Developer Guide – Cognitive Wellness & Memory Aid App

Author: Jessica Anioha
Project: Cognitive Wellness & Memory Aid App
Last Updated: December 2025

1. Overview

This developer guide provides a technical overview of the Cognitive Wellness & Memory Aid App, a Streamlit-based task management and reminder tool designed to support memory, routine building, and executive functioning.

The app allows users to:

- Create, update, and delete tasks
- Attach reminder times
- Receive real-time pop-up notifications triggered by the system
- Manage tasks with a clean UI and optional dark mode

This document explains the internal architecture, flow of user interaction, modules, core logic, and known issues. It also provides guidance for future developers who may extend or maintain the project.

2. Architecture & Directory Structure

Your final repository should look like this:

```
project/
│
├── main.py                 # Entry point to run the app
├── A_preprocess.py         # (Optional) Preprocessing or data setup
script
├── styles/
│   ├── style.css           # Light mode
│   └── dark_mode.css       # Dark mode theme (optional)
│
```

```
├── utils/
│   ├── reminder_engine.py  # Background reminder checking, modals
logic
│   ├── task_manager.py     # CRUD operations for tasks
│   ├── audio_handler.py    # Audio handling (if reminders include
sound)
│
├── data/
│   └── tasks.csv           # Saved task state
│
└── doc/
    ├── README.md           # User guide
    └── developer_guide.md  # This document
```

3. Final System Specification: What Is Implemented

Implemented Features

- Task creation and editing
- Reminder time setting
- Persistent storage in CSV / session state
- Real-time reminder checking
- Pop-up modal with:
    - *Mark Completed*
    - *Snooze 5 minutes*
- Light mode themes
- Clean separation between UI (main.py), logic (task_manager, reminder_engine), and resources (CSS)

Initial Specs Not Fully Implemented

- Push notifications outside Streamlit

- Cross-device syncing
- NLP task creation
- User authentication

4. Installation, Deployment & Admin Notes

4.1 Requirements

Python 3.10+
Install dependencies:

```
pip install -r requirements.txt
```

Main libraries:

- `streamlit`
- `pandas`
- `pydub` (if audio reminders are used)
- `datetime`

4.2 Running the App

Always run:

```
streamlit run main.py
```

4.3 Developer-Specific Notes

- Streamlit reruns code on every user interaction, so state must be stored in `st.session_state` or a CSV.
- Reminder checking is triggered in the main script every run , there is no background daemon unless you implement one.

- *Any new feature* should follow the module separation structure:
    - UI logic → `main.py`
    - Task logic → `task_manager.py`
    - Reminder logic → `reminder_engine.py`

5. User Flow & Code Walkthrough

5.1 User Interaction Summary

1. Users open the app (`main.py` loads UI).
2. Tasks load automatically from session state or CSV.
3. User:
    - Adds a task
    - Specifies reminder time
    - Saves
4. The system continuously checks whether a task's reminder time matches the current time.
5. When matched, a modal pops up.
6. User chooses:
    - Mark Completed
    - Snooze 5 minutes

5.2 Code Flow: Step-by-Step

Step 1 — main.py (UI Entrypoint)

Handles:

- Page layout

- Loading CSS
- Interacting with Streamlit session state
- Calling the reminder engine
- Calling the task manager
- Rendering tasks table

Imports:

```
from utils.task_manager import load_tasks, add_task, save_tasks
from utils.reminder_engine import check_for_reminders
```

Step 2 — task_manager.py

Core functions:

- `load_tasks()`
- `save_tasks()`
- `create_task()`
- `update_task()`
- `delete_task()`

This module maintains data consistency and provides the CRUD abstraction for tasks.

Step 3 — reminder_engine.py

Core functions:

- `check_for_reminders()`
- `display_reminder_modal()`
- `snooze_task()`
- `mark_task_completed()`

This file controls:

- Monitoring timestamps
- Comparing with system time
- Displaying popups with `st.modal`

Step 4 — styles/style.css

Controls layout, colors, spacing.

Step 5 — data/tasks.csv

Stores persistent task data:

```
Task,Reminder,Status
Drink water,08:00,Pending
Take vitamins,10:30,Completed
```

6. Module Breakdown

6.1 main.py

Responsible for:

- Initializing app state
- Importing logic modules
- Calling `check_for_reminders()` every rerun
- Rendering forms and tables
- Handling button actions

6.2 utils/task_manager.py

Handles:

- Converting CSV to DataFrame

- Storing in session state
- CRUD operations
- Safe saving

6.3 utils/reminder_engine.py

Handles:

- Detecting when reminder time == current time
- Displaying modal
- Snoozing (adds 5 minutes)
- Marking completed

Note: Streamlit does NOT support continuous background processes. The "engine" works by checking timestamps on every script rerun.

6.4 utils/audio_handler.py *(optional)*

If you include audio reminders.

7. Known Issues & Limitations

Minor Issues

- Streamlit reruns can cause modals to briefly reappear if timing overlaps.
- Large task lists may slow UI slightly due to re-rendering.

Major Issues

- Streamlit cannot run background tasks while idle, reminders depend on user activity or page refresh.
- If users closes the tab, reminders cannot trigger.

Potential Inefficiencies

- CSV storage is fine for small datasets but not scalable to thousands of tasks.
- Reminder checking could be optimized with caching or async.

8. Future Work

Short-term Improvements

- Database integration (SQLite or Firebase)
- User profiles and task categories
- Enhanced animations and transitions
- Voice reminders via Web Speech API

Long-term Extensions

- Full mobile app version
- Sync across devices
- AI-based task suggestions
- Full calendar integration

9. Ongoing Development

If continued evolution is expected:

Recommended practices

- Follow module-based separation: UI vs Logic vs Resources
- Write unit tests (pytest)
- For themes, separate all UI colors into CSS variables
- Use class-based models for tasks (instead of dict/DataFrame)
- Maintain backward compatibility with old CSV schemas

Adding New Features

When expanding the app:

- Add new logic inside `utils/`
- Import into `main.py`
- Document in this guide
- Add CSS if necessary

10. Contact & Contribution

If future developers wish to work on this project:

- Fork repo
- Create feature branch
- Submit PR with documentation
- Update developer guide