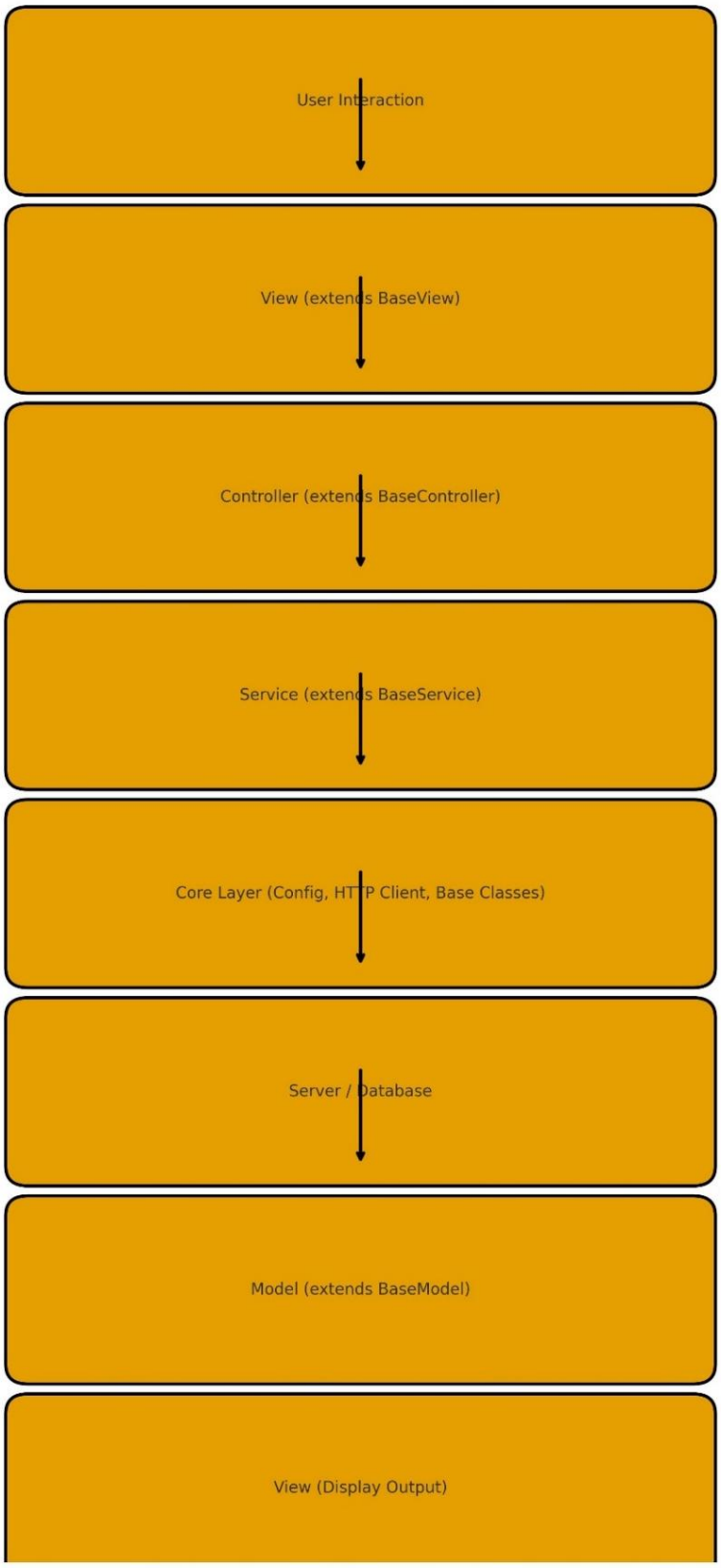
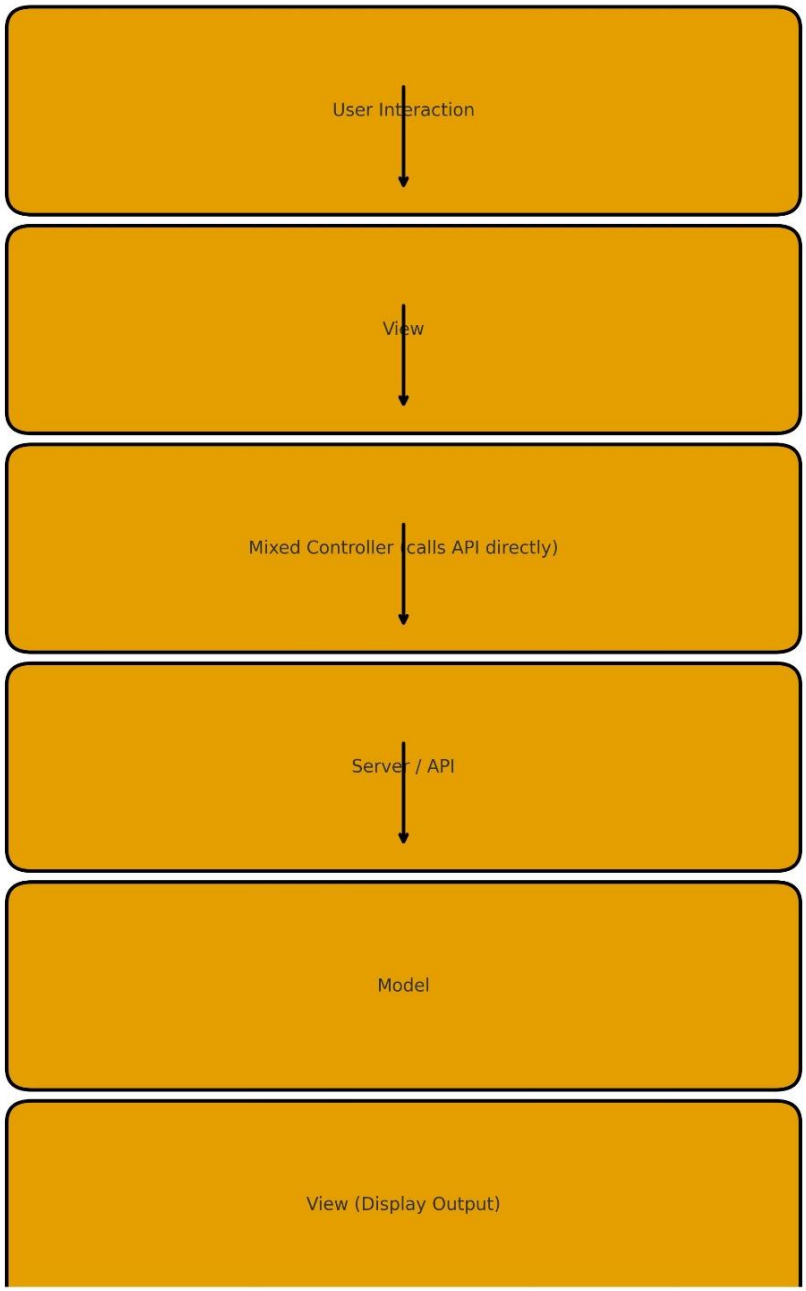
Struktur File Sesudah di refactor



Struktur File Sebelum Refactor



PERUBAHAN PALING BESAR SETELAH RESTRUKTUR (DENGAN BASE CLASS)

1. Aplikasi kini mempunyai “aturan inti” yang mengikat semua modul

Kami menambahkan **empat Base Class** di folder core/:

- BaseController
- BaseView
- BaseService
- BaseModel

Keempatnya berfungsi sebagai:

Fondasi utama
yang mengatur bagaimana *setiap file* di seluruh modul harus bekerja.

Ini membuat semua modul sekarang **konsisten**, **rapi**, dan **ikut aturan arsitektur yang sama**.

---

2. Semua modul sekarang diwariskan dari Base Class

Contoh:

```
class TransaksiController(BaseController):
    def __init__(self):
        super().__init__()
```

Dampaknya:

- Tidak ada controller liar yang berbeda dari modul lain
- Semua modul berjalan dengan **format dan aturan yang sama**
- Perubahan aturan di BaseClass langsung turun ke semua modul

Contoh benefit:

Jika nanti Anda ingin menambahkan debug logging ke semua controller,
cukup modifikasi 1 file: BaseController.
Semua modul otomatis ikut upgrade.

---

3. Pemisahan Tugas Menjadi Sangat Jelas

Dengan base class:

Layer	Tugas	Tidak Boleh
View	UI & event	Memanggil API
Controller	Mengatur alur	Simpan IP / logika API
Service	Logika bisnis + API	Sentuh UI
Model	Struktur data	Akses server

Ini membuat aplikasi:

- jauh lebih stabil
- mudah diperbaiki
- tidak mudah kacau
- cocok untuk tim besar

---

4. CORE sekarang menjadi pusat kekuatan aplikasi

Folder core/ berisi:

- konfigurasi global
- Base Classes
- http client
- logger
- db helper
- device utils

Sehingga:

kalau IP server berubah,
cukup ubah satu file → seluruh aplikasi ikut menyesuaikan.

Tidak ada lagi:

- IP tertanam di controller
- URL tercecer di 20 file berbeda

---

5. Modul benar-benar bersih & mandiri

Contoh modul penjualan/:

controllers/

models/

services/

views/

widgets/

Dengan BaseClass:

Semua modul selalu memiliki controller, service, model, dan view yang **selaras** dan **koheren**.

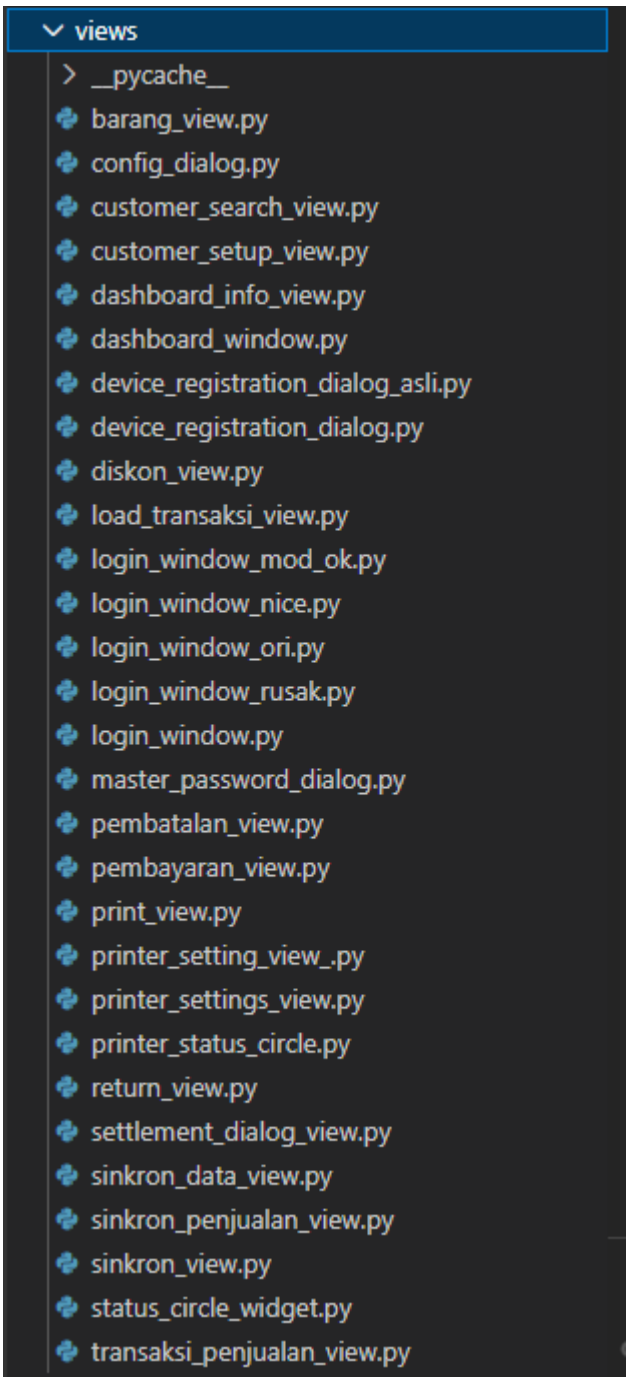
---

3. PERBEDAAN SEBELUM vs SESUDAH

Aspek	**Sebelum**	**Sesudah**
Struktur	Campur, tidak teratur	Modular per fitur
Konsistensi	Berbeda-beda	Seragam karena BaseClass
IP Server	Hardcoded di banyak file	Centralized di core/config_utils
Controller	Berantakan	Terkontrol via BaseController
Service	Tidak terstandarisasi	Semua ikut BaseService
Model	Bebas bentuk	Tertata via BaseModel
View	Bercampur logika	Bersih → event saja
Skalabilitas	Rendah	Sangat tinggi
Kemudahan Dev	Sulit	Mudah & cepat
Maintainability	Tinggi resiko error	Sangat aman
Clean Code	Tidak terjamin	Dijamin oleh BaseClass

Sebelum dilakukan restruktur, aplikasi tidak memiliki standar umum. Setiap file berdiri sendiri dan sulit diprediksi.
Setelah restruktur, kami menambahkan empat Base Class utama di dalam folder Core.
BaseClass ini menjadi landasan arsitektur, sehingga seluruh controller, view, service, dan model di setiap modul wajib mengikuti aturan yang sama. Dengan cara ini, semua modul menjadi konsisten, modular, mudah dipelihara, dan sangat scalable.
Jika aturan di BaseClass diperbarui, seluruh modul aplikasi akan otomatis ikut meningkat kualitasnya.

Struktur Lama



Sruktur lama:

- **Semua controller disimpan dalam satu folder di bagian paling atas (root folder).**

Controller itu ibarat “otak kecil” yang mengatur alur program. Namun karena semuanya ditumpuk jadi satu, setiap fungsi bercampur dan sulit mencari mana yang mengatur fitur tertentu.

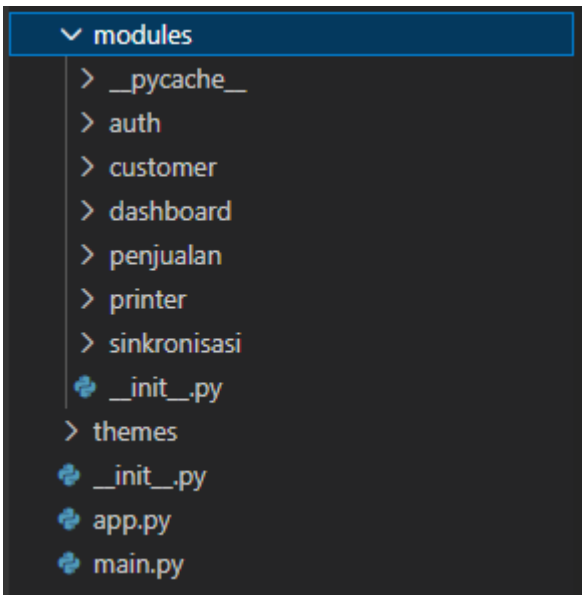
- Semua model juga disimpan dalam satu folder di root.

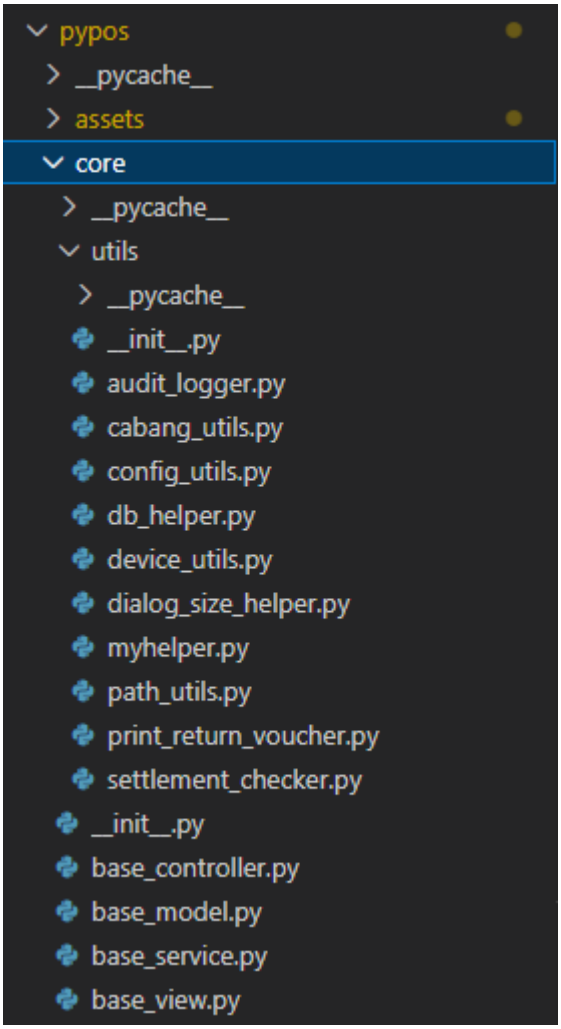
Model adalah bagian yang mengatur data. Semuanya ada di satu keranjang yang sama, sehingga semakin banyak fitur semakin sulit menemukan file yang tepat.

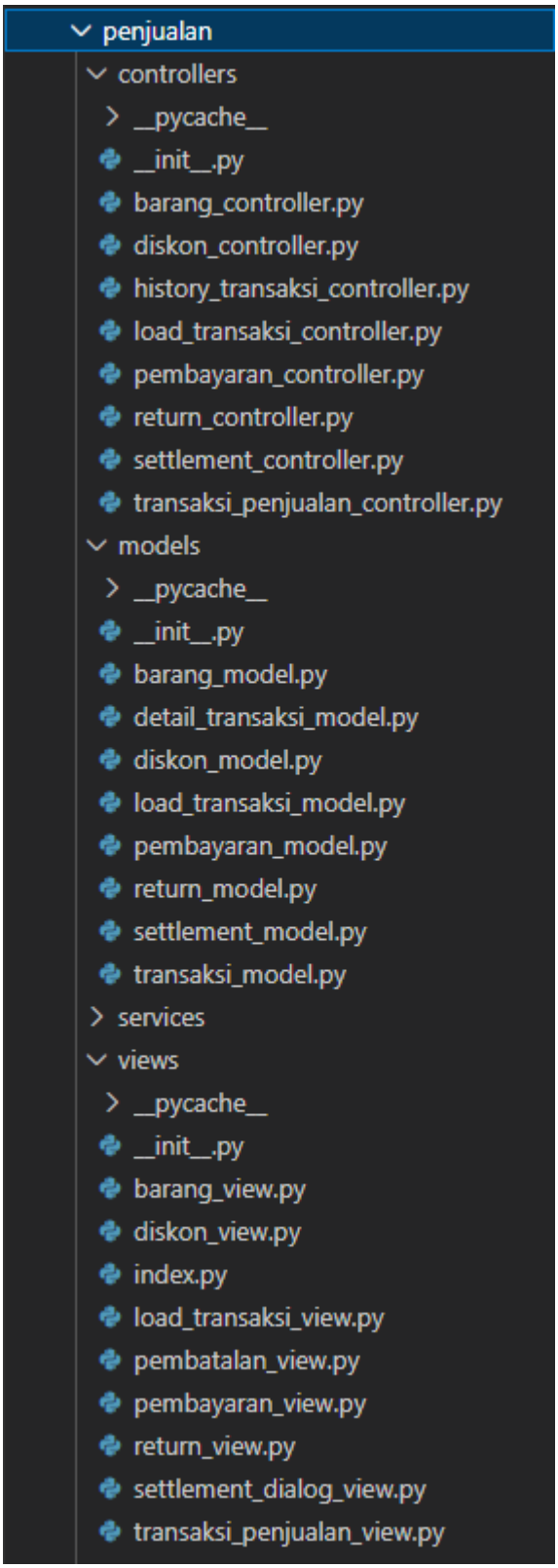
- **Semua view disimpan dalam satu folder di root.**

View adalah tampilan antarmuka. Karena disatukan tanpa pemisahan fitur, tampilan untuk halaman yang berbeda jadi bercampur dan membingungkan saat harus mencari atau memperbaikinya.

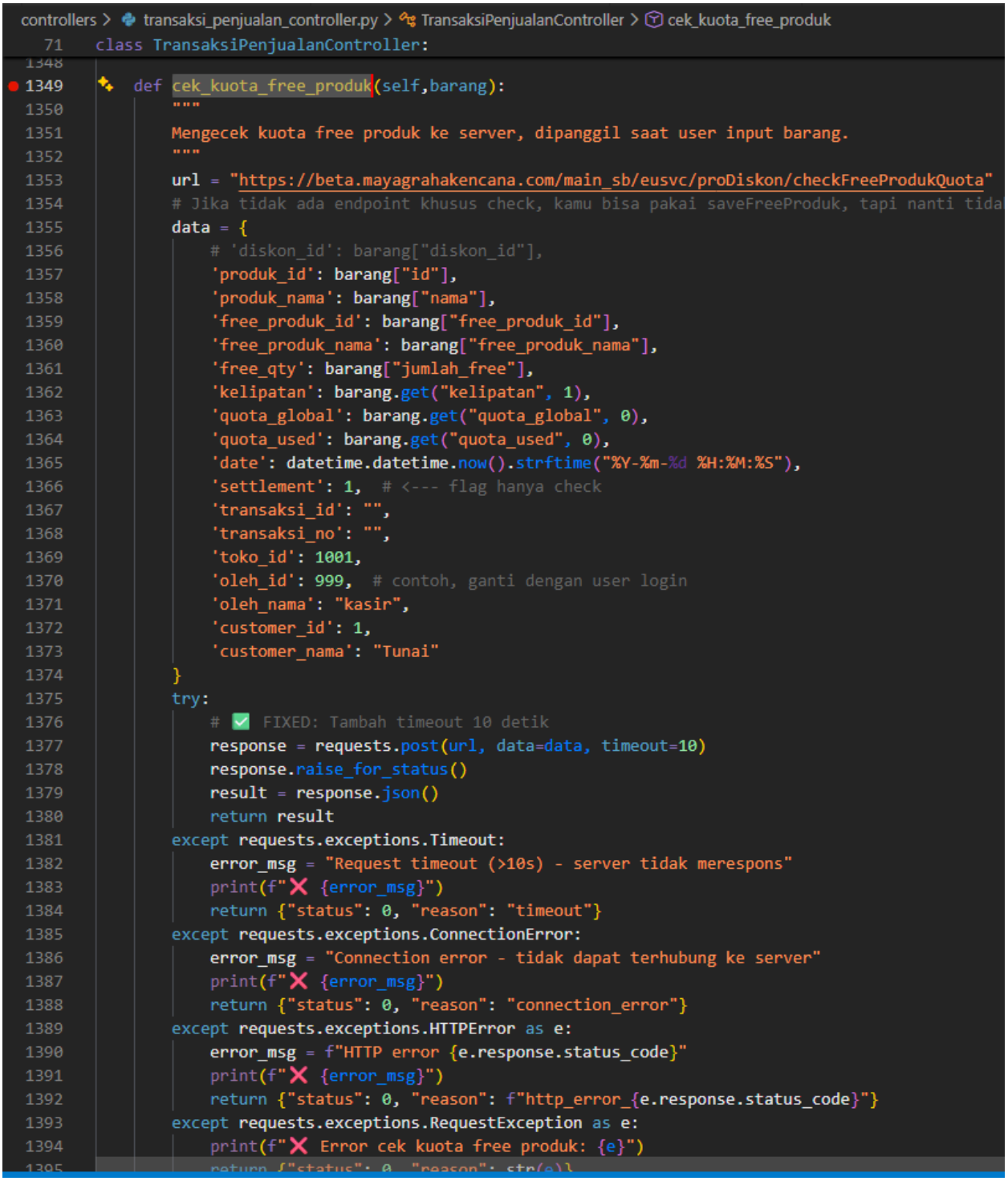
Struktur Baru







Contoh pemindahan yang seharusnya berada di models



#Ini adalah sebelum refactor

1. APA YANG DILAKUKAN KODE ANDA?

Kode tersebut:

- memanggil **server / API endpoint**
- mengirim request POST
- melakukan timeout + error handling
- mengembalikan JSON dari server

Artinya:
Kode ini adalah **komunikasi jaringan + business logic**.

Dan dalam arsitektur PyPOS (Model–View–Service–Manager), aturan utamanya adalah:

API / network call → 100% harus masuk ke SERVICE

#ini setelah refactor

pypos > modules > penjualan > controllers > transaksi_penjualan_controller.py > TransaksiPenjualanController > find_barang_row_by_id

```python
class TransaksiPenjualanController(BaseController):
    def cek_kuota_free_produk(self, barang):
        from pypos.modules.penjualan.services.transaksi_service import TransaksiService
        return TransaksiService.cek_kuota_free_produk(barang, self.user_info)

```

*dibawah ini file di models

pypos > modules > penjualan > services > transaksi_service.py > ...

```python
class TransaksiService:
    def cek_kuota_free_produk(self, barang, user_info=None):
        Mengecek kuota free produk ke server, dipanggil saat user input barang.
        """
        # Konfigurasi endpoint
        from pypos.core.utils.config_utils import load_config
        config = load_config()
        base_url = config.get("api_base_url", "https://beta.mayagrahakencana.com/main_sb").rstrip("/")
        path = config.get("ep_diskon_check_free_produk", "/eusvc/proDiskon/checkFreeProdukQuota")
        url = f"{base_url}{path}"
        data = {
            "produk_id": barang.get("id"),
            "produk_nama": barang.get("nama"),
            "free_produk_id": barang.get("free_produk_id"),
            "free_produk_nama": barang.get("free_produk_nama"),
            "free_qty": barang.get("jumlah_free"),
            "kelipatan": barang.get("kelipatan", 1),
            "quota_global": barang.get("quota_global", 0),
            "quota_used": barang.get("quota_used", 0),
            "date": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            "settlement": 1,
            "transaksi_id": "",
            "transaksi_no": "",
            "toko_id": 1001,
            "oleh_id": 999,
            "oleh_nama": "kasir",
            "customer_id": 1,
            "customer_nama": "Tunai",
        }
        if user_info:
            data["oleh_id"] = user_info.get("id", data["oleh_id"])
            data["oleh_nama"] = user_info.get("nama", data["oleh_nama"])
        try:
            response = requests.post(url, data=data, timeout=10)
            response.raise_for_status()
            return response.json()
        except requests.exceptions.Timeout:
            error_msg = "Request timeout (>10s) - server tidak merespons"
            print(f"[ERROR] {error_msg}")
            return {"status": 0, "reason": "timeout"}
        except requests.exceptions.ConnectionError:
            error_msg = "Connection error - tidak dapat terhubung ke server"
            print(f"[ERROR] {error_msg}")
            return {"status": 0, "reason": "connection_error"}
        except requests.exceptions.HTTPError as e:
            error_msg = f"HTTP error {e.response.status_code}"
            print(f"[ERROR] {error_msg}")
            return {"status": 0, "reason": f"http_error_{e.response.status_code}"}
```

*Controller (di Python menyebutnya manager/coordinator)

Controller hanya boleh:

- menerima event dari View
- memanggil Service
- mengirim hasil ke View

Controller TIDAK BOLEH:

- memanggil URL
- memanggil server
- mengurus timeout
- mengatur error handling request
- melakukan business logic berat

Tempat yang BENAR: SERVICE LAYER

Service adalah:

- jembatan antara Controller ↔ API
- tempat menyimpan semua logic transaksi
- tempat memanggil API
- tempat error handling