Super Entscheidung!
Hier kommt dein **Lernzettel** für:

📘 prompt_toolkit + asyncio: Eingabe & Hintergrundprozesse gleichzeitig

---

🎯 Ziel dieses Lernzettels

Du lernst, wie du mit `prompt_toolkit` und `asyncio` ein Terminalprogramm erstellst, das:

- gleichzeitig **Benutzereingaben** annimmt
- im **Hintergrund andere Tasks** ausführt (z. B. E-Mails prüfen)
- **nicht blockiert** und sauber reagiert

✅ Kein zusätzlicher Code – du kannst mit diesem Zettel **offline** arbeiten.
✅ Kompatibel mit deinem geplanten Projekt (z. B. E-Mail-Antworter + CLI-Modus)

🔧 1. Die Tools

Modul	Zweck
`prompt_toolkit`	Für Benutzereingaben (prompt, Shortcuts, Styling, etc.)
`asyncio`	Für parallele Abläufe im Hintergrund

🧠 2. Wichtige Begriffe

Begriff	Erklärung
Blocking Input	Das Programm „wartet“ auf eine Eingabe (z. B. bei `input()`) und tut in der Zeit **nichts anderes**
Async Input	Die Eingabe läuft „nebenbei“ – währenddessen können andere Prozesse weiterarbeiten
Eventloop	Das zentrale Steuersystem von `asyncio`, das Aufgaben nacheinander, aber scheinbar gleichzeitig ausführt

---

📦 3. Die Grundstruktur

```
from prompt_toolkit import PromptSession
import asyncio

session = PromptSession()

async def get_input():
```

```
    while True:
        user_input = await session.prompt_async(">> ")
        print(f"Du hast eingegeben: {user_input}")

async def background_task():
    while True:
        print("[INFO] Hintergrundprozess läuft...")
        await asyncio.sleep(5)  # Simuliere z. B. Email-Check

async def main():
    await asyncio.gather(
        get_input(),         # Benutzereingabe
        background_task()    # Nebenprozess
    )

asyncio.run(main())
```

🔄 4. Was passiert hier genau?

🔹 `PromptSession()`

Erstellt eine *Eingabe-Session*, mit Formatierung, History, Autovervollständigung, etc.

🔹 `session.prompt_async()`

Eingabe ohne Blockieren – das ist der zentrale Unterschied zu `prompt()`.

🔹 `asyncio.gather()`

Führt mehrere Aufgaben **gleichzeitig** aus (z. B. Eingabe und E-Mail prüfen).

💡 5. Eingabe verbessern (Toolbar, Passwort, Completer, etc.)

Du kannst `prompt_async()` **genauso wie `prompt()`** erweitern:

```
await session.prompt_async(
    "Passwort: ",
    is_password=True,
    bottom_toolbar="Dein IMAP-Passwort wird verschlüsselt gespeichert."
)
```

Oder mit Vervollständigung:

```
from prompt_toolkit.completion import WordCompleter

commands = WordCompleter(["start", "stop", "status", "exit"], ignore_case=True)

await session.prompt_async("Befehl: ", completer=commands)
```

⛑️ 6. Eingabe bei aktiven Prozessen (wichtiger Hinweis)

Die Ausgabe deines Hintergrundprozesses (z. B. `print()`) **kann die Eingabezeile unterbrechen**.

💡 Lösung: Verwende `prompt_toolkit.patch_stdout()`:

```
from prompt_toolkit.patch_stdout import patch_stdout

async def main():
    with patch_stdout():
        await asyncio.gather(
            get_input(),
            background_task()
        )
```

Damit bleibt deine Eingabe stabil, selbst wenn im Hintergrund `print()` läuft.

🛑 7. Programm sauber beenden (z. B. mit `exit`)

In `get_input()` kannst du prüfen:

```
if user_input.strip().lower() == "exit":
    print("Programm wird beendet.")
    break
```

Nach dem `break` wird `get_input()` beendet – und wenn du `asyncio.gather()` verwendest, wird auch das gesamte Programm beendet, **wenn alle Tasks fertig sind**.

Möchtest du **nur einen Task** beenden und andere laufen lassen? Dann brauchst du eine erweiterte `Task-Steuerung` (siehe Bonus unten).

🧱 8. So planst du dein echtes CLI-System

Teil	Umsetzung mit `prompt_toolkit` + `asyncio`
E-Mail regelmäßig prüfen	Als eigener `async`-Task mit `sleep(...)`
Benutzereingaben	Per `prompt_async()` in Loop
Passwort & Autovervoll.	Direkt im Prompt integriert
Shortcuts (z. B. Strg+C)	Siehe `Application`, optional

🎓 Bonus: Prozesse gezielt stoppen

Wenn du mehrere Tasks in `asyncio.gather()` verwendest, kannst du bestimmte Tasks auch gezielt abbrechen – etwa wenn `get_input()` „exit“ erkennt:

```
# tasks = asyncio.gather(get_input(), background_task())
# try:
#     await tasks
# except CancelledError:
#     tasks.cancel()
```

Dazu kann ich dir bei Bedarf gern einen eigenen Lernzettel machen („Tasksteuerung mit asyncio“).

✅ Zusammenfassung

Ziel	Technik
Eingaben nicht blockieren	`session.prompt_async()`
Prozesse gleichzeitig	`asyncio.gather()`
Ausgabe bleibt stabil	`with patch_stdout()`
Vervollständigung, Styling	`prompt_toolkit`-Features
Sauber beenden	`break` + ggf. `cancel()`

📬 Was du als Nächstes tun kannst

Wenn du willst, kann ich dir einen **passenden Plan zur Umsetzung in deinem eigenen Projekt** erstellen – z. B.:

- Wie `get_input()` deinen CLI-Modus steuert
- Wie `background_task()` Mails prüft und Aktionen triggert
- Wie du `prompt_toolkit` dynamisch je nach Modus konfigurierst

Sag einfach: **„Ja, bitte hilf mir bei der Projektstruktur!“**
Dann baue ich dir einen passenden Plan – Schritt für Schritt.