Discord Bot Python Project Structure

Python Discord bot using discord.py with cogs pattern. Modular commands, event handling, and slash command support.

#discord #python #bot #discord.py #cogs #slash-commands

PNG PDF Copy Prompt

Project Directory

```
discord-bot/
  src/ Bot source code
    __init__.py
    bot.py Bot class, star…
    cogs/ Command modules
      __init__.py
      general.py Basic commands
      moderation.py Mod commands
      fun.py
    events/ Event listeners
      __init__.py
      on_ready.py
      on_member_join.py
      error_handler.py
    utils/ Helper functions
      __init__.py
      checks.py Permission chec…
      embeds.py Embed builders
      database.py
    config.py Bot configurati…
  data/ Persistent data
    .gitkeep
  tests/
    __init__.py
    conftest.py
    test_commands.py
  main.py Entry point
  pyproject.toml
  .env.example DISCORD_TOKEN h…
  .gitignore
  README.md
```

Why This Structure?

This structure uses the Cogs pattern—discord.py's built-in way to organize commands and listeners into reusable classes. Each cog handles a specific feature (moderation, fun, etc.) and can be loaded/unloaded at runtime. Events are separated for cleaner error handling.

Key Directories

`src/bot.py` - Bot subclass with cog loading logic

`src/cogs/` - Command groups as Cog classes

`src/events/` - Event listeners separated from commands

`src/utils/` - Permission checks, embed helpers, DB access

Getting Started

1. `uv init discord-bot && cd discord-bot`
2. `uv add "discord.py[voice]" python-dotenv`
3. `cp .env.example .env` and add your bot token
4. `uv run python main.py`

Cog Pattern

```python
# src/cogs/general.py
from discord.ext import commands
from discord import app_commands

class General(commands.Cog):
    def __init__(self, bot):
        self.bot = bot

    @app_commands.command()
    async def ping(self, interaction):
        """Check bot latency."""
        await interaction.response.send_message(
            f"Pong! {round(self.bot.latency * 1000)}ms"
        )

async def setup(bot):
    await bot.add_cog(General(bot))
```

Dynamic Cog Loading

Cogs are loaded dynamically via `bot.load_extension('src.cogs.general')`. The `setup()` function at the bottom of each cog file is called automatically. Use `bot.reload_extension()` for hot-reloading during development.

When To Use This

- Building a Discord bot with multiple command categories
- Bots that need modular, reloadable features
- Projects with slash commands and context menus
- Teams familiar with Python async/await

Best Practices

- Use `app_commands` for slash commands (not prefix commands)
- Keep cogs focused—one feature per cog
- Handle errors in a central `error_handler.py` event
- Store tokens in `.env`, never commit them
- Use `discord.Intents` explicitly for required permissions

Trade-offs

`Async complexity` - Everything is async/await, debugging can be tricky

`Rate limits` - Discord API limits require careful command design

`Testing difficulty` - Mocking Discord API requires extra setup