Raeburn Brain AI – Full Module Overview

This document provides a complete, subsystem-level overview of every module currently designed, implemented, or planned within Raeburn Brain AI. It captures both core functionalities and optional extensions for memory, routing, agent identity, plugins, communications, model management, business logic, and orchestration.

🤖Core Modules

`core/memory_store.py`

- Interface to persistent memory (TinyDB, SQLite, PostgreSQL)
- Features:
- Sharded per-agent memory stores
- Importance-based pruning
- Expiry support
- Tag-based semantic indexing
- Full-text search (FTS5 or Postgres tsvector)

`core/memory_injector.py`

- Injects top N (typically 3) relevant memory entries into prompt context
- Filters by tags, importance, and recency
- Formats using Jinja2-compatible block structure

`core/router.py`

- Main model router logic
- Supports:
- UCB1, Thompson Sampling, Softmax selection
- Retry/backoff logic
- Failure handling and fallback
- Streaming output support (via `yield` / SSE)

`core/scoring.py`

- Calculates model score per output using:
- Latency
- Cost per token
- Failure rate
- Formula: `score = quality - (latency * 0.1) - (cost * 10) - (fail_rate * 5)`

`core/db_layer.py`

- Provides a uniform backend layer for DB access
- Abstracts over SQLite, TinyDB, and PostgreSQL
- Ensures schema setup, connection pooling, and FTS

`core/decision_engine.py`

- Maps context, agent traits, and tags to appropriate prompt templates
- Loads templates from `agent_templates/`
- Can use past output memory for adaptive selection

`core/event_bus.py` ***(Planned)***

- Internal pub/sub for memory updates, agent events, routing logs
- Useful for async pipelines and plugin messaging

🤓Agent System

`agents/identity_engine.py`

- Generates new agent identities
- Outputs:
- ID
- Name
- Traits
- Title
- Created timestamp
- Traits loaded from `agent_traits.json`

`agents/agent_registry.json`

- Stores agent metadata
- Includes stats like win rate, failure rate, last promotion, sandbox status

`agents/agent_templates/`

- Jinja2 prompt templates for agents
- Categorised by task type, tone, and role
- Used by `decision_engine.py`

`agents/chat_templates.json` ***(Planned)***

- Defines structured dialogue formats
- Supports agent-to-agent or agent-to-user chat threading
- Includes state, memory mood modifiers

😭Model Infrastructure

`tools/model_fetcher.py`

- Downloads local models (.gguf) from HuggingFace or mirrors
- Verifies:
- Size
- Hash

- Signature (optional)
- Decompression

`tools/model_health.py`

- Sends synthetic test prompts:
- Echo
- Logic
- Math
- Evaluates:
- Response latency
- Correctness
- Format compatibility
- Outputs to Prometheus or webhook

`config/model_registry.json`

- Stores metadata about each model:
- Latency
- Cost
- Avg score
- Failure rate

`config/models_installed.json`

- Tracks models installed via the fetcher
- Metadata for `model_fetcher.py`

🌙 Business Factory + Meta-Agent Engine

`business_factory/create_business.py`

- Orchestrates daily business spawning
- Assigns agent roles: marketing, dev, outreach, lead
- Uses highest scoring agents from registry

`meta_agent/daily_mission.py`

- Assigns KPIs, business goals, and mission briefs to teams
- Records progress via memory updates
- Can reassign or demote underperforming agents

`meta_agent/scoring_loop.py`

- Scores agents based on task performance, response quality, and feedback
- Promotes or demotes agents
- Flags for retirement or merging

`meta_agent/postmortem.py` ***(Planned)***

- Analyzes failed agents, tasks, and loops

- Writes structured logs to `logs/postmortem/`
- Suggests replacements or retraining needs

👹Observability & Services

`tools/service.py`

- Entrypoint for running the Raeburn system
- Exposes:
- `/healthz`
- `/metrics`
- `/uptime`
- Manages:
- Background threads (health, scheduler)
- Graceful shutdown

`tools/dashboard.py`

- Minimal Web UI with:
- Model status
- Agent list
- Logs + business status
- Basic auth support

`tools/preflight.py`

- Verifies:
- Python version
- DB connection
- Port availability
- Directory structure
- Permissions

`tools/daily_cron_health.py`

- Can be triggered by cron/systemd
- Re-validates model health + registry updates

👉🏽Deployment

`Dockerfile`

- Multi-stage build
- Non-root user

`docker-compose.yml`

- Starts Redis (for Celery), API service, and model services
- Healthcheck directive for service validation

`systemd/raeburn.service`

- Optional unit file for host OS boot/start
- Auto-restart with log persistence

`Makefile`

- `make production` or `make dev`
- Starts Docker build + run

`config/settings.py`

- Pydantic config loader
- Validates all critical environment variables on boot

👇Communication Modules

`comms/telegram_bot.py`

- Telegram bot integration
- Sends agent alerts, business updates
- Optional interactive commands for control

`comms/voice_agent.py` ***(Planned)***

- Whisper + TTS support for voice interactions
- Used in team briefings, outbound calls

`comms/slack_bridge.py` ***(Optional)***

- Post business milestones or messages to Slack channel

❤️Plugin System

`plugins/`

- Folder where all plugin modules reside
- Each contains `meta.json` with:
- `sandboxed`
- `description`
- `priority`

`plugins/raeburnEmailer/`

- Email marketing engine plugin
- Used by Factory Mode to handle outreach
- Includes segmentation, suppression, bounce handling

😜External Integration Hooks *(Optional/Planned)*

`external/openrouter.py`

- Sends requests to OpenRouter-hosted models

`external/huggingface.py`

- Queries HuggingFace inference API (e.g. CodeGemma, Qwen)

`external/searx_rag.py` ***(Planned)***

- Uses SearxNG or Brave API to gather factual context and inject into prompt

---

👉🏿Summary

This module-level architecture provides a complete overview of the Raeburn Brain AI system. Each module is independently testable, plug-in compatible, and designed to support real-time decision-making, agent evolution, autonomous output, and robust operations in either cloud, hybrid, or offline deployments.