Telegram Bot Python Project Structure

Python Telegram bot using python-telegram-bot. Modular handlers, conversation flows, and inline query support.

#telegram #python #bot #python-telegram-bot #async #webhooks

PNG PDF Copy Prompt

Project Directory

```
telegram-bot/
  src/ Bot source code
    __init__.py
    bot.py Application set…
    handlers/ Command and mes…
      __init__.py
      commands.py /start, /help
      messages.py Text handlers
      callbacks.py Inline button c…
      conversations.py Multi-step flows
    keyboards/ Reply and inlin…
      __init__.py
      reply.py
      inline.py
    services/ Business logic
      __init__.py
      user_service.py
      api_client.py
    utils/ Helper functions
      __init__.py
      filters.py Custom filters
      decorators.py
      helpers.py
    config.py Settings from e…
  data/ Persistent data
    .gitkeep
  tests/
    __init__.py
    conftest.py
    test_handlers.py
  main.py Entry point
  pyproject.toml
  .env.example BOT_TOKEN here
  .gitignore
  README.md
```

Why This Structure?

This structure separates concerns into handlers (user interaction), keyboards (UI), services (business logic), and utils (shared helpers). Handlers are grouped by interaction type—commands, messages, callbacks, and conversations—making it easy to find and modify specific features.

Key Directories

src/handlers/ - Command and message handlers by type

src/keyboards/ - Reply and inline keyboard builders

src/services/ - Business logic and external API clients

src/utils/ - Custom filters, decorators, helpers

Getting Started

1. `uv init telegram-bot && cd telegram-bot`
2. `uv add python-telegram-bot python-dotenv`
3. `cp .env.example .env` and add your bot token from @BotFather
4. `uv run python main.py`

Command Handlers

```python
# src/handlers/commands.py
from telegram import Update
from telegram.ext import ContextTypes

async def start(update: Update, ctx: ContextTypes.DEFAULT_
    await update.message.reply_text(
        "Welcome! Use /help to see available commands."
    )

async def help_command(update: Update, ctx: ContextTypes.D
    await update.message.reply_text("Available commands:\n
```

Conversation Flows

Use `ConversationHandler` for multi-step flows like registration or forms. Each state maps to a handler, with `entry_points`, `states`, and `fallbacks` defining the flow. Store conversation data in `context.user_data`.

When To Use This

- Building a Telegram bot with multiple commands
- Bots needing conversation flows (forms, wizards)
- Projects requiring inline keyboards and callbacks
- Teams familiar with Python async/await

Best Practices

- Use `ApplicationBuilder` for bot setup
- Group related handlers in separate modules
- Use `CallbackQueryHandler` patterns for inline buttons
- Store tokens in `.env`, never commit them
- Use `JobQueue` for scheduled tasks

Trade-offs

Async-only - v20+ is fully async, no sync fallback

Webhook complexity - Requires HTTPS endpoint for production webhooks

Rate limits - Telegram limits messages per second per chat