Perfekt. Du möchtest **wirklich jedes Detail** verstehen – von `BaseSession`, `prompt_toolkit`-Tools wie `WordCompleter`, bis zur **ganz konkreten Umsetzung**.

Ich erkläre dir jetzt **jede einzelne Komponente**, wie ein Profi mit 40 Jahren Erfahrung das für einen motivierten Anfänger aufbereiten würde.
Du wirst damit **offline in der Lage sein**, das ganze System **perfekt umzusetzen, zu erweitern und zu verstehen**.

🧱 Teil 1: `BaseSession` – Warum, wie, wo?

❓ Was ist `BaseSession`?

Ein **Basis-Baustein** (Basisklasse), von dem alle anderen "Session-Klassen" erben.

📌 Warum brauchst du das?

Weil du viele Sessions hast:

- `SMTPSession`
- `IMAPSession`
- `PasswordSession`
- `FilterSession`
- …

Alle sollen:

- eine `.run()` Methode haben
- ihre Daten irgendwo speichern (z. B. `self.result`)
- später evtl. Validierung, Speicherung oder Abbruchlogik gemeinsam nutzen

Wenn du das nicht zentral in `BaseSession` definierst, **wiederholst du Code** in jeder Klasse.

📁 Wo soll `BaseSession` hin?

```
projekt/sessions/base_session.py
```

Dort liegt sie gut getrennt vom Rest, wird von allen anderen Sessions importiert.

🧩 Wie sieht `BaseSession` konkret aus?

```
class BaseSession:
    def __init__(self):
        self.result = {}
```

```
    def run(self):
        raise NotImplementedError("Diese Methode muss in der abgeleiteten Klasse überschrieben werden.")

    def get_result(self):
        return self.result
```

🔄 Was passiert bei der Verwendung?

Wenn du z. B. `SMTPSession(BaseSession)` schreibst, **erbt** `SMTPSession` automatisch:

- `__init__()` → `self.result = {}` wird gesetzt
- `get_result()` → erlaubt Zugriff auf die Daten

Jede `Session` bekommt so automatisch **gleiche Struktur**, ist leicht zu warten und testen.

🧠 Und wenn du willst, kannst du später `BaseSession` erweitern:

- Mit Logging
- Mit Fehlerbehandlung
- Mit automatischer Speicherung
- Mit „Zurück“-Funktion

🧱 Teil 2: `prompt_toolkit`-Tools: `WordCompleter`, `AutoSuggest`, `Validator`, `bottom_toolbar` …

Jetzt erkläre ich **ganz genau**, was diese Tools machen, wann du sie brauchst und wie du sie **sauber in dein System einbaust.**

✅ `prompt()`

Die zentrale Funktion von `prompt_toolkit`. Sie fragt den Benutzer nach Eingaben – **mit Profi-Features**, wie:

Feature	Beschreibung
`validator=`	Prüft die Eingabe direkt
`bottom_toolbar=`	Zeigt kontextbezogene Hilfe
`is_password=True`	Macht die Eingabe unsichtbar
`completer=`	Vorschläge während der Eingabe (Autocomplete)
`default=`	Vorbelegung der Eingabe

Feature	Beschreibung
`auto_suggest=`	Vorschläge aus bisherigen Eingaben

🧠 Beispiel für ein prompt mit allem:

```
from prompt_toolkit import prompt
from prompt_toolkit.completion import WordCompleter
from prompt_toolkit.auto_suggest import AutoSuggestFromHistory
from prompt_toolkit.validation import Validator, ValidationError

email_completer = WordCompleter(["info@", "support@", "kontakt@"],
ignore_case=True)

class MyValidator(Validator):
    def validate(self, document):
        if "@" not in document.text:
            raise ValidationError(message="Keine gültige E-Mail!")

email = prompt(
    "Gib deine Mail ein: ",
    completer=email_completer,
    validator=MyValidator(),
    auto_suggest=AutoSuggestFromHistory(),
    bottom_toolbar=lambda: "E-Mail deines Providers eingeben"
)
```

🧠 Wann brauchst du `WordCompleter`?

- Wenn du Vorschläge **aus einer festen Liste** geben willst
- z. B. für Provider, Filtertypen, Kategorien, Länder, etc.

🧠 Wann brauchst du `AutoSuggestFromHistory`?

- Wenn du willst, dass frühere Eingaben automatisch vorgeschlagen werden – z. B. wenn man das Programm öfter nutzt

🧠 Wann brauchst du `Validator`?

- Immer, wenn die Eingabe **einen festen Aufbau** haben soll
- z. B. E-Mail-Adresse, Portnummer, Passwortlänge etc.

🧠 Wann brauchst du `bottom_toolbar`?

- Wenn du Hilfe zum aktuellen Feld geben willst – ohne die Frage selbst zu stören

---

🧱 Teil 3: So baust du deine Eingabe-Sessions jetzt professionell auf

---

🗂️ Ordnerstruktur (nochmal ergänzt):

```
projekt/
│
├── main.py                         ◀ Einstiegspunkt
│
├── sessions/                       ◀ Einzelne "Themenbereiche"
│   ├── base_session.py
│   ├── smtp_session.py
│   ├── imap_session.py
│   ├── password_session.py
│   └── filter_session.py
│
├── validators/                     ◀ Prüfungen für Eingaben
│   ├── smtp_validators.py
│   └── filter_validators.py
│
├── resources/                      ◀ Texte, Labels, Tooltips
│   ├── prompts.py
│   └── choices.py                  ◀ z. B. für WordCompleter
│
├── storage/                        ◀ Zum späteren Speichern
│   └── config_writer.py
```

📌 Wo genau kommt `prompt()` hin?

- Immer in `run()` der jeweiligen Session
- Dort fragst du alle Eingaben für **diese eine Aufgabe** ab (z. B. SMTP)

📌 Wo kommen `Validatoren` hin?

- Pro Themenbereich in eigene Datei unter `validators/`
- z. B. `PortValidator` in `smtp_validators.py`

📌 Wo definierst du `Completer`, `AutoSuggest`, `Tooltips`?

- In `resources/choices.py` z. B.:

```
PROVIDER_CHOICES = WordCompleter(["gmail.com", "outlook.com", "web.de"])
```

- In `resources/prompts.py` z. B.:

```
PROMPTS = {
    "smtp": {
        "host": {"text": "SMTP-Server: ", "hint": "z. B. smtp.gmail.com"},
        ...
    }
}
```

📌 Wie behandelst du Rückgabewerte?

- In der Session → `self.result = {...}`
- Im Hauptprogramm: `session.get_result()` oder direkt `session.result`

Du kannst sie dann:

- in `.json` speichern
- an ein Setup-Modul übergeben
- weiterverarbeiten

💡 PROFITIPP: Wiederverwendbare Eingabe-Komponenten

Mach dir ein Hilfsmodul, z. B. `input_helpers.py`, mit Funktionen wie:

```
def ask(prompt_text, validator=None, completer=None, default=None, hint=None,
password=False):
    return prompt(
        prompt_text,
        validator=validator,
        completer=completer,
        default=default,
        auto_suggest=AutoSuggestFromHistory(),
        bottom_toolbar=(lambda: hint) if hint else None,
        is_password=password
    )
```

So wird deine Session-Klasse super sauber:

```
from input_helpers import ask
from validators.smtp_validators import PortValidator
from resources.prompts import PROMPTS_SMTP

class SMTPSession(BaseSession):
    def run(self):
        host = ask(PROMPTS_SMTP["host"]["text"], hint=PROMPTS_SMTP["host"]
["hint"])
        port = ask(PROMPTS_SMTP["port"]["text"], validator=PortValidator(),
hint="1-65535")
        ...
```

✅ Fazit: Du kannst jetzt ALLES richtig umsetzen

Thema	Du weißt jetzt
`BaseSession`	Warum und wie man die verwendet, um alles einheitlich zu halten
`prompt()`	Wie es funktioniert und welche Parameter wann sinnvoll sind
`Validator`, `WordCompleter`	Wie man sie trennt und wiederverwendet

Thema	Du weißt jetzt
Rückgabe und Weiterverarbeitung	Wie du sauber Eingaben speicherst und verwendest
Struktur	Wo du welche Datei und Klasse hinlegst und warum

Wenn du willst, kann ich als Nächstes ein vollständiges Setup-Flow-Beispiel skizzieren (z. B. mit "Zurück", "Abbrechen", "Daten speichern").

Sag mir einfach, wohin du als Nächstes willst.