top# Rico 4.0 – Autopilot (vollautomatische Jobs)

Ziel

Das System soll automatisch Aufgaben abarbeiten können (z. B. Tages-Checkins, Social-Media-Posts, Analysen).
Die Steuerung läuft über eine zentrale Konfiguration (`data/autopilot/config.yaml`).
Das Frontend (Streamlit) erhält eine Sidebar mit Autopilot-Schalter und Statusanzeige.

Dateien & Module

1. Backend

- `backend/app/autopilot.py`
  - Enthält `AutopilotManager` (lädt Konfig, führt Tasks aus, speichert Status).
  - API-Endpunkte in `backend/app/routers/autopilot.py` :
    - `GET /api/v1/autopilot/status`
    - `POST /api/v1/autopilot/start`
    - `POST /api/v1/autopilot/stop`
    - `POST /api/v1/autopilot/run-task` (manueller Trigger).
- `backend/app/services/orchestrator.py`
  - `RicoOrchestrator` um Funktion `run_autopilot_task(self, task: dict)` erweitern.
  - Nutzt vorhandene LLM-Aufrufe (OpenAI, Claude, Perplexity etc.).

2. Frontend

- `frontend/streamlit_app.py`

- Sidebar-Erweiterung:
  - Autopilot-Schalter (Start/Stop).
  - Statusanzeige (laufend/gestoppt).
  - Liste der geplanten Tasks aus `config.yaml`.
  - Manuelle Trigger-Buttons je Task.

3. Daten

- `data/autopilot/config.yaml`
  - Enthält die geplanten Tasks (z. B. täglich 20:00 Uhr Check-In, Social-Media-Posts).
  - Felder:
    - `id`: eindeutige Task-ID
    - `description`: kurze Beschreibung
    - `schedule`: Cron-Syntax (z. B. `"0 20 * * *"`)
    - `enabled`: true/false
    - `provider`: auto | openai | claude | pplx
    - `task_type`: analysis | post | checkin
    - `prompt`: eigentlicher Prompt-Text

4. Tests

- `tests/test_autopilot.py`
  - Dummy-Test: Lädt Config, ruft AutopilotManager.run_once() auf.
  - Erwartet Rückgabe mit `ok=True` und Task-Resultat.

Ablauf

1. Beim Start lädt `AutopilotManager` die Config.
2. Jede Minute prüft ein Scheduler (`apscheduler` oder `asyncio.create_task`):
   - Welche Tasks sind fällig?

- Falls ja: Übergibt an `RicoOrchestrator.run_autopilot_task`.

3. Ergebnisse werden gespeichert (z. B. `logs/autopilot/` als JSON-Dateien).
4. Frontend zeigt Ergebnisse und Status an.

Beispiel `config.yaml`

```
tasks:
  - id: "daily_checkin"
    description: "Täglicher Abend-Checkin 20:00 Uhr"
    schedule: "0 20 * * *"
    enabled: true
    provider: "auto"
    task_type: "analysis"
    prompt: |
      Was ist heute passiert?
      Was sind die wichtigsten To-Dos für morgen?

  - id: "social_post"
    description: "Täglicher Social-Media-Post"
    schedule: "30 9 * * *"
    enabled: true
    provider: "openai"
    task_type: "post"
    prompt: |
      Schreibe einen motivierenden Social-Media-Post über Tiergesundheit

---

## Dateistruktur & Ablage-Regeln

### Dokumentationsablage
- **Alle Dokumentationsdateien** (.md, .pdf, .spec, Arbeitsbücher, Maste
- **Cursor erzeugt neue Dokumentationsdateien** immer direkt in `/docs`.
- **Technische Dateien** (Code, Tests, Skripte) bleiben in ihren Modulor
- **PDFs werden ebenfalls in `/docs` gespeichert**, außer es gibt expliz
- **Wichtig:** Cursor prüft bei neuen Dateien zuerst `/docs` und legt dc

### Aktuelle Dokumentationsstruktur
```

/docs/ ├── autopilot_spec.md # Diese Spezifikation ├── CLAUDE_HEADER_FIX_SUMMARY.md # Claude Header Fix ├── README_SECRETS_QUICKSTART.md # Secrets Quickstart ├──

README_UI_REDESIGN.md # UI Redesign Guide ├── RICO_4_0_OPTIMIERT.md # Rico 4.0 Optimiert ├── RICO_4_0_STATUS.md # Rico 4.0 Status ├── RICO_OPS_HUB_SETUP.md # Ops Hub Setup ├── Rico_MasterPrompt_Arbeitsbuch_V4_Premium.pdf # MasterPrompt V4 (Legacy) ├── Rico_MasterPrompt_Arbeitsbuch_V5.md # MasterPrompt V5 Markdown ├── Rico_MasterPrompt_Arbeitsbuch_V5.pdf # MasterPrompt V5 PDF └── SECURITY_SECRETS.md # Security & Secrets

```
### Ablage-Regeln für Cursor
1. **Neue Dokumentation** → Immer `/docs/` ablegen
2. **Technische Dateien** → In entsprechenden Modulordnern
3. **PDF-Exports** → Standardmäßig `/docs/`, bei Bedarf `/exports/`
4. **Automatische Prüfung** → Cursor prüft `/docs` zuerst bei neuen Dokument
```