**1) KIRO‑Hook: „Run targeted tests on save“**

Ziel: Bei Änderungen in core/, file\_handlers/, processors/, utils/ **nur die betroffenen Tests** starten (nicht die ganze Suite).

**In KIRO (Copy/Paste – Englisch):**

bash

KopierenBearbeiten

Create an Agent Hook named "Run targeted tests on save".

Trigger:

- On file save

- Scope: PM Assistant/core/\*\*, PM Assistant/file\_handlers/\*\*, PM Assistant/processors/\*\*, PM Assistant/utils/\*\*

Action (PowerShell on Windows; adapt to bash on macOS/Linux):

- Determine the saved file path as $env:KIRO\_CHANGED\_FILE

- Map it to the most likely test file:

- If path contains "file\_handlers\<name>\_handler.py" -> tests\test\_<name>\_handler.py

- Else if filename is "<name>.py" -> tests\test\_<name>.py

- Fallback: run last failed tests: pytest -q --lf

Command:

pwsh -NoProfile -Command "

$changed = $env:KIRO\_CHANGED\_FILE;

$base = Split-Path $changed -Leaf;

$stem = [System.IO.Path]::GetFileNameWithoutExtension($base);

$test = 'tests\test\_' + $stem + '.py';

if ($changed -match 'file\_handlers.\*\\(.+?)\_handler\.py') { $test = 'tests\test\_' + $Matches[1] + '\_handler.py' }

if (Test-Path $test) { python -m pytest $test -q } else { python -m pytest -q --lf --maxfail=1 }

"

Execution:

- Silent output unless failures

- Do not open a chat session

Warum so?

* **Gezielt** → schnell.
* **Fallback --lf** → wenn kein Mapping, laufen nur zuletzt gefallene Tests.
* **Kein Chat** → spart Interaktions‑Budget.

**2) Weitere sinnvolle Hooks (projektbezogen)**

**A) Lint/Format on save (schnell, ohne Chat)**

vbnet

KopierenBearbeiten

Create an Agent Hook: "Format + Lint on save"

Trigger: On file save (Python files)

Command (PowerShell):

pwsh -NoProfile -Command "python -m black . --quiet; python -m ruff check . --quiet"

Execution: silent unless errors

Du kannst auch isort integrieren; mit ruff ist’s meist nicht nötig.

**B) Typecheck on save (nur geänderte Datei)**

vbnet

KopierenBearbeiten

"Typecheck changed file"

Trigger: On file save (Python)

Command:

pwsh -NoProfile -Command "mypy --hide-error-codes --no-error-summary $env:KIRO\_CHANGED\_FILE"

**C) Prevent test edits (harte Guardrail)**

Blockt Commits, die tests/ verändern (außer du setzt einen Override‑Tag).

perl

KopierenBearbeiten

"Guard: prevent test modifications"

Trigger: On git commit

Command:

pwsh -NoProfile -Command "

git diff --cached --name-only | Select-String '^tests/' > $null

if ($?) {

$msg = git log -1 --pretty=%B

if ($msg -notmatch '\[ALLOW-TEST-CHANGE\]') {

Write-Host 'Commit blocked: modifying tests requires tag [ALLOW-TEST-CHANGE] in the commit message.' -ForegroundColor Red; exit 1

}

}

"

**D) Auto‑update Task‑Status nach Tests**

Trägt Subtasks als „done“ ein, wenn zugehörige Tests grün sind.

makefile

KopierenBearbeiten

"Update tasks.md after green tests"

Trigger: After successful pytest run

Command:

pwsh -NoProfile -Command "python manage\_tasks.py --update"

manage\_tasks.py müsstest du kurz erstellen (kann ich dir liefern), der mappt Test‑Ergebnisse → Häkchen in tasks.md.

**E) Config‑Validator**

Wenn config.yaml geändert wird, sofort prüfen.

vbnet

KopierenBearbeiten

"Validate config.yaml"

Trigger: On file save (config.yaml)

Command:

pwsh -NoProfile -Command "python -c 'from core.config\_manager import ConfigManager; ConfigManager(\"config.yaml\").load()'"

**F) Security quick scan (leichtgewichtig)**

vbnet

KopierenBearbeiten

"Security scan (Bandit) on commit"

Trigger: On git commit

Command:

pwsh -NoProfile -Command "bandit -q -r PM Assistant -ll || exit 1"

**3) Pre‑Commit (lokal, ohne KIRO, sehr zuverlässig)**

Das entlastet KIRO komplett und verhindert „kaputte“ Commits.

**pyproject.toml (Format + Lint config – Copy/Paste)**

toml

KopierenBearbeiten

[tool.black]

line-length = 100

target-version = ["py310"]

[tool.ruff]

line-length = 100

select = ["E","F","I","B","UP"]

ignore = ["E501"]

fix = true

[tool.mypy]

python\_version = "3.10"

ignore\_missing\_imports = true

strict\_optional = true

warn\_unused\_ignores = true

**.pre-commit-config.yaml**

yaml

KopierenBearbeiten

repos:

- repo: https://github.com/astral-sh/ruff-pre-commit

rev: v0.5.7

hooks:

- id: ruff

args: [--fix]

- id: ruff-format

- repo: https://github.com/psf/black

rev: 24.8.0

hooks:

- id: black

- repo: https://github.com/pre-commit/mirrors-mypy

rev: v1.10.0

hooks:

- id: mypy

additional\_dependencies: []

- repo: https://github.com/PyCQA/bandit

rev: 1.7.9

hooks:

- id: bandit

args: [-q, -r, PM Assistant]

- repo: local

hooks:

- id: pytest

name: pytest quick

entry: python -m pytest -q --maxfail=1 --last-failed

language: system

pass\_filenames: false

**Einrichten (einmalig)**

powershell

KopierenBearbeiten

pip install pre-commit ruff black mypy bandit

pre-commit install

**4) GitHub Actions (CI), schnell & hackathon‑tauglich**

.github/workflows/ci.yml

yaml

KopierenBearbeiten

name: CI

on:

push: { branches: ["main"] }

pull\_request:

jobs:

test:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- uses: actions/setup-python@v5

with: { python-version: "3.10" }

- name: Cache pip

uses: actions/cache@v4

with:

path: ~/.cache/pip

key: pip-${{ runner.os }}-${{ hashFiles('\*\*/requirements.txt') }}

- run: python -m pip install -U pip

- run: pip install -r requirements.txt

- run: pip install ruff black mypy pytest-cov

- run: ruff check .

- run: black --check .

- run: mypy PM\ Assistant || true

- run: python -m pytest -q --maxfail=1 --disable-warnings

**Was sind die Fallstricke?**

* **Dauerfeuer-Tests** beim Speichern können nerven → deshalb **targeted** + --lf + --maxfail=1.
* Hooks, die Chat‑Sessions öffnen, **kosten Tokens** → oben alles als **Silent‑Shell**.
* „Prevent test edits“ ist streng; du brauchst den Opt‑in‑Tag, wenn du legitime Testfixes machst.

Willst du, dass ich dir **manage\_tasks.py** (Task‑Auto‑Update nach Pytest) direkt erstelle? Dann setzt der Hook 2D sofort sichtbare Häkchen in tasks.md.

Quellen

**Du:**

Kannst Du mir die einzelnen Hooks noch mal erklären, was sie tun? z.B. verstehe ich den Sinn von cy.yml gar nicht genau

**ChatGPT:**

Klar, ich gehe die Liste noch mal Punkt für Punkt durch – diesmal so, dass du für jeden Hook genau weißt, **wozu** er da ist, **was** er macht, und **wann** er dir im KIRO-Projekt konkret etwas bringt.

**1️⃣ Run targeted tests on save**

**Zweck:**  
Sobald du eine Datei speicherst, laufen automatisch **nur die Tests**, die zu dieser Datei gehören.

* Spart Zeit im Vergleich zu pytest über alle Tests.
* Spart Interaktionsbudget, wenn KIRO involviert ist.
* Findet Fehler **sofort**, nicht erst beim manuellen Testlauf.

**Wann sinnvoll:**  
Immer dann, wenn du schnell Feedback über die Auswirkung einer Codeänderung brauchst.

**2️⃣ Format + Lint on save**

**Zweck:**  
Stellt sicher, dass dein Code **immer formatiert** (Black) und **stilistisch korrekt** (Ruff) ist, bevor er gespeichert oder committed wird.

* Entfernt manuelle Formatierarbeit.
* Hält Stil konsistent → wichtig, wenn KIRO Code ergänzt.

**Wann sinnvoll:**  
Bei jedem Projekt mit mehreren Dateien und einer klaren Code-Konvention.

**3️⃣ Typecheck on save**

**Zweck:**  
Prüft mit mypy, ob deine Typannotationen stimmen und ob es offensichtliche Typfehler gibt.

* Hilft, Bugs früh zu finden (z. B. falsche Argumenttypen, fehlende Rückgabewerte).
* Nützlich, weil KIRO bei Generierung zwar oft Typen setzt, aber nicht immer konsistent.

**Wann sinnvoll:**  
Wenn du mit vielen Dataclasses, Models und komplexen Strukturen wie in deinem Projekt arbeitest.

**4️⃣ Prevent test edits (Guardrail)**

**Zweck:**  
Verhindert versehentliches oder ungewolltes Ändern von Tests – außer, du setzt absichtlich einen Override-Tag in der Commit-Message ([ALLOW-TEST-CHANGE]).

* Vermeidet, dass man Tests anpasst, nur um Code „grün“ zu machen.
* Hält Test-Coverage glaubwürdig.

**Wann sinnvoll:**  
Besonders wichtig in einem Hackathon oder Wettbewerb, um die Integrität der Tests zu sichern.

**5️⃣ Auto-update Task-Status nach Tests**

**Zweck:**  
Lässt automatisch in deiner tasks.md die erledigten Tasks abhaken, wenn die dazugehörigen Tests erfolgreich sind.

* Spart manuelles Abhaken.
* Verbindet Fortschritt im Code mit Fortschritt im Plan.

**Wann sinnvoll:**  
Wenn du deine Fortschritte sauber dokumentieren möchtest, ohne KIRO jedes Mal fragen zu müssen.

**6️⃣ Config-Validator**

**Zweck:**  
Jedes Mal, wenn config.yaml gespeichert wird, wird sie mit deinem ConfigManager geladen, um Syntaxfehler oder fehlende Felder zu erkennen.

* Fehler in der Config werden sofort sichtbar.
* Spart Zeit, weil du nicht erst später beim Start merkst, dass YAML kaputt ist.

**Wann sinnvoll:**  
Immer, wenn Konfiguration zentral für den Lauf des Programms ist.

**7️⃣ Security scan (Bandit) on commit**

**Zweck:**  
Sucht automatisch nach **potenziellen Sicherheitslücken** (z. B. eval(), hartkodierte Passwörter).

* Blockiert Commits mit bekannten Patterns für Unsicherheiten.

**Wann sinnvoll:**  
Wenn du vorhast, das Projekt später öffentlich oder im Wettbewerb zu zeigen.

**8️⃣ .pre-commit Hooks (lokal, ohne KIRO)**

**Zweck:**  
Diese laufen **lokal** vor jedem Git-Commit, unabhängig von KIRO.

* Format, Lint, Typprüfung und Tests werden vor jedem Commit geprüft.
* Keine API-Kosten, keine KIRO-Interaktionen.

**Wann sinnvoll:**  
Immer – ersetzt viele manuelle Checks.

**9️⃣ GitHub CI (ci.yml)**

**Zweck:**  
Automatisiert das Testen im GitHub-Backend jedes Mal, wenn du pusht oder einen Pull-Request machst.

* CI = Continuous Integration.
* Prüft auf einem frischen System: Läuft dein Code wirklich überall, oder nur auf deinem Rechner?
* Fängt Probleme ab, die durch unterschiedliche Python-Versionen, fehlende Abhängigkeiten oder vergessene Dateien entstehen.

**Wann sinnvoll:**  
Wenn du:

* mit anderen zusammenarbeitest,
* im Hackathon ein sauberes Repo vorzeigen willst,
* sicherstellen willst, dass der Code auch in einer neutralen Umgebung funktioniert.

💡 Kurz gesagt:

* **Hooks 1–3**: schnelle, lokale Qualitätssicherung.
* **Hook 4**: verhindert unsaubere Testmanipulation.
* **Hook 5**: Fortschrittsanzeige automatisieren.
* **Hook 6**: Konfiguration absichern.
* **Hook 7**: Sicherheitsprüfung.
* **Pre-Commit**: lokales All-in-One-Sicherheitsnetz.
* **CI (ci.yml)**: externe, neutrale Prüfstelle im GitHub-Backend.