**Bot.py**

**1. Структура и основные компоненты**

Код состоит из нескольких ключевых частей:

* **Инициализация бота** – создание экземпляра приложения с помощью ApplicationBuilder.
* **Настройка логирования** – запись событий в консоль для отладки и мониторинга работы.
* **Регистрация обработчиков** – подключение различных модулей, отвечающих за команды и взаимодействие с пользователем.
* **Запуск бота** – переход в режим постоянного опроса сообщений (polling).

**2. Логирование**

Для удобства разработки настроен модуль logging, который выводит:

* Время события (%(asctime)s)
* Название модуля (%(name)s)
* Уровень важности (%(levelname)s)
* Текст сообщения (%(message)s)

Это помогает отслеживать ошибки и анализировать работу бота.

**3. Обработчики команд**

Функционал бота разбит на отдельные классы-обработчики:

* **StartHandler** – реагирует на команду /start (приветствие, начало работы).
* **RegistrationHandler** – отвечает за регистрацию пользователя (сбор данных, сохранение в БД).
* **FoodTrackingHandler** – позволяет вносить и анализировать информацию о питании.
* **ErrorHandler** – перехватывает и обрабатывает ошибки, чтобы бот не падал при сбоях.

Каждый обработчик, вероятно, содержит несколько подкоманд (например, /add\_food, /stats), которые добавляются в бота через метод get\_handlers().

**4. Запуск и работа**

Бот стартует в режиме polling – он постоянно запрашивает у серверов Telegram новые сообщения и команды. Токен для доступа к API берется из файла конфигурации (Config.TOKEN).

**5. Преимущества архитектуры**

* **Модульность** – каждый функционал вынесен в отдельный класс, что упрощает поддержку и расширение.
* **Централизованное управление** – все обработчики регистрируются в одном месте (setup\_handlers).
* **Обработка ошибок** – есть отдельный механизм для корректного реагирования на сбои.

**Bot\_alt.py**

При первом запуске бот предлагает пройти регистрацию, собирая данные:

* Пол (М/Ж)
* Возраст
* Рост (см)
* Вес (кг)
* Уровень физической активности (6 вариантов)

**Как рассчитывается норма калорий?**  
Используется **формула Миффлина-Сан Жеора**:

* Для мужчин:  
  BMR = (10 × вес) + (6.25 × рост) – (5 × возраст) + 5
* Для женщин:  
  BMR = (10 × вес) + (6.25 × рост) – (5 × возраст) – 161

Затем BMR умножается на **коэффициент активности**, чтобы получить **рекомендуемую дневную норму калорий**.

**2️⃣ Поиск калорийности продуктов через FatSecret API**

Пользователь может ввести название блюда, и бот:  
 Находит его в базе FatSecret  
Показывает калорийность на 100 г  
 Рассчитывает калории для указанного веса

**Пример работы:**  
➡ Пользователь вводит *"яблоко"*  
➡ Бот находит в API и сообщает: \*"Яблоко, 52 ккал/100 г"\*  
➡ Пользователь вводит *"150"* (грамм)  
➡ Бот выдаёт: *" Калорийность 150 г яблока: 78 ккал"*

**Сохранение данных пользователя**

Все введённые параметры (вес, рост, активность) сохраняются в **базе данных** (через класс Database).

**Техническая реализация**

**Структура кода**

1. **Инициализация бота**
   * Загрузка токенов из .env
   * Настройка логирования (logging)
   * Создание экземпляра ApplicationBuilder
2. **Обработчики команд**
   * /start – начало работы, меню
   * /profile – повторный расчёт нормы калорий
3. **Состояния диалога (ConversationHandler)**  
   Бот использует **машину состояний**, чтобы последовательно собирать данные:
   * GENDER → AGE → HEIGHT → WEIGHT → ACTIVITY\_LEVEL
   * ENTER\_DISH\_NAME → ENTER\_WEIGHT
4. **Работа с FatSecret API**
   * Авторизация через OAuth 2.0 (get\_access\_token)
   * Поиск продуктов (search\_food)
5. **Обработка ошибок**
   * Логирование через logger.error
   * Уведомление пользователя при сбоях

**Используемые технологии**

* **python-telegram-bot** – фреймворк для Telegram-ботов
* **FatSecret API** – база данных продуктов и их пищевой ценности
* **dotenv** – загрузка конфигурации из .env
* **logging** – логирование работы бота
* **requests** – HTTP-запросы к FatSecret API

**Сonfig.py**

Класс Config представляет собой централизованное хранилище конфигурационных параметров для бота, что является примером хорошей практики организации кода.

**Основные функции:**

1. **Хранение чувствительных данных**
   * TOKEN - токен Telegram-бота
   * FATSECRET\_CLIENT\_ID и FATSECRET\_CLIENT\_SECRET - credentials для работы с FatSecret API
   * Все данные загружаются из переменных окружения через os.getenv()
2. **Управление состояниями бота**
   * Определены все возможные состояния диалога (GENDER, AGE и др.) через range(8)
   * Это позволяет чётко отслеживать текущий этап взаимодействия с пользователем
3. **Настройки активности пользователя**
   * ACTIVITY\_LEVELS - список вариантов физической активности
   * ACTIVITY\_FACTORS - соответствующие коэффициенты для расчёта калорий

**Преимущества такого подхода:**

**Безопасность** - чувствительные данные не хранятся в коде  
 **Читаемость** - все важные настройки собраны в одном месте  
**Масштабируемость** - легко добавлять новые параметры  
 **Типизация** - состояния представлены как именованные константы

*# Используем конфиг*

state = Config.ENTER\_WEIGHT

Этот класс является ключевым компонентом архитектуры бота, обеспечивающим:

* Безопасное хранение настроек
* Чёткую структуру состояний диалога
* Простую модификацию параметров

Такой подход соответствует принципам SOLID, особенно:

* Single Responsibility (единственная ответственность)
* Open-Closed (легко расширять)

**Database.py**

Класс Database представляет собой модуль для работы с базой данных SQLite, обеспечивающий хранение и обработку данных пользователей и их пищевых привычек.

**Архитектурные особенности:**

1. **Инициализация базы данных**
   * Автоматическое создание таблиц при инициализации (\_init\_db)
   * Две основные таблицы:
     + users - профили пользователей
     + meals - история приемов пищи
2. **Основной функционал**:
   * save\_user\_data - сохраняет/обновляет данные пользователя
   * get\_user\_data - получает информацию о пользователе
   * save\_meal - записывает прием пищи
   * get\_daily\_nutrition - подсчет суммарного питания за день
   * get\_user\_meals - история последних приемов пищи
3. **Безопасность и валидация**:
   * Проверки диапазонов значений (CHECK-ограничения)
   * Транзакционность операций (with-контекст)
   * Логирование всех операций

**Base.py**

BaseHandler представляет собой абстрактный базовый класс для создания обработчиков команд и сообщений в Telegram-боте, реализующий паттерн "Шаблонный метод".

**Основное назначение:**

1. **Унифицированная структура** обработчиков:
   * Единый интерфейс для всех обработчиков бота
   * Стандартизированный способ регистрации команд
2. **Автоматизация** процесса:
   * Упрощение добавления новых команд
   * Централизованное управление обработчиками

**Ключевые компоненты:**

1.**Регистрация обработчиков**:

* + command\_handlers - хранит кортежи (команда, функция-обработчик)
  + message\_handlers - хранит кортежи (фильтр, функция-обработчик)

1. **Метод get\_handlers()**:
   * Преобразует внутренние структуры в объекты Handler для python-telegram-bot
   * Автоматически создает CommandHandler и MessageHandler
2. **Абстрактный метод handle()**:
   * Определяет интерфейс для всех наследников
   * Требует реализации в дочерних классах

**Преимущества архитектуры:**

**Единообразие** - все обработчики работают по одному принципу  
 **Масштабируемость** - легко добавлять новые команды  
 **Инкапсуляция** - логика регистрации скрыта в базовом классе  
 **Гибкость** - поддерживает как команды, так и текстовые сообщения

**Food\_tracking.py**

**Основные возможности:**

1. **Поиск продуктов** по названию через FatSecret API
2. **Расчет калорийности** на основе веса продукта
3. **Парсинг пищевой ценности** (БЖУ) из описания продукта
4. **Интерактивный диалог** с пользователем через состояния

**Архитектурные особенности:**

| **Метод** | **Назначение** |
| --- | --- |
| start\_tracking | Начало процесса подсчета калорий |
| handle\_food\_input | Роутинг ввода пользователя |
| handle\_dish\_name | Обработка названия блюда |
| handle\_weight | Расчет калорий по весу |
| \_parse\_nutrition\_info | Парсинг БЖУ (внутренний метод) |

1. Пользователь вводит название блюда
2. Система ищет продукт в FatSecret API
3. Пользователь указывает вес порции
4. Бот рассчитывает и выводит пищевую ценность

**Особенности реализации:**

**Интеграция с внешним API** - работа с FatSecret через отдельный класс  
 **Обработка ошибок** - корректная реакция на проблемы с API  
**Валидация ввода** - проверка веса на корректность  
 **Парсинг данных** - извлечение калорий и БЖУ из текстового описания  
**Состояния диалога** - четкое разделение этапов взаимодействия

**Пример вывода:**

Найдено: Куриная грудка

Описание: Calories: 165kcal | Protein: 31g | Fat: 3.6g

Куриная грудка - 200г

Калории: 330 ккал

Protein: 62g

Fat: 7.2g

**Преимущества решения:**

1. **Удобный интерфейс** - пошаговый ввод данных
2. **Точные расчеты** - на основе данных авторитетного источника
3. **Расширяемость** - легко добавить анализ других нутриентов
4. **Надежность** - обработка всех возможных ошибок ввода

**Registration.py**

Класс RegistrationHandler реализует многоэтапный процесс регистрации пользователей в фитнес-боте с расчетом их индивидуальной нормы калорий.

**Основные функции:**

1. **Пошаговая регистрация** пользователей через интерактивный диалог
2. **Сбор антропометрических данных** (пол, возраст, рост, вес)
3. **Определение уровня физической активности**
4. **Расчет суточной нормы калорий** по формуле Миффлина-Сан Жеора
5. **Сохранение данных** в базу данных

**Архитектурные особенности:**

**1. Инициализация:**

* Интеграция с базой данных
* Регистрация обработчиков для разных этапов регистрации

**2. Этапы регистрации:**

| **Этап** | **Метод обработки** | **Проверки данных** |
| --- | --- | --- |
| Старт | start\_registration | Проверка завершенной регистрации |
| Пол | handle\_gender | Только "М" или "Ж" |
| Возраст | handle\_age | 10-120 лет |
| Рост | handle\_height | 100-250 см |
| Вес | handle\_weight\_input | 30-300 кг |
| Активность | handle\_activity | Выбор из предустановленных |

**3. Расчет калорий:**  
Используется формула Миффлина-Сан Жеора с учетом:

* Пола пользователя
* Физических параметров (вес, рост, возраст)
* Коэффициента активности

**Особенности реализации:**

**Валидация ввода** - строгие проверки корректности данных  
 **Интерактивная клавиатура** - удобный выбор из вариантов  
 **Контекст пользователя** - сохранение данных между этапами  
 **Логика расчета** - точный подсчет нормы калорий  
 **Интеграция с БД** - сохранение полного профиля

**Пример диалога:**

**Преимущества решения:**

1. **Пошаговый процесс** - четкое руководство для пользователя
2. **Гибкость** - легко добавить новые параметры
3. **Безопасность** - проверка всех вводимых данных
4. **Персонализация** - индивидуальный расчет для каждого пользователя
5. **Удобство** - интерактивные элементы управления

Класс представляет собой законченный модуль для регистрации пользователей, который можно легко интегрировать в любую систему фитнес-трекинга или бота здорового питания.

**Start.py**

Класс StartHandler является обработчиком стартовых команд бота, реализующим базовый функционал приветствия и навигации.

**Основное назначение:**

1. **Обработка стартовых команд** /start и /profile
2. **Первичное приветствие** пользователя
3. **Предоставление главного меню** с основными опциями

**Ключевые особенности:**

**1. Инициализация обработчиков:**

self.command\_handlers = [("start", self.start), ("profile", self.start)]

* Регистрирует две команды с одинаковым обработчиком
* Наследует базовый функционал от BaseHandler

**2. Метод start():**

async def start(self, update: Update, context: ContextTypes.DEFAULT\_TYPE):

* Формирует персонализированное приветствие
* Создает клавиатуру с основными действиями
* Возвращает состояние выбора действия (CHOOSE\_ACTION)

**3. Элементы интерфейса:**

* Персонализированное обращение (использует first\_name)
* Интерактивная клавиатура с кнопками:
  + "Регистрация"
  + "Подсчёт ккал блюда"

**Преимущества реализации:**

**Простота** - минималистичный и понятный код  
 **Удобство** - интерактивная клавиатура для новых пользователей  
 **Гибкость** - единый обработчик для двух команд  
 **Стандартизация** - возврат состояния для машины состояний

**Пример работы:**

1. Пользователь отправляет /start
2. Бот отвечает:

Привет, Иван! Выбери действие:

[Регистрация] [Подсчёт ккал блюда]

**Рекомендации по расширению:**

1. Добавить проверку существующей регистрации
2. Реализовать разные сценарии для новых и существующих пользователей
3. Добавить краткую справку о возможностях бота

Этот класс представляет собой типичную реализацию стартового обработчика для Telegram-ботов, сочетающую простоту и базовый функционал для начала взаимодействия с пользователем.