # слово school с переводом "школа" и примерами
    "school": {"ru": "школа", "examples": ["Go to school", "School teacher"]},
    # слово university с переводом "университет" и примерами
    "university": {"ru": "университет", "examples": ["University student", "Study
at university"]},
    # слово teacher с переводом "учитель" и примерами
    "teacher": {"ru": "учитель", "examples": ["Math teacher", "Respect teacher"]},
    # слово student с переводом "студент" и примерами
    "student": {"ru": "студент", "examples": ["Good student", "Student life"]},
    # слово book с переводом "книга" и примерами
    "book": {"ru": "книга", "examples": ["Interesting book", "Read book"]},
}
}

# словарь с уроками грамматики
# каждый урок содержит подробное описание грамматической темы
GRAMMAR_LESSONS = {
    # урок по present simple
    "Present Simple": {
        # название урока

```

```
"name": "Present Simple (Настоящее простое время)",
# краткое описание
"description": "Используется для регулярных действий, привычек и общих фактов",
# основные правила
"rules": [
    " I/You/We/They + глагол без изменений",
    " He/She/It + глагол + s/es",
    " Отрицание: don't/doesn't + глагол",
    " Вопрос: Do/Does + подлежащее + глагол"
],
# примеры использования
"examples": [
    "I work every day. → Я работаю каждый день.",
    "She works in an office. → Она работает в офисе.",
    "They don't like coffee. → Они не любят кофе.",
    "Does he speak English? → Он говорит по-английски?"
],
# использование времени
"usage": [
    "Регулярные действия и привычки",
    "Общие факты и истины",
    "Расписания и программы",
    "Действия по графику"
],
# маркеры времени
"time_markers": [
    "always, usually, often",
    "sometimes, never",
    "every day/week/month",
    "on Mondays, in the morning"
]
},
# урок по past simple
"Past Simple": {
    # название урока
    "name": "Past Simple (Прошедшее простое время)",
    # краткое описание
    "description": "Используется для завершенных действий в прошлом",
    # основные правила
    "rules": [
        " Правильные глаголы: глагол + ed",
        " Неправильные глаголы: 2-я форма глагола",
        " Отрицание: didn't + глагол (1-я форма)",
        " Вопрос: Did + подлежащее + глагол (1-я форма)"
    ],
    # примеры использования
    "examples": [
        "I worked yesterday. → Я работал вчера.",
        "She went to school. → Она пошла в школу.",
        "They didn't visit us. → Они не посетили нас.",
        "Did you see that film? → Ты видел этот фильм?"
    ]
},
```

```
# использование времени
"usage": [
    "Завершенные действия в прошлом",
    "Последовательность действий",
    "Привычки в прошлом",
    "Конкретное время в прошлом"
],
# маркеры времени
"time_markers": [
    "yesterday, last week",
    "in 2020, 5 years ago",
    "when I was young",
    "at 5 o'clock yesterday"
]
},
# урок по future simple
"Future Simple": {
    # название урока
    "name": "Future Simple (Будущее простое время)",
    # краткое описание
    "description": "Используется для действий в будущем, решений и предсказаний",
    # основные правила
    "rules": [
        " Утверждение: will + глагол",
        " Отрицание: won't + глагол",
        " Вопрос: Will + подлежащее + глагол",
        " Сокращения: I'll, you'll, he'll"
    ],
    # примеры использования
    "examples": [
        "I will help you. → Я помогу тебе.",
        "She will be a doctor. → Она будет врачом.",
        "They won't come tomorrow. → Они не придут завтра.",
        "Will you marry me? → Ты выйдешь за меня?"
    ],
    # использование времени
    "usage": [
        "Спонтанные решения",
        "Предсказания без доказательств",
        "Обещания и предложения",
        "Действия в будущем"
    ],
    # маркеры времени
    "time_markers": [
        "tomorrow, next week",
        "in the future, soon",
        "in 2030, in 5 years",
        "later, tonight"
    ]
},
# урок по present continuous
"Present Continuous": {
    # название урока
```

```
"name": "Present Continuous (Настоящее продолженное время)",  
    # краткое описание  
    "description": "Используется для действий, происходящих прямо сейчас или в  
ближайшем будущем",  
    # основные правила  
    "rules": [  
        " Утверждение: am/is/are + глагол + ing",  
        " Отрицание: am not/isn't/aren't + глагол + ing",  
        " Вопрос: Am/Is/Are + подлежащее + глагол + ing"  
    ],  
    # примеры использования  
    "examples": [  
        "I am studying now. → Я учусь сейчас.",  
        "He is watching TV. → Он смотрит телевизор.",  
        "They aren't working today. → Они не работают сегодня.",  
        "Are you listening to me? → Ты меня слушаешь?"  
    ],  
    # использование времени  
    "usage": [  
        "Действия в момент речи",  
        "Временные ситуации",  
        "Изменяющиеся ситуации",  
        "Запланированные будущие действия"  
    ],  
    # маркеры времени  
    "time_markers": [  
        "now, at the moment",  
        "currently, right now",  
        "today, this week",  
        "Look!, Listen!"  
    ]  
,  
    # урок по артиклям  
    "Артикли": {  
        # название урока  
        "name": "Артикли (a/an/the)",  
        # краткое описание  
        "description": "Определенные и неопределенные артикли в английском языке",  
        # основные правила  
        "rules": [  
            " a – перед согласным звуком: a book, a university",  
            " an – перед гласным звуком: an apple, an hour",  
            " the – определенный артикль: the sun, the book I read",  
            " Без артикля – общие понятия: love, water, English"  
        ],  
        # примеры использования  
        "examples": [  
            "I have a book. → У меня есть книга.",  
            "She is an engineer. → Она инженер.",  
            "The sun is bright. → Солнце яркое.",  
            "I speak English. → Я говорю по-английски."  
        ],  
        # использование артиклей
```

```
"usage": [
    "a/an – один из многих (впервые упоминается)",
    "the – конкретный предмет (уже известен)",
    "– – неисчисляемые существительные",
    "– – названия стран, городов, языков"
],
# исключения из правил
"exceptions": [
    "a university (согласный звук [j])",
    "an hour (гласный звук [аʊ])",
    "the USA (сокращения стран)",
    "the Netherlands (страны во мн. числе)"
],
# особенности использования
"features": [
    "a/an = один/какой-то",
    "the = конкретный/уже известный",
    "a/an + исчисляемые существительные",
    "the + уникальные предметы (the sun, the moon)"
]
},
# урок по предлогам
"Предлоги": {
    # название урока
    "name": "Предлоги времени и места",
    # краткое описание
    "description": "Предлоги времени in/on/at и предлоги места",
    # основные правила
    "rules": [
        " at – точное время: at 5 o'clock, at night",
        " on – дни и даты: on Monday, on 1st May",
        " in – месяцы, годы, периоды: in June, in 2024",
        " in – внутри: in the room, in the box",
        " on – на поверхности: on the table, on the wall",
        " at – у конкретного места: at home, at school"
    ],
    # примеры использования
    "examples": [
        "I get up at 7 AM. → Я встаю в 7 утра.",
        "We meet on Monday. → Мы встречаемся в понедельник.",
        "She was born in 1990. → Она родилась в 1990 году.",
        "The book is on the table. → Книга на столе.",
        "He lives in London. → Он живет в Лондоне.",
        "She is at the park. → Она в парке."
    ],
    # использование предлогов
    "usage": [
        "at + время: at 3 PM, at midnight",
        "on + дни: on Friday, on my birthday",
        "in + периоды: in summer, in the morning",
        "in для больших пространств, at для точек",
        "on для поверхностей и улиц"
    ]
},
```

```
# часто используемые предлоги
"common_prepositions": [
    "in: in the car, in the city, in the world",
    "on: on the bus, on the street, on the floor",
    "at: at work, at the door, at the station",
    "to: go to school, come to me, return to home",
    "from: from Russia, from 9 to 5, from my friend"
],
},
# урок по модальным глаголам
"Модальные глаголы": {
    # название урока
    "name": "Модальные глаголы",
    # краткое описание
    "description": "Глаголы, выражающие возможность, необходимость, разрешение",
    # основные правила
    "rules": [
        " can/could – умение, возможность",
        " may/might – разрешение, вероятность",
        " must – обязанность, необходимость",
        " should – совет, рекомендация",
        " will/would – будущее, вежливые просьбы",
        " Нет –s в 3-м лице: He can swim (NOT cans)",
        " Вопросы без do/does: Can you swim? (NOT Do you can swim?)"
    ],
    # примеры использования
    "examples": [
        "I can swim. → Я умею плавать.",
        "You must study. → Ты должен учиться.",
        "She should rest. → Ей следует отдохнуть.",
        "May I come in? → Можно войти?",
        "Could you help me? → Не могли бы вы помочь мне?",
        "It might rain tomorrow. → Возможно, завтра будет дождь."
    ],
    # использование модальных глаголов
    "usage": [
        "can – физическая возможность (I can speak English)",
        "could – вежливая просьба или прошлая возможность",
        "must – сильная необходимость (You must wear a seatbelt)",
        "should – рекомендация (You should see a doctor)",
        "may – формальное разрешение (May I leave the room?)",
        "might – небольшая вероятность (It might snow)"
    ],
    # особенности модальных глаголов
    "features": [
        "Не добавляют –s в 3 лице (He can, NOT He cans)",
        "За ними глагол без to (I can swim, NOT I can to swim)",
        "Не используют вспомогательные глаголы в вопросах",
        "Имеют одинаковую форму для всех лиц",
        "Не имеют инфинитива и причастий"
    ],
    # отрицательные формы
    "negation": [

```

```
        "can → can't/cannot",
        "could → couldn't",
        "must → mustn't (запрет)",
        "should → shouldn't",
        "may → may not",
        "might → might not"
    ],
},
}

# импорт необходимых модулей
import random
from datetime import datetime
# импорт словаря тем из конфигурации
from config import VOCABULARY_BY_TOPIC

# создание словарей для хранения данных пользователей
# словарь для хранения словарей пользователей
user_vocabulary = {}

# словарь для хранения прогресса пользователей
user_progress = {}

# словарь для хранения настроек пользователей
user_settings = {}

# функция инициализации данных пользователя
def init_user_data(user_id: int) -> bool:
    # проверка, есть ли уже данные пользователя
    if user_id not in user_vocabulary:
        # создание пустого списка для словаря пользователя
        user_vocabulary[user_id] = []
        # создание словаря для прогресса пользователя
        user_progress[user_id] = {
            # количество выученных слов
            "words_learned": 0,
            # количество выполненных упражнений
            "exercises_completed": 0,
            # текущая серия правильных ответов
            "current_streak": 0,
            # точность ответов в процентах
            "accuracy": 0.0,
            # общее количество попыток
            "total_attempts": 0,
            # количество правильных попыток
            "correct_attempts": 0,
            # время последней активности в формате строки iso
            "last_active": datetime.now().isoformat()
        }
        # создание словаря для настроек пользователя
        user_settings[user_id] = {
            # текущая тема для изучения
            "current_topic": "Еда",
            # ежедневная цель по количеству слов
            "daily_goal": 5,
            # уровень сложности
        }
    return True
```

```

        "difficulty": "средняя"
    }

    # создание списка базовых слов для нового пользователя
    basic_words = [
        {"word": "hello", "translation": "привет", "learned": False,
    "review_count": 0, "topic": "Базовые"},
        {"word": "goodbye", "translation": "до свидания", "learned": False,
    "review_count": 0, "topic": "Базовые"},
        {"word": "please", "translation": "пожалуйста", "learned": False,
    "review_count": 0, "topic": "Базовые"},
        {"word": "thank you", "translation": "спасибо", "learned": False,
    "review_count": 0, "topic": "Базовые"},
        {"word": "sorry", "translation": "извините", "learned": False,
    "review_count": 0, "topic": "Базовые"},
    ]
    # добавление базовых слов в словарь пользователя
    user_vocabulary[user_id].extend(basic_words)
    # возврат успешного выполнения
    return True

# функция получения словаря пользователя
def get_user_vocabulary(user_id: int):
    # возврат словаря пользователя или пустого списка если пользователя нет
    return user_vocabulary.get(user_id, [])

# функция получения настроек пользователя
def get_user_settings(user_id: int):
    # возврат настроек пользователя или пустого словаря если пользователя нет
    return user_settings.get(user_id, {})

# функция получения прогресса пользователя
def get_user_progress(user_id: int):
    # возврат прогресса пользователя или пустого словаря если пользователя нет
    return user_progress.get(user_id, {})

# функция получения статистики пользователя
def get_user_stats(user_id: int):
    # получение словаря пользователя
    vocabulary = get_user_vocabulary(user_id)
    # получение прогресса пользователя
    progress = get_user_progress(user_id)

    # подсчет общего количества слов
    total_words = len(vocabulary)
    # подсчет выученных слов (где learned = True)
    learned_words = sum(1 for w in vocabulary if w.get("learned", False))

    # возврат словаря со статистикой
    return {
        "total_words": total_words,
        "learned_words": learned_words,
        "exercises_completed": progress.get("exercises_completed", 0),

```

```
"current_streak": progress.get("current_streak", 0),
"accuracy": progress.get("accuracy", 0.0)
}

# функция обновления прогресса пользователя
def update_user_progress(user_id: int, key: str, value):
    # проверка существования прогресса пользователя
    if user_id in user_progress:
        # обновление значения по ключу
        user_progress[user_id][key] = value
        return True
    return False

# функция обновления настроек пользователя
def update_user_settings(user_id: int, key: str, value):
    # проверка существования настроек пользователя
    if user_id in user_settings:
        # обновление значения по ключу
        user_settings[user_id][key] = value
        return True
    return False

# функция добавления слова в словарь пользователя
def add_word_to_vocabulary(user_id: int, word_data: dict):
    # создание пустого списка если пользователя нет
    if user_id not in user_vocabulary:
        user_vocabulary[user_id] = []

    # получение списка существующих слов пользователя
    existing_words = [w.get("word") for w in user_vocabulary[user_id]]
    # проверка, что слово еще не добавлено
    if word_data.get("word") not in existing_words:
        # добавление слова в словарь
        user_vocabulary[user_id].append(word_data)
        return True
    return False

# функция получения случайного слова из словаря пользователя
def get_random_word(user_id: int, only_unlearned: bool = False):
    # проверка существования словаря пользователя и что он не пустой
    if user_id not in user_vocabulary or not user_vocabulary[user_id]:
        return None

    # если нужно только невыученные слова
    if only_unlearned:
        # фильтрация невыученных слов
        unlearned_words = [w for w in user_vocabulary[user_id] if not w.get("learned",
False)]
        # если невыученных слов нет
        if not unlearned_words:
            return None
        # возврат случайного невыученного слова
        return random.choice(unlearned_words)
```

```
# возврат случайного слова из всего словаря
return random.choice(user_vocabulary[user_id])

# функция обновления точности пользователя
def update_accuracy(user_id: int, is_correct: bool):
    # получение прогресса пользователя
    progress = get_user_progress(user_id)
    if not progress:
        return

    # увеличение общего количества попыток
    progress["total_attempts"] = progress.get("total_attempts", 0) + 1
    # если ответ правильный
    if is_correct:
        # увеличение количества правильных попыток
        progress["correct_attempts"] = progress.get("correct_attempts", 0) + 1
        # увеличение текущей серии правильных ответов
        progress["current_streak"] = progress.get("current_streak", 0) + 1
        # увеличение количества выполненных упражнений
        progress["exercises_completed"] = progress.get("exercises_completed", 0) + 1
    else:
        # сброс серии правильных ответов при неправильном ответе
        progress["current_streak"] = 0

    # получение общего количества попыток и правильных ответов
    total = progress.get("total_attempts", 0)
    correct = progress.get("correct_attempts", 0)
    # расчет точности если были попытки
    if total > 0:
        progress["accuracy"] = round(correct / total * 100, 1)

# функция увеличения счетчика повторений слова
def increment_word_review(user_id: int, word: str):
    # проверка существования словаря пользователя
    if user_id in user_vocabulary:
        # поиск слова в словаре пользователя
        for w in user_vocabulary[user_id]:
            if w.get("word") == word:
                # увеличение счетчика повторений
                w["review_count"] = w.get("review_count", 0) + 1
                # если слово повторено 2 или более раз, помечаем как выученное
                if w["review_count"] >= 2:
                    w["learned"] = True
        return True
    return False

# функция пометки слова как выученного
def mark_word_as_learned(user_id: int, word: str):
    # проверка существования словаря пользователя
    if user_id in user_vocabulary:
        # поиск слова в словаре пользователя
        for w in user_vocabulary[user_id]:
```

```

        if w.get("word") == word:
            # пометка слова как выученного
            w["learned"] = True
            return True
    return False

# функция получения ежедневных фраз
def get_daily_phrases():
    # создание списка всех доступных фраз
    all_phrases = [
        {"en": "How are you?", "ru": "Как дела?", "context": "Приветствие"},
        {"en": "I'm fine, thank you!", "ru": "Хорошо, спасибо!", "context": "Ответ на приветствие"},
        {"en": "What's your name?", "ru": "Как тебя зовут?", "context": "Знакомство"},
        {"en": "Nice to meet you!", "ru": "Приятно познакомиться!", "context": "Знакомство"},
        {"en": "Where are you from?", "ru": "Откуда ты?", "context": "Разговор о происхождении"},
        {"en": "Can you help me?", "ru": "Можешь помочь мне?", "context": "Просьба о помощи"},
        {"en": "I don't understand", "ru": "Я не понимаю", "context": "Непонимание"},
        {"en": "How much is it?", "ru": "Сколько это стоит?", "context": "Покупки"},
        {"en": "What time is it?", "ru": "Который час?", "context": "Время"},
        {"en": "Where is the bathroom?", "ru": "Где ванная комната?", "context": "Поиск места"},
        {"en": "I'm hungry", "ru": "Я голоден", "context": "Еда"},
        {"en": "Let's go!", "ru": "Пошли!", "context": "Предложение"},
        {"en": "See you later!", "ru": "Увидимся позже!", "context": "Прощание"},
        {"en": "Have a nice day!", "ru": "Хорошего дня!", "context": "Пожелание"},
        {"en": "You're welcome!", "ru": "Пожалуйста!", "context": "Ответ на спасибо"},
        {"en": "Excuse me", "ru": "Извините", "context": "Привлечение внимания"},
        {"en": "I love you", "ru": "Я тебя люблю", "context": "Чувства"},
        {"en": "Good luck!", "ru": "Удачи!", "context": "Пожелание"},
        {"en": "Happy birthday!", "ru": "С днем рождения!", "context": "Поздравление"},
        {"en": "Merry Christmas!", "ru": "С Рождеством!", "context": "Поздравление"}
    ]

    # возврат 5 случайных фраз из списка
    return random.sample(all_phrases, 5)

# импорт необходимых компонентов
from aiogram import types
from config import VOCABULARY_BY_TOPIC

# функция для получения основной клавиатуры
def get_main_keyboard():
    # создание клавиатуры с кнопками для основных режимов
    return types.ReplyKeyboardMarkup(
        keyboard=[
            # первая строка: теория и практика
            [types.KeyboardButton(text="Теория"),
             types.KeyboardButton(text="Практика")],
            # вторая строка: фразы дня и главное меню

```

```
[types.KeyboardButton(text="Фразы дня"),
types.KeyboardButton(text="Главное меню")]
],
# автоматическое изменение размера клавиатуры
resize_keyboard=True
)

# функция для получения клавиатуры теории
def get_theory_keyboard():
    # создание клавиатуры для режима теории
    return types.ReplyKeyboardMarkup(
        keyboard=[
            # первая строка: новые слова и грамматика
            [types.KeyboardButton(text="Новые слова"),
             types.KeyboardButton(text="Грамматика")],
            # вторая строка: сменить тему и главное меню
            [types.KeyboardButton(text="Сменить тему"),
             types.KeyboardButton(text="Главное меню")]
        ],
        # автоматическое изменение размера клавиатуры
        resize_keyboard=True
    )

# функция для получения клавиатуры с темами
def get_topics_keyboard():
    # получение списка доступных тем из конфигурации
    topics = list(VOCABULARY_BY_TOPIC.keys())
    # создание пустого списка для кнопок
    keyboard = []

    # организация кнопок тем в ряды по 2
    for i in range(0, len(topics), 2):
        # выбор двух тем для текущего ряда
        row = topics[i:i+2]
        # добавление кнопок в клавиатуру
        keyboard.append([types.KeyboardButton(text=topic) for topic in row])

    # добавление кнопки для возврата к теории
    keyboard.append([types.KeyboardButton(text="К теории")])

    # создание и возврат клавиатуры
    return types.ReplyKeyboardMarkup(keyboard=keyboard, resize_keyboard=True)

# функция для получения клавиатуры с вариантами ответов
def get_exercise_keyboard(options):
    # создание пустого списка для кнопок
    buttons = []
    # создание пустого ряда для текущих кнопок
    row = []

    # организация вариантов ответов в ряды по 2
    for option in options:
        # добавление варианта в текущий ряд
        buttons.append(types.KeyboardButton(text=option))
        if len(buttons) == 2:
            row.append(buttons)
            buttons = []
```

```
row.append(types.KeyboardButton(text=option))
# если в ряду 2 кнопки, добавляем его в клавиатуру
if len(row) == 2:
    buttons.append(row)
    # сброс текущего ряда
    row = []

# если остались недобавленные кнопки, добавляем их
if row:
    buttons.append(row)

# добавление кнопки для возврата к практике
buttons.append([types.KeyboardButton(text="Назад к практике")])

# создание и возврат клавиатуры
return types.ReplyKeyboardMarkup(keyboard=buttons, resize_keyboard=True)

# функция для получения клавиатуры обратной связи после упражнения
def get_exercise_feedback_keyboard():
    # создание клавиатуры с кнопками для навигации после упражнения
    return types.ReplyKeyboardMarkup(
        keyboard=[
            # строка с кнопками: следующий вопрос и назад к практике
            [types.KeyboardButton(text="Следующий вопрос"),
             types.KeyboardButton(text="Назад к практике")]
        ],
        # автоматическое изменение размера клавиатуры
        resize_keyboard=True
    )

# функция для получения клавиатуры с темами грамматики
def get_grammar_keyboard():
    # создание клавиатуры с темами грамматики
    return types.ReplyKeyboardMarkup(
        keyboard=[
            # первый ряд: простые времена
            [types.KeyboardButton(text="Present Simple"),
             types.KeyboardButton(text="Past Simple")],
            # второй ряд: продолженные времена и будущее время
            [types.KeyboardButton(text="Future Simple"),
             types.KeyboardButton(text="Present Continuous")],
            # третий ряд: артикли и предлоги
            [types.KeyboardButton(text="Артикли"),
             types.KeyboardButton(text="Предлоги")],
            # четвертый ряд: модальные глаголы и возврат к теории
            [types.KeyboardButton(text="Модальные глаголы"),
             types.KeyboardButton(text="К теории")]
        ],
        # автоматическое изменение размера клавиатуры
        resize_keyboard=True
    )

# импорт асинхронной библиотеки
```

```
import asyncio
# импорт модуля для логирования
import logging
# импорт необходимых компонентов из aiogram
from aiogram import Bot, Dispatcher, types, F
from aiogram.filters import Command
from aiogram.fsm.storage.memory import MemoryStorage
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup
# импорт модуля для случайных значений
import random

# импорт конфигурационных данных
from config import *
# импорт функций для работы с базой данных
from database import *
# импорт функций для создания клавиатур
from keyboards import *

# настройка логирования
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
# создание логгера для текущего модуля
logger = logging.getLogger(__name__)

# создание экземпляра бота с токеном из конфига
bot = Bot(token=API_TOKEN)
# создание диспетчера с хранилищем состояний в памяти
dp = Dispatcher(storage=MemoryStorage())

# определение состояний для режима теории
class TheoryStates(StatesGroup):
    # состояние для работы с теорией
    THEORY = State()

# определение состояний для режима практики
class PracticeStates(StatesGroup):
    # состояние для выбора типа практики
    PRACTICE = State()
    # состояние ожидания ответа от пользователя
    WAITING_FOR_ANSWER = State()

# обработчик команды отмены
@dp.message(Command("cancel"))
async def cancel_handler(message: types.Message, state: FSMContext):
    # получение текущего состояния
    current_state = await state.get_state()
    # если состояния нет, ничего не делаем
    if current_state is None:
        return
```

```
# очистка состояния
await state.clear()
# отправка сообщения об отмене
await message.answer(
    "действие отменено. возвращаю в главное меню.",
    reply_markup=get_main_keyboard()
)

# обработчик команды старт
@dp.message(Command("start"))
async def cmd_start(message: types.Message):
    # получение id пользователя
    user_id = message.from_user.id
    # инициализация данных пользователя
    init_user_data(user_id)

    # создание приветственного сообщения
    welcome_message = (
        f"*добро пожаловать в языковой бот!*\n\n"
        f"я помогу вам изучать английский язык эффективно.\n\n"
        f"*доступные режимы:*\n"
        f"• теория – изучение новых слов и грамматики\n"
        f"• практика – упражнения для закрепления\n"
        f"• фразы дня – полезные выражения\n\n"
        f"выберите режим:"
    )

    # отправка приветственного сообщения
    await message.answer(
        welcome_message,
        reply_markup=get_main_keyboard(),
        parse_mode="Markdown"
    )

# обработчик кнопки "главное меню"
@dp.message(F.text == "Главное меню")
async def main_menu(message: types.Message, state: FSMContext):
    # очистка состояния
    await state.clear()
    # возврат к стартовому сообщению
    await cmd_start(message)

# обработчик кнопки "фразы дня"
@dp.message(F.text == "Фразы дня")
async def show_phrases_of_day(message: types.Message):
    # получение фраз дня
    phrases = get_daily_phrases()

    # формирование ответа
    response = "*фразы дня:*\n\n"
    for i, phrase in enumerate(phrases, 1):
        response += f"{i}. {phrase['en']}\n"
        response += f"    перевод: {phrase['ru']}\n"

    await message.answer(response)
```

```
response += f"    контекст: {phrase['context']}\n\n"

# отправка ответа с кастомной клавиатурой
await message.answer(
    response,
    reply_markup=types.ReplyKeyboardMarkup(
        keyboard=[
            [types.KeyboardButton(text="Новые фразы")],
            [types.KeyboardButton(text="Главное меню")]
        ],
        resize_keyboard=True
    ),
    parse_mode="Markdown"
)

# обработчик кнопки "новые фразы"
@dp.message(F.text == "Новые фразы")
async def get_new_phrases(message: types.Message):
    # получение фраз дня
    phrases = get_daily_phrases()

    # формирование ответа
    response = "*новые фразы дня:*\n\n"
    for i, phrase in enumerate(phrases, 1):
        response += f"{i}. *{phrase['en']}*\n"
        response += f"    перевод: {phrase['ru']}\n"
        response += f"    контекст: {phrase['context']}\n\n"

    # отправка ответа с кастомной клавиатурой
    await message.answer(
        response,
        reply_markup=types.ReplyKeyboardMarkup(
            keyboard=[
                [types.KeyboardButton(text="Еще фразы")],
                [types.KeyboardButton(text="Главное меню")]
            ],
            resize_keyboard=True
        ),
        parse_mode="Markdown"
    )

# обработчик кнопки "еще фразы"
@dp.message(F.text == "Еще фразы")
async def get_more_phrases(message: types.Message):
    # просто вызывает функцию для получения новых фраз
    await get_new_phrases(message)

# обработчик кнопки "теория"
@dp.message(F.text == "Теория")
async def start_theory(message: types.Message, state: FSMContext):
    # установка состояния теории
    await state.set_state(TheoryStates.THEORY)
    # получение id пользователя
```

```
user_id = message.from_user.id
# получение настроек пользователя
settings = get_user_settings(user_id)
# получение текущей темы или использование темы по умолчанию
topic = settings.get("current_topic", "Еда")

# отправка сообщения с выбором типа материала
await message.answer(
    f"*режим теории*\n\ntекущая тема: *{topic}*\nвыберите тип материала:",
    reply_markup=get_theory_keyboard(),
    parse_mode="Markdown"
)

# обработчик кнопки "новые слова" в состоянии теории
@dp.message(TheoryStates.THEORY, F.text == "Новые слова")
async def show_new_words(message: types.Message, state: FSMContext):
    # получение id пользователя
    user_id = message.from_user.id
    # получение настроек пользователя
    settings = get_user_settings(user_id)
    # получение текущей темы или использование темы по умолчанию
    topic = settings.get("current_topic", "Еда")
    # получение словаря по теме
    vocabulary = VOCABULARY_BY_TOPIC.get(topic, {})

    # если словарь не пустой
    if vocabulary:
        response = f"\n*новые слова по теме '{topic}'*:\n\n"
        # получение списка слов
        word_items = list(vocabulary.items())
        # выбор случайных слов (не более 5)
        selected_words = random.sample(word_items, min(5, len(word_items)))

        # формирование ответа для каждого слова
        for i, (en_word, data) in enumerate(selected_words, 1):
            ru_word = data["ru"]
            # выбор случайного примера
            example = random.choice(data.get("examples", [""]))

            response += f"\n{i}. *{en_word}* - {ru_word}\n"
            response += f"\n    пример: {_example_}\n\n"

        # добавление слова в словарь пользователя
        add_word_to_vocabulary(user_id, {
            "word": en_word,
            "translation": ru_word,
            "learned": False,
            "review_count": 0,
            "topic": topic
        })

    # создание кастомной клавиатуры
    keyboard = types.ReplyKeyboardMarkup(
        keyboard=[
```

```

        [types.KeyboardButton(text="Запомнил слова")],
        [types.KeyboardButton(text="К теории")]
    ],
    resize_keyboard=True
)
response += "нажмите 'запомнил слова', когда будете готовы к практике!"
else:
    # если слов нет
    response = "для этой темы пока нет слов."
    keyboard = types.ReplyKeyboardMarkup(
        keyboard=[[types.KeyboardButton(text="К теории")]],
        resize_keyboard=True
)

# отправка ответа
await message.answer(response, reply_markup=keyboard, parse_mode="Markdown")

# обработчик кнопки "сменить тему" в состоянии теории
@dp.message(TheoryStates.THEORY, F.text == "Сменить тему")
async def change_topic(message: types.Message):
    # отправка сообщения с выбором темы
    await message.answer(
        "*выберите тему для изучения:*\n\nкаждая тема содержит новые слова с
примерами.",
        reply_markup=get_topics_keyboard(),
        parse_mode="Markdown"
    )

# обработчик выбора темы в состоянии теории
@dp.message(TheoryStates.THEORY, F.text.in_(list(VOCABULARY_BY_TOPIC.keys())))
async def handle_topic_selection(message: types.Message):
    # получение id пользователя
    user_id = message.from_user.id
    # получение выбранной темы
    selected_topic = message.text

    # обновление настроек пользователя
    update_user_settings(user_id, "current_topic", selected_topic)

    # отправка подтверждения
    await message.answer(
        f"Тема изменена на: *{selected_topic}*\n\n"
        f"Теперь вы можете изучать новые слова по этой теме.",
        parse_mode="Markdown",
        reply_markup=get_theory_keyboard()
    )

# обработчик кнопки "грамматика" в состоянии теории
@dp.message(TheoryStates.THEORY, F.text == "Грамматика")
async def show_grammar(message: types.Message):
    # отправка сообщения с выбором темы грамматики
    await message.answer(
        "*выберите тему грамматики:*\n\n"

```

```

"изучите правила, примеры и выполните упражнения для закрепления.\n\n"
"• доступные темы:\n"
"• present simple – настоящеe простое\n"
"• past simple – прошедшее простое\n"
"• future simple – будущее простое\n"
"• present continuous – настоящеe продолженное\n"
"• артикли – a/an/the\n"
"• предлоги – in/on/at\n"
"• модальныe глаголы – can/must/should",
reply_markup=get_grammar_keyboard(),
parse_mode="Markdown"
)

# обработчик кнопки "практика"
@dp.message(F.text == "Практика")
async def start_practice(message: types.Message, state: FSMContext):
    # установка состояния практики
    await state.set_state(PracticeStates.PRACTICE)
    # получение id пользователя
    user_id = message.from_user.id
    # инициализация данных пользователя
    init_user_data(user_id)

    # получение словаря пользователя
    vocabulary = get_user_vocabulary(user_id)
    # получение статистики пользователя
    stats = get_user_stats(user_id)

    # создание клавиатуры для практики
    keyboard = types.ReplyKeyboardMarkup(
        keyboard=[
            [types.KeyboardButton(text="Слова: перевод"),
             types.KeyboardButton(text="Слова: выбор")],
            [types.KeyboardButton(text="Главное меню")]
        ],
        resize_keyboard=True
    )

    # отправка сообщения с статистикой
    await message.answer(
        f"• режим практики\n\n• ваша статистика:\n"
        f"• слов в словаре: *{len(vocabulary)}*\n"
        f"• выучено слов: *{stats.get('learned_words', 0)}*\n"
        f"• точность: *{stats.get('accuracy', 0)}%*\n"
        f"• серия правильных ответов: *{stats.get('current_streak', 0)}*\n\n"
        f"• выберите тип упражнения:",
        reply_markup=keyboard,
        parse_mode="Markdown"
    )

    # обработчик кнопки "слова: перевод" в состоянии практики
    @dp.message(PracticeStates.PRACTICE, F.text == "Слова: перевод")
    async def practice_translation(message: types.Message, state: FSMContext):

```

```

# отправка упражнения на перевод
await send_word_exercise(message.from_user.id, message, state, "translation")

# обработчик кнопки "слова: выбор" в состоянии практики
@dp.message(PracticeStates.PRACTICE, F.text == "Слова: выбор")
async def practice_choose_translation(message: types.Message, state: FSMContext):
    # отправка упражнения на выбор перевода
    await send_word_exercise(message.from_user.id, message, state,
"choose_translation")

# функция отправки упражнения
async def send_word_exercise(user_id: int, message: types.Message, state: FSMContext,
exercise_type: str = "translation"):
    # получение словаря пользователя
    vocabulary = get_user_vocabulary(user_id)

    # если словарь пустой
    if not vocabulary:
        await message.answer(
            "нет слов для практики.\n\nсначала изучите слова в режиме теория!",
            reply_markup=types.ReplyKeyboardMarkup(
                keyboard=[[types.KeyboardButton(text="Теория")]],
                resize_keyboard=True
            )
        )
        return

    # получение случайного слова
    word_data = get_random_word(user_id)
    # если не удалось получить слово
    if not word_data:
        await message.answer(
            "не удалось получить слово для упражнения.",
            reply_markup=types.ReplyKeyboardMarkup(
                keyboard=[[types.KeyboardButton(text="Назад к практике")]],
                resize_keyboard=True
            )
        )
        return

    # подготовка упражнения в зависимости от типа
    if exercise_type == "translation":
        word = word_data["word"]
        correct_answer = word_data["translation"]
        # получение всех неправильных вариантов
        all_options = [w["translation"] for w in vocabulary if w["translation"] != correct_answer]
        question = f"*переведите слово:{*\n\n**{word}**\n\nвыберите правильный\nперевод:"

    else:
        translation = word_data["translation"]
        correct_answer = word_data["word"]
        # получение всех неправильных вариантов

```

```

all_options = [w["word"] for w in vocabulary if w["word"] != correct_answer]
question = f"выберите слово для перевода:{translation}**выберите\nправильное английское слово:"

# создание списка вариантов ответов
if len(all_options) >= 3:
    wrong_options = random.sample(all_options, 3)
else:
    # если недостаточно вариантов, создаем фейковые
    fake_options = ["стол", "окно", "дверь"] if exercise_type == "translation"
    else ["table", "window", "door"]
    wrong_options = random.sample([o for o in fake_options if o != correct_answer], 3)

# смешивание вариантов ответов
options = [correct_answer] + wrong_options
random.shuffle(options)

# сохранение данных в состояние
await state.update_data(
    correct_answer=correct_answer,
    exercise_type=exercise_type,
    word_data=word_data
)

# дополнительные данные в зависимости от типа упражнения
if exercise_type == "choose_translation":
    await state.update_data(translation=translation)
else:
    await state.update_data(word=word_data["word"])

# отправка вопроса
await message.answer(
    question,
    reply_markup=get_exercise_keyboard(options),
    parse_mode="Markdown"
)
# установка состояния ожидания ответа
await state.set_state(PracticeStates.WAITING_FOR_ANSWER)

# обработчик кнопки "следующий вопрос"
@dp.message(F.text == "Следующий вопрос")
async def handle_next_question(message: types.Message, state: FSMContext):
    # возврат в состояние практики
    await state.set_state(PracticeStates.PRACTICE)
    # выбор случайного типа упражнения
    exercises = [practice_translation, practice_choose_translation]
    await random.choice(exercises)(message, state)

# обработчик ответа в состоянии ожидания ответа
@dp.message(PracticeStates.WAITING_FOR_ANSWER)
async def handle_practice_answer(message: types.Message, state: FSMContext):
    # получение ответа пользователя

```

```
user_answer = message.text
# получение данных из состояния
data = await state.get_data()
correct_answer = data.get("correct_answer", "")

# проверка правильности ответа
is_correct = user_answer == correct_answer
# обновление статистики точности
update_accuracy(message.from_user.id, is_correct)

# если ответ правильный, увеличиваем счетчик повторений слова
if is_correct:
    if "word_data" in data:
        word = data["word_data"]["word"]
        increment_word_review(message.from_user.id, word)

# формирование префикса в зависимости от правильности ответа
if is_correct:
    prefix = "*правильно!@"
else:
    prefix = "*неправильно!@"

# формирование ответа в зависимости от типа упражнения
if data.get("exercise_type") == "translation":
    word = data.get("word", "")
    correct = data.get("correct_answer", "")
    if is_correct:
        response = f"{prefix}\n\n**{word}** → **{correct}**"
    else:
        response = f"{prefix}\n\n**{word}** → **{correct}**\n\nваш ответ:\n{user_answer}"
else:
    translation = data.get("translation", "")
    correct = data.get("correct_answer", "")
    if is_correct:
        response = f"{prefix}\n\n**{translation}** → **{correct}**"
    else:
        response = f"{prefix}\n\n**{translation}** → **{correct}**\n\nваш ответ:\n{user_answer}"

# добавление примеров использования слова
if "word_data" in data and "examples" in data["word_data"]:
    examples = data["word_data"]["examples"][:2]
    if examples:
        response += f"\n\n*примеры использования:*\n"
        for example in examples:
            response += f"\n• {example}"

# получение статистики пользователя
stats = get_user_stats(message.from_user.id)
response += f"\n\n*статистика:*\n"
response += f"упражнений выполнено: *{stats.get('exercises_completed', 0)}*\n"
response += f"серия: *{stats.get('current_streak', 0)}*\n"
```

```
response += f"выучено слов: *{stats.get('learned_words', 0)}*\n"
response += f"точность: *{stats.get('accuracy', 0)}%*"

# отправка ответа с клавиатурой для обратной связи
await message.answer(
    response,
    reply_markup=get_exercise_feedback_keyboard(),
    parse_mode="Markdown"
)
# очистка состояния
await state.clear()

# обработчик кнопки "к теории"
@dp.message(F.text == "К теории")
async def back_to_theory(message: types.Message, state: FSMContext):
    # установка состояния теории
    await state.set_state(TheoryStates.THEORY)
    # переход к началу теории
    await start_theory(message, state)

# обработчик кнопки "запомнил слова"
@dp.message(F.text == "Запомнил слова")
async def words_learned(message: types.Message):
    # получение id пользователя
    user_id = message.from_user.id
    # получение прогресса пользователя
    progress = get_user_progress(user_id)
    # получение словаря пользователя
    vocabulary = get_user_vocabulary(user_id)
    # подсчет выученных слов
    learned_count = len([w for w in vocabulary if w.get("learned", False)])
    # обновление прогресса
    update_user_progress(user_id, "words_learned", learned_count)

    # отправка подтверждения
    await message.answer(
        f"*отлично!* вы запомнили новые слова!\n\nтеперь переходите в режим практика!",
        reply_markup=types.ReplyKeyboardMarkup(
            keyboard=[
                [types.KeyboardButton(text="Практика"),
                 types.KeyboardButton(text="Теория")]
            ],
            resize_keyboard=True
        ),
        parse_mode="Markdown"
    )

# обработчик кнопки "назад к практике"
@dp.message(F.text.contains("Назад к практике"))
async def back_to_practice_handler(message: types.Message, state: FSMContext):
    # установка состояния практики
    await state.set_state(PracticeStates.PRACTICE)
```

```
# переход к началу практики
await start_practice(message, state)

# обработчик выбора темы грамматики
@dp.message(F.text.in_(list(GRAMMAR_LESSONS.keys())))
async def handle_grammar_topic(message: types.Message):
    # получение выбранной темы
    topic = message.text
    # получение урока грамматики
    lesson = GRAMMAR_LESSONS.get(topic)

    # если урок не найден
    if not lesson:
        await message.answer("этот тема грамматики еще не добавлена.")
        return

    # формирование ответа с уроком
    response = f"{lesson['name']}\n\n"
    response += f"*{lesson['description']}*\n\n"

    # добавление правил
    response += "*правила:*\n"
    for rule in lesson["rules"]:
        response += f"\n{rule}\n"

    # добавление примеров
    response += "\n*примеры:*\n"
    for example in lesson["examples"]:
        response += f"\n• {example}\n"

    # добавление использования (если есть)
    if "usage" in lesson:
        response += "\n*использование:*\n"
        for usage in lesson["usage"]:
            response += f"\n• {usage}\n"

    # добавление маркеров времени (если есть)
    if "time_markers" in lesson:
        response += "\n*маркеры времени:*\n"
        for marker in lesson["time_markers"]:
            response += f"\n• {marker}\n"

    # добавление исключений (если есть)
    if "exceptions" in lesson:
        response += "\n*исключения:*\n"
        for exception in lesson["exceptions"]:
            response += f"\n• {exception}\n"

    # добавление особенностей (если есть)
    if "features" in lesson:
        response += "\n*особенности:*\n"
        for feature in lesson["features"]:
            response += f"\n• {feature}\n"
```

```

# создание клавиатуры для навигации
keyboard = types.ReplyKeyboardMarkup(
    keyboard=[
        [types.KeyboardButton(text="Другая тема"), types.KeyboardButton(text="К
теории")]
    ],
    resize_keyboard=True
)

# отправка урока
await message.answer(response, reply_markup=keyboard, parse_mode="Markdown")

# обработчик кнопки "другая тема"
@dp.message(F.text == "Другая тема")
async def another_grammar_topic(message: types.Message):
    # возврат к выбору темы грамматики
    await show_grammar(message)

# отладочный обработчик для всех сообщений
@dp.message()
async def debug_handler(message: types.Message):
    print(f" получено: '{message.text}'")
    print(f" содержит 'назад'? {'назад' in message.text}")
    print(f" содержит 'назад к практике'? {'назад к практике' in message.text}")

# основная асинхронная функция
async def main():
    print("бот запускается")
    print(f"бот готов к работе")
    print(f"основные режимы: теория, практика, фразы дня")
    print(f"команды: /start, /cancel")
    print(f"начните с команды /start")
    print()

    # запуск опроса бота
    await dp.start_polling(bot)

# точка входа в программу
if __name__ == "__main__":
    # запуск основной асинхронной функции
    asyncio.run(main())

```

## Скриншот работы приложения

```

○ (venv) daraprosvirakova@MacBook-Air-Dara-2 telegram-bot % python3 main.py
Бот запускается
Бот готов к работе
Основные режимы: Теория, Практика, Фразы дня
Команды: /start, /cancel
Начните с команды /start

2025-12-23 13:31:56,994 - aiogram.dispatcher - INFO - Start polling
2025-12-23 13:31:57,160 - aiogram.dispatcher - INFO - Run polling for bot @english_training_bot id=8201918414 - 'english_practice'
2025-12-23 13:31:57,429 - aiogram.event - INFO - Update id=203423872 is handled. Duration 210 ms by bot id=8201918414

```

