Bedienungsanleitung – social_posts (Kaspio)

Diese Anleitung beschreibt Installation, Konfiguration und tägliche Nutzung des Skripts social_posts – mit Schwerpunkt auf der Datenablage im Unterordner data sowie den Dateien ingredients_overrides.json und ingredients_meta.json. Alle Pfade beziehen sich auf das Projekt-Root.

Datenstruktur – data/

```
1) Verzeichnisstruktur (relevant)
   project_root/
   ├─ src/social_post/...                  # Quellcode
   ├─ data/                                # Arbeitsdaten (von dir pflegbar)
   │   ├─ used_products.json               # Merkliste bereits geposteter Produkte
   │   ├─ anlass_kalender.json             # Datum → Anlasstext
   │   ├─ quotes.json                      # Zitate-Liste für Posts
   │   ├─ menu.json                        # Aktuelle Speisekarte (optional, falls genutzt)
   │   ├─ ingredients_overrides.json       # Manuell gepflegte Zutaten-Fakten (deine Quelle der
Wahrheit)
   │   ├─ ingredients_auto.json            # Automatisch ermittelte Zutaten (vom Skript erzeugt)
   │   └─ ingredients_meta.json            # Klassifikation/Regeln je Zutat (beverage/food,
cookable, …)
   └─ .env                                 # API-Keys + IDs (Notion, OpenAI, Drive)

Hinweis: Die Dateien in data werden vom Skript gelesen/aktualisiert. Du kannst overrides und
meta direkt pflegen.
```

Ablauf & Generierung

2) Ablauf (High-Level)
- CLI-Aufruf (python -m social_post.cli ...)
- Skript lädt Daten aus data/* (Anlässe, Zitate, Menü, Auto-Zutaten, Overrides, Meta)
- Für jeden Tag wird anhand des Feed-Musters ein Post-Typ gewählt (produkt, zitat, ingredient_fact; an Ruhetagen: zitat/ingredient_fact)
- Bei ingredient_fact und gesetztem --carousel-ingredients wird ein Carousel-Plan erzeugt (AI oder Platzhalter) und als JSON in AI-Vorschlag nach Notion geschrieben
- Eintrag wird in Notion erstellt: Titel, Text, Hashtags, Geplanter Zeitpunkt, Status=Entwurf, Post-Typ, Plattform (multi-select), Medientyp (Bild/Video), AI-Vorschlag (JSON), sowie optional Media Folder/Link
- Optional: Google-Drive-Ordner pro Post wird angelegt und Link in Notion gespeichert
- n8n liest Notion, prüft Plattform (multi-select) und postet je Kanal

ingredients_overrides.json – Zweck & Schema

3) ingredients_overrides.json – Zweck und Schema

Zweck: Redaktionelle Kurztexte zu Zutaten manuell pflegen/überschreiben. Diese Facts nutzt das Skript für ingredient_fact-Posts und zur AI-Anreicherung als Ausgangspunkt.

Minimales Schema (vom Code erwartet):

```
{
  "ingredients": [
    { "name": "Aperol", "fact": "Bitterorangig, ideal für Aperitivo; ..." },
    { "name": "Burrata", "fact": "Cremiger Frischkäse aus Mozzarella und Sahne; ..." }
  ]
}
```

- name: Klarname der Zutat (wird intern lowercased indiziert)
- fact: kurzer, sachlicher Infotext (100–400 Zeichen), keine Heilversprechen, keine Markenclaims
- Der Loader baut daraus ein Mapping: name_lower → {name, fact}
- Mit --enrich-ingredients kann das Skript zu kurze/fehlende Facts via KI ergänzen; mit --write-enriched-overrides werden die neuen Texte in diese Datei zurückgeschrieben

ingredients_meta.json – Zweck & Schema

4) ingredients_meta.json – Zweck und Schema

Zweck: Steuerung/Klassifikation je Zutat – u. a. ob sie als Getränk gilt, ob sie kochbar ist, und ob sie als Ingredient-Post zulässig ist. Das beeinflusst Text-Logik (z. B. keine Kochbehauptungen bei Getränken) und Carousel-Generierung.

Empfohlenes, vom Code gut nutzbares Schema:

```
{
  "meta": {
    "aperol":   { "category": "beverage", "cookable": false, "allow_ingredient_post": true },
    "burrata":  { "category": "food",     "cookable": true,  "allow_ingredient_post": true }
  },
  "aliases": [
    { "from": "allioli", "to": "aioli" },
    { "from": "hahnchen", "to": "hähnchen" }
  ]
}
```

- meta: Dictionary (Kleinbuchstaben) → Objekt mit Feldern:
  - category: "beverage" | "food"
  - cookable: true/false
  - allow_ingredient_post: true/false
- aliases (optional): Normalisierung von Varianten/Nebenschreibweisen
- Falls ein Eintrag fehlt, greift eine Heuristik: alkoholische Hinweise → beverage/cookable=false; sonst food/true

ingredients_auto.json – Info

5) ingredients_auto.json – Info
   Wird vom Skript erzeugt/aktualisiert und enthält die automatisch aus der Karte extrahierten Zutaten   inkl. Menü-Signatur-Hash, um unnötige Neuberechnungen zu vermeiden. Du musst diese Datei nicht manuell pflegen.

Typische Nutzung (CLI)

6) Typische Nutzung

- Notion-Felder einrichten/prüfen:
  python -m social_post.cli --setup-notion-fields

- 30 Tage planen, mit Ingredient-Carousels (6 Slides):
  python -m social_post.cli --start 2025-09-01 --days 30 --carousel-ingredients --carousel-slides 6

- Testlauf ohne OpenAI (schnell, Platzhalter):
  python -m social_post.cli --start 2025-09-01 --days 7 --carousel-ingredients --skip-ai --dry-run --verbose

- Zutaten anreichern und speichern:
  python -m social_post.cli --enrich-ingredients --enrich-limit 10 --write-enriched-overrides

Best Practices

7) Best Practices
   • overrides: Halte die Facts prägnant (100–300 Zeichen), sachlich, keine Heilsversprechen
   • meta: Pflege Ausnahmen (Getränke, nicht kochbar) explizit ein; erleichtert robuste Texte
   • Multi-Plattform: In Notion ist „Plattform“ ein Multi-Select – n8n sollte pro ausgewählter Plattform posten
   • Carousel: AI-Vorschlag bevorzugt als JSON speichern; dein n8n-Parser repariert/liest das robust
   • Drive: Wenn genutzt, Media Link + Folder befüllen; `Media Status` in n8n auf generated/ready/posted setzen

Troubleshooting

8) Troubleshooting (kurz)
   • Property fehlt in Notion → --setup-notion-fields ausführen
   • Carousel wird in n8n nicht erkannt → in n8n das Feld „Plattform“ (multi-select) prüfen; nicht „Medientyp“
   • Keine Drive-Ordner/Links → .env prüfen: GOOGLE_DRIVE_SA_FILE + DRIVE_PARENT_FOLDER_ID
   • AI-Vorschlag kaputt → n8n-Reparatur aktiv lassen; im Skript kompakte JSONs verwenden