Круто, это очень хорошее усиление структуры. Ниже — короткий «дифф-пакет»: что принять + готовые шаблоны файлов и настроек под **src-layout**.

**Что принимаю как есть**

* src/ + mf\_core/ — 👍
* Новые файлы: api/dependencies.py, api/errors.py, api/middleware.py, agents/base.py, llm/caching.py, workers/tasks.py, telemetry/tracing.py, services/\*, scripts/prestart.sh, .pre-commit-config.yaml.
* Тесты: tests/{unit,integration} + conftest.py.
* K8s манифесты для Redis/Qdrant.

**Готовые шаблоны (вставляй как есть)**

**pyproject.toml — src-layout + dev-инструменты**

[build-system]

requires = ["setuptools>=65.0", "wheel"]

build-backend = "setuptools.build\_meta"

[tool.setuptools]

package-dir = {"" = "src"}

packages = { find = { where = ["src"] } }

[project]

name = "mindforge-core"

version = "0.1.0"

description = "MindForge Core: knowledge/RAG/LLM platform"

requires-python = ">=3.11"

readme = "README.md"

license = { text = "MIT" }

dependencies = [

"fastapi>=0.115",

"uvicorn[standard]>=0.30",

"pydantic>=2.6",

"python-dotenv>=1.0",

"httpx>=0.27",

"requests>=2.32",

"sentence-transformers>=3.0",

"qdrant-client>=1.9",

"faiss-cpu>=1.8.0",

"redis>=5.0",

"prometheus-client>=0.20",

"celery>=5.4",

"opentelemetry-sdk>=1.26.0",

"opentelemetry-instrumentation-fastapi>=0.47b0",

"opentelemetry-exporter-otlp>=1.26.0"

]

[project.optional-dependencies]

dev = [

"pytest>=8.2",

"pytest-cov>=5.0",

"mypy>=1.10",

"ruff>=0.5",

"black>=24.8",

"isort>=5.13",

"pre-commit>=3.7"

]

[tool.ruff]

line-length = 100

select = ["E","F","I","UP"]

[tool.mypy]

python\_version = "3.11"

ignore\_missing\_imports = true

strict = false

[tool.pytest.ini\_options]

addopts = "-q --maxfail=1 --disable-warnings"

**.pre-commit-config.yaml**

repos:

- repo: https://github.com/charliermarsh/ruff-pre-commit

rev: v0.5.6

hooks: [{ id: ruff, args: ["--fix"] }]

- repo: https://github.com/psf/black

rev: 24.8.0

hooks: [{ id: black }]

- repo: https://github.com/pycqa/isort

rev: 5.13.2

hooks: [{ id: isort }]

- repo: https://github.com/pre-commit/mirrors-mypy

rev: v1.10.0

hooks: [{ id: mypy, additional\_dependencies: ["pydantic>=2.6"] }]

**src/mf\_core/api/dependencies.py**

from typing import Annotated

from fastapi import Depends

from mf\_core.services.search\_service import SearchService

from mf\_core.services.summary\_service import SummaryService

from mf\_core.services.file\_service import FileService

def get\_search\_service() -> SearchService:

return SearchService()

def get\_summary\_service() -> SummaryService:

return SummaryService()

def get\_file\_service() -> FileService:

return FileService()

SearchDep = Annotated[SearchService, Depends(get\_search\_service)]

SummaryDep = Annotated[SummaryService, Depends(get\_summary\_service)]

FileDep = Annotated[FileService, Depends(get\_file\_service)]

**src/mf\_core/api/errors.py**

from fastapi import Request

from fastapi.responses import JSONResponse

from fastapi.exceptions import RequestValidationError

from starlette.status import HTTP\_422\_UNPROCESSABLE\_ENTITY, HTTP\_500\_INTERNAL\_SERVER\_ERROR

from mf\_core.telemetry.logging import configure\_logging

log = configure\_logging()

class AppError(Exception):

def \_\_init\_\_(self, message: str, code: str = "app\_error"):

self.message = message

self.code = code

def register\_exception\_handlers(app):

@app.exception\_handler(AppError)

async def app\_error\_handler(\_: Request, exc: AppError):

return JSONResponse({"error": exc.code, "message": exc.message}, status\_code=400)

@app.exception\_handler(RequestValidationError)

async def validation\_handler(\_: Request, exc: RequestValidationError):

return JSONResponse({"error": "validation\_error", "details": exc.errors()}, status\_code=HTTP\_422\_UNPROCESSABLE\_ENTITY)

@app.middleware("http")

async def catch\_all(request: Request, call\_next):

try:

return await call\_next(request)

except Exception as e: # noqa

log.exception("Unhandled error")

return JSONResponse({"error": "internal\_error", "message": str(e)}, status\_code=HTTP\_500\_INTERNAL\_SERVER\_ERROR)

**src/mf\_core/api/middleware.py**

from fastapi import FastAPI

from fastapi.middleware.cors import CORSMiddleware

from mf\_core.telemetry.metrics import REQUESTS

def setup\_middleware(app: FastAPI) -> None:

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"], allow\_credentials=True,

allow\_methods=["\*"], allow\_headers=["\*"],

)

@app.middleware("http")

async def metrics\_mw(request, call\_next):

resp = await call\_next(request)

try:

REQUESTS.labels(endpoint=request.url.path).inc()

except Exception:

pass

return resp

**src/mf\_core/api/app.py (сборка)**

from fastapi import FastAPI

from mf\_core.api.v1.health import router as health\_router

from mf\_core.api.v1.files import router as files\_router

from mf\_core.api.v1.search import router as search\_router

from mf\_core.api.v1.summary import router as summary\_router

from mf\_core.api.v1.status import router as status\_router

from mf\_core.api.middleware import setup\_middleware

from mf\_core.api.errors import register\_exception\_handlers

from mf\_core.telemetry.tracing import init\_tracer

app = FastAPI(title="MindForge API", version="0.1.0")

setup\_middleware(app)

register\_exception\_handlers(app)

init\_tracer(app)

app.include\_router(health\_router, prefix="/v1")

app.include\_router(files\_router, prefix="/v1")

app.include\_router(search\_router, prefix="/v1")

app.include\_router(summary\_router,prefix="/v1")

app.include\_router(status\_router, prefix="/v1")

@app.get("/healthz")

def healthz():

return {"status": "ok"}

**src/mf\_core/agents/base.py**

from abc import ABC, abstractmethod

from typing import Any, Dict

class BaseAgent(ABC):

name: str = "agent"

@abstractmethod

def run(self, payload: Dict[str, Any]) -> Dict[str, Any]:

...

**src/mf\_core/llm/caching.py (Redis)**

import json, os

from typing import Any, Optional

from redis import Redis

\_TTL = int(os.getenv("MF\_LLM\_CACHE\_TTL", "3600"))

class LLMCache:

def \_\_init\_\_(self, redis: Optional[Redis] = None):

self.r = redis or Redis(host=os.getenv("REDIS\_HOST","localhost"), port=int(os.getenv("REDIS\_PORT","6379")), db=1)

def \_key(self, prompt: str, model: str) -> str:

return f"llm:{model}:{hash(prompt)}"

def get(self, prompt: str, model: str) -> Optional[Any]:

v = self.r.get(self.\_key(prompt, model))

return json.loads(v) if v else None

def set(self, prompt: str, model: str, value: Any) -> None:

self.r.setex(self.\_key(prompt, model), \_TTL, json.dumps(value))

**src/mf\_core/workers/celery\_app.py + tasks.py**

# celery\_app.py

from celery import Celery

import os

celery\_app = Celery(

"mindforge",

broker=os.getenv("CELERY\_BROKER\_URL", "redis://localhost:6379/0"),

backend=os.getenv("CELERY\_RESULT\_BACKEND", "redis://localhost:6379/0"),

)

# tasks.py

from .celery\_app import celery\_app

@celery\_app.task(name="mindforge.summarize")

def summarize\_task(doc\_id: str) -> dict:

# TODO: plug into services/summary\_service.py

return {"doc\_id": doc\_id, "summary": "TBD"}

**src/mf\_core/telemetry/tracing.py (OpenTelemetry)**

import os

from fastapi import FastAPI

from opentelemetry import trace

from opentelemetry.instrumentation.fastapi import FastAPIInstrumentor

from opentelemetry.sdk.resources import Resource

from opentelemetry.sdk.trace import TracerProvider

from opentelemetry.sdk.trace.export import BatchSpanProcessor

from opentelemetry.exporter.otlp.proto.http.trace\_exporter import OTLPSpanExporter

def init\_tracer(app: FastAPI) -> None:

endpoint = os.getenv("OTEL\_EXPORTER\_OTLP\_ENDPOINT", "http://localhost:4318")

resource = Resource.create({"service.name": "mindforge-api"})

provider = TracerProvider(resource=resource)

span\_exporter = OTLPSpanExporter(endpoint=f"{endpoint}/v1/traces")

provider.add\_span\_processor(BatchSpanProcessor(span\_exporter))

trace.set\_tracer\_provider(provider)

FastAPIInstrumentor.instrument\_app(app)

**src/mf\_core/services/\***

# file\_service.py

from pathlib import Path

from mf\_core.common.paths import PROJECT\_ROOT

class FileService:

def \_\_init\_\_(self, base: Path | None = None):

self.base = base or (PROJECT\_ROOT / "data" / "uploads")

self.base.mkdir(parents=True, exist\_ok=True)

def save(self, filename: str, content: bytes) -> Path:

p = self.base / filename

p.write\_bytes(content)

return p

# summary\_service.py

class SummaryService:

def summarize(self, text: str) -> str:

# TODO: chain to LLM + RAG

return "TBD summary"

# search\_service.py

class SearchService:

def search(self, query: str) -> list[dict]:

# TODO: dense + BM25 + rerank

return []

**src/mf\_core/api/v1/\*.py — примеры с зависимостями**

# search.py

from fastapi import APIRouter, Query

from mf\_core.api.dependencies import SearchDep

router = APIRouter(tags=["search"])

@router.get("/search")

def search(q: str = Query(min\_length=2), svc: SearchDep = None):

return {"query": q, "results": svc.search(q)}

# summary.py

from fastapi import APIRouter, Body

from mf\_core.api.dependencies import SummaryDep

router = APIRouter(tags=["summary"])

@router.post("/summary")

def summary(text: str = Body(..., embed=True), svc: SummaryDep = None):

return {"summary": svc.summarize(text)}

**tests/conftest.py + примеры тестов**

# tests/conftest.py

import pytest

from fastapi.testclient import TestClient

from mf\_core.api.app import app

@pytest.fixture()

def client():

return TestClient(app)

# tests/integration/test\_search\_api.py

def test\_search\_endpoint(client):

r = client.get("/v1/search", params={"q": "test"})

assert r.status\_code == 200

assert "results" in r.json()

**scripts/prestart.sh**

#!/usr/bin/env bash

set -euo pipefail

echo "[prestart] Applying migrations / warming caches (TBD)"

**deploy/docker/docker-compose.yaml (дополнено брокером/вектором)**

version: "3.9"

services:

api:

build: { context: ../.., dockerfile: deploy/docker/Dockerfile.api }

environment:

- MF\_ENV=dev

- REDIS\_HOST=redis

- REDIS\_PORT=6379

- OTEL\_EXPORTER\_OTLP\_ENDPOINT=http://otelcol:4318

ports: ["8000:8000"]

depends\_on: [redis, qdrant, otelcol]

worker:

build: { context: ../.., dockerfile: deploy/docker/Dockerfile.worker }

environment:

- CELERY\_BROKER\_URL=redis://redis:6379/0

- CELERY\_RESULT\_BACKEND=redis://redis:6379/0

depends\_on: [redis]

redis:

image: redis:7-alpine

ports: ["6379:6379"]

qdrant:

image: qdrant/qdrant:latest

ports: ["6333:6333","6334:6334"]

volumes: ["./qdrant\_storage:/qdrant/storage"]

otelcol:

image: otel/opentelemetry-collector:latest

command: ["--config=/etc/otelcol-config.yml"]

volumes:

- ./otelcol-config.yml:/etc/otelcol-config.yml

ports: ["4318:4318"]

**Makefile (минимальные правки под src-layout)**

run:

\tuvicorn mf\_core.api.app:app --reload --host 0.0.0.0 --port 8000

lint:

\truff check src

\tmypy src

fmt:

\truff check --select I --fix src

\tblack src

\tisort src

Если нужно, соберу это в один «инициализационный» скрипт (создаст папки и файлы) или добавлю GitHub Actions (lint.yml, tests.yml) под эту структуру.

Начало формы



Конец формы