1 Strategic-Level Code Chunks

(major folders / top-level modules you will commit to the repo)

Folder / File	**Purpose**	**Typical Size**
super_earth/	Package root; houses all source.	—
__init__.py	Makes the package importable.	tiny
main.py	*Entry point*. Builds Tk root, styles, global queues, loads config, instantiates feature controllers, launches threads, traps shutdown.	150-250 loc
config.py	Reads settings.yaml (camera URLs, API keys, SDR frequencies...). Provides a singleton Settings object.	80-120 loc
ui/	Pure Tkinter & ttk widgets—no blocking I/O.	—
dashboard.py	Lays out frames (camera, calendar, radio, aircraft) and exposes register_panel(name, frame) for features.	120-180 loc
threads.py	Lightweight helper to start/stop worker threads and marshal callbacks back into the Tk loop (root.after).	60-90 loc
calendar/	Google Calendar integration.	—
camera/	Reolink video capture.	—
sdr/	RTL-SDR / rtl_433 scanning.	—
adsb/	dump1090 aircraft listener.	—
map/	TkinterMapView & marker persistence.	—
tests/	pytest folder with unit tests for parsers & helpers.	—
settings.yaml	User-editable config (credentials, IPs, lat/lon, etc.).	—
requirements.txt	Exact PyPI dependencies; plus comment lines for external EXEs (rtl_433.exe, dump1090.exe).	—

2 Tactical-Level Tasks per Feature

2.1 Base Infrastructure (super_earth.main, dashboard, threads, config)

Task	Key Functions / Classes to Write	Notes

Bootstrap	def main():
	root = tk.Tk()
	Dashboard(root)
		Full-screen logic, DPI scaling, theme load.

Style loader	apply_global_style(root)	ttk Style, large fonts, dark theme.
Graceful shutdown	AppController.stop_all()	Sets stop events, waits on threads, destroys root.
Thread helper	class TkThread(threading.Thread) with safe_callback(fn,*a) → marshals to Tk via root.after(0,fn).	Prevents GUI calls from worker threads.
Global queues	radio_queue, adsb_queue, etc.	queue.Queue(maxsize=N) for each feed.
Config loader	Settings.from_file("settings.yaml")	Parses YAML, validates, exposes typed fields (camera list, calendar ID, etc.).

2.2 Calendar Module (calendar/service.py, calendar/panel.py)

Layer	Code to Write	Details
Service (worker)	class CalendarService(TkThread): authenticate() (OAuth flow, save token.json) fetch_events() → returns list[Event]	Thread sleeps 10 min, refetches. Emits on_events(events) to UI.

UI panel	class CalendarPanel(ttk.Frame):
	update(events)
		Treeview or Listbox, date formatting helper (pretty_date).

2.3 Camera Module (camera/feed.py, camera/panel.py)

Layer	Code to Write	Details

Capture thread	class CameraFeed(TkThread):
	run() opens cv2.VideoCapture(url)
	Loop → read frame → convert BGR→RGB → ImageTk.PhotoImage → safe_callback(ui.draw, img)
		Reconnect logic on failure; FPS throttle.

UI widget	class CameraPanel(ttk.Frame): Holds tk.Label for image. draw(img) replaces label image.	Optionally rotate among feeds via next_camera().

2.4 SDR Module (sdr/rtl433.py, sdr/panel.py)

Layer	Code	Details
Subprocess wrapper	run_rtl433() launches rtl_433 -F json with subprocess.Popen.	Parse stdout lines: json.loads(line) → dict.
Parser	def format_msg(d): returns user string ("Temp 72 °F, ID 101").	Map Celsius→F; identify sensor model.
Worker thread	class Rtl433Thread(TkThread) reads lines, safe_callback(queue.put, msg).	Optional second thread for scanning.
UI	class RadioPanel(ttk.Frame) shows scrolling tk.Text or Listbox of latest N messages.	Colors by sensor type (weather, alert, etc.).

2.5 ADS-B Module (adsb/listener.py, adsb/panel.py)

Layer	Code	Details
Socket listener	class AdsbListener(TkThread): Connect to ('127.0.0.1', 30003); buffer recv(4096). Split by \r\n, call parse_sbs(msg)	Maintain aircraft: dict[id] → AircraftState.
Parser	parse_sbs(line) extracts callsign, altitude, ground speed.	Handle MSG types 1–8; return tuple.
Data model	class AircraftState (id, callsign, alt, speed, ts_last).	Cleanup stale (>60 s) in thread.

Layer	Code	Details
UI	class AircraftPanel(ttk.Frame) with ttk.Treeview columns: Flight, Alt(ft), Speed(kts). refresh() reads snapshot under lock.	Refresh every 1 s via after.

2.6 Map Module (map/view.py, map/storage.py)

Layer	Code	Details
Map widget	class MapWindow(tk.Toplevel) builds TkinterMapView. Sets satellite tile server. set_position(settings.lat, settings.lon)	Buttons: *Add Note*, *Save*, *Close*.
Marker add flow	On *Add Note* toggle → next click callback returns lat/lon → simple askstring() dialog for label → call set_marker.	
Persistence	save_markers(path) dumps list[{lat,lon,label}] to JSON; load_markers() on start.	
Integration button	In Dashboard add “Map” button; opens MapWindow.	

2.7 Testing & Utilities

Category	Code	Notes
Unit tests	tests/test_adsb_parse.py, tests/test_rtl433_parse.py, etc.	Use captured sample lines.
Mock helpers	Fake Google Calendar JSON, fake SBS lines for tests.	
Installer script	scripts/post_install.ps1 to copy rtl_433.exe & dump1090.exe next to Python exe; set ENV PATH.	Optional but helpful.

---

3 Approximate Effort (per feature)

Feature	New LOC (est.)	Comments
Core app / dashboard	350-450	includes style, shutdown, config

Feature	New LOC (est.)	Comments
Calendar	120-180	most work is API auth boilerplate
Camera	150-220	frame loop + UI; heavy but straightforward
SDR	180-250	subprocess mgmt, JSON parsing, UI log
ADS-B	160-220	low-level parsing + table UI
Map	140-200	interactive widget + save/load
Total	**~1 100-1 500 LOC**	excluding tests & comments

---

4 Implementation Order

1. **Skeleton**: main.py, dashboard, config.
2. **Camera** (quick visual win & tests threading).
3. **Calendar** (introduces Google auth flow).
4. **SDR (rtl_433)** – verify subprocess & queue pattern.
5. **ADS-B** – socket parsing, table refresh.
6. **Map** – embed TkinterMapView; add persistence.
7. **Polish** – graceful exit, hot reload settings, high-DPI tweaks.
8. **Unit tests & packaging**.