Федеральное государственное автономное образовательное учреждение высшего образования
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий
Кафедра «Информатика и информационные технологии»

Направление подготовки/ специальность: «Автоматизированные системы обработки информации и управления»

ОТЧЕТ

по проектной практике

Студент: Люкин Евгений Алексеевич Группа: 243-331

Место прохождения практики: Московский Политех, кафедра «Информатика и информационные технологии»

Отчет принят с оценкой _____ Дата _____

Руководитель практики: _____

Москва 2025

Оглавление

Введение

Aiogram-Base-App - это готовый шаблон для быстрого старта разработки Telegram-ботов с использованием фреймворка Aiogram. Проект избавляет разработчиков от рутинной настройки и инициализации, предоставляя уже настроенную структуру с основными функциями.

Цели проекта:

1. **Создать базовый шаблон бота для мессенджера Telegram.**
2. **Стандартизация кода.** Предоставить «чистый код» для того, чтобы разработчик смог его легко прочитать и дополнять
3. **Масштабируемость.** Обеспечить модульность для независимого добавления нового функционала
4. **Документированность.** Создать понятные README.md файлы с примерами, как установить и настроить проект
5. **Поддержка обновлений.** Легко адаптироваться к новым версиям используемых в проекте зависимостей

Задачи проекта

Реализовать и обеспечить:

- Асинхронную архитектуру
- Использовать лучшие практики архитектур Telegram ботов
- Простое и надёжное управление зависимостями
- Кэширование данных
- Валидацию данных
- Простую конфигурацию
- Взаимодействие с базой данных
- Автоматические тесты
- Развёртывание проекта

Выбор технологий, инструментов, фреймворков:

- Aiogram 3.x - Асинхронный фреймворк для Telegram-ботов.
- Asyncpg + SQLAlchemy - Для асинхронной работы с PostgreSQL.
- Alembic - Управление миграциями БД.
- Redis - Кэширование и хранение состояний (FSM).
- Pydantic - Валидация данных
- Poetry - Управление зависимостями.
- Ruff + Муру - Линтинг и проверка типов.
- Pre-commit - Git хук на линтинг Ruff
- Pytest - Автоматическое тестирование.
- Docker - Контейнеризация.

Архитектура директорий проекта:

```
|-- app
|   |-- core
|   |   |-- models
|   |   |   `-- mixins
|   |   `-- schemas
|   |-- handlers
|   |-- middlewares
|   |-- repository
|   `-- utils
|-- docs
|   `-- images
|-- migrations
|   `-- versions
|-- scripts
|-- tests
|   |-- integration_tests
```

| | |-- mock_data
| | `-- test_handlers
| `-- unit_tests
`-- texts

Основные компоненты системы:

app/ - Ядро приложения

- core/
    - models/ - ORM-модели базы данных (SQLAlchemy)
    - schemas/ - Pydantic-схемы для валидации/конвертации данных
    - Содержит базовую бизнес-логику приложения
- handlers/ - Обработчики Telegram-команд и сообщений
- middlewares/ - Промежуточное ПО (антифлуд, логирование)
- repository/ - Паттерн Repository для работы с БД
- utils/ - Вспомогательные утилиты (парсеры, хелперы)

migrations/ - Управление миграциями БД

- versions/ - Версии миграций БД
- Обеспечивает эволюционное изменение схемы БД без потери данных

tests/ - Тестирование

- unit_tests/ - Модульные тесты
- integration_tests/:
    - mock_data/ - Тестовые данные
    - test_handlers/ - Интеграционные тесты обработчиков
    - Другие интеграционные тесты

Вспомогательные директории :

- docs/ - Документация (включая images/)

- scripts/ - Полезные скрипты (развертывание, CI/CD)
- texts/ - Локализация и текстовые шаблоны бота

Несколько примеров листинга кода:

ORM-модель пользователя

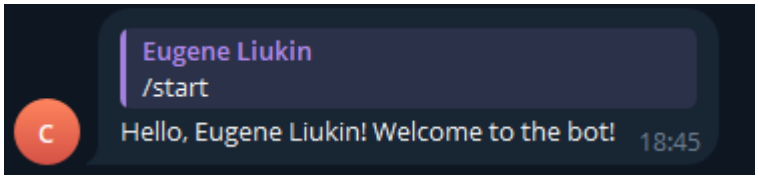
```
from sqlalchemy import BigInteger, Boolean, String
from sqlalchemy.orm import Mapped, mapped_column


from app.core.models.base import BaseOrm
from app.core.models.mixins import TimestampMixin



class UserOrm(BaseOrm, TimestampMixin):
    tg_id: Mapped[int] = mapped_column(BigInteger, unique=True)
    first_name: Mapped[str] = mapped_column(String(30))
    username: Mapped[str | None] = mapped_column(String(50), unique=True)
    last_name: Mapped[str | None] = mapped_column(String(30))

    is_active: Mapped[bool] = mapped_column(Boolean, default=True)
```

Паттерн репозиторий пользователя:

```
from __future__ import annotations


from typing import TYPE_CHECKING


from app.core.models import UserOrm
from app.core.schemas import UserCreateS
from app.core.schemas.user import UserUpdateS
from app.repository.base import BaseRepository


if TYPE_CHECKING:
    from sqlalchemy import ScalarResult
    from sqlalchemy.ext.asyncio import AsyncSession


class UserRepository(BaseRepository[UserOrm, UserCreateS, UserUpdateS]):
    model_class: type[UserOrm] = UserOrm


    @classmethod
    async def get_by_tg_id(cls, session: AsyncSession, tg_id: int) -> UserOrm | None:
        scalar_result: ScalarResult[UserOrm] = await cls._get_by_fields(session=session, tg_id=tg_id)
        user: UserOrm | None = scalar_result.one_or_none()
        return user


    @classmethod
    async def update_by_tg_id(cls, session: AsyncSession, tg_id: int, update_schema: UserUpdateS) -> None:
        await cls._update_by_filter_by(
            session=session,
            update_schema=update_schema,
            tg_id=tg_id,
        )
```

```
@classmethod
async def delete_by_tg_id(cls, session: AsyncSession, tg_id: int) -> None:
    await cls._delete_by_filter_by(session=session, tg_id=tg_id)
```

Пример работы обработчика /start



Этот пользователь будет добавлен в БД:

	id	tg_id	first_name	last_name	username	is_active
1	1	2104928…	Eugene	Liukin	EugeneLiuk…	true

Достигнутые результаты

В ходе выполнения проектной практики мной был разработан базовый шаблон Telegram-бота на основе фреймворка Aiogram 3.x, который полностью соответствует всем поставленным целям и задачам. Ниже представлен детальный анализ достигнутых результатов.

1. Создание универсального шаблона бота

Разработанная архитектура проекта включает все необходимые компоненты для быстрого старта разработки:

- Полноценная модульная структура с четким разделением ответственности между компонентами
- Готовые обработчики базовых команд (/start)
- Интеграция с системой логгирования для мониторинга работы бота

2. Стандартизация кода и архитектуры

В проекте реализованы современные подходы к разработке:

- Строгое соблюдение принципов SOLID
- Реализация паттерна Repository для работы с базой данных
- Использование type hints для улучшения читаемости кода
- Разделение моделей (ORM) и схем данных (Pydantic)
- Применение dependency injection для управления зависимостями
- Реализация unit of work для управления транзакциями

3. Масштабируемость системы

Архитектура проекта позволяет легко расширять функциональность:

- Возможность добавления новых модулей без изменения существующего кода
- Гибкая система маршрутизации сообщений

4. Полноценная документация

Проект содержит исчерпывающую документацию:

- Подробный README файл с инструкциями по установке и настройке

5. Готовность к промышленной эксплуатации

Проект включает все необходимое для production-развертывания:

- Конфигурация для Docker-контейнеризации
- Настройки для CI/CD
- Поддержка webhook и polling режимов

6. Обеспечение качества кода

- Реализованы инструменты для поддержания высокого качества:
- Настройка pre-commit хуков

- Интеграция с линтерами (Ruff, Mypy)
- Написание unit и интеграционных тестов

7. Реализация дополнительных возможностей

Помимо базовых требований, в проекте реализованы:

- Система кэширования с Redis
- Поддержка мультиязычности

В результате выполнения проекта был создан полноценный шаблон, который значительно ускоряет процесс разработки Telegram-ботов, обеспечивая при этом высокое качество кода и возможность масштабирования. Все поставленные цели и задачи были успешно достигнуты, а в некоторых аспектах даже превзойдены.