ReMailD Account Management Offline Coding Guide

PDF Created by Grok: August 10, 2025

Introduction

Hey there, future Python pro! This is your full, concrete todo list guide to coding the entire account management system for ReMailD. As a seasoned Python expert who's built countless OO apps, I'll walk you through every step professionally—thinking in classes, encapsulation, and reusability. But as your crazy cool teacher, I'll keep explanations short, beginner-friendly, and fun: easy words so you can follow without overwhelm. You'll code it all yourself offline (except testing IMAP/SMTP connections, which need internet). I'll explain why each step matters (if not obvious), provide mini-code examples (just 2-5 lines of key parts, saying where they go), and include all needed info like a big dict of email providers' IMAP/SMTP settings (compiled from reliable sources). This "PDF" is text-based—copy it to a word processor like LibreOffice, format with headings/bullets, and save as PDF for printing/offline use. It's long (20+ "pages" worth), but broken into clear sections. Let's build this OO masterpiece: Providers for auto-detect, Account for per-folder data, AccountManager for handling creates/loads. You got this—code step-by-step, test what you can offline!**Todo 3.6:**

Section 1: Project Setup (Why: A clean structure keeps your app organized and scalable, like a pro OO project)

Todo 1.1: Create main project folder called "remaild". Inside, make subfolders: "accounts" (for account folders) and "src" (for code files). Why: Separates data from code for security and modularity.

Todo 1.2: In "src", create empty files: __init__.py (makes it a package), main.py (CLI entry), providers.py (for auto-detect), account.py (per-account logic), account_manager.py (handles multiples).

Todo 1.3: Ensure you have these Python libs (standard or pre-installed before offline): json, os, getpass, imaplib, smtplib. For extras: prompt_toolkit (for prompts/completers), keyring (for secure storage). If not installed, note: pip install prompt-toolkit keyring (do before offline). Why: These are lightweight; keyring uses OS vault for passwords.

Todo 1.4: In main.py, add basic CLI skeleton with argparse for commands like "create <name>", "load <name>".
Mini-example (in main.py, at top after imports):
```python
import argparse
from src.account_manager import AccountManager
parser = argparse.ArgumentParser()
parser.add_argument('command', choices=['create', 'load'])
parser.add_argument('account_name')
```
Easy: This parses commands—args = parser.parse_args() later.

Section 2: Build Providers Class (Why: Auto-detects settings from domain, cutting user input—makes app fast and user-friendly)

Todo 2.1: In providers.py, define class Providers. Use a classmethod to get settings by domain. Why: Classmethod for static access, no instance needed.

Todo 2.2: Create a big dict of providers. Keys: domains (e.g., 'gmail.com'). Values: dict with 'imap_host', 'imap_port', 'imap_security' (e.g., 'SSL'), 'smtp_host', 'smtp_port', 'smtp_security'. From sources:
- gmail.com: imap_host: 'imap.gmail.com', imap_port: 993, imap_security: 'SSL', smtp_host: 'smtp.gmail.com', smtp_port: 587, smtp_security: 'TLS' (alt port 465 'SSL')
- outlook.com/hotmail.com: imap_host: 'outlook.office365.com', imap_port: 993, imap_security: 'TLS', smtp_host: 'smtp-mail.outlook.com', smtp_port: 587, smtp_security: 'STARTTLS'
- yahoo.com: imap_host: 'imap.mail.yahoo.com', imap_port: 993, imap_security: 'SSL', smtp_host: 'smtp.mail.yahoo.com', smtp_port: 465, smtp_security: 'SSL' (alt 587 'TLS')
- icloud.com/me.com: imap_host: 'imap.mail.me.com', imap_port: 993, imap_security: 'SSL/TLS', smtp_host: 'smtp.mail.me.com', smtp_port: 587, smtp_security: 'STARTTLS'
- protonmail.com/proton.me: Note: No standard support; requires Bridge app. Use empty dict or handle specially.
- zoho.com (personal): imap_host: 'imap.zoho.com', imap_port: 993, imap_security: 'SSL', smtp_host: 'smtp.zoho.com', smtp_port: 465, smtp_security: 'SSL'
- zoho.com (org): imap_host: 'imappro.zoho.com', imap_port: 993, imap_security: 'SSL', smtp_host: 'smtppro.zoho.com', smtp_port: 587, smtp_security: 'TLS'
- aol.com: imap_host: 'imap.aol.com', imap_port: 993, imap_security: 'SSL', smtp_host: 'smtp.aol.com', smtp_port: 465, smtp_security: 'SSL/TLS'
- gmx.com: imap_host: 'imap.gmx.com', imap_port: 993, imap_security: 'SSL', smtp_host: 'mail.gmx.com', smtp_port: 587, smtp_security: 'STARTTLS'
- fastmail.com: imap_host: 'imap.fastmail.com', imap_port: 993, imap_security: 'SSL/TLS', smtp_host: 'smtp.fastmail.com', smtp_port: 465, smtp_security: 'SSL/TLS'
- tutanota.com/tuta.com: No standard support; requires app.
- yandex.com: imap_host: 'imap.yandex.com', imap_port: 993, imap_security: 'SSL/TLS', smtp_host: 'smtp.yandex.com', smtp_port: 465, smtp_security: 'SSL/TLS'
- Add more if you know: e.g., hotmail.com same as outlook.com. Why: This dict is your offline database—auto-guess saves time. For security: 'SSL' means IMAP4_SSL, 'TLS' means starttls after connect.

Todo 2.3: In get_settings, return dict or empty if no match.
Mini-example (in providers.py, inside Providers class):
```python
providers = {
    'gmail.com': {'imap_host': 'imap.gmail.com', 'imap_port': 993, 'imap_security': 'SSL', ...},
    # Add all from list above
}
@classmethod
def get_settings(cls, domain: str) -> dict:
    return cls.providers.get(domain, {})
```

Easy: Split email by '@' for domain, call this—gets settings or empty (then prompt user).

Section 3: Build Account Class (Why: Encapsulates one account's data and actions, OO way—keeps logic per-folder)

Todo 3.1: In account.py, define class Account with init taking name, main_email, recovery_email, etc. Create folder "./accounts/{name}". Why: Folder per account for isolation.

Todo 3.2: Store non-secrets in JSON: main_email, recovery_email, imap_host/port/security, smtp_host/port/security (same for recovery if needed).

Todo 3.3: Store passwords in keyring: main_pass, recovery_pass. Why: OS-secure, not in files.

Todo 3.4: Add methods: save() to write JSON and keyring, load() classmethod to read from folder.

Todo 3.5: Add connect_imap(): Use imaplib, handle SSL vs TLS. Why: For auto-answering later.
Mini-example (in account.py, inside __init__):
```python
import os
import json
import keyring
os.makedirs(f'accounts/{self.name}', exist_ok=True)
self.config_path = f'accounts/{self.name}/config.json'
```
Easy: Makes folder, sets path—save() will json.dump there.

Todo 3.6: For save(), dump config dict to JSON, set keyring passwords.
Mini-example (in account.py, inside save method):
```python
config = {'main_email': self.main_email, 'recovery_email': self.recovery_email, 'imap_host': self.imap_host, ...}
with open(self.config_path, 'w') as f:
    json.dump(config, f)
keyring.set_password('ReMailD', f'{self.name}_main_pass', self.main_pass)
keyring.set_password('ReMailD', f'{self.name}_recovery_pass', self.recovery_pass)
```
Easy: Config to JSON, passwords to keyring—secure!

Todo 3.7: For load(), read JSON, get passwords from keyring.

Todo 3.8: For connect_imap(), if security 'SSL': use IMAP4_SSL, else IMAP4 then starttls().
Mini-example (in account.py, inside connect_imap method):
```python
import imaplib
if self.imap_security == 'SSL':
    self.imap = imaplib.IMAP4_SSL(self.imap_host, self.imap_port)
else:
    self.imap = imaplib.IMAP4(self.imap_host, self.imap_port)
    self.imap.starttls()
self.imap.login(self.main_email, self.main_pass)
```
Easy: Connects securely—test offline by mocking, real test needs net.
Todo 3.9: Add connect_smtp() similar, using smtplib.SMTP or SMTP_SSL, then login.

Todo 3.10: Add send_notification(): Use recovery SMTP to send email. Why: For alerts, like logins.
Mini-example (in account.py, inside send_notification method):
```python
import smtplib
from email.message import EmailMessage
msg = EmailMessage()
msg['Subject'] = 'ReMailD Alert'
msg['From'] = self.recovery_email
msg['To'] = self.recovery_email
msg.set_content('Login detected!')
with smtplib.SMTP(self.recovery_smtp_host, self.recovery_smtp_port) as server:
    if self.recovery_smtp_security == 'TLS':
        server.starttls()
    server.login(self.recovery_email, self.recovery_pass)
    server.send_message(msg)
```
Easy: Sends self-email for notification—encapsulate in class.

Section 4: Build AccountManager Class (Why: Manages multiples, like a factory/controller—OO separation of concerns)

Todo 4.1: In account_manager.py, define AccountManager. Init with accounts_dir = 'accounts'.

Todo 4.2: Add create_account(name): Prompt for main_email, recovery_email using prompt_toolkit.

Todo 4.3: For prompts, build WordCompleter from existing accounts' emails (scan JSONs). Why: Reuse data, less typing.
Mini-example (in account_manager.py, inside create_account):
```python
import os
import json
from prompt_toolkit import prompt
from prompt_toolkit.completion import WordCompleter
existing_main = []
for dir in os.listdir(self.accounts_dir):
    config = json.load(open(f'{self.accounts_dir}/{dir}/config.json'))
    existing_main.append(config['main_email'])
completer = WordCompleter(existing_main + existing_recovery)  # Add recovery too
main_email = prompt('Main email: ', completer=completer)
```
Easy: Scans folders, loads JSON, builds completer—fast offline.

Todo 4.4: After email, split domain, call Providers.get_settings(domain). If empty, prompt for host/port/security.

Todo 4.5: Do same for recovery_email settings (SMTP mainly for sends).
Todo 4.6: Prompt passwords with getpass.getpass()—secure input.
Mini-example (in create_account, after settings):
```python
import getpass
main_pass = getpass.getpass('Main password: ')
recovery_pass = getpass.getpass('Recovery password: ')
```
Easy: No echo on input—safe.

Todo 4.7: Create Account instance, set attrs, call save().

Todo 4.8: Add load_account(name): Check folder exists, call Account.load(name).

Todo 4.9: Add list_accounts(): os.listdir(accounts_dir), print them. Why: For CLI usability.

Section 5: Integrate into Main CLI (Why: Ties it all together for running)

Todo 5.1: In main.py, create AccountManager instance.

Todo 5.2: If command 'create', call manager.create_account(args.account_name).

Todo 5.3: If 'load', load and maybe connect/test (offline: just load).
Mini-example (in main.py, after parsing args):
```python
manager = AccountManager()
if args.command == 'create':
    manager.create_account(args.account_name)
elif args.command == 'load':
    account = manager.load_account(args.account_name)
    # account.connect_imap()  # For later, needs net
```
Easy: Central control—expand for auto-answer logic later.

Section 6: Handling Recovery Email Fully (Why: Critical for control, integrated like main but SMTP-focused)

Todo 6.1: In Account, add recovery_smtp_host/port/security attrs, set during create via Providers.

Todo 6.2: In create_account, after recovery_email prompt, auto-detect or prompt SMTP for it.

Todo 6.3: In save/load, include recovery SMTP in JSON.

Todo 6.4: Use in send_notification as above. Why: Allows offline coding, online testing.

Section 7: Security and Edge Cases (Why: Pro apps handle errors, safe by design)

Todo 7.1: Add try/except in prompts for invalid input (e.g., email format regex).

Todo 7.2: For unknown domains, prompt: 'IMAP host (leave blank if unknown): ', default to None.

Todo 7.3: Note: For providers like ProtonMail/Tutanota, warn user "May need bridge app, manual settings."

Todo 7.4: Offline testing: Mock imaplib/smtplib with print statements. Why: Simulate without net.

Section 8: Auto-Answer Stub (Why: Placeholder for core feature, build on this)

Todo 8.1: In Account, add auto_answer(): After connect_imap, select 'INBOX', search unseen, reply via SMTP.

Todo 8.2: But offline: Just code the structure, test connections later.

Section 9: Full Providers Dict Code (Copy this into providers.py)

Use this exact dict—it's your offline ref:

```
providers = {
    'gmail.com': {'imap_host': 'imap.gmail.com', 'imap_port': 993, 'imap_security': 'SSL', 'smtp_host':
'smtp.gmail.com', 'smtp_port': 587, 'smtp_security': 'TLS'},
    'outlook.com': {'imap_host': 'outlook.office365.com', 'imap_port': 993, 'imap_security': 'TLS',
'smtp_host': 'smtp-mail.outlook.com', 'smtp_port': 587, 'smtp_security': 'STARTTLS'},
    'hotmail.com': same as outlook.com,
    'yahoo.com': {'imap_host': 'imap.mail.yahoo.com', 'imap_port': 993, 'imap_security': 'SSL',
'smtp_host': 'smtp.mail.yahoo.com', 'smtp_port': 465, 'smtp_security': 'SSL'},
    'icloud.com': {'imap_host': 'imap.mail.me.com', 'imap_port': 993, 'imap_security': 'SSL/TLS',
'smtp_host': 'smtp.mail.me.com', 'smtp_port': 587, 'smtp_security': 'STARTTLS'},
    'me.com': same as icloud.com,
    'zoho.com': {'imap_host': 'imap.zoho.com', 'imap_port': 993, 'imap_security': 'SSL', 'smtp_host':
'smtp.zoho.com', 'smtp_port': 465, 'smtp_security': 'SSL'},  # Personal; for org use imappro etc.
    'aol.com': {'imap_host': 'imap.aol.com', 'imap_port': 993, 'imap_security': 'SSL', 'smtp_host':
'smtp.aol.com', 'smtp_port': 465, 'smtp_security': 'SSL/TLS'},
    'gmx.com': {'imap_host': 'imap.gmx.com', 'imap_port': 993, 'imap_security': 'SSL', 'smtp_host':
'mail.gmx.com', 'smtp_port': 587, 'smtp_security': 'STARTTLS'},
    'fastmail.com': {'imap_host': 'imap.fastmail.com', 'imap_port': 993, 'imap_security': 'SSL/TLS',
'smtp_host': 'smtp.fastmail.com', 'smtp_port': 465, 'smtp_security': 'SSL/TLS'},
    'yandex.com': {'imap_host': 'imap.yandex.com', 'imap_port': 993, 'imap_security': 'SSL/TLS',
'smtp_host': 'smtp.yandex.com', 'smtp_port': 465, 'smtp_security': 'SSL/TLS'},
    # ProtonMail & Tutanota: {}  # Empty, warn user
}
```
Why: Comprehensive from sources—add POP if needed, but focus IMAP/SMTP.

Section 10: Testing Offline (Why: Verify logic without net)

Todo 10.1: Run create, check folders/JSON/keyring (use keyring.get_password to test).

Todo 10.2: Mock connections: In connect_imap, add if offline_mode: print("Mock connect").

Todo 10.3: Add unit tests in a tests.py using unittest—e.g., test Providers.get_settings.

Section 11: Advanced OO Tips (Why: Make it pro-level)

Todo 11.1: Use @property for attrs like main_pass: return keyring.get... Why: Encapsulates access.

Todo 11.2: Inherit if subaccounts later: class SubAccount(Account).

Todo 11.3: Error handling: Raise custom exceptions, e.g., class AccountError(Exception).

Section 12: Final Polish and Expansion

Todo 12.1: Add docstrings to classes/methods. Why: Pro documentation.

Todo 12.2: For selling: Add --help to CLI, user-friendly messages.

Todo 12.3: Next: Implement auto-reply logic in Account—search emails, craft responses.
Congrats—you've built it offline! If questions (when back online), ask. Code on! 🚀