МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

КУРСОВАЯ РАБОТА

по дисциплине «Объектно-ориентированное программирование»

Тема: Создание бота в Telegram

Студент гр. 3351 Баринов А.А.

Преподаватель Кулагин М.В.

Санкт-Петербург
2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Баринов А.А.

Группа 3351

Тема работы: Создание бота в Telegram

Исходные данные:
Python 3.12, Используемое ПО PyCharm Community Edition 2022

Содержание пояснительной записки:
Перечисляются требуемые разделы пояснительной записки (разделы «Содержание», «Введение», «Заключение», «Список использованных источников» и т.д.)

Предполагаемый объем пояснительной записки:
Не менее 20 страниц.

Дата выдачи задания: 09.10.2024

Дата сдачи реферата: 25.12.2024

Дата защиты реферата: 26.12.2024

Студент Баринов А.А.

Преподаватель Кулагин М.В.

АННОТАЦИЯ

В рамках курсового проекта был разработан Telegram-бот для автоматической обработки изображений. Бот позволяет пользователям загружать фоны и шаблоны, комбинировать их и генерировать изображения в заданном количестве. В процессе работы использованы основные принципы объектно-ориентированного программирования: инкапсуляция, наследование и полиморфизм. Разработаны ключевые классы, такие как ImageManager для управления изображениями и AutoCreoBot для обработки пользовательских команд. Методы включают загрузку файлов, их обработку и отправку результатов пользователю. В результате был создан функциональный бот, который эффективно взаимодействует с пользователем, обрабатывает изображения и генерирует итоговые файлы. Полученные результаты показывают высокую модульность системы, которая может быть легко расширена и адаптирована под новые задачи.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной курсовой работы является разработка Telegram-бота, который позволяет пользователям загружать фото или ZIP-архивы с изображениями фонов и шаблонов, а затем генерировать новые изображения, комбинируя эти элементы. Бот предоставляет пользователю несколько опций: он может отправить изображения по одному или в виде ZIP-архива, а также очистить все текущие данные и начать процесс заново. С помощью бота можно получить как определенное количество генераций изображений, так и сгенерировать все наборы для всех фоноа.

Для реализации функциональности бота использованы библиотеки python-telegram-bot и Pillow, которые обеспечивают удобную работу с Telegram API и обработку изображений. Бот предоставляет интуитивно понятный интерфейс с кнопками, что делает его удобным для пользователей с различным уровнем подготовки.

В рамках работы были решены задачи разработки обработки различных типов файлов, генерации изображений и организации взаимодействия с пользователем через Telegram.

1. Создание Telegram бота

1.1. Получение токена через BotFather

Для создания и настройки Telegram-бота, который будет использоваться для получения данных о погоде, необходимо сначала получить токен, который используется для взаимодействия с API Telegram. Это можно сделать через официального бота BotFather. Ниже приведены шаги, которые были выполнены для получения токена и подключения бота к Telegram:

- После начала общения с BotFather была использована команда /start для начала работы
- Далее была введена команда /newbot, чтобы создать нового бота
- Был выбран уникальный username и имя бота.

После создания бота, BotFather выдал уникальный токен для доступа к API Telegram, который был использован для инициализации бота в коде.

1.2. Используемые библиотеки и фреймворки

Для разработки Telegram-бота, который выполняет обработку изображений и взаимодействует с пользователями, были использованы следующие библиотеки и фреймворки:

1. **python-telegram-bot**
   Это одна из самых популярных библиотек для работы с Telegram Bot API. Она предоставляет удобный и высокоуровневый интерфейс для создания и управления ботами в Telegram. С помощью этой библиотеки реализуются все взаимодействия с пользователем, включая обработку команд, кнопок и сообщений. Библиотека поддерживает синхронную и асинхронную обработку запросов, что позволяет эффективно работать с несколькими пользователями одновременно. В данном проекте библиотека используется для:
   - Обработки сообщений и команд, отправляемых пользователями.
   - Реализации кнопок и меню в Telegram.
   - Отправки файлов (изображений, архивов) и текстовых сообщений.
2. **Pillow (PIL Fork)**
   Pillow — это популярная библиотека для обработки изображений в Python. Библиотека предоставляет широкий набор инструментов для открытия, изменения и сохранения различных форматов изображений, таких как PNG, JPEG, BMP и другие. В данном проекте библиотека используется для:
   - Открытия и обработки изображений (фонов и шаблонов).
   - Наложения шаблонов на фоны с использованием прозрачности (альфа-канала).
   - Изменения размеров изображений для корректного наложения.
   - Сохранения полученных изображений в формате PNG.

3. **zipfile**
   Стандартная библиотека Python для работы с ZIP-архивами. Она предоставляет инструменты для создания, чтения и извлечения содержимого архивов. В данном проекте библиотека используется для:
   - Извлечения файлов из загруженных пользователем ZIP-архивов (например, с фонами или шаблонами).
   - Архивирования сгенерированных изображений перед отправкой пользователю.
4. **tempfile**
   Модуль Python для работы с временными файлами и директориями. В данном проекте используется для создания временных файлов для сохранения изображений и архивов, которые затем отправляются пользователю или используются для обработки. Это позволяет избежать необходимости хранить файлы на постоянной основе, что важно для обработки больших объемов данных.
5. **logging**
   Стандартная библиотека Python для ведения журналов (логирования). Она используется для отслеживания различных событий и ошибок в процессе работы бота, что помогает в отладке и мониторинге состояния приложения. В проекте она настроена для записи ошибок и информационных сообщений, таких как успешная обработка изображений или возникшие исключения.
6. **asyncio** (через python-telegram-bot)
   В библиотеке python-telegram-bot используется асинхронная обработка сообщений и запросов. Это позволяет боту обрабатывать несколько запросов от пользователей одновременно, не блокируя выполнение других операций, что особенно важно для ботов с большим количеством пользователей.

2. Описание структуры программы

2.1. Диаграмма вариантов использования системы

Рис. 1. Диаграмма вариантов использования системы (use cases).

2.2 UML-Диаграмма классов

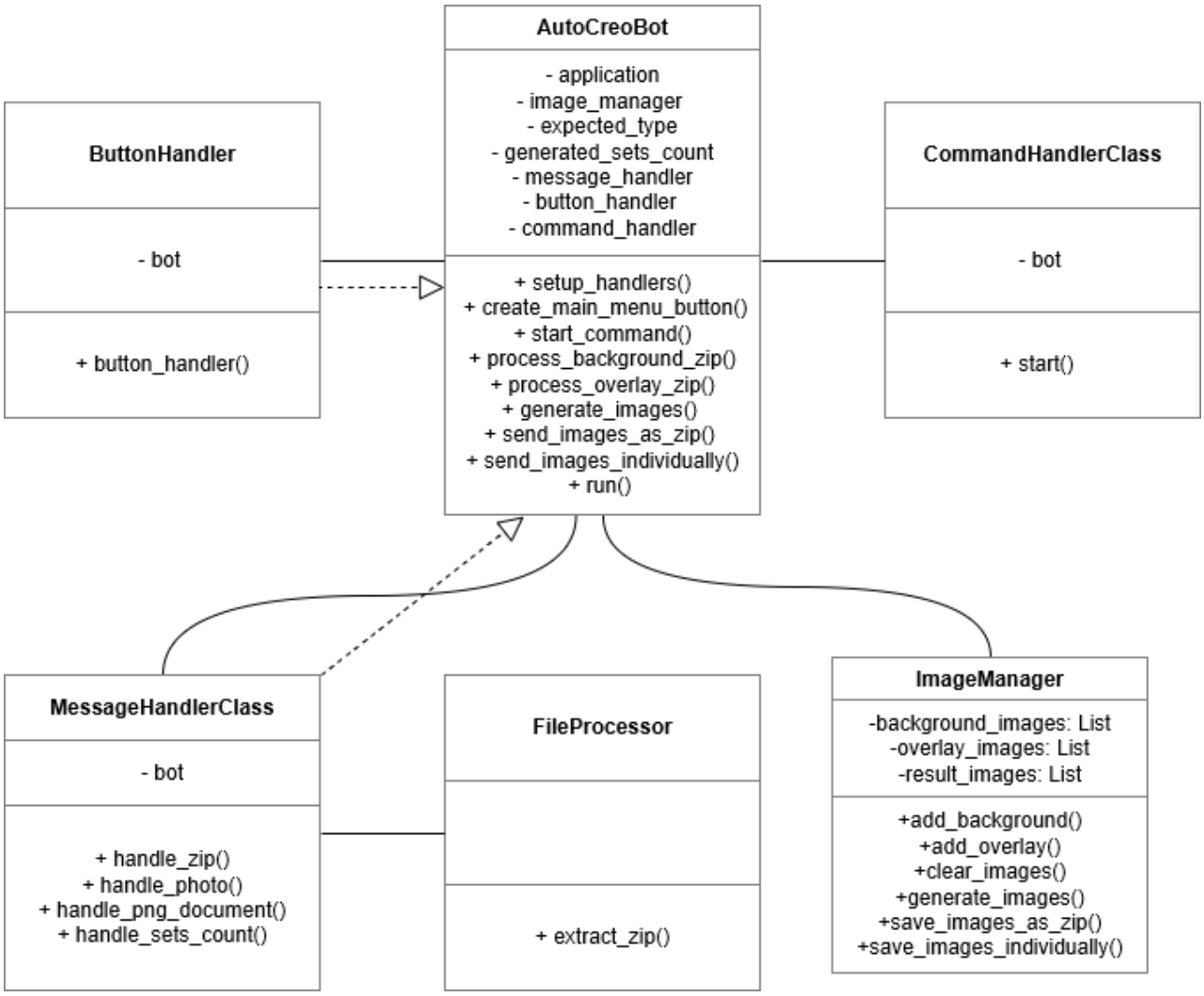

Рис. 2 – UML-Диаграмма классов

2.3 Спецификация классов.

Класс: AutoCreoBot

Поля	**Описание полей**
application ()	Управляет взаимодействием с API Telegram.
image_manager ()	Управляет изображениями.
expected_type (str or None)	Ожидаемый тип загружаемых файлов (например, background или overlay).

generated_sets_count (int)	Количество наборов изображений для генерации.
message_handler ()	Обрабатывает различные типы сообщений.
button_handler ()	Обрабатывает нажатия на кнопки.
command_handler ()	Обрабатывает команды от пользователя.

методы	**Описание методов**
setup_handlers()	Настройка обработчиков для команд и сообщений.
process_background_zip()	Обрабатывает загрузку и извлечение фонов из ZIP-архива.
process_overlay_zip()	Обрабатывает загрузку и извлечение наложений (шаблонов) из ZIP-архива.
generate_images()	Генерирует изображения с использованием фонов и наложений.
send_images_as_zip()	Отправляет сгенерированные изображения в виде ZIP-архива.
run()	Запускает бота.
create_main_menu_button()	Создает и возвращает клавиатуру с кнопками для главного меню.

Класс: ImageManager

Поля	**Описание полей**
background_images (list)	Список фоновых изображений.
overlay_images (list)	Список изображений наложений.
result_images (list)	Список сгенерированных изображений.

методы	**Описание методов**
add_background(image)	Добавляет фоновое изображение в список.
add_overlay(image)	Добавляет изображение наложения в список.
clear_images()	Очищает все списки изображений.
generate_images(sets_count: int)	Генерирует изображения, комбинируя фоны с наложениями.
save_images_as_zip()	Сохраняет все сгенерированные изображения в ZIP-архив.

save_images_individually()	Сохраняет все сгенерированные изображения по отдельности и возвращает их пути.

Класс: ImageManager

Поля	**Описание полей**
background_images (list)	Список фоновых изображений.
overlay_images (list)	Список изображений наложений.
result_images (list)	Список сгенерированных изображений.

методы	**Описание методов**
add_background(image)	Добавляет фоновое изображение в список.
add_overlay(image)	Добавляет изображение наложения в список.
clear_images()	Очищает все списки изображений.
generate_images(sets_count: int)	Генерирует изображения, комбинируя фоны с наложениями.
save_images_as_zip()	Сохраняет все сгенерированные изображения в ZIP-архив.
save_images_individually()	Сохраняет все сгенерированные изображения по отдельности и возвращает их пути.

Класс: MessageHandlerClass

Поля	**Описание полей**
bot ()	Ссылка на основной объект бота

методы	**Описание методов**
handle_zip()	Обрабатывает отправку ZIP-архива, извлекает его содержимое и передает данные соответствующему обработчику (фоны или наложения).
handle_photo()	Обрабатывает отправку фото и добавляет его как фон.
handle_png_document()	Обрабатывает отправку PNG-документа и добавляет его как наложение.
handle_sets_count()	Обрабатывает ввод количества наборов для генерации изображений.

Класс: ButtonHandler

Поля	Описание полей
bot ()	Ссылка на основной объект бота для доступа к его методам и данным.

методы	Описание методов
button_handler()	Обрабатывает нажатия на кнопки и выполняет соответствующие действия (например, загрузка фонов, генерация изображений, очистка данных и т.д.).

Класс: CommandHandlerClass

Поля	Описание полей
bot ()	Ссылка на основной объект бота для доступа к его методам и данным.

методы	Описание методов
start()	Обрабатывает команду /start, приветствует пользователя и предлагает действия.

Класс: FileProcessor

методы	Описание методов
extract_zip()	Извлекает содержимое ZIP-архива в указанную временную директорию.

2.4 Код классов объектной модели.

```
class AutoCreoBot:
    def __init__(self, token: str):
        self.application = Application.builder().token(token).build()
        self.image_manager = ImageManager()
        self.expected_type = None
        self.generated_sets_count = 0
        self.message_handler = MessageHandlerClass(self)
        self.button_handler = ButtonHandler(self)
        self.command_handler = CommandHandlerClass(self)
        self.setup_handlers()

    def setup_handlers(self):
```

```
        self.application.add_handler(CommandHandler("start", self.command_handler.start))
        self.application.add_handler(CallbackQueryHandler(self.button_handler.button_handler))
        self.application.add_handler(MessageHandler(filters.Document.MimeType("application/zip"), self.message_handler.handle_zip))
        self.application.add_handler(MessageHandler(filters.PHOTO, self.message_handler.handle_photo))
        self.application.add_handler(MessageHandler(filters.Document.MimeType("image/png"),
self.message_handler.handle_png_document))
        self.application.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, self.message_handler.handle_sets_count))

    def create_main_menu_button(self):
        keyboard = [[InlineKeyboardButton("Вернуться в главное меню", callback_data="back_to_main_menu")]]
        return InlineKeyboardMarkup(keyboard)

    async def start_command(self, update: Update, context: CallbackContext):
        await self.command_handler.start(update, context)

    async def process_background_zip(self, temp_dir: str, update: Update):
        found_backgrounds = False
        for filename in os.listdir(temp_dir):
            file_path = os.path.join(temp_dir, filename)
            if filename.lower().endswith(('png', 'jpg', 'jpeg')):
                try:
                    self.image_manager.add_background(Image.open(file_path))
                    found_backgrounds = True
                except Exception as e:
                    logger.error(f"Ошибка при открытии файла {filename}: {e}")

        if found_backgrounds:
            reply_markup = self.create_main_menu_button()
            await update.message.reply_text(f"Фоны получены, всего фонов: {len(self.image_manager.background_images)}.",
                                            reply_markup=reply_markup)
        else:
            reply_markup = self.create_main_menu_button()
            await update.message.reply_text("Не было найдено фонов в архиве.", reply_markup=reply_markup)

    async def process_overlay_zip(self, temp_dir: str, update: Update):
        found_overlay = False
        for filename in os.listdir(temp_dir):
            if filename.lower().endswith('.png'):
                try:
                    file_path = os.path.join(temp_dir, filename)
                    self.image_manager.add_overlay(Image.open(file_path))
                    found_overlay = True
                except Exception as e:
                    logger.error(f"Ошибка при обработке файла {filename}: {e}")
        if found_overlay:
            reply_markup = self.create_main_menu_button()
            await update.message.reply_text(f"Шаблоны (PNG) получены: {len(self.image_manager.overlay_images)}.",
                                            reply_markup=reply_markup)
        else:
            reply_markup = self.create_main_menu_button()
            await update.message.reply_text("В архиве не найдено PNG изображений.", reply_markup=reply_markup)

    async def generate_images(self, message):
        sets_to_generate = self.generated_sets_count if self.generated_sets_count > 0 else len(
            self.image_manager.background_images)
        self.image_manager.generate_images(sets_to_generate)

        keyboard = [
            [InlineKeyboardButton("Отправить как ZIP файл", callback_data="send_as_zip")],
            [InlineKeyboardButton("Отправить по очереди", callback_data="send_individually")]
        ]
        reply_markup = InlineKeyboardMarkup(keyboard)
        await message.reply_text("Изображения успешно сгенерированы. Выберите способ отправки:",
                                 reply_markup=reply_markup)

    async def send_images_individually(self, message):
        temp_files = self.image_manager.save_images_individually()
        for file_path in temp_files:
            await message.reply_document(document=open(file_path, 'rb'), filename=os.path.basename(file_path))
            os.remove(file_path)
```

```
        # Отправляем сообщение о завершении и добавляем кнопку возврата в главное меню
        keyboard = [[InlineKeyboardButton("Вернуться в главное меню", callback_data="back_to_main_menu")]]
        reply_markup = InlineKeyboardMarkup(keyboard)
        await message.reply_text("Фото успешно отправлены! 📡", reply_markup=reply_markup)

    def run(self):
        self.application.run_polling()

class ImageManager:

    def __init__(self):
        self.background_images = []
        self.overlay_images = []
        self.result_images = []

    def add_background(self, image):
        self.background_images.append(image)

    def add_overlay(self, image):
        self.overlay_images.append(image)

    def clear_images(self):
        self.background_images.clear()
        self.overlay_images.clear()
        self.result_images.clear()

    def generate_images(self, sets_count: int):
        self.result_images = []
        for i in range(sets_count):
            bg = self.background_images[i % len(self.background_images)]
            for overlay in self.overlay_images:
                bg = bg.convert("RGBA")
                overlay = overlay.convert("RGBA")
                overlay_resized = overlay.resize(bg.size, Image.Resampling.LANCZOS)
                result = Image.alpha_composite(bg, overlay_resized)
                self.result_images.append(result)

    def save_images_as_zip(self, temp_zip):
        with zipfile.ZipFile(temp_zip, 'w') as zipf:
            for idx, result in enumerate(self.result_images):
                img_filename = f"image_{idx + 1}.png"
                with tempfile.NamedTemporaryFile(delete=False, suffix=".png") as temp_file:
                    result.save(temp_file, "PNG")
                    temp_file.seek(0)
                    zipf.write(temp_file.name, img_filename)

    def save_images_individually(self):
        temp_files = []
        for result in self.result_images:
            with tempfile.NamedTemporaryFile(delete=False, suffix=".png") as temp_file:
                result.save(temp_file, "PNG")
                temp_files.append(temp_file.name)
        return temp_files

class MessageHandlerClass:

    def __init__(self, bot):
        self.bot = bot

    async def handle_zip(self, update: Update, context: CallbackContext):
        file = update.message.document
        file_info = await file.get_file()
        file_bytes = await file_info.download_as_bytearray()

        with tempfile.TemporaryDirectory() as temp_dir:
            if not FileProcessor.extract_zip(file_bytes, temp_dir):
                await update.message.reply_text("Ошибка: это не ZIP файл или файл поврежден.")
                return

            if self.bot.expected_type == 'background':
                await self.bot.process_background_zip(temp_dir, update)
```

```
            elif self.bot.expected_type == 'overlay':
                await self.bot.process_overlay_zip(temp_dir, update)

    async def handle_photo(self, update: Update, context: CallbackContext):
        file = update.message.photo[-1]
        file_info = await file.get_file()
        file_path = os.path.join(tempfile.gettempdir(), f"temp_{file.file_id}.jpg")
        await file_info.download_to_drive(file_path)

        if self.bot.expected_type == 'background':
            self.bot.image_manager.add_background(Image.open(file_path))
            reply_markup = self.bot.create_main_menu_button()
            await update.message.reply_text("Фон получен.", reply_markup=reply_markup)
        else:
            reply_markup = self.bot.create_main_menu_button()
            await update.message.reply_text("Ошибка: фото принимаются только для фонов.", reply_markup=reply_markup)

    async def handle_png_document(self, update: Update, context: CallbackContext):
        if self.bot.expected_type != 'overlay':
            reply_markup = self.bot.create_main_menu_button()
            await update.message.reply_text("Ошибка: PNG файлы принимаются только для шаблонов.",
                                            reply_markup=reply_markup)
            return

        file = update.message.document
        file_info = await file.get_file()
        file_path = os.path.join(tempfile.gettempdir(), f"overlay_{file.file_id}.png")
        await file_info.download_to_drive(file_path)

        self.bot.image_manager.add_overlay(Image.open(file_path))
        reply_markup = self.bot.create_main_menu_button()
        await update.message.reply_text("Шаблон (PNG) получен.", reply_markup=reply_markup)

    async def handle_sets_count(self, update: Update, context: CallbackContext):
        try:
            sets_count = int(update.message.text)
            if sets_count > 0:
                if sets_count > len(self.bot.image_manager.background_images):
                    reply_markup = self.bot.create_main_menu_button()
                    await update.message.reply_text(
                        f"Ошибка: недостаточно фонов для создания {sets_count} наборов. Пожалуйста, загрузите больше фонов.",
                        reply_markup=reply_markup
                    )
                else:
                    self.bot.generated_sets_count = sets_count
                    await update.message.reply_text(
                        f"Количество наборов установлено: {sets_count}. Генерация начнется.")
                    await self.bot.generate_images(update.message)
            else:
                await update.message.reply_text("Пожалуйста, введите положительное число.")
        except ValueError:
            await update.message.reply_text("Ошибка: Пожалуйста, введите число.")

class ButtonHandler:

    def __init__(self, bot):
        self.bot = bot

    async def button_handler(self, update: Update, context: CallbackContext):
        query = update.callback_query
        await query.answer()
        message = query.message

        if query.data == "send_background":
            self.bot.expected_type = 'background'
            reply_markup = self.bot.create_main_menu_button()
            await message.edit_text("Пожалуйста, отправьте ZIP архив с фонами или изображения (форматы: PNG, JPG).",
                                    reply_markup=reply_markup)

        elif query.data == "send_overlay":
            self.bot.expected_type = 'overlay'
```

```
        reply_markup = self.bot.create_main_menu_button()
        await message.edit_text("Пожалуйста, отправьте ZIP архив с шаблонами или изображения (форматы: PNG, JPG).",
                                reply_markup=reply_markup)

    elif query.data == "generate_images":
        if self.bot.image_manager.background_images and self.bot.image_manager.overlay_images:
            keyboard = [
                [InlineKeyboardButton("Генерировать для всех фонов", callback_data="generate_all_sets")],
                [InlineKeyboardButton("Введите количество наборов", callback_data="input_sets_count")]
            ]
            reply_markup = InlineKeyboardMarkup(keyboard)
            await message.edit_text("Как вы хотите генерировать изображения?", reply_markup=reply_markup)
        else:
            await message.edit_text(
                "Не все изображения получены. Пожалуйста, отправьте архивы с фонами и шаблонами.")
            reply_markup = self.bot.create_main_menu_button()
            await message.reply_text("Вы можете вернуться в главное меню.", reply_markup=reply_markup)

    elif query.data == "clear_all":
        self.bot.image_manager.clear_images()
        await message.edit_text("Все изображения и настройки очищены. Пожалуйста, отправьте новые архивы.")
        reply_markup = self.bot.create_main_menu_button()
        await message.reply_text("Вы можете вернуться в главное меню.", reply_markup=reply_markup)

    elif query.data == "back_to_main_menu":
        await self.bot.start_command(update, context)

    elif query.data == "send_as_zip":
        await self.bot.send_images_as_zip(message)

    elif query.data == "send_individually":
        await self.bot.send_images_individually(message)

    elif query.data == "generate_all_sets":
        self.bot.generated_sets_count = len(self.bot.image_manager.background_images)
        await self.bot.generate_images(message)

    elif query.data == "input_sets_count":
        await message.edit_text("Введите количество наборов для генерации:")
```

3. Описание интерфейса пользователя программы.

Для того чтобы начать использование бота, необходимо написать в чат команду /start

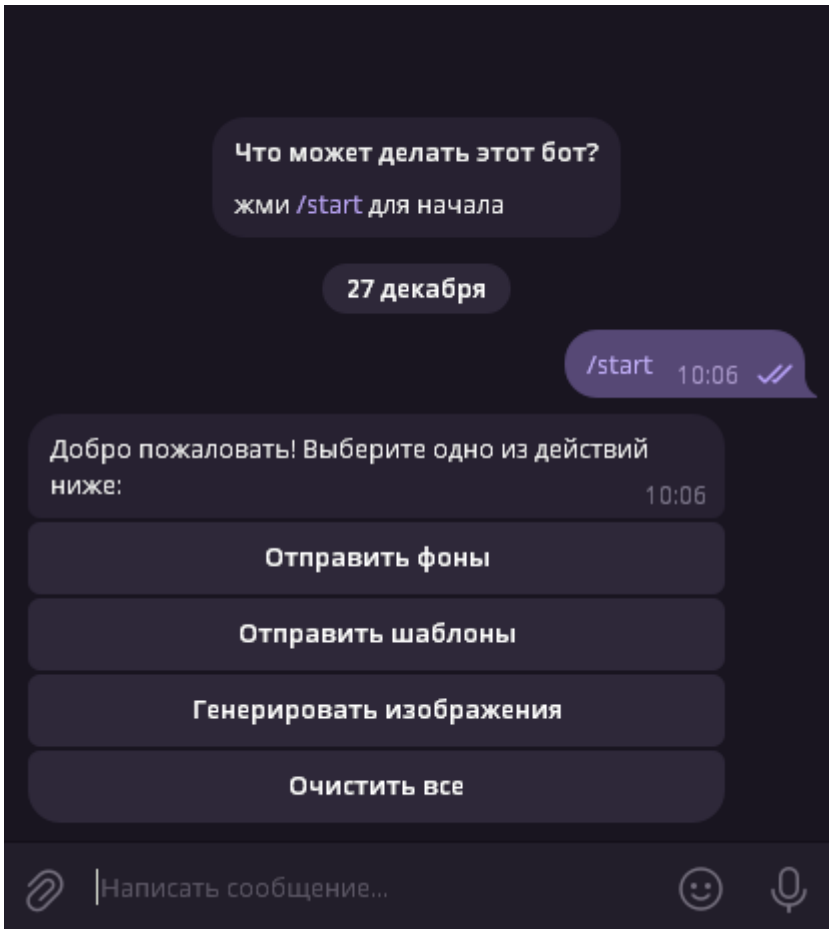

Рисунок 3 – Ввод команды /start

Далее пользователю необходимо отправить фотографии для фонов или шаблонов:

- Для фонов пользователь может отправить фото как в сжатом формате через чат (рис.4), либо через zip файл (рис.5).

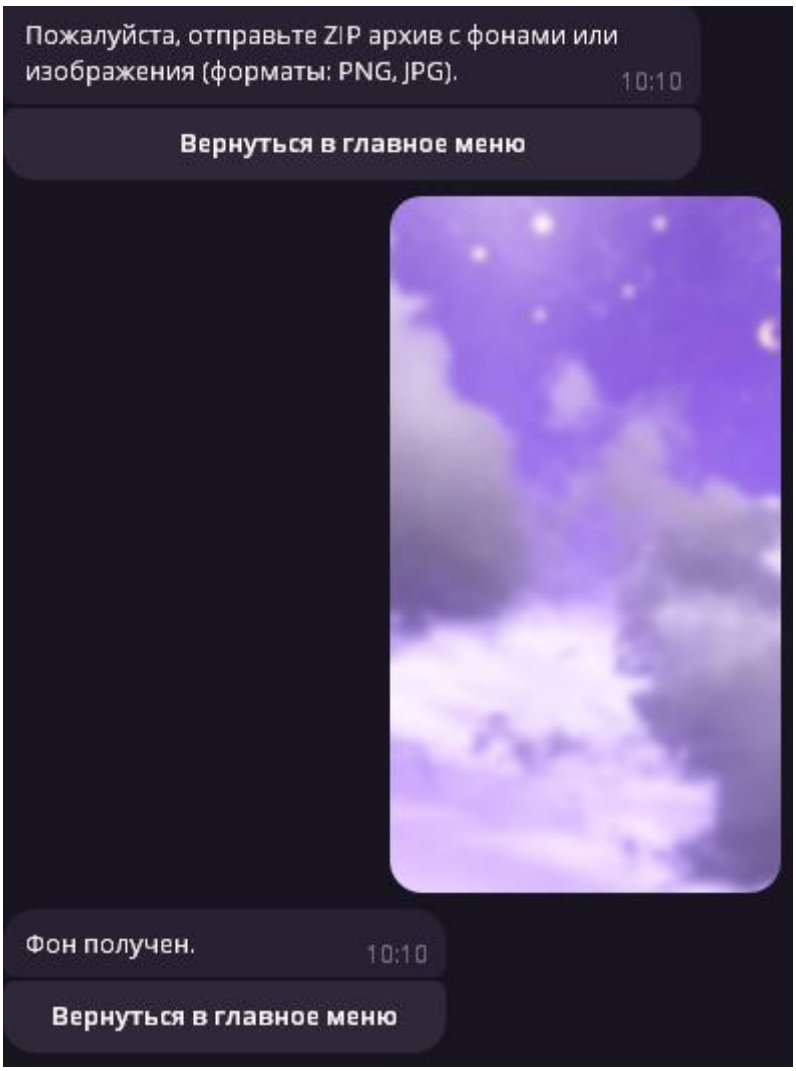

Рисунок 4 – Отправка фона в формате изображения

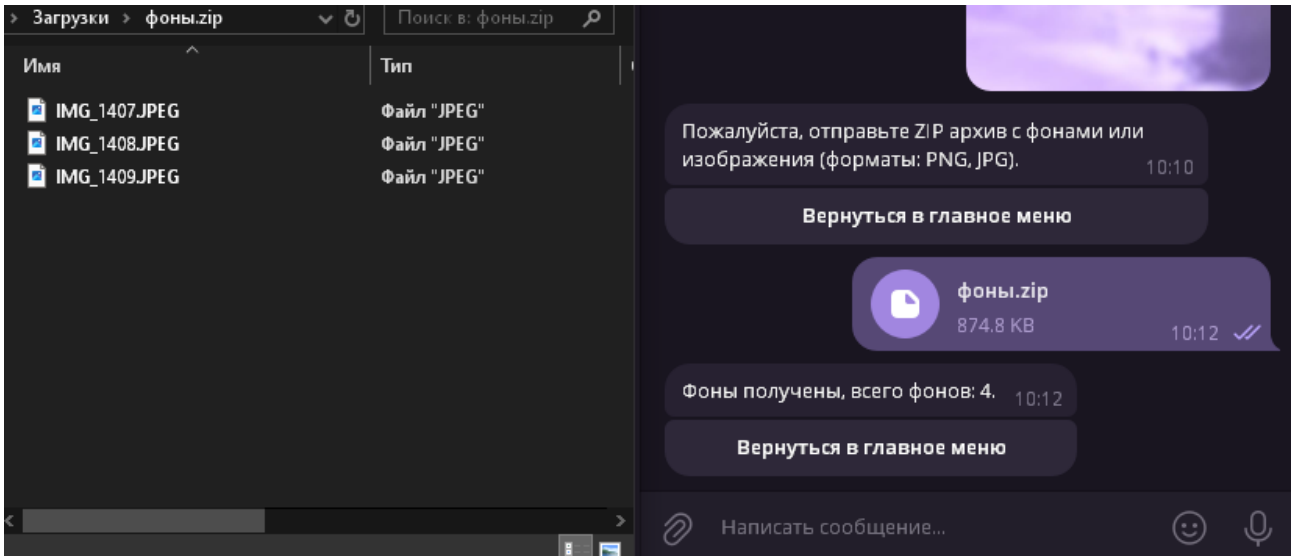

Рисунок 5 – Отправка фонов в формате zip.

- Для шаблонов пользователь может отправить фото как в несжатом PNG RGBA формате с использованием прозрачного фона через чат, либо через zip файл, аналогично, как на рисунке 5, но с соблюдением формата фото PNG (рис.6).

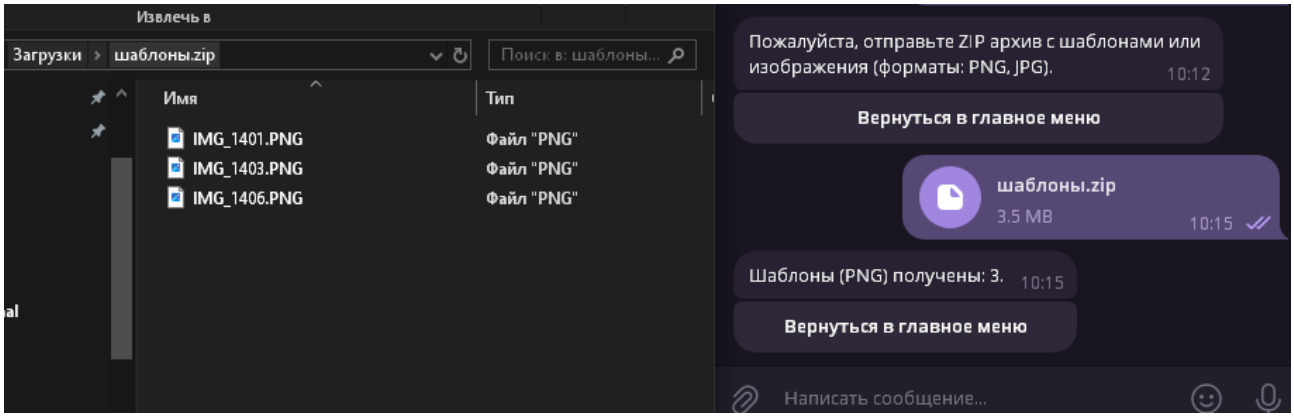

Рисунок 6 – Отправка шаблонов в формате zip.

Далее пользователь переходит в главное меню и нажимает кнопку «генерировать изображение», после нажатия пользователю будет предложена генерация наборов для всех фонов или же генерация определенного количества наборов, где 1 набор соответствует 1 генерации всех шаблонов для 1 фона. (рис. 7)

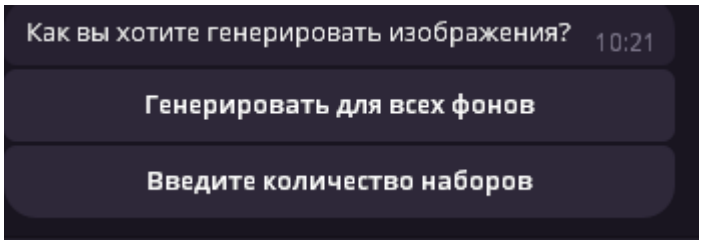

Рисунок 7 – Выбор количества генераций.

После выбора количества генераций, пользователь выбирает в каком формате будут присланы сгенерированные изображения. Формат: Zip файл, фото, отправленные по очереди (рис. 8).

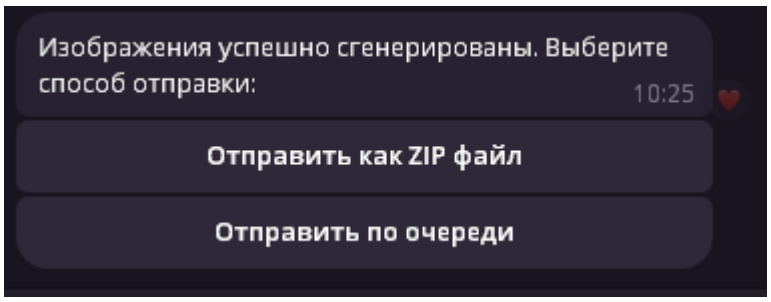

Рисунок 8 – Выбор формата генераций.

- При выборе Zip файла (Рис. 9):

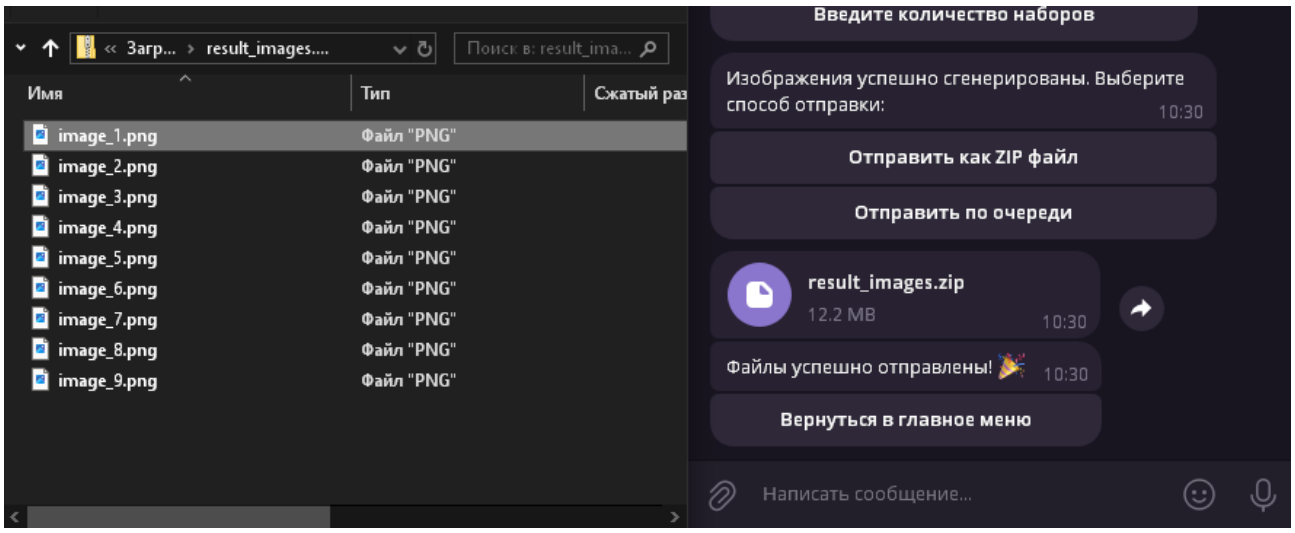

Рисунок 9 – Отправка в Zip файле.

- При выборе отправки по очереди (Рис. 10):

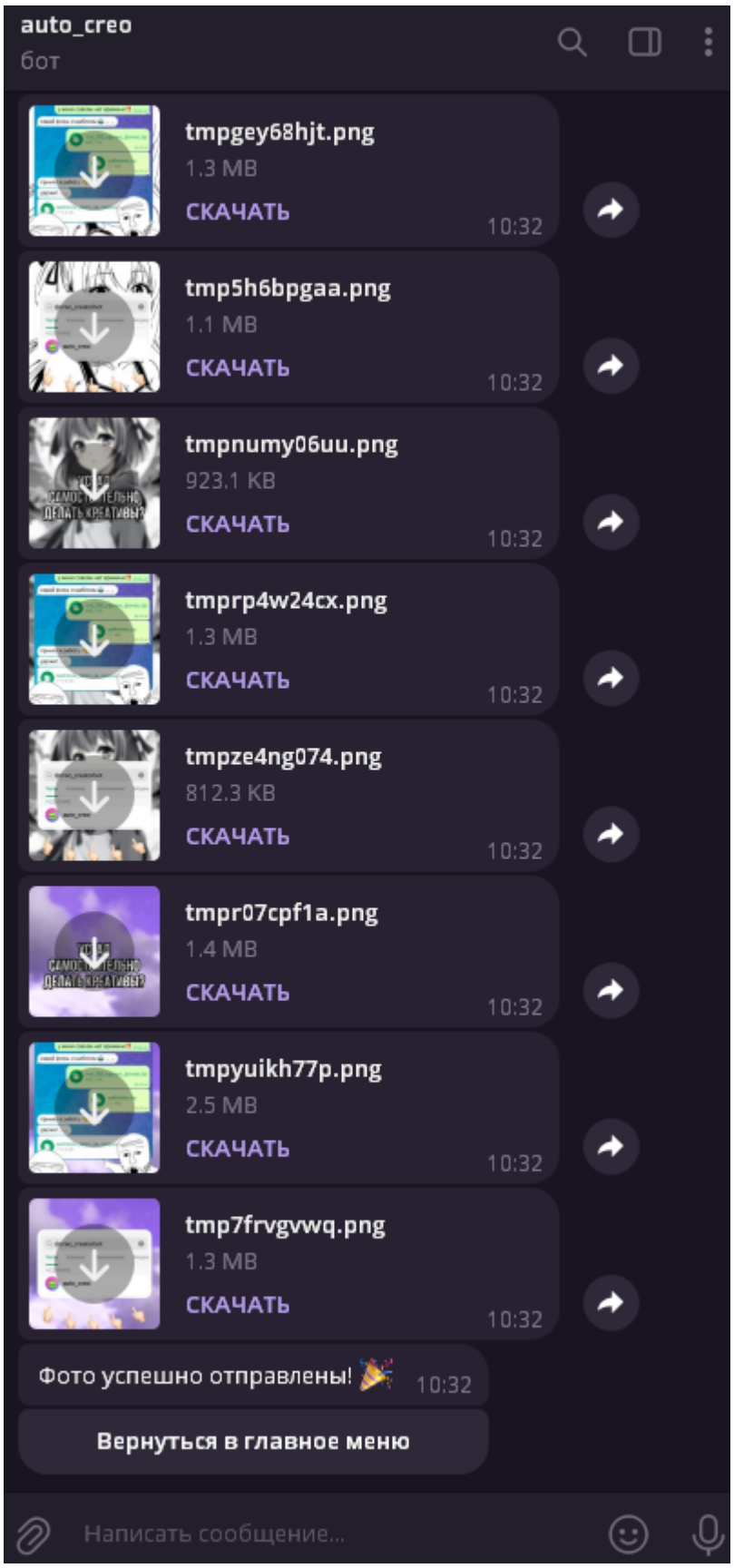

Рисунок 10 – Отправка в Zip файле.

В любой момент пользователю может понадобиться очистить хранилище с фонами и шаблонами, в таком случае пользователь нажимает кнопку «Очистить все» (Рис. 11):

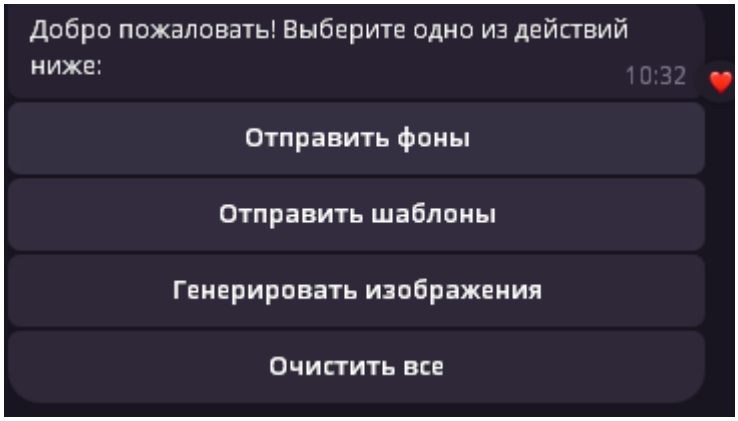

Рисунок 11 –Кнопка очистки.

После нажатия происходит очистка, бот предлагает вернуться в главное меню (рис. 12):

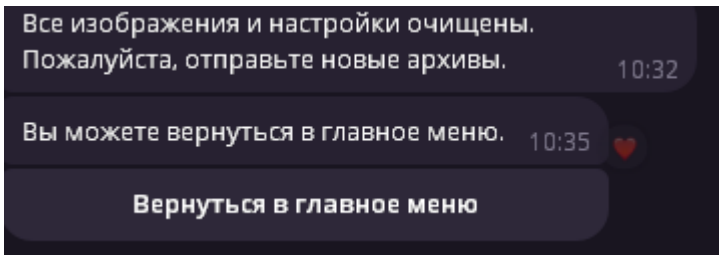

Рисунок 12 – Очистка.

Пример сгенерированных изображений (рис. 13):

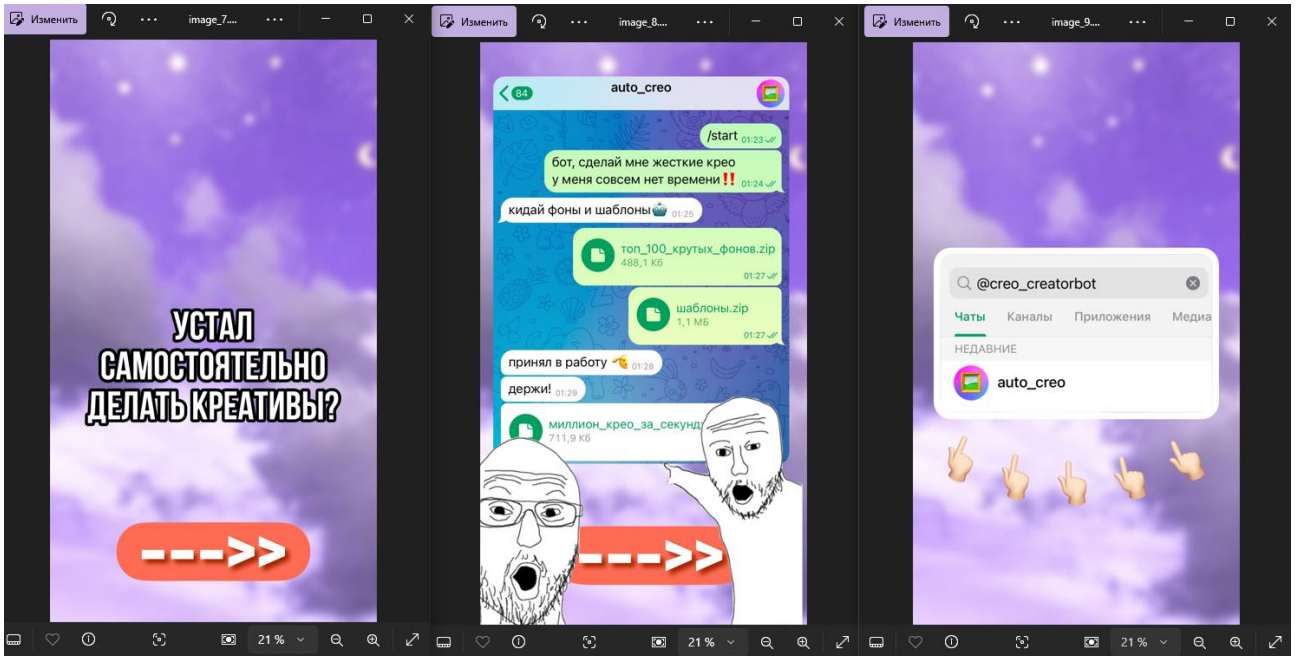

Рисунок 13 – Пример генерации.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была разработана и реализована объектно-ориентированная модель для Telegram-бота, предназначенного для обработки изображений, включая фоны и шаблоны. Процесс разработки включал в себя проектирование классов, их взаимодействие, а также определение необходимых методов и полей для эффективной работы с изображениями.
ссылка на git: https://github.com/barinovartur/OOP_CW1.git

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. telegram.ext package Documentation (telegram.ext package - python-telegram-bot v21.9)

2. Практическая стеганография. Скрытие информации в изображениях PNG/ Александр Якубович (Практическая стеганография. Скрытие информации в изображениях PNG / Хабр)

3. Паттерны объектно-ориентированного проектирования/ Гамма Эрих, Хелм Ричард, Влиссидес Джон\ Издательство Питер.

ПРИЛОЖЕНИЕ А

AUTO_CREO

MAIN.PY

```
from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup
from telegram.ext import Application, CommandHandler, MessageHandler, filters, CallbackContext, CallbackQueryHandler
from PIL import Image
import piexif
import io
import os
import tempfile
import logging
from ImageManager import ImageManager
from MessageHandler import MessageHandlerClass
from ButtonHandler import ButtonHandler
from CommandHandlerClass import CommandHandlerClass


# Настройка логирования
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
                    level=logging.INFO)
logger = logging.getLogger(__name__)

class AutoCreoBot:
    def __init__(self, token: str):
        self.application = Application.builder().token(token).build()
        self.image_manager = ImageManager()
        self.expected_type = None
        self.generated_sets_count = 0
        self.message_handler = MessageHandlerClass(self)
        self.button_handler = ButtonHandler(self)
        self.command_handler = CommandHandlerClass(self)
        self.setup_handlers()

    def setup_handlers(self):
        self.application.add_handler(CommandHandler("start", self.command_handler.start))
        self.application.add_handler(CallbackQueryHandler(self.button_handler.button_handler))
        self.application.add_handler(MessageHandler(filters.Document.MimeType("application/zip"), self.message_handler.handle_zip))
        self.application.add_handler(MessageHandler(filters.PHOTO, self.message_handler.handle_photo))
        self.application.add_handler(MessageHandler(filters.Document.MimeType("image/png"),
self.message_handler.handle_png_document))
        self.application.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, self.message_handler.handle_sets_count))

    def create_main_menu_button(self):
        keyboard = [[InlineKeyboardButton("Вернуться в главное меню", callback_data="back_to_main_menu")]]
        return InlineKeyboardMarkup(keyboard)

    async def start_command(self, update: Update, context: CallbackContext):
        await self.command_handler.start(update, context)

    async def process_background_zip(self, temp_dir: str, update: Update):
        found_backgrounds = False
        for filename in os.listdir(temp_dir):
            file_path = os.path.join(temp_dir, filename)
            if filename.lower().endswith(('png', 'jpg', 'jpeg')):
                try:
                    self.image_manager.add_background(Image.open(file_path))
                    found_backgrounds = True
                except Exception as e:
                    logger.error(f"Ошибка при открытии файла {filename}: {e}")

        if found_backgrounds:
            reply_markup = self.create_main_menu_button()
            await update.message.reply_text(f"Фоны получены, всего фонов: {len(self.image_manager.background_images)}.",
                                            reply_markup=reply_markup)
        else:
            reply_markup = self.create_main_menu_button()
```

```
        await update.message.reply_text("Не было найдено фонов в архиве.", reply_markup=reply_markup)

async def process_overlay_zip(self, temp_dir: str, update: Update):
    found_overlay = False
    for filename in os.listdir(temp_dir):
        if filename.lower().endswith('.png'):
            try:
                file_path = os.path.join(temp_dir, filename)
                self.image_manager.add_overlay(Image.open(file_path))
                found_overlay = True
            except Exception as e:
                logger.error(f"Ошибка при обработке файла {filename}: {e}")
    if found_overlay:
        reply_markup = self.create_main_menu_button()
        await update.message.reply_text(f"Шаблоны (PNG) получены: {len(self.image_manager.overlay_images)}.",
                                        reply_markup=reply_markup)
    else:
        reply_markup = self.create_main_menu_button()
        await update.message.reply_text("В архиве не найдено PNG изображений.", reply_markup=reply_markup)

async def generate_images(self, message):
    sets_to_generate = self.generated_sets_count if self.generated_sets_count > 0 else len(
        self.image_manager.background_images)
    self.image_manager.generate_images(sets_to_generate)

    keyboard = [
        [InlineKeyboardButton("Отправить как ZIP файл", callback_data="send_as_zip")],
        [InlineKeyboardButton("Отправить по очереди", callback_data="send_individually")]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await message.reply_text("Изображения успешно сгенерированы. Выберите способ отправки:",
                             reply_markup=reply_markup)

def remove_metadata(self, image: Image.Image) -> Image.Image:
    # Для PNG просто игнорируем метаданные
    clean_image = image.copy()
    if 'exif' in clean_image.info:
        del clean_image.info['exif']  # Удаляем EXIF данные, если они есть
    return clean_image

def remove_exif_from_jpeg(self, image: Image.Image) -> Image.Image:
    if image.format == "JPEG":
        # Получаем EXIF данные (если они есть)
        exif_dict = piexif.load(image.info.get("exif", b""))

        # Убираем EXIF данные
        exif_dict['0th'] = {}
        exif_dict['Exif'] = {}
        exif_dict['GPS'] = {}
        exif_dict['1st'] = {}
        exif_bytes = piexif.dump(exif_dict)

        # Создаем новое изображение без EXIF
        clean_image = image.copy()
        clean_image.save(io.BytesIO(), "JPEG", exif=exif_bytes)  # Сохраняем JPEG без EXIF
        return clean_image
    else:
        return image

def remove_metadata_from_all(self):

    self.result_images = [self.remove_metadata(result) for result in self.result_images]
    self.result_images = [self.remove_exif_from_jpeg(result) for result in self.result_images]

async def send_images_as_zip(self, message):
    with tempfile.NamedTemporaryFile(delete=False, suffix=".zip") as temp_zip:
        self.image_manager.save_images_as_zip(temp_zip.name)
        await message.reply_document(document=open(temp_zip.name, 'rb'), filename="result_images.zip")
    # Отправляем сообщение о завершении и добавляем кнопку возврата в главное меню
    keyboard = [[InlineKeyboardButton("Вернуться в главное меню", callback_data="back_to_main_menu")]]
    reply_markup = InlineKeyboardMarkup(keyboard)
```

```
        await message.reply_text("Файлы успешно отправлены! 🚀", reply_markup=reply_markup)

    async def send_images_individually(self, message):
        temp_files = self.image_manager.save_images_individually()
        for file_path in temp_files:
            await message.reply_document(document=open(file_path, 'rb'), filename=os.path.basename(file_path))
            os.remove(file_path)

        # Отправляем сообщение о завершении и добавляем кнопку возврата в главное меню
        keyboard = [[InlineKeyboardButton("Вернуться в главное меню", callback_data="back_to_main_menu")]]
        reply_markup = InlineKeyboardMarkup(keyboard)
        await message.reply_text("Фото успешно отправлены! 🚀", reply_markup=reply_markup)

    def run(self):
        self.application.run_polling()


if __name__ == "__main__":
    with open('api_token.txt', 'r') as file:
        api_token = file.read().strip()
    bot = AutoCreoBot(api_token)
    bot.run()
```

ImageManager.py

```
from PIL import Image
import zipfile

import tempfile

class ImageManager:

    def __init__(self):
        self.background_images = []
        self.overlay_images = []
        self.result_images = []

    def add_background(self, image):
        self.background_images.append(image)

    def add_overlay(self, image):
        self.overlay_images.append(image)

    def clear_images(self):
        self.background_images.clear()
        self.overlay_images.clear()
        self.result_images.clear()

    def generate_images(self, sets_count: int):
        self.result_images = []
        for i in range(sets_count):
            bg = self.background_images[i % len(self.background_images)]
            for overlay in self.overlay_images:
                bg = bg.convert("RGBA")
                overlay = overlay.convert("RGBA")
                overlay_resized = overlay.resize(bg.size, Image.Resampling.LANCZOS)
                result = Image.alpha_composite(bg, overlay_resized)
                self.result_images.append(result)

    def save_images_as_zip(self, temp_zip):
        with zipfile.ZipFile(temp_zip, 'w') as zipf:
            for idx, result in enumerate(self.result_images):
                img_filename = f"image_{idx + 1}.png"
                with tempfile.NamedTemporaryFile(delete=False, suffix=".png") as temp_file:
                    result.save(temp_file, "PNG")
                    temp_file.seek(0)
                    zipf.write(temp_file.name, img_filename)

    def save_images_individually(self):
```

```
        temp_files = []
        for result in self.result_images:
            with tempfile.NamedTemporaryFile(delete=False, suffix=".png") as temp_file:
                result.save(temp_file, "PNG")
                temp_files.append(temp_file.name)
        return temp_files
```

MassegeHandler

```
from telegram import Update
from telegram.ext import CallbackContext
from PIL import Image
import os
import tempfile
from FileProcessor import FileProcessor

class MessageHandlerClass:

    def __init__(self, bot):
        self.bot = bot

    async def handle_zip(self, update: Update, context: CallbackContext):
        file = update.message.document
        file_info = await file.get_file()
        file_bytes = await file_info.download_as_bytearray()

        with tempfile.TemporaryDirectory() as temp_dir:
            if not FileProcessor.extract_zip(file_bytes, temp_dir):
                await update.message.reply_text("Ошибка: это не ZIP файл или файл поврежден.")
                return

            if self.bot.expected_type == 'background':
                await self.bot.process_background_zip(temp_dir, update)
            elif self.bot.expected_type == 'overlay':
                await self.bot.process_overlay_zip(temp_dir, update)

    async def handle_photo(self, update: Update, context: CallbackContext):
        file = update.message.photo[-1]
        file_info = await file.get_file()
        file_path = os.path.join(tempfile.gettempdir(), f"temp_{file.file_id}.jpg")
        await file_info.download_to_drive(file_path)

        if self.bot.expected_type == 'background':
            self.bot.image_manager.add_background(Image.open(file_path))
            reply_markup = self.bot.create_main_menu_button()
            await update.message.reply_text("Фон получен.", reply_markup=reply_markup)
        else:
            reply_markup = self.bot.create_main_menu_button()
            await update.message.reply_text("Ошибка: фото принимаются только для фонов.", reply_markup=reply_markup)

    async def handle_png_document(self, update: Update, context: CallbackContext):
        if self.bot.expected_type != 'overlay':
            reply_markup = self.bot.create_main_menu_button()
            await update.message.reply_text("Ошибка: PNG файлы принимаются только для шаблонов.",
                                            reply_markup=reply_markup)
            return

        file = update.message.document
        file_info = await file.get_file()
        file_path = os.path.join(tempfile.gettempdir(), f"overlay_{file.file_id}.png")
        await file_info.download_to_drive(file_path)

        self.bot.image_manager.add_overlay(Image.open(file_path))
        reply_markup = self.bot.create_main_menu_button()
        await update.message.reply_text("Шаблон (PNG) получен.", reply_markup=reply_markup)

    async def handle_sets_count(self, update: Update, context: CallbackContext):
        try:
            sets_count = int(update.message.text)
```

```
            if sets_count > 0:
                if sets_count > len(self.bot.image_manager.background_images):
                    reply_markup = self.bot.create_main_menu_button()
                    await update.message.reply_text(
                        f"Ошибка: недостаточно фонов для создания {sets_count} наборов. Пожалуйста, загрузите больше фонов.",
                        reply_markup=reply_markup
                    )
                else:
                    self.bot.generated_sets_count = sets_count
                    await update.message.reply_text(
                        f"Количество наборов установлено: {sets_count}. Генерация начнется.")
                    await self.bot.generate_images(update.message)
            else:
                await update.message.reply_text("Пожалуйста, введите положительное число.")
        except ValueError:
            await update.message.reply_text("Ошибка: Пожалуйста, введите число.")
```

ButtonHendler.py

```
from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup
from telegram.ext import CallbackContext


class ButtonHandler:

    def __init__(self, bot):
        self.bot = bot

    async def button_handler(self, update: Update, context: CallbackContext):
        query = update.callback_query
        await query.answer()
        message = query.message

        if query.data == "send_background":
            self.bot.expected_type = 'background'
            reply_markup = self.bot.create_main_menu_button()
            await message.edit_text("Пожалуйста, отправьте ZIP архив с фонами или изображения (форматы: PNG, JPG).",
                                    reply_markup=reply_markup)

        elif query.data == "send_overlay":
            self.bot.expected_type = 'overlay'
            reply_markup = self.bot.create_main_menu_button()
            await message.edit_text("Пожалуйста, отправьте ZIP архив с шаблонами или изображения (форматы: PNG, JPG).",
                                    reply_markup=reply_markup)

        elif query.data == "generate_images":
            if self.bot.image_manager.background_images and self.bot.image_manager.overlay_images:
                keyboard = [
                    [InlineKeyboardButton("Генерировать для всех фонов", callback_data="generate_all_sets")],
                    [InlineKeyboardButton("Введите количество наборов", callback_data="input_sets_count")]
                ]
                reply_markup = InlineKeyboardMarkup(keyboard)
                await message.edit_text("Как вы хотите генерировать изображения?", reply_markup=reply_markup)
            else:
                await message.edit_text(
                    "Не все изображения получены. Пожалуйста, отправьте архивы с фонами и шаблонами.")
                reply_markup = self.bot.create_main_menu_button()
                await message.reply_text("Вы можете вернуться в главное меню.", reply_markup=reply_markup)

        elif query.data == "clear_all":
            self.bot.image_manager.clear_images()
            await message.edit_text("Все изображения и настройки очищены. Пожалуйста, отправьте новые архивы.")
            reply_markup = self.bot.create_main_menu_button()
            await message.reply_text("Вы можете вернуться в главное меню.", reply_markup=reply_markup)

        elif query.data == "back_to_main_menu":
            await self.bot.start_command(update, context)

        elif query.data == "send_as_zip":
```

```
            await self.bot.send_images_as_zip(message)

        elif query.data == "send_individually":
            await self.bot.send_images_individually(message)

        elif query.data == "generate_all_sets":
            self.bot.generated_sets_count = len(self.bot.image_manager.background_images)
            await self.bot.generate_images(message)

        elif query.data == "input_sets_count":
            await message.edit_text("Введите количество наборов для генерации:")
```

CommanfHendlerClass.py

```
from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup
from telegram.ext import CallbackContext


class CommandHandlerClass:

    def __init__(self, bot):
        self.bot = bot

    async def start(self, update: Update, context: CallbackContext):
        keyboard = [
            [InlineKeyboardButton("Отправить фоны", callback_data="send_background")],
            [InlineKeyboardButton("Отправить шаблоны", callback_data="send_overlay")],
            [InlineKeyboardButton("Генерировать изображения", callback_data="generate_images")],
            [InlineKeyboardButton("Очистить все", callback_data="clear_all")],
        ]
        reply_markup = InlineKeyboardMarkup(keyboard)
        if update.message:
            await update.message.reply_text("Добро пожаловать! Выберите одно из действий ниже:",
                                            reply_markup=reply_markup)
        elif update.callback_query:
            await update.callback_query.message.edit_text("Добро пожаловать! Выберите одно из действий ниже:",
                                                          reply_markup=reply_markup)
```

FileProcessor.py

```
import zipfile
from io import BytesIO
import logging


# Настройка логирования
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
                    level=logging.INFO)
logger = logging.getLogger(__name__)

class FileProcessor:

    @staticmethod
    def extract_zip(file_bytes: bytes, temp_dir: str):
        try:
            with zipfile.ZipFile(BytesIO(file_bytes), 'r') as zip_ref:
                zip_ref.extractall(temp_dir)
            return True
        except zipfile.BadZipFile:
            logger.error("Ошибка: это не ZIP файл или файл поврежден.")
            return False
```