Zusammenfassung: Methoden zum Erstellen eines Accounts in ReMailD

Methodenübersicht

Methode	Datei	Was reingehört	Warum hier?	Warum nicht woanders?
init_new_account(name: str) -> AccountMeta	`manager.py`	- Hole ID von `paths_manager.get_next_id()` (z. B. 1, 2, …).- Erstelle `AccountMeta(id, name, path="accounts/{name}")`.- Erstelle Ordner `accounts/name` und leere `accountData.json`.- Gib `AccountMeta` zurück.	Steuerungslogik: ID-Generierung und Ordnererstellung sind Vorbereitung, nicht für einen Account. `manager.py` koordiniert mit `paths_manager.py` und weiß, wie Name kommt (z. B. `remaild create acc1`).	In `repository.py` wäre falsch, da es nur für einen Account (Dateien schreiben/lesen) ist. ID/Ordner sind allgemein, würden `repository.py` überladen und Single Responsibility Principle (SRP) brechen.
ask_acc_info() -> AccountData, passwords	`manager.py`	- Frage Haupt-/Wiederherstellungs-E-Mail mit `prompt_toolkit` (Defaults aus `paths_manager`).- Frage IMAP/SMTP (Defaults von `Providers`, z. B. `imap.gmail.com`).- Frage Passwörter mit `getpass` (sicher, keine Defaults).- Gib `AccountData` und `(main_pass, recovery_pass)` zurück.	Eingaben sind Steuerungslogik: `manager.py` entscheidet, was/wie gefragt wird (mit Defaults für Schnelligkeit). Koordiniert mit `paths_manager`/`Providers`. Unabhängig von einem Account.	In `repository.py` wäre falsch, da es nur Dateien/`keyring` handhabt. Eingaben würden es aufblähen, SRP brechen und GUI-Änderungen erschweren.
create_account(data: AccountDat	`repository.py`	- Schreibe `AccountData` in `accountData.json`	Speichern ist spezifisch für einen Account:	In `manager.py` wäre falsch, da es Steuerung mit Speicherlogik mischt.

Methode a, passwords: tuple[str, str])	Datei	Was reingehört (via `self.meta.path` aus `Repository.__init__`).- Speichere Passwörter in `keyring` (z. B. `ReMailD_{meta.id}_main_pass`).- Keine Rückgabe.	Warum hier? `repository.py` weiß via `AccountMeta`, wo es speichert. Passt zu SRP, hält Datei-/`keyring`-Zugriffe sicher und klar.	Warum nicht woanders? Würde Tests erschweren und SRP brechen.
create_account(name: str)	`manager.py`	- Rufe `init_new_account(name)` → `AccountMeta`.- Rufe `ask_acc_info()` → `AccountData`, Passwörter.- Erstelle `Repository(meta)`, rufe `repo.create_account(data, passwords)`.- Rufe `paths_manager.add_path(meta)` für `accountPaths.json`.	Hauptsteuerfunktion: Verknüpft Vorbereitung, Eingaben, Speichern, Registrierung. `manager.py` ist der „Chef“, von `main.py` aufgerufen (z. B. `remaild create acc1`).	In `repository.py` wäre falsch, da es nur einen Account handhabt, nicht Koordination (Eingaben, IDs). Würde SRP brechen und `repository.py` überladen.

Warum so und nicht anders?

- **Objektorientiert (OO) und SRP**: Jede Methode hat eine klare Aufgabe. `manager.py` steuert (Logik, Eingaben), `repository.py` handhabt einen Account (Speichern). Alles in `repository.py` würde Eingaben, IDs und Speichern mischen, was unübersichtlich ist und Tests/GUI-Änderungen erschwert.
- **Schnell/Verkaufbar**: Defaults in `ask_acc_info` (in `manager.py`) machen Eingaben schnell (Nutzer drückt Enter). Steuerung in `manager.py` hält CLI klar.
- **Offline**: Nur JSON/`keyring`, kein Netz.
- **Wartbar**: Getrennte Methoden erlauben Änderungen (z. B. GUI), ohne alles umzuschreiben.

Warum nicht alles in `repository.py`?

- `repository.py` würde für alles zuständig sein (IDs, Eingaben, Speichern), was gegen OO ist. Es müsste IDs generieren (`paths_manager`) und Nutzer fragen (`manager.py`), was es überlädt. GUI-Änderungen wären schwer.

Warum nicht alles in `manager.py`?

- Dateioperationen (`accountData.json`, `keyring`) in `manager.py` mischen Steuerung mit Speicherlogik, brechen SRP und erschweren Tests. `repository.py` kapselt Dateizugriffe sicher (z. B. via `meta.path`).

Lern-Tipp (Cooler Lehrer)

- **Stell dir vor**: `manager.py` ist ein Regisseur (sagt: „Bau, frag, speicher!“), `repository.py` ein Techniker (macht Dateien für einen Account).
- **Test-Idee**: Teste `create_account("acc1")` offline, prüfe `accounts/acc1/accountData.json` und `accountPaths.json`.
- **Herausforderung**: Füge in `ask_acc_info` Default für IMAP (z. B. `imap.gmail.com`) hinzu.
- **Wenn unklar**: Frag, wenn z. B. `Repository`-Initialisierung oder Defaults haken – Mini-Tipp kommt!