Assignment Discord Bot

Architecture & Design Document

A concise, engineer-focused overview of how the bot is structured and how it works.

Document Overview

- Purpose: Technical architecture and design for the Assignment Discord Bot
- Audience: Developers maintaining or extending the bot
- Last Updated: November 2025

Table of Contents

1. System Overview

Description

A Python-based Discord bot that syncs assignments from Canvas, stores them in SQLite, and helps users plan study sessions with automated reminders and weekly summaries.

- Language: Python 3.9+
- Framework: discord.py
- Storage: SQLite via aiosqlite
- External API: Canvas LMS (REST)
- Runtime: Long-running bot process

Key Features

- Weekly assignment summary to a configured channel (Mondays)
- Interactive planning of study sessions per assignment
- Automated reminders (24h, 1h, at-time) for planned sessions
- Due date reminders (2d, 1d, 12h) for upcoming assignments
- Per-user completion tracking and weekly congratulation message
- Manual sync with Canvas

2. Architecture & Layers

Layered Structure

```
assignment-discord-bot/
├── bot.py                  # Discord bot entry & orchestration
├── config.py               # Environment configuration
├── constants.py            # App-wide constants
├── canvas_api/             # Canvas HTTP client & endpoint helpers
├── database/               # SQLite data access & schema
├── services/               # Presentation helpers (formatting)
└── utils/                  # Utilities (datetime, weekly UI, sync)
```

Responsibilities

- bot.py: Commands, background tasks, coordination
- canvas_api/: Authenticated HTTP client + endpoints for courses/assignments
- database/: Schema init/migrations, CRUD, reminder queries, completion tracking
- utils/: Datetime conversions, weekly messaging, sync orchestration
- services/: Format Canvas data for display in Discord

Dependency Flow

```
Discord (commands & tasks)
    ↓
Utilities & Services (weekly UI, sync)
    ↓
Database (SQLite via aiosqlite)
    ↓
Canvas API (requests → Canvas)
```

- Datetimes stored as UTC ISO strings; presented in local time
- Composite keys used to avoid cross-course ID collisions

3. Data Model & Storage

SQLite Schema (Simplified)

```
courses(
  id PRIMARY KEY,
  name TEXT,
  course_code TEXT,
  start_at TEXT,
  end_at TEXT
)

assignments(
  id INTEGER,
  course_id INTEGER,
  name TEXT,
  due_at TEXT,      -- UTC ISO 'Z'
  week_number INT,
  html_url TEXT,
  submitted INT,
  due_reminder_2d_sent INT,
  due_reminder_1d_sent INT,
  due_reminder_12h_sent INT,
  PRIMARY KEY(course_id, id)
)

study_plans(
  user_id TEXT,
  course_id INTEGER,
  assignment_id INTEGER,
  planned_at_utc TEXT,  -- UTC ISO 'Z'
  notes TEXT,
  reminder_24h_sent INT,
  reminder_1h_sent INT,
  reminder_now_sent INT,
  PRIMARY KEY(user_id, course_id, assignment_id)
)

user_assignment_status(
  user_id TEXT,
  course_id INTEGER,
  assignment_id INTEGER,
  completed INT,
  completed_at_utc TEXT,
  PRIMARY KEY(user_id, course_id, assignment_id)
)

week_completion_notifications(
  user_id TEXT,
  week_key TEXT, -- 'YYYY-WW'
  notified INT,
  PRIMARY KEY(user_id, week_key)
)
```

4. Canvas Integration

HTTP Client

- Module: `canvas_api/client.py`
- Auth: Bearer token (`CANVAS_TOKEN`)
- Pagination: Follows `Link` headers (`per_page=100`)
- Errors: Raises `CanvasAPIError` on request failures

Endpoints

- Module: `canvas_api/endpoints.py`
- `get_courses()` : id, name, course_code, start/end dates
- `get_assignments(course_id)` : id, name, due_at, url, submitted flag

5. Discord Commands & Tasks

Commands (bot.py)

- `!sync` : Fetch latest courses/assignments from Canvas and store locally
- `!thisweek` : List assignments due this week; prompt for scheduling
- `!plans` : Show user-planned study sessions for the week
- `!complete [query]` : Mark selected assignments complete
- `!reschedule` : Change the planned time of a session

Background Tasks

- Weekly summary: Monday at configured time
- Reminder loop: Every minute
  - Work session reminders (24h, 1h, now)
  - Due date reminders (2d, 1d, 12h)
  - Noon check: weekly completion congratulations

6. Reminder Logic

Work Session Reminders

- Trigger windows:
  - 24h: planned_at ±1 minute
  - 1h: planned_at ±1 minute
  - Now: planned_at ±1 minute
- Flags updated per reminder type to prevent duplicates

Due Date Reminders

- Trigger windows relative to `due_at`:
  - 2 days, 1 day, 12 hours (±1 minute)
- Only for current week and incomplete assignments
- Flags stored per assignment

7. Configuration & Environment

Environment Variables

Loaded via `python-dotenv` from `.env`:

Required:

- `BOT_TOKEN`: Discord bot token
- `CANVAS_TOKEN`: Canvas API token
- `CHANNEL_ID`: Discord channel ID (int)

Optional:

- `CANVAS_BASE_URL`: Canvas API base (default: `https://canvas.instructure.com/api/v1/`)
- `DB_PATH`: SQLite file path (default: `data/canvas_bot.db`)
- `WEEKLY_NOTIFICATION_HOUR`: Hour (default: 9)
- `WEEKLY_NOTIFICATION_MINUTE`: Minute (default: 0)

8. Testing Strategy

Tests Structure

```
tests/
├── unit/          # datetime utils, endpoints, config
├── integration/   # database ops, sync flow
├── regression/    # schema, startup, parsing
└── acceptance/    # user workflows
```

- Prefer unit tests for utilities and endpoints
- Integration tests for DB + sync
- Regression tests for schema and startup behavior

9. Deployment & Operations

Running Locally

```
# Windows PowerShell
python -m venv .venv; .\.venv\Scripts\Activate.ps1
pip install -r requirements.txt
copy .env.example .env
# edit .env with your tokens
python bot.py
```

Operational Notes

- Ensure bot has permissions and intents enabled
- Keep process running (PM2/Task Scheduler/Service)
- Back up `data/canvas_bot.db` periodically
- Rotate `CANVAS_TOKEN` as needed