# 📁 Folder & File Structure

railway-pg-test/

📄 .coverage

📄 .dockerignore

📄 .env

📄 .env.local

📄 .env.prod

📄 .env.staging

📁 .github/

📁 workflows/

📄 cron-sync.yml

📄 python-tests.yml

📄 .gitignore

📄 .pylintrc

📁 .vscode/

📄 Dockerfile

📄 README.md

📄 SmartCoach\_GPT\_Handoff\_Summary\_CLEAN.docx

📄 activity\_ingestion.log

📁 alembic/

📄 README

🐍 env.py

📄 script.py.mako

📁 versions/

🐍 07abffe60681\_make\_lap\_index\_nullable.py

🐍 280430412206\_add\_tokens\_table.py

🐍 475a80332c46\_fix\_split\_column\_type\_to\_boolean.py

🐍 a01b22564ade\_initial\_baseline.py

🐍 f23968f5fa38\_fix\_split\_column\_to\_integer.py

📄 alembic.ini

📁 app/

🐍 staging\_auth\_app.py

📁 debug\_dumps/

📄 docker-compose.yml

📄 final\_project\_map.docx

📁 frontend/

📄 .env

📄 .env.local

📄 .env.production

📄 .env.staging

📄 .gitignore

📄 README.md

📄 eslint.config.js

📁 public/

📄 vite.svg

📁 src/

📄 App.css

📄 App.jsx

📄 AskGptMvpUI.jsx

📄 OnboardScreen.jsx

📄 PostOAuthSuccess.jsx

📄 PostOAuthSync.jsx

📁 assets/

📄 react.svg

📁 components/

📄 PostOAuthProgress.jsx

📄 index.css

📄 main.jsx

📄 switch-env.ps1

📄 vite.config.js

📁 htmlcov/

📄 .gitignore

📄 coverage\_html\_cb\_497bf287.js

📄 favicon\_32\_cb\_58284776.png

📄 keybd\_closed\_cb\_ce680311.png

📄 style\_cb\_db813965.css

📁 migrations/

📄 README

🐍 env.py

📄 script.py.mako

📁 versions/

📄 nixpacks.tom

📄 prod.clean.dump

📄 prod.cleaned.sql

📄 prod.dump

📄 prod.sql

📄 prod.sql.cleaned

📄 pytest\_log.txt

📄 requirements.txt

📄 reset\_and\_init\_db.sh

🐍 run.py

📁 src/

🐍 \_\_init\_\_.py

🐍 app.py

📁 db/

🐍 \_\_init\_\_.py

📁 dao/

🐍 \_\_init\_\_.py

🐍 activity\_dao.py

🐍 activity\_stats\_dao.py

🐍 athlete\_dao.py

🐍 split\_dao.py

🐍 token\_dao.py

🐍 db\_session.py

📁 models/

🐍 \_\_init\_\_.py

🐍 activities.py

🐍 athletes.py

🐍 splits.py

🐍 tokens.py

📁 linter\_plugins/

📁 tests/

📁 dao\_db\_logic/

📁 enrichment\_boundary/

📁 service\_purity/

📁 routes/

🐍 \_\_init\_\_.py

🐍 activity\_routes.py

🐍 admin\_routes.py

🐍 ask\_routes.py

🐍 auth\_routes.py

🐍 health\_routes.py

📁 scripts/

🐍 \_\_init\_\_.py

🐍 activity\_query\_cli.py

🐍 debug\_token.py

🐍 main\_pipeline.py

🐍 manual\_verify.py

🐍 oauth\_cli.py

🐍 scheduled\_ingest.py

📄 sync\_athlete.sh

📁 services/

🐍 \_\_init\_\_.py

🐍 activity\_service.py

🐍 ingestion\_orchestrator\_service.py

🐍 strava\_access\_service.py

🐍 token\_service.py

📁 utils/

🐍 \_\_init\_\_.py

🐍 config.py

🐍 conversions.py

🐍 db\_check.py

🐍 generate\_gpt\_handoff\_summary.py

🐍 gpt\_ops.py

🐍 jwt\_utils.py

🐍 logger.py

🐍 map\_and\_extract\_Railway.py

🐍 sample\_data.py

🐍 seeder.py

📁 static/

📁 assets/

📄 index-Dtn62Xmo.css

📄 index-tay6b22Q.js

📄 vite.svg

📄 switch\_env.sh

📄 sync-test-db.ps1

🐍 test\_enrichment\_sync.py

📁 tests/

🐍 conftest.py

🐍 test\_activity\_dao.py

🐍 test\_activity\_routes.py

🐍 test\_activity\_service.py

🐍 test\_app.py

🐍 test\_ask\_routes.py

🐍 test\_athlete\_dao.py

🐍 test\_auth.py

🐍 test\_auth\_routes.py

📁 test\_data/

🐍 sample\_activities.py

🐍 test\_enrichment\_with\_splits.py

🐍 test\_extract\_hr\_zone\_percentages.py

🐍 test\_format\_prompt.py

🐍 test\_full\_ingestion\_flow.py

🐍 test\_health.py

🐍 test\_hr\_zone\_api.py

🐍 test\_ingestion\_orchestrator.py

🐍 test\_integration\_ask\_prompt.py

🐍 test\_integration\_ingestion.py

🐍 test\_jwt\_utils.py

🐍 test\_main\_pipeline.py

🐍 test\_oauth\_flow.py

🐍 test\_split\_dao.py

🐍 test\_split\_upsert\_idempotency.py

🐍 test\_splits.py

🐍 test\_strava\_access\_service.py

🐍 test\_sync.py

🐍 test\_token\_service.py

🐍 utils.py

🐍 validate\_envs.py

🐍 wsgi.py

# 🧠 Code & Content

### 📄 README.md

# 🏃 Smart Marathon Coach API  
  
This is a Flask-based API for syncing Strava activity data, enriching runs, and generating training insights. This repo is part of a multi-phase project — currently in \*\*Milestone 1: Setup & Plumbing\*\*.  
  
---  
  
## 🚀 Getting Started  
  
### 1. Clone the repo  
  
```bash  
git clone https://github.com/your-username/railway-pg-test.git  
cd railway-pg-test  
  
  
### 2. Set up your virtual environment  
  
```bash  
python -m venv venv  
venv\Scripts\activate # On Windows  
# or  
source venv/bin/activate # On Mac/Linux  
```  
  
### 3. Install dependencies  
  
```bash  
pip install -r requirements.txt  
```  
  
---  
  
## 🛠️ Environment Configuration  
  
Copy the example `.env` file and fill in your secrets:  
  
```bash  
cp .env.example .env  
```  
  
Or manually create `.env` with values like:  
  
```env  
DATABASE\_URL=postgresql://smartcoach:devpass@postgres:5432/smartcoach  
  
# NOTE: For local dev without Docker, change "postgres" → "localhost"  
# DATABASE\_URL=postgresql://smartcoach:devpass@localhost:5432/smartcoach  
  
STRAVA\_CLIENT\_ID=your\_client\_id  
STRAVA\_CLIENT\_SECRET=your\_client\_secret  
REDIRECT\_URI=http://127.0.0.1:5000/oauth/callback  
ADMIN\_USER=admin  
ADMIN\_PASS=secret  
SECRET\_KEY=supersecretkey  
CRON\_SECRET\_KEY=your\_cron\_key  
INTERNAL\_API\_KEY=your\_internal\_key  
  
```  
  
---  
  
## 💻 Running Locally  
  
```bash  
python run.py  
```  
  
Then open [http://127.0.0.1:5000/ping](http://127.0.0.1:5000/ping)   
You should see: `pong`  
  
---  
  
## 🧪 Endpoints (Milestone 1)  
  
| Route | Description |  
|-----------------|----------------------------------|  
| `/ping` | Health check |  
| `/init-db` | Creates DB tables |  
| `/auth/login` | Basic credential-based login |  
| `/auth/logout` | Clear session |  
| `/enrich/status`| Returns enrichment status (stub) |  
  
> More functionality is coming in Milestone 2  
  
---  
  
## 🧬 GitHub Actions  
  
We’ve added a skeleton workflow in `.github/workflows/cron-sync.yml` that:  
  
- Runs every 6 hours  
- Supports manual trigger  
- Runs placeholder logic (future expansion)  
  
---  
  
## 🧩 Project Structure  
  
```bash  
railway-pg-test/  
├── src/  
│ ├── app.py → Flask app entrypoint  
│ ├── routes/ → Route blueprints  
│ ├── services/ → Business logic  
│ ├── db/ → Database models + sessions  
│ └── utils/ → Utility functions  
├── schema.sql → Creates core DB tables  
├── run.py → Runs the app  
├── requirements.txt  
└── .env → Environment variables  
  
```  
  
---  
  
## 📦 Requirements  
  
- Python 3.11+  
- SQLite or Postgres  
- A Strava API App (https://www.strava.com/settings/api)# Triggering redeploy

### 📄 alembic\env.py

import sys  
import os  
  
from dotenv import load\_dotenv  
  
# --- Add project root to sys.path ---  
project\_root = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))  
sys.path.insert(0, project\_root)  
  
# --- Add 'src' folder to sys.path ---  
sys.path.insert(0, os.path.join(project\_root, "src"))  
  
# ✅ Load environment variables from explicit .env path with override  
# ✅ Load the appropriate .env file based on FLASK\_ENV  
# Use FLASK\_ENV or fallback to RAILWAY\_ENVIRONMENT or default to "development"  
env\_mode = os.getenv("FLASK\_ENV") or os.getenv("RAILWAY\_ENVIRONMENT") or "development"  
  
if env\_mode == "testing":  
 env\_file = ".env.test"  
elif env\_mode == "production":  
 env\_file = ".env.prod"  
else:  
 env\_file = ".env"  
  
dotenv\_path = os.path.join(project\_root, env\_file)  
load\_dotenv(dotenv\_path, override=True)  
  
print(f"✅ [Alembic] Loaded environment: {env\_file}")  
  
  
  
print("🚨 DATABASE\_URL =", os.getenv("DATABASE\_URL"))  
  
from logging.config import fileConfig  
from sqlalchemy import engine\_from\_config, create\_engine, pool  
from alembic import context  
  
# DATABASE\_URL after .env is loaded  
DATABASE\_URL = os.getenv("DATABASE\_URL")  
print(f"DEBUG: Using DATABASE\_URL = {DATABASE\_URL}")  
  
if not DATABASE\_URL:  
 raise RuntimeError("DATABASE\_URL environment variable is not set. Alembic cannot continue.")  
  
# Import SQLAlchemy Base AFTER sys.path is fully patched  
from src.db.db\_session import Base  
  
# ✅ Import all models so Alembic can detect schema  
import src.db.models.activities  
import src.db.models.tokens  
import src.db.models.splits  
import src.db.models.athletes  
  
# Alembic Config object  
config = context.config  
  
# Configure logging  
if config.config\_file\_name is not None:  
 fileConfig(config.config\_file\_name)  
  
# Set target metadata for autogenerate  
target\_metadata = Base.metadata  
  
# Inject DATABASE\_URL dynamically for Alembic migrations  
config.set\_main\_option("sqlalchemy.url", DATABASE\_URL)  
  
  
def run\_migrations\_offline():  
 """Run migrations in 'offline' mode (no DB connection)."""  
 url = config.get\_main\_option("sqlalchemy.url")  
 context.configure(  
 url=url,  
 target\_metadata=target\_metadata,  
 literal\_binds=True,  
 compare\_type=True,  
 )  
  
 with context.begin\_transaction():  
 context.run\_migrations()  
  
  
def run\_migrations\_online():  
 """Run migrations in 'online' mode (DB connection active)."""  
 # ✅ Replace with explicit engine creation and echo enabled  
 connectable = create\_engine(  
 DATABASE\_URL,  
 echo=True, # 🔍 Enable SQL echoing  
 poolclass=pool.NullPool,  
 )  
  
 with connectable.connect() as connection:  
 context.configure(  
 connection=connection,  
 target\_metadata=target\_metadata,  
 compare\_type=True,  
 )  
  
 with context.begin\_transaction():  
 context.run\_migrations()  
  
  
if context.is\_offline\_mode():  
 run\_migrations\_offline()  
else:  
 run\_migrations\_online()

### 📄 alembic\versions\07abffe60681\_make\_lap\_index\_nullable.py

"""Make lap\_index nullable  
  
Revision ID: 07abffe60681  
Revises: 475a80332c46  
Create Date: 2025-06-16 18:55:24.799414  
"""  
  
from typing import Sequence, Union  
from alembic import op  
import sqlalchemy as sa  
  
# revision identifiers, used by Alembic.  
revision: str = '07abffe60681'  
down\_revision: Union[str, None] = '475a80332c46'  
branch\_labels: Union[str, Sequence[str], None] = None  
depends\_on: Union[str, Sequence[str], None] = None  
  
def upgrade() -> None:  
 """Allow 'lap\_index' in 'splits' table to be nullable."""  
 op.alter\_column(  
 'splits',  
 'lap\_index',  
 existing\_type=sa.Integer(),  
 nullable=True  
 )  
  
def downgrade() -> None:  
 """Revert 'lap\_index' column in 'splits' table to NOT NULL."""  
 op.alter\_column(  
 'splits',  
 'lap\_index',  
 existing\_type=sa.Integer(),  
 nullable=False  
 )

### 📄 alembic\versions\280430412206\_add\_tokens\_table.py

"""Add tokens table  
  
Revision ID: 280430412206  
Revises: a01b22564ade  
Create Date: 2025-06-14 23:41:27.438877  
"""  
  
from typing import Sequence, Union  
from alembic import op  
import sqlalchemy as sa  
  
# revision identifiers, used by Alembic.  
revision: str = '280430412206'  
down\_revision: Union[str, None] = 'a01b22564ade'  
branch\_labels: Union[str, Sequence[str], None] = None  
depends\_on: Union[str, Sequence[str], None] = None  
  
def upgrade() -> None:  
 """Upgrade schema."""  
 op.create\_table(  
 'tokens',  
 sa.Column('athlete\_id', sa.BigInteger(), nullable=False),  
 sa.Column('access\_token', sa.String(), nullable=False),  
 sa.Column('refresh\_token', sa.String(), nullable=False),  
 sa.Column('expires\_at', sa.BigInteger(), nullable=False),  
 sa.PrimaryKeyConstraint('athlete\_id')  
 )  
  
def downgrade() -> None:  
 """Downgrade schema."""  
 op.drop\_table('tokens')

### 📄 alembic\versions\475a80332c46\_fix\_split\_column\_type\_to\_boolean.py

"""Fix split column type to Boolean  
  
Revision ID: 475a80332c46  
Revises: 280430412206  
Create Date: 2025-06-16 15:05:23.135910  
"""  
  
from typing import Sequence, Union  
from alembic import op  
import sqlalchemy as sa  
  
# revision identifiers  
revision: str = '475a80332c46'  
down\_revision: Union[str, None] = '280430412206'  
branch\_labels: Union[str, Sequence[str], None] = None  
depends\_on: Union[str, Sequence[str], None] = None  
  
def upgrade() -> None:  
 """Convert 'split' column in 'splits' table from INTEGER to BOOLEAN."""  
 op.execute("""  
 ALTER TABLE splits   
 ALTER COLUMN split TYPE BOOLEAN   
 USING CASE  
 WHEN split::INTEGER = 1 THEN TRUE  
 WHEN split::INTEGER = 0 THEN FALSE  
 ELSE NULL  
 END  
 """)  
  
def downgrade() -> None:  
 """Downgrade not implemented due to potential data loss converting from BOOLEAN back to INTEGER."""  
 pass

### 📄 alembic\versions\a01b22564ade\_initial\_baseline.py

"""initial baseline  
  
Revision ID: a01b22564ade  
Revises:   
Create Date: 2025-06-14 23:36:21.153362  
  
"""  
from typing import Sequence, Union  
  
from alembic import op  
import sqlalchemy as sa  
  
  
# revision identifiers, used by Alembic.  
revision: str = 'a01b22564ade'  
down\_revision: Union[str, None] = None  
branch\_labels: Union[str, Sequence[str], None] = None  
depends\_on: Union[str, Sequence[str], None] = None  
  
  
def upgrade() -> None:  
 """Upgrade schema."""  
 # ### commands auto generated by Alembic - please adjust! ###  
 op.create\_table('activities',  
 sa.Column('activity\_id', sa.BigInteger(), nullable=False),  
 sa.Column('athlete\_id', sa.BigInteger(), nullable=False),  
 sa.Column('name', sa.String(), nullable=True),  
 sa.Column('type', sa.String(), nullable=True),  
 sa.Column('start\_date', sa.DateTime(), nullable=True),  
 sa.Column('distance', sa.Float(), nullable=True),  
 sa.Column('elapsed\_time', sa.Integer(), nullable=True),  
 sa.Column('moving\_time', sa.Integer(), nullable=True),  
 sa.Column('total\_elevation\_gain', sa.Float(), nullable=True),  
 sa.Column('external\_id', sa.String(), nullable=True),  
 sa.Column('timezone', sa.String(), nullable=True),  
 sa.Column('average\_speed', sa.Float(), nullable=True),  
 sa.Column('max\_speed', sa.Float(), nullable=True),  
 sa.Column('suffer\_score', sa.Float(), nullable=True),  
 sa.Column('average\_heartrate', sa.Float(), nullable=True),  
 sa.Column('max\_heartrate', sa.Float(), nullable=True),  
 sa.Column('calories', sa.Float(), nullable=True),  
 sa.Column('conv\_distance', sa.Float(), nullable=True),  
 sa.Column('conv\_elevation\_feet', sa.Float(), nullable=True),  
 sa.Column('conv\_avg\_speed', sa.Float(), nullable=True),  
 sa.Column('conv\_max\_speed', sa.Float(), nullable=True),  
 sa.Column('conv\_moving\_time', sa.String(), nullable=True),  
 sa.Column('conv\_elapsed\_time', sa.String(), nullable=True),  
 sa.Column('hr\_zone\_1', sa.Float(), nullable=True),  
 sa.Column('hr\_zone\_2', sa.Float(), nullable=True),  
 sa.Column('hr\_zone\_3', sa.Float(), nullable=True),  
 sa.Column('hr\_zone\_4', sa.Float(), nullable=True),  
 sa.Column('hr\_zone\_5', sa.Float(), nullable=True),  
 sa.PrimaryKeyConstraint('activity\_id')  
 )  
 op.create\_index(op.f('ix\_activities\_activity\_id'), 'activities', ['activity\_id'], unique=False)  
 op.create\_index(op.f('ix\_activities\_athlete\_id'), 'activities', ['athlete\_id'], unique=False)  
 op.create\_table('athletes',  
 sa.Column('id', sa.Integer(), nullable=False),  
 sa.Column('strava\_athlete\_id', sa.BigInteger(), nullable=False),  
 sa.Column('name', sa.String(), nullable=True),  
 sa.Column('email', sa.String(), nullable=True),  
 sa.Column('created\_at', sa.DateTime(), server\_default=sa.text('now()'), nullable=True),  
 sa.PrimaryKeyConstraint('id'),  
 sa.UniqueConstraint('strava\_athlete\_id')  
 )  
 op.create\_table('splits',  
 sa.Column('id', sa.Integer(), nullable=False),  
 sa.Column('activity\_id', sa.BigInteger(), nullable=False),  
 sa.Column('lap\_index', sa.Integer(), nullable=False),  
 sa.Column('distance', sa.Float(), nullable=True),  
 sa.Column('elapsed\_time', sa.Integer(), nullable=True),  
 sa.Column('moving\_time', sa.Integer(), nullable=True),  
 sa.Column('average\_speed', sa.Float(), nullable=True),  
 sa.Column('max\_speed', sa.Float(), nullable=True),  
 sa.Column('start\_index', sa.Integer(), nullable=True),  
 sa.Column('end\_index', sa.Integer(), nullable=True),  
 sa.Column('split', sa.Boolean(), nullable=True),  
 sa.Column('average\_heartrate', sa.Float(), nullable=True),  
 sa.Column('pace\_zone', sa.Integer(), nullable=True),  
 sa.Column('created\_at', sa.TIMESTAMP(), server\_default=sa.text('now()'), nullable=True),  
 sa.Column('conv\_distance', sa.Float(), nullable=True),  
 sa.Column('conv\_avg\_speed', sa.Float(), nullable=True),  
 sa.Column('conv\_moving\_time', sa.String(), nullable=True),  
 sa.Column('conv\_elapsed\_time', sa.String(), nullable=True),  
 sa.ForeignKeyConstraint(['activity\_id'], ['activities.activity\_id'], ondelete='CASCADE'),  
 sa.PrimaryKeyConstraint('id'),  
 sa.UniqueConstraint('activity\_id', 'lap\_index', name='uq\_activity\_lap')  
 )  
 # ### end Alembic commands ###  
  
  
def downgrade() -> None:  
 """Downgrade schema."""  
 # ### commands auto generated by Alembic - please adjust! ###  
 op.drop\_table('splits')  
 op.drop\_table('athletes')  
 op.drop\_index(op.f('ix\_activities\_athlete\_id'), table\_name='activities')  
 op.drop\_index(op.f('ix\_activities\_activity\_id'), table\_name='activities')  
 op.drop\_table('activities')  
 # ### end Alembic commands ###

### 📄 alembic\versions\f23968f5fa38\_fix\_split\_column\_to\_integer.py

"""Fix split column back to INTEGER"""  
  
from typing import Sequence, Union  
from alembic import op  
import sqlalchemy as sa  
  
# revision identifiers  
revision: str = 'f23968f5fa38'  
down\_revision: Union[str, None] = '07abffe60681'  
branch\_labels: Union[str, Sequence[str], None] = None  
depends\_on: Union[str, Sequence[str], None] = None  
  
def upgrade() -> None:  
 op.execute("""  
 ALTER TABLE splits   
 ALTER COLUMN split TYPE INTEGER   
 USING split::INTEGER  
 """)  
  
def downgrade() -> None:  
 op.execute("""  
 ALTER TABLE splits   
 ALTER COLUMN split TYPE INTEGER   
 USING CASE  
 WHEN split = TRUE THEN 1  
 WHEN split = FALSE THEN 0  
 ELSE NULL  
 END  
 """)

### 📄 app\staging\_auth\_app.py

# staging\_auth\_app.py  
  
import os  
import sys  
sys.path.insert(0, os.path.join(os.path.dirname(\_\_file\_\_), "src"))  
  
from flask import Flask, redirect, request, jsonify  
from src.services.token\_service import get\_authorization\_url, store\_tokens\_from\_callback  
from src.db.db\_session import get\_session  
  
app = Flask(\_\_name\_\_)  
  
@app.route("/auth/login")  
def strava\_login():  
 """  
 Redirect to Strava OAuth URL (called manually from browser)  
 """  
 return redirect(get\_authorization\_url())  
  
  
@app.route("/auth/callback")  
def strava\_callback():  
 """  
 Handle OAuth redirect from Strava, exchange code, and persist tokens.  
 """  
 session = get\_session()  
 try:  
 code = request.args.get("code")  
 if not code:  
 return "Missing OAuth code", 400  
  
 athlete\_id = store\_tokens\_from\_callback(code, session)  
 return jsonify({  
 "status": "✅ Token stored",  
 "athlete\_id": athlete\_id  
 })  
 except Exception as e:  
 return jsonify({"error": str(e)}), 500  
 finally:  
 session.close()  
  
  
@app.route("/ping")  
def ping():  
 return "pong from staging OAuth app"

### 📄 frontend\README.md

# React + Vite  
  
This template provides a minimal setup to get React working in Vite with HMR and some ESLint rules.  
  
Currently, two official plugins are available:  
  
- [@vitejs/plugin-react](https://github.com/vitejs/vite-plugin-react/blob/main/packages/plugin-react) uses [Babel](https://babeljs.io/) for Fast Refresh  
- [@vitejs/plugin-react-swc](https://github.com/vitejs/vite-plugin-react/blob/main/packages/plugin-react-swc) uses [SWC](https://swc.rs/) for Fast Refresh  
  
## Expanding the ESLint configuration  
  
If you are developing a production application, we recommend using TypeScript with type-aware lint rules enabled. Check out the [TS template](https://github.com/vitejs/vite/tree/main/packages/create-vite/template-react-ts) for information on how to integrate TypeScript and [`typescript-eslint`](https://typescript-eslint.io) in your project.

### 📄 frontend\eslint.config.js

import js from '@eslint/js'  
import globals from 'globals'  
import reactHooks from 'eslint-plugin-react-hooks'  
import reactRefresh from 'eslint-plugin-react-refresh'  
import { defineConfig, globalIgnores } from 'eslint/config'  
  
export default defineConfig([  
 globalIgnores(['dist']),  
 {  
 files: ['\*\*/\*.{js,jsx}'],  
 extends: [  
 js.configs.recommended,  
 reactHooks.configs['recommended-latest'],  
 reactRefresh.configs.vite,  
 ],  
 languageOptions: {  
 ecmaVersion: 2020,  
 globals: globals.browser,  
 parserOptions: {  
 ecmaVersion: 'latest',  
 ecmaFeatures: { jsx: true },  
 sourceType: 'module',  
 },  
 },  
 rules: {  
 'no-unused-vars': ['error', { varsIgnorePattern: '^[A-Z\_]' }],  
 },  
 },  
])

### 📄 frontend\src\App.css

#root {  
 max-width: 1280px;  
 margin: 0 auto;  
 padding: 2rem;  
 text-align: center;  
}  
  
.logo {  
 height: 6em;  
 padding: 1.5em;  
 will-change: filter;  
 transition: filter 300ms;  
}  
.logo:hover {  
 filter: drop-shadow(0 0 2em #646cffaa);  
}  
.logo.react:hover {  
 filter: drop-shadow(0 0 2em #61dafbaa);  
}  
  
@keyframes logo-spin {  
 from {  
 transform: rotate(0deg);  
 }  
 to {  
 transform: rotate(360deg);  
 }  
}  
  
@media (prefers-reduced-motion: no-preference) {  
 a:nth-of-type(2) .logo {  
 animation: logo-spin infinite 20s linear;  
 }  
}  
  
.card {  
 padding: 2em;  
}  
  
.read-the-docs {  
 color: #888;  
}

### 📄 frontend\src\index.css

:root {  
 font-family: system-ui, Avenir, Helvetica, Arial, sans-serif;  
 line-height: 1.5;  
 font-weight: 400;  
  
 color-scheme: light dark;  
 color: rgba(255, 255, 255, 0.87);  
 background-color: #242424;  
  
 font-synthesis: none;  
 text-rendering: optimizeLegibility;  
 -webkit-font-smoothing: antialiased;  
 -moz-osx-font-smoothing: grayscale;  
}  
  
a {  
 font-weight: 500;  
 color: #646cff;  
 text-decoration: inherit;  
}  
a:hover {  
 color: #535bf2;  
}  
  
body {  
 margin: 0;  
 display: flex;  
 place-items: center;  
 min-width: 320px;  
 min-height: 100vh;  
}  
  
h1 {  
 font-size: 3.2em;  
 line-height: 1.1;  
}  
  
button {  
 border-radius: 8px;  
 border: 1px solid transparent;  
 padding: 0.6em 1.2em;  
 font-size: 1em;  
 font-weight: 500;  
 font-family: inherit;  
 background-color: #1a1a1a;  
 cursor: pointer;  
 transition: border-color 0.25s;  
}  
button:hover {  
 border-color: #646cff;  
}  
button:focus,  
button:focus-visible {  
 outline: 4px auto -webkit-focus-ring-color;  
}  
  
@media (prefers-color-scheme: light) {  
 :root {  
 color: #213547;  
 background-color: #ffffff;  
 }  
 a:hover {  
 color: #747bff;  
 }  
 button {  
 background-color: #f9f9f9;  
 }  
}

### 📄 frontend\vite.config.js

import { defineConfig, loadEnv } from 'vite';  
import react from '@vitejs/plugin-react';  
import path from 'path';  
  
export default defineConfig(({ mode }) => {  
 // Load .env files based on mode (e.g., .env.local, .env.staging, .env.production)  
 const env = loadEnv(mode, process.cwd(), '');  
  
 return {  
 base: '/',  
 plugins: [react()],  
 resolve: {  
 alias: {  
 '@': path.resolve(\_\_dirname, './src'),  
 },  
 },  
 server: {  
 proxy: {  
 '/auth': {  
 target: env.VITE\_BACKEND\_URL,  
 changeOrigin: true,  
 secure: false,  
 },  
 '/admin': {  
 target: env.VITE\_BACKEND\_URL,  
 changeOrigin: true,  
 secure: false,  
 },  
 '/sync': {  
 target: env.VITE\_BACKEND\_URL,  
 changeOrigin: true,  
 secure: false,  
 },  
 },  
 },  
 };  
});

### 📄 htmlcov\coverage\_html\_cb\_497bf287.js

// Licensed under the Apache License: http://www.apache.org/licenses/LICENSE-2.0  
// For details: https://github.com/nedbat/coveragepy/blob/master/NOTICE.txt  
  
// Coverage.py HTML report browser code.  
/\*jslint browser: true, sloppy: true, vars: true, plusplus: true, maxerr: 50, indent: 4 \*/  
/\*global coverage: true, document, window, $ \*/  
  
coverage = {};  
  
// General helpers  
function debounce(callback, wait) {  
 let timeoutId = null;  
 return function(...args) {  
 clearTimeout(timeoutId);  
 timeoutId = setTimeout(() => {  
 callback.apply(this, args);  
 }, wait);  
 };  
};  
  
function checkVisible(element) {  
 const rect = element.getBoundingClientRect();  
 const viewBottom = Math.max(document.documentElement.clientHeight, window.innerHeight);  
 const viewTop = 30;  
 return !(rect.bottom < viewTop || rect.top >= viewBottom);  
}  
  
function on\_click(sel, fn) {  
 const elt = document.querySelector(sel);  
 if (elt) {  
 elt.addEventListener("click", fn);  
 }  
}  
  
// Helpers for table sorting  
function getCellValue(row, column = 0) {  
 const cell = row.cells[column] // nosemgrep: eslint.detect-object-injection  
 if (cell.childElementCount == 1) {  
 var child = cell.firstElementChild;  
 if (child.tagName === "A") {  
 child = child.firstElementChild;  
 }  
 if (child instanceof HTMLDataElement && child.value) {  
 return child.value;  
 }  
 }  
 return cell.innerText || cell.textContent;  
}  
  
function rowComparator(rowA, rowB, column = 0) {  
 let valueA = getCellValue(rowA, column);  
 let valueB = getCellValue(rowB, column);  
 if (!isNaN(valueA) && !isNaN(valueB)) {  
 return valueA - valueB;  
 }  
 return valueA.localeCompare(valueB, undefined, {numeric: true});  
}  
  
function sortColumn(th) {  
 // Get the current sorting direction of the selected header,  
 // clear state on other headers and then set the new sorting direction.  
 const currentSortOrder = th.getAttribute("aria-sort");  
 [...th.parentElement.cells].forEach(header => header.setAttribute("aria-sort", "none"));  
 var direction;  
 if (currentSortOrder === "none") {  
 direction = th.dataset.defaultSortOrder || "ascending";  
 }  
 else if (currentSortOrder === "ascending") {  
 direction = "descending";  
 }  
 else {  
 direction = "ascending";  
 }  
 th.setAttribute("aria-sort", direction);  
  
 const column = [...th.parentElement.cells].indexOf(th)  
  
 // Sort all rows and afterwards append them in order to move them in the DOM.  
 Array.from(th.closest("table").querySelectorAll("tbody tr"))  
 .sort((rowA, rowB) => rowComparator(rowA, rowB, column) \* (direction === "ascending" ? 1 : -1))  
 .forEach(tr => tr.parentElement.appendChild(tr));  
  
 // Save the sort order for next time.  
 if (th.id !== "region") {  
 let th\_id = "file"; // Sort by file if we don't have a column id  
 let current\_direction = direction;  
 const stored\_list = localStorage.getItem(coverage.INDEX\_SORT\_STORAGE);  
 if (stored\_list) {  
 ({th\_id, direction} = JSON.parse(stored\_list))  
 }  
 localStorage.setItem(coverage.INDEX\_SORT\_STORAGE, JSON.stringify({  
 "th\_id": th.id,  
 "direction": current\_direction  
 }));  
 if (th.id !== th\_id || document.getElementById("region")) {  
 // Sort column has changed, unset sorting by function or class.  
 localStorage.setItem(coverage.SORTED\_BY\_REGION, JSON.stringify({  
 "by\_region": false,  
 "region\_direction": current\_direction  
 }));  
 }  
 }  
 else {  
 // Sort column has changed to by function or class, remember that.  
 localStorage.setItem(coverage.SORTED\_BY\_REGION, JSON.stringify({  
 "by\_region": true,  
 "region\_direction": direction  
 }));  
 }  
}  
  
// Find all the elements with data-shortcut attribute, and use them to assign a shortcut key.  
coverage.assign\_shortkeys = function () {  
 document.querySelectorAll("[data-shortcut]").forEach(element => {  
 document.addEventListener("keypress", event => {  
 if (event.target.tagName.toLowerCase() === "input") {  
 return; // ignore keypress from search filter  
 }  
 if (event.key === element.dataset.shortcut) {  
 element.click();  
 }  
 });  
 });  
};  
  
// Create the events for the filter box.  
coverage.wire\_up\_filter = function () {  
 // Populate the filter and hide100 inputs if there are saved values for them.  
 const saved\_filter\_value = localStorage.getItem(coverage.FILTER\_STORAGE);  
 if (saved\_filter\_value) {  
 document.getElementById("filter").value = saved\_filter\_value;  
 }  
 const saved\_hide100\_value = localStorage.getItem(coverage.HIDE100\_STORAGE);  
 if (saved\_hide100\_value) {  
 document.getElementById("hide100").checked = JSON.parse(saved\_hide100\_value);  
 }  
  
 // Cache elements.  
 const table = document.querySelector("table.index");  
 const table\_body\_rows = table.querySelectorAll("tbody tr");  
 const no\_rows = document.getElementById("no\_rows");  
  
 // Observe filter keyevents.  
 const filter\_handler = (event => {  
 // Keep running total of each metric, first index contains number of shown rows  
 const totals = new Array(table.rows[0].cells.length).fill(0);  
 // Accumulate the percentage as fraction  
 totals[totals.length - 1] = { "numer": 0, "denom": 0 }; // nosemgrep: eslint.detect-object-injection  
  
 var text = document.getElementById("filter").value;  
 // Store filter value  
 localStorage.setItem(coverage.FILTER\_STORAGE, text);  
 const casefold = (text === text.toLowerCase());  
 const hide100 = document.getElementById("hide100").checked;  
 // Store hide value.  
 localStorage.setItem(coverage.HIDE100\_STORAGE, JSON.stringify(hide100));  
  
 // Hide / show elements.  
 table\_body\_rows.forEach(row => {  
 var show = false;  
 // Check the text filter.  
 for (let column = 0; column < totals.length; column++) {  
 cell = row.cells[column];  
 if (cell.classList.contains("name")) {  
 var celltext = cell.textContent;  
 if (casefold) {  
 celltext = celltext.toLowerCase();  
 }  
 if (celltext.includes(text)) {  
 show = true;  
 }  
 }  
 }  
  
 // Check the "hide covered" filter.  
 if (show && hide100) {  
 const [numer, denom] = row.cells[row.cells.length - 1].dataset.ratio.split(" ");  
 show = (numer !== denom);  
 }  
  
 if (!show) {  
 // hide  
 row.classList.add("hidden");  
 return;  
 }  
  
 // show  
 row.classList.remove("hidden");  
 totals[0]++;  
  
 for (let column = 0; column < totals.length; column++) {  
 // Accumulate dynamic totals  
 cell = row.cells[column] // nosemgrep: eslint.detect-object-injection  
 if (cell.classList.contains("name")) {  
 continue;  
 }  
 if (column === totals.length - 1) {  
 // Last column contains percentage  
 const [numer, denom] = cell.dataset.ratio.split(" ");  
 totals[column]["numer"] += parseInt(numer, 10); // nosemgrep: eslint.detect-object-injection  
 totals[column]["denom"] += parseInt(denom, 10); // nosemgrep: eslint.detect-object-injection  
 }  
 else {  
 totals[column] += parseInt(cell.textContent, 10); // nosemgrep: eslint.detect-object-injection  
 }  
 }  
 });  
  
 // Show placeholder if no rows will be displayed.  
 if (!totals[0]) {  
 // Show placeholder, hide table.  
 no\_rows.style.display = "block";  
 table.style.display = "none";  
 return;  
 }  
  
 // Hide placeholder, show table.  
 no\_rows.style.display = null;  
 table.style.display = null;  
  
 const footer = table.tFoot.rows[0];  
 // Calculate new dynamic sum values based on visible rows.  
 for (let column = 0; column < totals.length; column++) {  
 // Get footer cell element.  
 const cell = footer.cells[column]; // nosemgrep: eslint.detect-object-injection  
 if (cell.classList.contains("name")) {  
 continue;  
 }  
  
 // Set value into dynamic footer cell element.  
 if (column === totals.length - 1) {  
 // Percentage column uses the numerator and denominator,  
 // and adapts to the number of decimal places.  
 const match = /\.([0-9]+)/.exec(cell.textContent);  
 const places = match ? match[1].length : 0;  
 const { numer, denom } = totals[column]; // nosemgrep: eslint.detect-object-injection  
 cell.dataset.ratio = `${numer} ${denom}`;  
 // Check denom to prevent NaN if filtered files contain no statements  
 cell.textContent = denom  
 ? `${(numer \* 100 / denom).toFixed(places)}%`  
 : `${(100).toFixed(places)}%`;  
 }  
 else {  
 cell.textContent = totals[column]; // nosemgrep: eslint.detect-object-injection  
 }  
 }  
 });  
  
 document.getElementById("filter").addEventListener("input", debounce(filter\_handler));  
 document.getElementById("hide100").addEventListener("input", debounce(filter\_handler));  
  
 // Trigger change event on setup, to force filter on page refresh  
 // (filter value may still be present).  
 document.getElementById("filter").dispatchEvent(new Event("input"));  
 document.getElementById("hide100").dispatchEvent(new Event("input"));  
};  
coverage.FILTER\_STORAGE = "COVERAGE\_FILTER\_VALUE";  
coverage.HIDE100\_STORAGE = "COVERAGE\_HIDE100\_VALUE";  
  
// Set up the click-to-sort columns.  
coverage.wire\_up\_sorting = function () {  
 document.querySelectorAll("[data-sortable] th[aria-sort]").forEach(  
 th => th.addEventListener("click", e => sortColumn(e.target))  
 );  
  
 // Look for a localStorage item containing previous sort settings:  
 let th\_id = "file", direction = "ascending";  
 const stored\_list = localStorage.getItem(coverage.INDEX\_SORT\_STORAGE);  
 if (stored\_list) {  
 ({th\_id, direction} = JSON.parse(stored\_list));  
 }  
 let by\_region = false, region\_direction = "ascending";  
 const sorted\_by\_region = localStorage.getItem(coverage.SORTED\_BY\_REGION);  
 if (sorted\_by\_region) {  
 ({  
 by\_region,  
 region\_direction  
 } = JSON.parse(sorted\_by\_region));  
 }  
  
 const region\_id = "region";  
 if (by\_region && document.getElementById(region\_id)) {  
 direction = region\_direction;  
 }  
 // If we are in a page that has a column with id of "region", sort on  
 // it if the last sort was by function or class.  
 let th;  
 if (document.getElementById(region\_id)) {  
 th = document.getElementById(by\_region ? region\_id : th\_id);  
 }  
 else {  
 th = document.getElementById(th\_id);  
 }  
 th.setAttribute("aria-sort", direction === "ascending" ? "descending" : "ascending");  
 th.click()  
};  
  
coverage.INDEX\_SORT\_STORAGE = "COVERAGE\_INDEX\_SORT\_2";  
coverage.SORTED\_BY\_REGION = "COVERAGE\_SORT\_REGION";  
  
...truncated...

### 📄 htmlcov\style\_cb\_db813965.css

@charset "UTF-8";  
/\* Licensed under the Apache License: http://www.apache.org/licenses/LICENSE-2.0 \*/  
/\* For details: https://github.com/nedbat/coveragepy/blob/master/NOTICE.txt \*/  
/\* Don't edit this .css file. Edit the .scss file instead! \*/  
html, body, h1, h2, h3, p, table, td, th { margin: 0; padding: 0; border: 0; font-weight: inherit; font-style: inherit; font-size: 100%; font-family: inherit; vertical-align: baseline; }  
  
body { font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Ubuntu, Cantarell, "Helvetica Neue", sans-serif; font-size: 1em; background: #fff; color: #000; }  
  
@media (prefers-color-scheme: dark) { body { background: #1e1e1e; } }  
  
@media (prefers-color-scheme: dark) { body { color: #eee; } }  
  
html > body { font-size: 16px; }  
  
a:active, a:focus { outline: 2px dashed #007acc; }  
  
p { font-size: .875em; line-height: 1.4em; }  
  
table { border-collapse: collapse; }  
  
td { vertical-align: top; }  
  
table tr.hidden { display: none !important; }  
  
p#no\_rows { display: none; font-size: 1.15em; font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Ubuntu, Cantarell, "Helvetica Neue", sans-serif; }  
  
a.nav { text-decoration: none; color: inherit; }  
  
a.nav:hover { text-decoration: underline; color: inherit; }  
  
.hidden { display: none; }  
  
header { background: #f8f8f8; width: 100%; z-index: 2; border-bottom: 1px solid #ccc; }  
  
@media (prefers-color-scheme: dark) { header { background: black; } }  
  
@media (prefers-color-scheme: dark) { header { border-color: #333; } }  
  
header .content { padding: 1rem 3.5rem; }  
  
header h2 { margin-top: .5em; font-size: 1em; }  
  
header h2 a.button { font-family: inherit; font-size: inherit; border: 1px solid; border-radius: .2em; background: #eee; color: inherit; text-decoration: none; padding: .1em .5em; margin: 1px calc(.1em + 1px); cursor: pointer; border-color: #ccc; }  
  
@media (prefers-color-scheme: dark) { header h2 a.button { background: #333; } }  
  
@media (prefers-color-scheme: dark) { header h2 a.button { border-color: #444; } }  
  
header h2 a.button.current { border: 2px solid; background: #fff; border-color: #999; cursor: default; }  
  
@media (prefers-color-scheme: dark) { header h2 a.button.current { background: #1e1e1e; } }  
  
@media (prefers-color-scheme: dark) { header h2 a.button.current { border-color: #777; } }  
  
header p.text { margin: .5em 0 -.5em; color: #666; font-style: italic; }  
  
@media (prefers-color-scheme: dark) { header p.text { color: #aaa; } }  
  
header.sticky { position: fixed; left: 0; right: 0; height: 2.5em; }  
  
header.sticky .text { display: none; }  
  
header.sticky h1, header.sticky h2 { font-size: 1em; margin-top: 0; display: inline-block; }  
  
header.sticky .content { padding: 0.5rem 3.5rem; }  
  
header.sticky .content p { font-size: 1em; }  
  
header.sticky ~ #source { padding-top: 6.5em; }  
  
main { position: relative; z-index: 1; }  
  
footer { margin: 1rem 3.5rem; }  
  
footer .content { padding: 0; color: #666; font-style: italic; }  
  
@media (prefers-color-scheme: dark) { footer .content { color: #aaa; } }  
  
#index { margin: 1rem 0 0 3.5rem; }  
  
h1 { font-size: 1.25em; display: inline-block; }  
  
#filter\_container { float: right; margin: 0 2em 0 0; line-height: 1.66em; }  
  
#filter\_container #filter { width: 10em; padding: 0.2em 0.5em; border: 2px solid #ccc; background: #fff; color: #000; }  
  
@media (prefers-color-scheme: dark) { #filter\_container #filter { border-color: #444; } }  
  
@media (prefers-color-scheme: dark) { #filter\_container #filter { background: #1e1e1e; } }  
  
@media (prefers-color-scheme: dark) { #filter\_container #filter { color: #eee; } }  
  
#filter\_container #filter:focus { border-color: #007acc; }  
  
#filter\_container :disabled ~ label { color: #ccc; }  
  
@media (prefers-color-scheme: dark) { #filter\_container :disabled ~ label { color: #444; } }  
  
#filter\_container label { font-size: .875em; color: #666; }  
  
@media (prefers-color-scheme: dark) { #filter\_container label { color: #aaa; } }  
  
header button { font-family: inherit; font-size: inherit; border: 1px solid; border-radius: .2em; background: #eee; color: inherit; text-decoration: none; padding: .1em .5em; margin: 1px calc(.1em + 1px); cursor: pointer; border-color: #ccc; }  
  
@media (prefers-color-scheme: dark) { header button { background: #333; } }  
  
@media (prefers-color-scheme: dark) { header button { border-color: #444; } }  
  
header button:active, header button:focus { outline: 2px dashed #007acc; }  
  
header button.run { background: #eeffee; }  
  
@media (prefers-color-scheme: dark) { header button.run { background: #373d29; } }  
  
header button.run.show\_run { background: #dfd; border: 2px solid #00dd00; margin: 0 .1em; }  
  
@media (prefers-color-scheme: dark) { header button.run.show\_run { background: #373d29; } }  
  
header button.mis { background: #ffeeee; }  
  
@media (prefers-color-scheme: dark) { header button.mis { background: #4b1818; } }  
  
header button.mis.show\_mis { background: #fdd; border: 2px solid #ff0000; margin: 0 .1em; }  
  
@media (prefers-color-scheme: dark) { header button.mis.show\_mis { background: #4b1818; } }  
  
header button.exc { background: #f7f7f7; }  
  
@media (prefers-color-scheme: dark) { header button.exc { background: #333; } }  
  
header button.exc.show\_exc { background: #eee; border: 2px solid #808080; margin: 0 .1em; }  
  
@media (prefers-color-scheme: dark) { header button.exc.show\_exc { background: #333; } }  
  
header button.par { background: #ffffd5; }  
  
@media (prefers-color-scheme: dark) { header button.par { background: #650; } }  
  
header button.par.show\_par { background: #ffa; border: 2px solid #bbbb00; margin: 0 .1em; }  
  
@media (prefers-color-scheme: dark) { header button.par.show\_par { background: #650; } }  
  
#help\_panel, #source p .annotate.long { display: none; position: absolute; z-index: 999; background: #ffffcc; border: 1px solid #888; border-radius: .2em; color: #333; padding: .25em .5em; }  
  
#source p .annotate.long { white-space: normal; float: right; top: 1.75em; right: 1em; height: auto; }  
  
#help\_panel\_wrapper { float: right; position: relative; }  
  
#keyboard\_icon { margin: 5px; }  
  
#help\_panel\_state { display: none; }  
  
#help\_panel { top: 25px; right: 0; padding: .75em; border: 1px solid #883; color: #333; }  
  
#help\_panel .keyhelp p { margin-top: .75em; }  
  
#help\_panel .legend { font-style: italic; margin-bottom: 1em; }  
  
.indexfile #help\_panel { width: 25em; }  
  
.pyfile #help\_panel { width: 18em; }  
  
#help\_panel\_state:checked ~ #help\_panel { display: block; }  
  
kbd { border: 1px solid black; border-color: #888 #333 #333 #888; padding: .1em .35em; font-family: SFMono-Regular, Menlo, Monaco, Consolas, monospace; font-weight: bold; background: #eee; border-radius: 3px; }  
  
#source { padding: 1em 0 1em 3.5rem; font-family: SFMono-Regular, Menlo, Monaco, Consolas, monospace; }  
  
#source p { position: relative; white-space: pre; }  
  
#source p \* { box-sizing: border-box; }  
  
#source p .n { float: left; text-align: right; width: 3.5rem; box-sizing: border-box; margin-left: -3.5rem; padding-right: 1em; color: #999; user-select: none; }  
  
@media (prefers-color-scheme: dark) { #source p .n { color: #777; } }  
  
#source p .n.highlight { background: #ffdd00; }  
  
#source p .n a { scroll-margin-top: 6em; text-decoration: none; color: #999; }  
  
@media (prefers-color-scheme: dark) { #source p .n a { color: #777; } }  
  
#source p .n a:hover { text-decoration: underline; color: #999; }  
  
@media (prefers-color-scheme: dark) { #source p .n a:hover { color: #777; } }  
  
#source p .t { display: inline-block; width: 100%; box-sizing: border-box; margin-left: -.5em; padding-left: 0.3em; border-left: 0.2em solid #fff; }  
  
@media (prefers-color-scheme: dark) { #source p .t { border-color: #1e1e1e; } }  
  
#source p .t:hover { background: #f2f2f2; }  
  
@media (prefers-color-scheme: dark) { #source p .t:hover { background: #282828; } }  
  
#source p .t:hover ~ .r .annotate.long { display: block; }  
  
#source p .t .com { color: #008000; font-style: italic; line-height: 1px; }  
  
@media (prefers-color-scheme: dark) { #source p .t .com { color: #6a9955; } }  
  
#source p .t .key { font-weight: bold; line-height: 1px; }  
  
#source p .t .str, #source p .t .fst { color: #0451a5; }  
  
@media (prefers-color-scheme: dark) { #source p .t .str, #source p .t .fst { color: #9cdcfe; } }  
  
#source p.mis .t { border-left: 0.2em solid #ff0000; }  
  
#source p.mis.show\_mis .t { background: #fdd; }  
  
@media (prefers-color-scheme: dark) { #source p.mis.show\_mis .t { background: #4b1818; } }  
  
#source p.mis.show\_mis .t:hover { background: #f2d2d2; }  
  
@media (prefers-color-scheme: dark) { #source p.mis.show\_mis .t:hover { background: #532323; } }  
  
#source p.run .t { border-left: 0.2em solid #00dd00; }  
  
#source p.run.show\_run .t { background: #dfd; }  
  
@media (prefers-color-scheme: dark) { #source p.run.show\_run .t { background: #373d29; } }  
  
#source p.run.show\_run .t:hover { background: #d2f2d2; }  
  
@media (prefers-color-scheme: dark) { #source p.run.show\_run .t:hover { background: #404633; } }  
  
#source p.exc .t { border-left: 0.2em solid #808080; }  
  
#source p.exc.show\_exc .t { background: #eee; }  
  
@media (prefers-color-scheme: dark) { #source p.exc.show\_exc .t { background: #333; } }  
  
#source p.exc.show\_exc .t:hover { background: #e2e2e2; }  
  
@media (prefers-color-scheme: dark) { #source p.exc.show\_exc .t:hover { background: #3c3c3c; } }  
  
#source p.par .t { border-left: 0.2em solid #bbbb00; }  
  
#source p.par.show\_par .t { background: #ffa; }  
  
@media (prefers-color-scheme: dark) { #source p.par.show\_par .t { background: #650; } }  
  
#source p.par.show\_par .t:hover { background: #f2f2a2; }  
  
@media (prefers-color-scheme: dark) { #source p.par.show\_par .t:hover { background: #6d5d0c; } }  
  
#source p .r { position: absolute; top: 0; right: 2.5em; font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Ubuntu, Cantarell, "Helvetica Neue", sans-serif; }  
  
#source p .annotate { font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Ubuntu, Cantarell, "Helvetica Neue", sans-serif; color: #666; padding-right: .5em; }  
  
@media (prefers-color-scheme: dark) { #source p .annotate { color: #ddd; } }  
  
#source p .annotate.short:hover ~ .long { display: block; }  
  
#source p .annotate.long { width: 30em; right: 2.5em; }  
  
#source p input { display: none; }  
  
#source p input ~ .r label.ctx { cursor: pointer; border-radius: .25em; }  
  
#source p input ~ .r label.ctx::before { content: "▶ "; }  
  
#source p input ~ .r label.ctx:hover { background: #e8f4ff; color: #666; }  
  
@media (prefers-color-scheme: dark) { #source p input ~ .r label.ctx:hover { background: #0f3a42; } }  
  
@media (prefers-color-scheme: dark) { #source p input ~ .r label.ctx:hover { color: #aaa; } }  
  
#source p input:checked ~ .r label.ctx { background: #d0e8ff; color: #666; border-radius: .75em .75em 0 0; padding: 0 .5em; margin: -.25em 0; }  
  
@media (prefers-color-scheme: dark) { #source p input:checked ~ .r label.ctx { background: #056; } }  
  
@media (prefers-color-scheme: dark) { #source p input:checked ~ .r label.ctx { color: #aaa; } }  
  
#source p input:checked ~ .r label.ctx::before { content: "▼ "; }  
  
#source p input:checked ~ .ctxs { padding: .25em .5em; overflow-y: scroll; max-height: 10.5em; }  
  
#source p label.ctx { color: #999; display: inline-block; padding: 0 .5em; font-size: .8333em; }  
  
@media (prefers-color-scheme: dark) { #source p label.ctx { color: #777; } }  
  
#source p .ctxs { display: block; max-height: 0; overflow-y: hidden; transition: all .2s; padding: 0 .5em; font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Ubuntu, Cantarell, "Helvetica Neue", sans-serif; white-space: nowrap; background: #d0e8ff; border-radius: .25em; margin-right: 1.75em; text-align: right; }  
  
@media (prefers-color-scheme: dark) { #source p .ctxs { background: #056; } }  
  
#index { font-family: SFMono-Regular, Menlo, Monaco, Consolas, monospace; font-size: 0.875em; }  
  
#index table.index { margin-left: -.5em; }  
  
#index td, #index th { text-align: right; padding: .25em .5em; border-bottom: 1px solid #eee; }  
  
@media (prefers-color-scheme: dark) { #index td, #index th { border-color: #333; } }  
  
#index td.name, #index th.name { text-align: left; width: auto; font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Ubuntu, Cantarell, "Helvetica Neue", sans-serif; min-width: 15em; }  
  
#index th { font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Ubuntu, Cantarell, "Helvetica Neue", sans-serif; font-style: italic; color: #333; cursor: pointer; }  
  
@media (prefers-color-scheme: dark) { #index th { color: #ddd; } }  
  
...truncated...

### 📄 migrations\env.py

from logging.config import fileConfig  
  
from sqlalchemy import engine\_from\_config  
from sqlalchemy import pool  
  
from alembic import context  
  
# this is the Alembic Config object, which provides  
# access to the values within the .ini file in use.  
config = context.config  
  
# Interpret the config file for Python logging.  
# This line sets up loggers basically.  
if config.config\_file\_name is not None:  
 fileConfig(config.config\_file\_name)  
  
# add your model's MetaData object here  
# for 'autogenerate' support  
# from myapp import mymodel  
# target\_metadata = mymodel.Base.metadata  
target\_metadata = None  
  
# other values from the config, defined by the needs of env.py,  
# can be acquired:  
# my\_important\_option = config.get\_main\_option("my\_important\_option")  
# ... etc.  
  
  
def run\_migrations\_offline() -> None:  
 """Run migrations in 'offline' mode.  
  
 This configures the context with just a URL  
 and not an Engine, though an Engine is acceptable  
 here as well. By skipping the Engine creation  
 we don't even need a DBAPI to be available.  
  
 Calls to context.execute() here emit the given string to the  
 script output.  
  
 """  
 url = config.get\_main\_option("sqlalchemy.url")  
 context.configure(  
 url=url,  
 target\_metadata=target\_metadata,  
 literal\_binds=True,  
 dialect\_opts={"paramstyle": "named"},  
 )  
  
 with context.begin\_transaction():  
 context.run\_migrations()  
  
  
def run\_migrations\_online() -> None:  
 """Run migrations in 'online' mode.  
  
 In this scenario we need to create an Engine  
 and associate a connection with the context.  
  
 """  
 connectable = engine\_from\_config(  
 config.get\_section(config.config\_ini\_section, {}),  
 prefix="sqlalchemy.",  
 poolclass=pool.NullPool,  
 )  
  
 with connectable.connect() as connection:  
 context.configure(  
 connection=connection, target\_metadata=target\_metadata  
 )  
  
 with context.begin\_transaction():  
 context.run\_migrations()  
  
  
if context.is\_offline\_mode():  
 run\_migrations\_offline()  
else:  
 run\_migrations\_online()

### 📄 requirements.txt

alembic==1.16.1  
annotated-types==0.7.0  
anyio==4.9.0  
arrow==1.3.0  
astroid==3.3.10  
blinker==1.9.0  
certifi==2025.4.26  
charset-normalizer==3.4.2  
click==8.2.1  
colorama==0.4.6  
coverage==7.9.1  
croniter==6.0.0  
dill==0.4.0  
distro==1.9.0  
Flask==3.1.0  
flexcache==0.3  
flexparser==0.4  
greenlet==3.2.2  
gunicorn==23.0.0  
h11==0.16.0  
httpcore==1.0.9  
httpx==0.28.1  
idna==3.10  
iniconfig==2.1.0  
isort==6.0.1  
itsdangerous==2.2.0  
Jinja2==3.1.6  
jiter==0.10.0  
lxml==5.4.0  
Mako==1.3.10  
MarkupSafe==3.0.2  
mccabe==0.7.0  
openai==1.92.2  
packaging==25.0  
Pint==0.24.4  
platformdirs==4.3.8  
pluggy==1.6.0  
psycopg2-binary==2.9.10  
pydantic==2.11.5  
pydantic\_core==2.33.2  
PyJWT==2.10.1  
pylint==3.3.7  
pytest==8.3.5  
pytest-cov==6.2.1  
python-dateutil==2.9.0.post0  
python-docx==1.1.2  
python-dotenv==1.1.0  
pytz==2025.2  
requests==2.31.0  
six==1.17.0  
sniffio==1.3.1  
SQLAlchemy==2.0.41  
stravalib==2.3  
tomlkit==0.13.3  
tqdm==4.67.1  
types-python-dateutil==2.9.0.20250516  
typing-inspection==0.4.1  
typing\_extensions==4.13.2  
urllib3==2.4.0  
Werkzeug==3.1.3  
flask-cors==4.0.0

### 📄 run.py

import sys  
import os  
from pathlib import Path  
from datetime import datetime  
from dotenv import load\_dotenv  
  
# ─────────────────────────────  
# 📦 Setup  
# ─────────────────────────────  
sys.path.insert(0, str(Path(\_\_file\_\_).resolve().parent))  
  
# Check and print the FLASK\_ENV to confirm it's being set correctly  
env\_mode = os.getenv("FLASK\_ENV", "production")  
env\_file = {  
 "local": ".env.local",  
 "staging": ".env.staging",  
 "production": ".env.prod"  
}.get(env\_mode, ".env")  
  
load\_dotenv(env\_file, override=True)  
print(f"🔍 Loaded environment file: {env\_file}", flush=True)  
print("DATABASE\_URL at runtime:", os.getenv("DATABASE\_URL"), flush=True)  
  
import src.utils.config as config  
from src.db.db\_session import get\_session  
from src.services.ingestion\_orchestrator\_service import run\_full\_ingestion\_and\_enrichment  
  
# ─────────────────────────────  
# 🚀 Flask App Setup for Gunicorn (top-level for Gunicorn import)  
# ─────────────────────────────  
from src.app import create\_app  
  
app = create\_app()  
print("✅ App created via create\_app()", flush=True)  
  
# ─────────────────────────────  
# 🧪 Local + Cron Execution Only  
# ─────────────────────────────  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("📦 Starting run.py...", flush=True)  
  
 # Patch DB URL if running locally  
 if config.IS\_LOCAL and config.DATABASE\_URL and 'postgres@postgres:' in config.DATABASE\_URL:  
 patched\_db\_url = config.DATABASE\_URL.replace('postgres@postgres:', 'postgres@localhost:')  
 os.environ['DATABASE\_URL'] = patched\_db\_url  
 print(f"🔧 DATABASE\_URL rewritten for local: {patched\_db\_url}", flush=True)  
 else:  
 print(f"✅ DATABASE\_URL used as-is: {config.DATABASE\_URL}", flush=True)  
  
 # Run as CRON job if enabled  
 if os.getenv("RUN\_CRON") == "true":  
 print(f"[CRON SYNC] ✅ Sync job started at {datetime.utcnow().isoformat()}", flush=True)  
 try:  
 session = get\_session()  
 athlete\_id = int(os.getenv("ATHLETE\_ID", "123456"))  
 result = run\_full\_ingestion\_and\_enrichment(session, athlete\_id)  
 print(f"[CRON SYNC] ✅ Sync complete: {result}", flush=True)  
 except Exception as e:  
 print(f"[CRON SYNC] ❌ Error during sync: {e}", flush=True)  
 import traceback  
 traceback.print\_exc()  
 sys.exit(0)  
  
 # Dev mode — test DB connection and run server  
 port = config.PORT  
 config\_db\_url = app.config.get("DATABASE\_URL") or config.DATABASE\_URL  
 print("🔍 ENV DATABASE\_URL =", config.DATABASE\_URL, flush=True)  
 print("🔍 CONFIG DATABASE\_URL =", config\_db\_url, flush=True)  
  
 if not config\_db\_url:  
 print("❗ DATABASE\_URL not set — exiting", flush=True)  
 sys.exit(1)  
  
 try:  
 from psycopg2 import connect  
 print(f"🔌 Attempting psycopg2.connect() to: {config\_db\_url}", flush=True)  
 sanitized\_url = config\_db\_url.replace("postgresql+psycopg2://", "postgresql://").split("#")[0].strip()  
 conn = connect(sanitized\_url)  
 with conn.cursor() as cur:  
 cur.execute("SELECT 1;")  
 cur.fetchone()  
 print("✅ DB test query succeeded!", flush=True)  
 except Exception as e:  
 print("⚠️ DB test query failed:", e, flush=True)  
 import traceback  
 traceback.print\_exc()  
  
 print(f"🚀 Starting app locally on 0.0.0.0:{port}", flush=True)  
 app.run(host="0.0.0.0", port=port, debug=True)

### 📄 src\\_\_init\_\_.py

### 📄 src\app.py

import os  
from dotenv import load\_dotenv  
from pathlib import Path  
  
# 📦 Environment Setup  
raw\_env\_mode = os.environ.get("FLASK\_ENV", "production")  
env\_path = {  
 "local": ".env.local",  
 "staging": ".env.staging",  
 "production": ".env.prod"  
}.get(raw\_env\_mode, ".env")  
  
load\_dotenv(env\_path, override=True)  
print(f"✅ Loaded environment: {env\_path}", flush=True)  
  
if env\_path != ".env":  
 print("🚫 Skipping .env fallback", flush=True)  
else:  
 print("ℹ️ Default .env used", flush=True)  
  
print(f"📍 STRAVA\_REDIRECT\_URI = {os.getenv('STRAVA\_REDIRECT\_URI')}", flush=True)  
  
# 🌐 Flask Setup  
from flask import Flask, send\_from\_directory, redirect  
from flask\_cors import CORS  
import src.utils.config as config  
from src.routes.admin\_routes import admin\_bp  
from src.routes.auth\_routes import auth\_bp  
from src.routes.activity\_routes import activity\_bp  
from src.routes.health\_routes import health\_bp  
from src.routes.ask\_routes import ask\_bp  
  
# ✅ Always serve from absolute resolved build path  
FRONTEND\_DIST = os.path.join(os.path.dirname(\_\_file\_\_), "frontend", "dist")  
  
def create\_app(test\_config=None):  
 app = Flask(\_\_name\_\_, static\_folder=FRONTEND\_DIST, static\_url\_path="/")  
 CORS(app, supports\_credentials=True)  
  
 # 🔐 Configuration  
 app.config.from\_mapping(  
 SECRET\_KEY=config.SECRET\_KEY,  
 DATABASE\_URL=os.getenv("DATABASE\_URL"),  
 CRON\_SECRET\_KEY=config.CRON\_SECRET\_KEY,  
 INTERNAL\_API\_KEY=config.INTERNAL\_API\_KEY,  
 SESSION\_TYPE="filesystem"  
 )  
  
 if test\_config:  
 app.config.update(test\_config)  
  
 # 🔗 Blueprints  
 app.register\_blueprint(auth\_bp, url\_prefix="/auth")  
 app.register\_blueprint(admin\_bp, url\_prefix="/admin")  
 app.register\_blueprint(activity\_bp, url\_prefix="/sync")  
 app.register\_blueprint(health\_bp)  
 app.register\_blueprint(ask\_bp)  
  
 # 🧪 Utility Endpoints  
 @app.route("/ping")  
 def ping():  
 return "pong", 200  
  
 @app.route("/startup")  
 def startup():  
 return {  
 "status": "started",  
 "env\_PORT": os.getenv("PORT"),  
 "env\_DATABASE\_URL": os.getenv("DATABASE\_URL"),  
 "config\_DATABASE\_URL": config.DATABASE\_URL,  
 "cwd": os.getcwd(),  
 "files": [p.name for p in Path(".").iterdir()],  
 }  
  
 @app.route("/db-check")  
 def db\_check():  
 try:  
 from sqlalchemy import create\_engine, inspect  
 db\_url = os.getenv("DATABASE\_URL")  
 engine = create\_engine(db\_url)  
 insp = inspect(engine)  
 columns = insp.get\_columns("splits")  
 split\_col = next((c for c in columns if c["name"] == "split"), None)  
 return {  
 "status": "ok",  
 "db": True,  
 "split\_column": {  
 "name": split\_col["name"],  
 "type": str(split\_col["type"]),  
 "nullable": split\_col["nullable"]  
 } if split\_col else "not found"  
 }  
 except Exception as e:  
 import traceback  
 traceback.print\_exc()  
 return {"status": "fail", "error": str(e)}, 500  
  
 @app.route("/post-oauth")  
 def post\_oauth():  
 if raw\_env\_mode == "local":  
 redirect\_url = os.getenv("FRONTEND\_REDIRECT")  
 return redirect(redirect\_url)  
 index\_path = os.path.join(app.static\_folder, "index.html")  
 if os.path.exists(index\_path):  
 return send\_from\_directory(app.static\_folder, "index.html")  
 return "❌ Frontend not found", 404  
  
 @app.errorhandler(404)  
 def spa\_fallback(e):  
 index\_path = os.path.join(app.static\_folder, "index.html")  
 if os.path.exists(index\_path):  
 return send\_from\_directory(app.static\_folder, "index.html")  
 return "404 Not Found", 404  
  
 # 🧭 Serve frontend (for production/staging)  
 @app.route("/", defaults={"path": ""})  
 @app.route("/<path:path>")  
 def serve\_frontend(path):  
 if raw\_env\_mode == "local":  
 return "✅ Dev mode — frontend served by Vite", 200  
  
 full\_path = os.path.join(app.static\_folder, path)  
 if path and os.path.exists(full\_path) and not path.endswith("/"):  
 return send\_from\_directory(app.static\_folder, path)  
 return send\_from\_directory(app.static\_folder, "index.html")  
  
 return app  
  
# 👟 Entry point for local dev  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = create\_app()  
 app.run(host="127.0.0.1", port=5000)

### 📄 src\db\\_\_init\_\_.py

### 📄 src\db\dao\activity\_dao.py

from sqlalchemy.dialects.postgresql import insert  
from sqlalchemy.orm import Session  
from src.db.models.activities import Activity  
  
from src.utils.conversions import convert\_metrics  
from src.utils.logger import get\_logger  
from typing import List, Dict  
  
logger = get\_logger(\_\_name\_\_)  
  
class ActivityDAO:  
 @staticmethod  
 def upsert\_activities(session: Session, athlete\_id: int, activities: List[Dict]) -> int:  
 """  
 Upsert activities into the database, filtering only 'Run' types.  
 """  
 if not activities:  
 return 0  
  
 rows = []  
 for act in activities:  
 if act.get("type") != "Run":  
 logger.warning(f"⚠️ Skipping non-Run activity {act.get('id')} — type={act.get('type')}")  
 continue  
  
 name = act.get("name", "").lower()  
 is\_treadmill = "treadmill" in name  
  
 required\_fields = ["id", "start\_date", "distance", "moving\_time", "elapsed\_time"]  
 if not is\_treadmill:  
 required\_fields.append("external\_id")  
  
 missing = [f for f in required\_fields if not act.get(f)]  
 if missing:  
 logger.error(f"❌ Skipping activity {act.get('id')} due to missing required fields: {missing}")  
 continue  
  
 conv\_input = {  
 "distance": act.get("distance"),  
 "elevation": act.get("total\_elevation\_gain"),  
 "average\_speed": act.get("average\_speed"),  
 "max\_speed": act.get("max\_speed"),  
 "moving\_time": act.get("moving\_time"),  
 "elapsed\_time": act.get("elapsed\_time")  
 }  
 conv\_fields = ["distance", "elevation", "average\_speed", "max\_speed", "moving\_time", "elapsed\_time"]  
 conv = convert\_metrics(conv\_input, conv\_fields)  
  
 row = {  
 "activity\_id": act["id"],  
 "athlete\_id": athlete\_id,  
 "name": act.get("name"),  
 "type": act.get("type"),  
 "start\_date": act.get("start\_date"),  
 "distance": act.get("distance"),  
 "elapsed\_time": act.get("elapsed\_time"),  
 "moving\_time": act.get("moving\_time"),  
 "total\_elevation\_gain": act.get("total\_elevation\_gain"),  
 "external\_id": act.get("external\_id"),  
 "timezone": act.get("timezone"),  
 "hr\_zone\_1": act.get("hr\_zone\_1"),  
 "hr\_zone\_2": act.get("hr\_zone\_2"),  
 "hr\_zone\_3": act.get("hr\_zone\_3"),  
 "hr\_zone\_4": act.get("hr\_zone\_4"),  
 "hr\_zone\_5": act.get("hr\_zone\_5"),  
 "conv\_distance": conv.get("conv\_distance"),  
 "conv\_elevation\_feet": conv.get("conv\_elevation\_feet"),  
 "conv\_avg\_speed": conv.get("conv\_avg\_speed"),  
 "conv\_max\_speed": conv.get("conv\_max\_speed"),  
 "conv\_moving\_time": conv.get("conv\_moving\_time"),  
 "conv\_elapsed\_time": conv.get("conv\_elapsed\_time"),  
 }  
 rows.append(row)  
  
 if not rows:  
 return 0  
  
 stmt = insert(Activity).values(rows)  
 update\_cols = {  
 col.name: getattr(stmt.excluded, col.name)  
 for col in Activity.\_\_table\_\_.columns  
 if col.name != "activity\_id"  
 }  
 stmt = stmt.on\_conflict\_do\_update(index\_elements=["activity\_id"], set\_=update\_cols)  
  
 result = session.execute(stmt)  
 session.commit()  
 return result.rowcount  
  
 @staticmethod  
 def get\_by\_id(session: Session, activity\_id: int) -> Activity | None:  
 """  
 Retrieve a single activity by its activity\_id.  
 """  
 return session.query(Activity).filter\_by(activity\_id=activity\_id).first()  
   
   
 @staticmethod  
 def get\_activities\_by\_athlete(session: Session, athlete\_id: int) -> list[Activity]:  
 return (  
 session.query(Activity)  
 .filter(Activity.athlete\_id == athlete\_id)  
 .order\_by(Activity.start\_date.desc())  
 .all()  
 )

### 📄 src\db\dao\activity\_stats\_dao.py

from datetime import datetime, timedelta  
from sqlalchemy import func, extract, select  
from src.db.models.activities import Activity  
from sqlalchemy.orm import Session  
from typing import List, Dict, Optional  
  
class ActivityStatsDAO:  
  
 @staticmethod  
 def get\_recent\_activities(session: Session, athlete\_id: int, days: int) -> List[Activity]:  
 cutoff = datetime.utcnow() - timedelta(days=days)  
 stmt = select(Activity).where(  
 Activity.athlete\_id == athlete\_id,  
 Activity.type == "Run",  
 Activity.start\_date >= cutoff  
 ).order\_by(Activity.start\_date.desc())  
 return session.scalars(stmt).all()  
  
 @staticmethod  
 def get\_activities\_by\_date\_range(session: Session, athlete\_id: int, start\_date: datetime, end\_date: datetime) -> List[Activity]:  
 stmt = select(Activity).where(  
 Activity.athlete\_id == athlete\_id,  
 Activity.type == "Run",  
 Activity.start\_date.between(start\_date, end\_date)  
 ).order\_by(Activity.start\_date)  
 return session.scalars(stmt).all()  
  
 @staticmethod  
 def get\_total\_distance(session: Session, athlete\_id: int, start\_date: datetime, end\_date: datetime) -> float:  
 stmt = select(func.sum(Activity.distance)).where(  
 Activity.athlete\_id == athlete\_id,  
 Activity.type == "Run",  
 Activity.start\_date.between(start\_date, end\_date)  
 )  
 return session.scalar(stmt) or 0.0  
  
 @staticmethod  
 def get\_average\_pace(session: Session, athlete\_id: int, days: int) -> Optional[float]:  
 cutoff = datetime.utcnow() - timedelta(days=days)  
 stmt = select(  
 func.avg(Activity.moving\_time / (Activity.distance / 1000))  
 ).where(  
 Activity.athlete\_id == athlete\_id,  
 Activity.type == "Run",  
 Activity.start\_date >= cutoff,  
 Activity.distance > 0  
 )  
 return session.scalar(stmt)  
  
 @staticmethod  
 def get\_longest\_run(session: Session, athlete\_id: int, days: int) -> Optional[Activity]:  
 cutoff = datetime.utcnow() - timedelta(days=days)  
 stmt = select(Activity).where(  
 Activity.athlete\_id == athlete\_id,  
 Activity.type == "Run",  
 Activity.start\_date >= cutoff  
 ).order\_by(Activity.distance.desc()).limit(1)  
 return session.scalar(stmt)  
  
 @staticmethod  
 def get\_fastest\_run(session: Session, athlete\_id: int, days: int) -> Optional[Activity]:  
 cutoff = datetime.utcnow() - timedelta(days=days)  
 stmt = select(Activity).where(  
 Activity.athlete\_id == athlete\_id,  
 Activity.type == "Run",  
 Activity.start\_date >= cutoff,  
 Activity.distance > 0  
 ).order\_by((Activity.moving\_time / (Activity.distance / 1000)).asc()).limit(1)  
 return session.scalar(stmt)  
  
 @staticmethod  
 def get\_weekly\_summary(session: Session, athlete\_id: int, past\_weeks: int = 4) -> List[Dict]:  
 cutoff = datetime.utcnow() - timedelta(weeks=past\_weeks)  
 stmt = select(  
 extract('year', Activity.start\_date).label('year'),  
 extract('week', Activity.start\_date).label('week'),  
 func.sum(Activity.distance).label('total\_distance'),  
 func.sum(Activity.moving\_time).label('total\_time')  
 ).where(  
 Activity.athlete\_id == athlete\_id,  
 Activity.type == "Run",  
 Activity.start\_date >= cutoff  
 ).group\_by('year', 'week').order\_by('year', 'week')  
 return [dict(row.\_mapping) for row in session.execute(stmt)]  
  
 @staticmethod  
 def get\_hr\_zone\_summary(session: Session, athlete\_id: int, days: int) -> Dict[str, float]:  
 cutoff = datetime.utcnow() - timedelta(days=days)  
 stmt = select(  
 func.avg(Activity.hr\_zone\_1),  
 func.avg(Activity.hr\_zone\_2),  
 func.avg(Activity.hr\_zone\_3),  
 func.avg(Activity.hr\_zone\_4),  
 func.avg(Activity.hr\_zone\_5)  
 ).where(  
 Activity.athlete\_id == athlete\_id,  
 Activity.type == "Run",  
 Activity.start\_date >= cutoff  
 )  
 row = session.execute(stmt).first()  
 return {  
 "zone\_1": row[0] or 0.0,  
 "zone\_2": row[1] or 0.0,  
 "zone\_3": row[2] or 0.0,  
 "zone\_4": row[3] or 0.0,  
 "zone\_5": row[4] or 0.0,  
 }  
  
 @staticmethod  
 def get\_treadmill\_vs\_outdoor\_stats(session: Session, athlete\_id: int, days: int) -> Dict[str, int]:  
 cutoff = datetime.utcnow() - timedelta(days=days)  
 stmt = select(Activity.name).where(  
 Activity.athlete\_id == athlete\_id,  
 Activity.type == "Run",  
 Activity.start\_date >= cutoff  
 )  
 runs = session.scalars(stmt).all()  
 treadmill = sum(1 for r in runs if r and "treadmill" in r.lower())  
 return {"treadmill": treadmill, "outdoor": len(runs) - treadmill}  
  
 @staticmethod  
 def get\_runs\_by\_weekday(session: Session, athlete\_id: int, days: int) -> Dict[str, int]:  
 cutoff = datetime.utcnow() - timedelta(days=days)  
 stmt = select(extract("dow", Activity.start\_date), func.count()).where(  
 Activity.athlete\_id == athlete\_id,  
 Activity.type == "Run",  
 Activity.start\_date >= cutoff  
 ).group\_by(extract("dow", Activity.start\_date))  
 result = session.execute(stmt).all()  
 return {str(int(day)): count for day, count in result}  
  
 @staticmethod  
 def get\_time\_of\_day\_stats(session: Session, athlete\_id: int, days: int) -> Dict[str, int]:  
 cutoff = datetime.utcnow() - timedelta(days=days)  
 stmt = select(Activity.start\_date).where(  
 Activity.athlete\_id == athlete\_id,  
 Activity.type == "Run",  
 Activity.start\_date >= cutoff  
 )  
 results = session.scalars(stmt).all()  
 morning = sum(1 for r in results if r and r.hour < 12)  
 return {"morning": morning, "evening": len(results) - morning}  
  
 @staticmethod  
 def get\_trend\_metrics(session: Session, athlete\_id: int, metric: str, window\_size: int = 7) -> List[Dict]:  
 metric\_col = getattr(Activity, metric, None)  
 if not metric\_col:  
 raise ValueError(f"Invalid metric field: {metric}")  
  
 cutoff = datetime.utcnow() - timedelta(days=window\_size \* 6)  
 stmt = select(  
 func.date\_trunc('week', Activity.start\_date).label('week'),  
 func.avg(metric\_col).label('avg\_metric')  
 ).where(  
 Activity.athlete\_id == athlete\_id,  
 Activity.type == "Run",  
 Activity.start\_date >= cutoff  
 ).group\_by('week').order\_by('week')  
 return [dict(row.\_mapping) for row in session.execute(stmt)]

### 📄 src\db\dao\athlete\_dao.py

# src/db/dao/athlete\_dao.py  
  
from typing import Optional, List  
from sqlalchemy.orm import Session  
from sqlalchemy.dialects.postgresql import insert as pg\_insert  
  
from src.db.models.athletes import Athlete # ✅ Import only the ORM class  
  
def get\_athlete\_by\_strava\_id(session: Session, strava\_id: int) -> Optional[Athlete]:  
 """  
 Retrieve the athlete record by their Strava athlete ID.  
 """  
 return session.query(Athlete).filter\_by(strava\_athlete\_id=strava\_id).first()  
  
  
def get\_athlete\_id\_from\_strava\_id(session: Session, strava\_id: int) -> Optional[int]:  
 """  
 Retrieve internal athlete ID from the Strava athlete ID.  
 """  
 result = session.query(Athlete.id).filter\_by(strava\_athlete\_id=strava\_id).first()  
 return result.id if result else None  
  
  
def insert\_athlete(session: Session, strava\_athlete\_id: int, name: Optional[str] = None, email: Optional[str] = None) -> int:  
 """  
 Insert a new athlete record and return the internal athlete ID.  
 """  
 new\_athlete = Athlete(  
 strava\_athlete\_id=strava\_athlete\_id,  
 name=name,  
 email=email  
 )  
 session.add(new\_athlete)  
 session.commit()  
 session.refresh(new\_athlete)  
 return new\_athlete.id  
  
  
def get\_all\_athletes(session: Session) -> List[Athlete]:  
 """  
 Retrieve all athletes from the database.  
 """  
 return session.query(Athlete).all()  
  
  
def upsert\_athlete(  
 session: Session,  
 athlete\_id: int,  
 strava\_athlete\_id: int,  
 name: Optional[str],  
 email: Optional[str]  
) -> None:  
 """  
 Upsert athlete's name and email using internal athlete ID.  
 strava\_athlete\_id is only used during insert, never updated.  
 """  
 stmt = pg\_insert(Athlete.\_\_table\_\_).values(  
 id=athlete\_id,  
 strava\_athlete\_id=strava\_athlete\_id,  
 name=name,  
 email=email  
 ).on\_conflict\_do\_update(  
 index\_elements=["id"],  
 set\_={  
 "name": name,  
 "email": email  
 # ❌ do NOT include "strava\_athlete\_id"  
 }  
 )  
 session.execute(stmt)  
 session.commit()

### 📄 src\db\dao\split\_dao.py

from sqlalchemy.dialects.postgresql import insert  
from src.db.models.splits import Split  
from src.utils.conversions import convert\_metrics  
  
def upsert\_splits(session, splits: list) -> int:  
 """  
 Upserts multiple split records into the 'splits' table.  
 Applies conversion logic centrally before inserting.  
 Ensures 'split' field is consistently a valid integer type.  
 """  
 if not splits:  
 return 0  
  
 converted = []  
 for s in splits:  
 # Ensure 'split' is a true int, not a bool or invalid type  
 split\_raw = s.get("split")  
 split\_value = None  
 if split\_raw is not None:  
 if isinstance(split\_raw, bool):  
 split\_value = 1 if split\_raw else 0  
 elif isinstance(split\_raw, (int, float, str)):  
 try:  
 split\_value = int(split\_raw)  
 except (ValueError, TypeError):  
 split\_value = None # fallback for invalid string etc.  
  
 conv\_fields = ["distance", "average\_speed", "moving\_time", "elapsed\_time"]  
 conv\_data = {  
 "distance": s.get("distance"),  
 "average\_speed": s.get("average\_speed"),  
 "moving\_time": s.get("moving\_time"),  
 "elapsed\_time": s.get("elapsed\_time"),  
 }  
 enriched = convert\_metrics(conv\_data, conv\_fields)  
  
 converted.append({  
 "activity\_id": s["activity\_id"],  
 "lap\_index": s["lap\_index"],  
 "distance": s["distance"],  
 "elapsed\_time": s["elapsed\_time"],  
 "moving\_time": s["moving\_time"],  
 "average\_speed": s["average\_speed"],  
 "max\_speed": s["max\_speed"],  
 "start\_index": s["start\_index"],  
 "end\_index": s["end\_index"],  
 "split": split\_value,  
 "average\_heartrate": s.get("average\_heartrate"),  
 "pace\_zone": s.get("pace\_zone"),  
 "conv\_distance": enriched.get("conv\_distance"),  
 "conv\_avg\_speed": enriched.get("conv\_avg\_speed"),  
 "conv\_moving\_time": enriched.get("conv\_moving\_time"),  
 "conv\_elapsed\_time": enriched.get("conv\_elapsed\_time"),  
 })  
  
 # 🔍 Debug output to verify types  
 print(f"[DAO DEBUG] split values: {[row['split'] for row in converted]} | types: {[type(row['split']) for row in converted]}")  
  
 stmt = insert(Split).values(converted)  
  
 update\_map = {  
 col.name: getattr(stmt.excluded, col.name)  
 for col in Split.\_\_table\_\_.columns  
 if col.name not in ("activity\_id", "lap\_index")  
 }  
  
 stmt = stmt.on\_conflict\_do\_update(  
 index\_elements=["activity\_id", "lap\_index"],  
 set\_=update\_map  
 )  
  
 result = session.execute(stmt)  
 session.commit()  
 return result.rowcount

### 📄 src\db\dao\token\_dao.py

from sqlalchemy.exc import NoResultFound  
from sqlalchemy.dialects.postgresql import insert  
from src.db.models.tokens import Token  
  
  
def get\_tokens\_sa(session, athlete\_id: int) -> dict | None:  
 """  
 Retrieves access, refresh, and expiration tokens for the given athlete.  
 Returns None if not found.  
 """  
 try:  
 token = session.query(Token).filter\_by(athlete\_id=athlete\_id).one()  
 return {  
 "access\_token": token.access\_token,  
 "refresh\_token": token.refresh\_token,  
 "expires\_at": token.expires\_at,  
 }  
 except NoResultFound:  
 return None  
  
  
def insert\_token\_sa(session, athlete\_id: int, access\_token: str, refresh\_token: str, expires\_at: int) -> None:  
 """  
 Inserts or updates a token record for the given athlete using upsert.  
 """  
 stmt = insert(Token).values(  
 athlete\_id=athlete\_id,  
 access\_token=access\_token,  
 refresh\_token=refresh\_token,  
 expires\_at=expires\_at  
 ).on\_conflict\_do\_update(  
 index\_elements=["athlete\_id"],  
 set\_={  
 "access\_token": access\_token,  
 "refresh\_token": refresh\_token,  
 "expires\_at": expires\_at  
 }  
 )  
  
 session.execute(stmt)  
 session.commit()  
  
  
def delete\_tokens\_sa(session, athlete\_id: int) -> int:  
 """  
 Deletes the token record for the given athlete.  
 Returns the number of rows deleted.  
 """  
 result = session.query(Token).filter\_by(athlete\_id=athlete\_id).delete()  
 session.commit()  
 return result

### 📄 src\db\db\_session.py

from sqlalchemy import create\_engine  
from sqlalchemy.orm import declarative\_base, sessionmaker  
  
import src.utils.config as config  
  
# Global declarative base — shared across models  
Base = declarative\_base()  
  
def get\_engine(db\_url=None):  
 """  
 Create a SQLAlchemy engine.  
 Allows optional db\_url override for tests or special cases.  
 """  
 db\_url = db\_url or config.DATABASE\_URL  
 if not db\_url:  
 raise RuntimeError("DATABASE\_URL is not set in configuration.")  
 return create\_engine(db\_url, echo=False, future=True)  
  
def get\_session(engine=None):  
 """  
 Create a new SQLAlchemy sessionmaker (not a global session).  
 Allows optional engine injection for test harnesses.  
 """  
 engine = engine or get\_engine()  
 SessionLocal = sessionmaker(bind=engine, autoflush=False, autocommit=False, future=True)  
 return SessionLocal()

### 📄 src\db\models\activities.py

from sqlalchemy import Column, BigInteger, Integer, String, Float, DateTime  
from src.db.db\_session import Base  
  
class Activity(Base):  
 \_\_tablename\_\_ = "activities"  
  
 activity\_id = Column(BigInteger, primary\_key=True, index=True)  
 athlete\_id = Column(BigInteger, nullable=False, index=True)  
 name = Column(String)  
 type = Column(String)  
 start\_date = Column(DateTime)  
 distance = Column(Float)  
 elapsed\_time = Column(Integer)  
 moving\_time = Column(Integer)  
 total\_elevation\_gain = Column(Float)  
 external\_id = Column(String)  
 timezone = Column(String)  
  
 average\_speed = Column(Float)  
 max\_speed = Column(Float)  
 suffer\_score = Column(Float)  
 average\_heartrate = Column(Float)  
 max\_heartrate = Column(Float)  
 calories = Column(Float)  
  
 conv\_distance = Column(Float)  
 conv\_elevation\_feet = Column(Float)  
 conv\_avg\_speed = Column(Float)  
 conv\_max\_speed = Column(Float)  
 conv\_moving\_time = Column(String)  
 conv\_elapsed\_time = Column(String)  
  
 # HR Zone enrichment (test field)  
 hr\_zone\_1 = Column(Float, nullable=True)  
 hr\_zone\_2 = Column(Float, nullable=True)  
 hr\_zone\_3 = Column(Float, nullable=True)  
 hr\_zone\_4 = Column(Float, nullable=True)  
 hr\_zone\_5 = Column(Float, nullable=True)

### 📄 src\db\models\athletes.py

from sqlalchemy import Column, Integer, BigInteger, String, DateTime, func  
from src.db.db\_session import Base  
  
class Athlete(Base):  
 \_\_tablename\_\_ = "athletes"  
  
 id = Column(Integer, primary\_key=True)  
 strava\_athlete\_id = Column(BigInteger, unique=True, nullable=False)  
 name = Column(String, nullable=True)  
 email = Column(String, nullable=True)  
 created\_at = Column(DateTime, server\_default=func.now())

### 📄 src\db\models\splits.py

from sqlalchemy import (  
 Column,  
 Integer,  
 Float,  
 ForeignKey,  
 TIMESTAMP,  
 BigInteger,  
 String,  
 UniqueConstraint,  
 Boolean, # ✅ Added Boolean  
)  
from sqlalchemy.sql import func  
from sqlalchemy.dialects.postgresql import insert  
from src.db.db\_session import Base, get\_session  
  
  
class Split(Base):  
 \_\_tablename\_\_ = "splits"  
  
 id = Column(Integer, primary\_key=True)  
 activity\_id = Column(BigInteger, ForeignKey("activities.activity\_id", ondelete="CASCADE"), nullable=False)  
 lap\_index = Column(Integer, nullable=False)  
 distance = Column(Float)  
 elapsed\_time = Column(Integer)  
 moving\_time = Column(Integer)  
 average\_speed = Column(Float)  
 max\_speed = Column(Float)  
 start\_index = Column(Integer)  
 end\_index = Column(Integer)  
 split = Column(Integer, nullable=True)  
 average\_heartrate = Column(Float)  
 pace\_zone = Column(Integer)  
 created\_at = Column(TIMESTAMP, server\_default=func.now())  
 conv\_distance = Column(Float)  
 conv\_avg\_speed = Column(Float)  
 conv\_moving\_time = Column(String)  
 conv\_elapsed\_time = Column(String)  
  
 \_\_table\_args\_\_ = (  
 UniqueConstraint("activity\_id", "lap\_index", name="uq\_activity\_lap"),  
 )  
  
  
def upsert\_splits(session, splits\_data: list[dict]):  
 if not splits\_data:  
 return  
  
 stmt = insert(Split).values(splits\_data)  
 update\_cols = {  
 c.name: getattr(stmt.excluded, c.name)  
 for c in Split.\_\_table\_\_.columns  
 if c.name not in {"id", "created\_at"}  
 }  
  
 stmt = stmt.on\_conflict\_do\_update(  
 constraint="uq\_activity\_lap",  
 set\_=update\_cols  
 )  
  
 session.execute(stmt)  
 session.commit()

### 📄 src\db\models\tokens.py

from sqlalchemy import Column, BigInteger, String  
from src.db.db\_session import Base # ✅ use shared Base  
  
class Token(Base):  
 \_\_tablename\_\_ = "tokens"  
  
 athlete\_id = Column(BigInteger, primary\_key=True)  
 access\_token = Column(String, nullable=False)  
 refresh\_token = Column(String, nullable=False)  
 expires\_at = Column(BigInteger, nullable=False)

### 📄 src\routes\\_\_init\_\_.py

### 📄 src\routes\activity\_routes.py

"""  
activity\_routes.py  
  
Enrichment-related endpoints only. All Strava activity ingestion must be run via CLI:  
  
 ✅ USE THIS:  
 python -m src.scripts.main\_pipeline --athlete\_id <id> --lookback\_days <N>  
  
 ⛔ DO NOT USE:  
 /sync/<athlete\_id> route — this is deprecated and will return 410 in prod.  
  
Why:  
- Ensures consistent ingestion logic  
- Avoids API parameter drift  
- Matches production cron jobs and test paths  
"""  
  
import os  
import traceback  
from flask import Blueprint, jsonify, request  
from sqlalchemy import text  
  
import src.utils.config as config  
from src.services.activity\_service import ActivityIngestionService, run\_enrichment\_batch  
from src.db.db\_session import get\_session  
  
activity\_bp = Blueprint("activity", \_\_name\_\_)  
  
# -------- Enrichment Routes --------  
  
@activity\_bp.route("/enrich/status", methods=["GET"])  
def enrich\_status():  
 """Quick health check"""  
 return jsonify({"enrich": "ok"}), 200  
  
  
@activity\_bp.route("/enrich/activity/<int:activity\_id>", methods=["POST"])  
def enrich\_single(activity\_id):  
 """Trigger enrichment for a single activity"""  
 session = get\_session()  
 try:  
 row = session.execute(  
 text("SELECT athlete\_id FROM activities WHERE activity\_id = :id"),  
 {"id": activity\_id}  
 ).fetchone()  
  
 if not row:  
 return jsonify({"error": f"Activity {activity\_id} not found"}), 404  
  
 athlete\_id = row.athlete\_id  
 service = ActivityIngestionService(session, athlete\_id)  
 service.enrich\_single\_activity(activity\_id)  
  
 return jsonify({"status": f"Activity {activity\_id} enriched"}), 200  
  
 except Exception as e:  
 traceback.print\_exc()  
 return jsonify({"error": str(e)}), 500  
  
 finally:  
 session.close()  
  
  
@activity\_bp.route("/enrich/batch", methods=["POST"])  
def enrich\_batch():  
 """Enrich a batch of activities for a given athlete"""  
 athlete\_id = request.args.get("athlete\_id", type=int)  
 batch = request.args.get("batch", default=20, type=int)  
  
 if not athlete\_id:  
 return jsonify({"error": "Missing athlete\_id"}), 400  
  
 session = get\_session()  
 try:  
 enriched = run\_enrichment\_batch(session, athlete\_id, batch\_size=batch)  
 return jsonify({"status": "Batch enrichment complete", "count": enriched}), 200  
 except Exception as e:  
 traceback.print\_exc()  
 return jsonify({"error": str(e)}), 500  
 finally:  
 session.close()  
  
  
# -------- Deprecated Sync Route --------  
  
@activity\_bp.route("/sync/<int:athlete\_id>")  
def sync\_strava\_to\_db(athlete\_id):  
 """  
 ⚠️ DEPRECATED in production. Used only for test validation.  
 """  
 if os.getenv("FLASK\_ENV") != "test":  
 return jsonify({  
 "error": "This sync route is deprecated. Use CLI ingestion instead.",  
 "hint": "python -m src.scripts.main\_pipeline --athlete\_id <id> --lookback\_days <N>"  
 }), 410  
  
 lookback = request.args.get("lookback", default=14, type=int)  
 limit = request.args.get("limit", default=None, type=int)  
 key = request.args.get("key")  
  
 if key != config.CRON\_SECRET\_KEY:  
 return jsonify({"error": "Unauthorized"}), 401  
  
 session = get\_session()  
 try:  
 service = ActivityIngestionService(session, athlete\_id)  
 inserted = service.ingest\_recent(lookback\_days=lookback, max\_activities=limit)  
 return jsonify({"inserted": inserted}), 200  
 except Exception as e:  
 traceback.print\_exc()  
 return jsonify({"error": str(e)}), 500  
 finally:  
 session.close()

### 📄 src\routes\admin\_routes.py

from flask import Blueprint, jsonify, request  
from src.services.ingestion\_orchestrator\_service import run\_full\_ingestion\_and\_enrichment  
from src.db.db\_session import get\_session  
  
admin\_bp = Blueprint("admin", \_\_name\_\_)  
  
@admin\_bp.route("/ping")  
def ping():  
 return "pong from admin"  
  
@admin\_bp.route("/trigger-ingest/<int:athlete\_id>", methods=["POST"])  
def trigger\_ingestion(athlete\_id):  
 print(f"⏱️ Received trigger-ingest for athlete {athlete\_id}", flush=True)  
 session = get\_session()  
 try:  
   
 lookback\_days = request.args.get("lookback\_days", default=None, type=int)  
 max\_activities = request.args.get("max\_activities", default=10, type=int)  
  
 result = run\_full\_ingestion\_and\_enrichment(  
 session=session,  
 athlete\_id=athlete\_id,  
 lookback\_days=lookback\_days,  
 max\_activities=max\_activities,  
 batch\_size=10,  
 per\_page=200  
)  
 print(f"✅ Ingestion result: {result}", flush=True)  
 return jsonify({"status": "success", "result": result}), 200  
 except Exception as e:  
 print(f"❌ Ingestion error: {e}", flush=True)  
 return jsonify({"status": "error", "message": str(e)}), 500  
 finally:  
 session.close()

### 📄 src\routes\ask\_routes.py

from flask import Blueprint, request, jsonify  
from src.utils.gpt\_ops import format\_prompt, get\_gpt\_response  
from src.db.db\_session import get\_session  
from src.db.dao.activity\_dao import ActivityDAO  
from datetime import datetime, timedelta  
  
ask\_bp = Blueprint('ask', \_\_name\_\_)  
  
@ask\_bp.route('/ask', methods=['POST'])  
def ask():  
 if not request.is\_json:  
 print("Error: Content-Type is not JSON")  
 return jsonify({"error": "Request content-type must be application/json"}), 400  
  
 data = request.get\_json()  
 print(f"Received data: {data}")  
  
 if not data:  
 print("Error: Missing JSON payload")  
 return jsonify({"error": "Missing JSON payload"}), 400  
  
 question = data.get("question")  
 athlete\_id = data.get("athlete\_id")  
  
 if not isinstance(question, str) or not question.strip():  
 print("Error: Invalid or missing 'question'")  
 return jsonify({"error": "Invalid or missing 'question'"}), 400  
  
 try:  
 athlete\_id = int(athlete\_id)  
 if athlete\_id <= 0:  
 raise ValueError  
 except (ValueError, TypeError):  
 print("Error: Invalid or missing 'athlete\_id'")  
 return jsonify({"error": "Invalid or missing 'athlete\_id' (must be a positive integer)"}), 400  
  
 sanitized\_question = " ".join(question.strip().split())  
  
   
 from datetime import date, timedelta  
 # Get start of current week (Monday)  
 today = date.today()  
 start\_date = today - timedelta(days=today.weekday())  
  
 session = get\_session()  
 try:  
 activities = ActivityDAO.get\_activities\_by\_athlete(session, athlete\_id)  
 filtered = [  
 a for a in activities  
 if a.start\_date and start\_date <= a.start\_date.date() <= today  
 ]  
  
 activity\_data = [  
 {  
 "start\_date": a.start\_date.strftime("%Y-%m-%d %H:%M:%S"),  
 "conv\_distance": round(a.conv\_distance, 2),  
 "duration": f"{round(a.moving\_time / 60)} minutes"  
 }  
 for a in filtered  
 ]  
 finally:  
 session.close()  
  
 print(f"Activity data: {activity\_data}")  
  
 prompt = format\_prompt(sanitized\_question, activity\_data)  
 print(f"Generated prompt: {prompt}")  
  
 gpt\_response = get\_gpt\_response(prompt)  
 print(f"Full GPT Response: {gpt\_response}")  
  
 return jsonify({  
 "message": "✅ GPT response generated",  
 "athlete\_id": athlete\_id,  
 "question": sanitized\_question,  
 "response": gpt\_response  
 }), 200

### 📄 src\routes\auth\_routes.py

from flask import Blueprint, redirect, request, jsonify, session as flask\_session, current\_app  
import traceback  
import requests  
from datetime import datetime, timedelta  
import jwt  
import os  
  
import src.utils.config as config  
from src.services.token\_service import (  
 refresh\_token\_if\_expired,  
 delete\_athlete\_tokens  
)  
from src.db.db\_session import get\_session  
  
auth\_bp = Blueprint("auth", \_\_name\_\_)  
  
@auth\_bp.route('/whoami', methods=['GET'])  
def whoami():  
 athlete\_id = flask\_session.get("athlete\_id")  
 if not athlete\_id:  
 return jsonify({"error": "Not logged in"}), 401  
 return jsonify({"athlete\_id": athlete\_id})  
  
@auth\_bp.route("/login", methods=["POST"])  
def admin\_login():  
 try:  
 data = request.get\_json()  
 username = data.get("username")  
 password = data.get("password")  
  
 if username == config.ADMIN\_USER and password == config.ADMIN\_PASS:  
 access\_token = jwt.encode(  
 {"sub": username, "exp": datetime.utcnow() + timedelta(seconds=config.ACCESS\_TOKEN\_EXP)},  
 config.SECRET\_KEY, algorithm="HS256"  
 )  
 refresh\_token = jwt.encode(  
 {"sub": username, "exp": datetime.utcnow() + timedelta(seconds=config.REFRESH\_TOKEN\_EXP)},  
 config.SECRET\_KEY, algorithm="HS256"  
 )  
 return jsonify({  
 "access\_token": access\_token,  
 "refresh\_token": refresh\_token  
 }), 200  
 else:  
 return jsonify({"error": "Unauthorized"}), 401  
 except Exception as e:  
 traceback.print\_exc()  
 return jsonify({"error": str(e)}), 500  
  
@auth\_bp.route("/login", methods=["GET"])  
def strava\_login():  
 redirect\_uri = os.getenv("STRAVA\_REDIRECT\_URI")  
 client\_id = os.getenv("STRAVA\_CLIENT\_ID")  
  
 print(f"🌐 OAuth Login Triggered", flush=True)  
 print(f"🔑 STRAVA\_CLIENT\_ID = {client\_id}", flush=True)  
 print(f"📍 STRAVA\_REDIRECT\_URI = {redirect\_uri}", flush=True)  
  
 url = (  
 f"https://www.strava.com/oauth/authorize"  
 f"?client\_id={client\_id}"  
 f"&response\_type=code"  
 f"&redirect\_uri={redirect\_uri}"  
 f"&scope=read,activity:read\_all"  
 )  
 return redirect(url)  
  
@auth\_bp.route("/callback")  
def callback():  
 from src.services.token\_service import store\_tokens\_from\_callback  
 session = get\_session()  
 try:  
 code = request.args.get("code")  
 if not code:  
 print("❌ Missing code param", flush=True)  
 return "❌ Missing OAuth code", 400  
  
 print(f"➡️ Received OAuth code: {code}", flush=True)  
  
 token\_payload = {  
 "client\_id": os.getenv("STRAVA\_CLIENT\_ID"),  
 "client\_secret": os.getenv("STRAVA\_CLIENT\_SECRET"),  
 "code": code,  
 "grant\_type": "authorization\_code",  
 "redirect\_uri": os.getenv("STRAVA\_REDIRECT\_URI"),  
 }  
  
 print("🚨 Token request payload being sent to Strava:")  
 import pprint  
 pprint.PrettyPrinter(indent=2).pprint(token\_payload)  
  
 athlete\_id = store\_tokens\_from\_callback(code, session)  
 flask\_session["athlete\_id"] = athlete\_id  
  
 print(f"✅ Stored token and session for athlete\_id: {athlete\_id}", flush=True)  
  
 if current\_app.config.get("TESTING"):  
 return f"Token stored for athlete\_id: {athlete\_id}", 200  
  
 return redirect("/post-oauth?authed=true")  
  
 except requests.exceptions.HTTPError as e:  
 print(f"🔥 Callback HTTP error: {e}", flush=True)  
 return jsonify({"error": "Strava OAuth token exchange failed"}), 502  
  
 except Exception as e:  
 traceback.print\_exc()  
 return f"❌ Callback error: {str(e)}", 500  
  
 finally:  
 session.close()  
  
@auth\_bp.route("/refresh/<int:athlete\_id>", methods=["POST"])  
def refresh\_token(athlete\_id):  
 session = get\_session()  
 try:  
 auth\_header = request.headers.get("Authorization", "")  
 if not auth\_header.startswith("Bearer "):  
 return jsonify({"error": "Missing or invalid Authorization header"}), 401  
  
 token = auth\_header.split(" ")[1]  
 try:  
 jwt.decode(token, config.SECRET\_KEY, algorithms=["HS256"])  
 except jwt.ExpiredSignatureError:  
 return jsonify({"error": "Refresh token expired"}), 401  
 except jwt.InvalidTokenError:  
 return jsonify({"error": "Invalid token"}), 401  
  
 refreshed = refresh\_token\_if\_expired(session, athlete\_id)  
 return jsonify({"refreshed": refreshed}), 200  
 except Exception as e:  
 traceback.print\_exc()  
 return jsonify({"error": str(e)}), 500  
 finally:  
 session.close()  
  
@auth\_bp.route("/logout/<int:athlete\_id>", methods=["POST"])  
def logout(athlete\_id):  
 session = get\_session()  
 try:  
 deleted = delete\_athlete\_tokens(session, athlete\_id)  
 return jsonify({"deleted": deleted}), 200  
 except Exception as e:  
 traceback.print\_exc()  
 return jsonify({"error": str(e)}), 500  
 finally:  
 session.close()  
  
@auth\_bp.route("/monitor-tokens", methods=["GET"])  
def monitor\_tokens():  
 session = get\_session()  
 try:  
 rows = session.execute("SELECT athlete\_id, expires\_at FROM tokens ORDER BY expires\_at").fetchall()  
 data = [{"athlete\_id": r.athlete\_id, "expires\_at": r.expires\_at} for r in rows]  
 return jsonify(data), 200  
 except Exception as e:  
 traceback.print\_exc()  
 return jsonify({"error": str(e)}), 500  
 finally:  
 session.close()  
  
@auth\_bp.route("/profile", methods=["POST"])  
def save\_athlete\_profile():  
 from src.db.dao.athlete\_dao import upsert\_athlete  
 session = get\_session()  
 try:  
 data = request.get\_json()  
 athlete\_id = data.get("athlete\_id")  
 name = data.get("name", "").strip()  
 email = data.get("email", "").strip()  
  
 if not athlete\_id:  
 return jsonify({"error": "Missing athlete\_id"}), 400  
 if not name and not email:  
 return jsonify({"error": "At least one of name or email must be provided"}), 400  
  
 upsert\_athlete(session, athlete\_id, strava\_athlete\_id=athlete\_id, name=name, email=email)  
 return jsonify({"status": "✅ Profile saved"}), 200  
  
 except Exception as e:  
 traceback.print\_exc()  
 return jsonify({"error": str(e)}), 500  
 finally:  
 session.close()

### 📄 src\routes\health\_routes.py

# src/routes/health\_routes.py  
  
from flask import Blueprint, jsonify  
from sqlalchemy import text  
from src.db.db\_session import get\_session  
  
health\_bp = Blueprint("health", \_\_name\_\_)  
  
@health\_bp.route("/health", methods=["GET"])  
def health\_check():  
 try:  
 session = get\_session()  
 session.execute(text("SELECT 1"))  
 return jsonify({"status": "ok", "db": "connected"}), 200  
 except Exception as e:  
 return jsonify({"status": "error", "db": "disconnected", "error": str(e)}), 500

### 📄 src\scripts\\_\_init\_\_.py

### 📄 src\scripts\activity\_query\_cli.py

import argparse  
from datetime import datetime, timedelta  
from src.db.db\_session import get\_session  
from src.db.dao.activity\_stats\_dao import ActivityStatsDAO  
from src.utils.logger import get\_logger  
from src.utils.config import load\_env  
  
logger = get\_logger(\_\_name\_\_)  
  
def main(athlete\_id: int, lookback\_days: int, mode: str, metric: str = None):  
 load\_env()  
 session = get\_session()  
   
 match mode:  
 case "recent\_activities":  
 data = ActivityStatsDAO.get\_recent\_activities\_by\_athlete(session, athlete\_id, lookback\_days)  
 print(f"\n✅ Retrieved {len(data)} recent activities:")  
 for r in data:  
 print(f"- {r['start\_date']}: {r['name']}")  
  
 case "fastest\_run":  
 r = ActivityStatsDAO.get\_fastest\_run(session, athlete\_id, lookback\_days)  
 print(f"\n🚀 Fastest run: {r['name']} on {r['start\_date']} @ pace {r['pace']:.2f} min/km")  
  
 case "longest\_run":  
 r = ActivityStatsDAO.get\_longest\_run(session, athlete\_id, lookback\_days)  
 print(f"\n🏃‍♂️ Longest run: {r['name']} on {r['start\_date']} — {r['distance\_km']:.2f} km")  
  
 case "total\_distance":  
 end = datetime.utcnow()  
 start = end - timedelta(days=lookback\_days)  
 stats = ActivityStatsDAO.get\_total\_distance(session, athlete\_id, start, end)  
 print(f"\n📊 Total distance: {stats['distance\_km']:.2f} km, Time: {stats['duration\_hours']:.2f} hrs")  
  
 case "weekly\_summary":  
 data = ActivityStatsDAO.get\_weekly\_summary(session, athlete\_id)  
 print("\n📅 Weekly summary:")  
 for d in data:  
 print(f"- Week of {d['week\_start']}: {d['total\_distance\_km']:.2f} km in {d['total\_time\_hr']:.2f} hrs")  
  
 case "hr\_zone\_summary":  
 zones = ActivityStatsDAO.get\_hr\_zone\_summary(session, athlete\_id, lookback\_days)  
 print("\n❤️ HR Zone Summary:")  
 for k, v in zones.items():  
 print(f"- {k}: {v:.2f} min")  
  
 case "treadmill\_stats":  
 stats = ActivityStatsDAO.get\_treadmill\_vs\_outdoor\_stats(session, athlete\_id, lookback\_days)  
 print("\n🏠 vs 🌳:")  
 for k, v in stats.items():  
 print(f"- {k}: {v:.2f} %")  
  
 case "weekday\_pattern":  
 days = ActivityStatsDAO.get\_runs\_by\_weekday(session, athlete\_id, lookback\_days)  
 print("\n🗓️ Runs by weekday:")  
 for k, v in days.items():  
 print(f"- {k}: {v}")  
  
 case "time\_of\_day":  
 times = ActivityStatsDAO.get\_time\_of\_day\_stats(session, athlete\_id, lookback\_days)  
 print("\n🌞 vs 🌙:")  
 for k, v in times.items():  
 print(f"- {k}: {v}")  
  
 case "trend\_metric":  
 if not metric:  
 print("❌ Please provide --metric for trend\_metric mode")  
 return  
 trend = ActivityStatsDAO.get\_trend\_metrics(session, athlete\_id, metric)  
 print(f"\n📈 {metric} trend:")  
 for t in trend:  
 print(f"- {t['period']}: {t['value']:.2f}")  
  
 case \_:  
 print(f"❌ Unknown mode: {mode}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 parser = argparse.ArgumentParser()  
 parser.add\_argument("--athlete\_id", type=int, required=True)  
 parser.add\_argument("--lookback\_days", type=int, default=30)  
 parser.add\_argument("--mode", type=str, required=True)  
 parser.add\_argument("--metric", type=str, help="Used for trend\_metric mode")  
 args = parser.parse\_args()  
 main(args.athlete\_id, args.lookback\_days, args.mode, args.metric)

### 📄 src\scripts\debug\_token.py

# scripts/debug\_token.py  
  
import os  
from dotenv import load\_dotenv  
from src.db.db\_session import get\_session  
from src.db.dao.token\_dao import get\_tokens\_sa  
  
load\_dotenv(".env.prod")  
session = get\_session()  
token = get\_tokens\_sa(session, 347085)  
print("🔐 Token from DB:", token)  
session.close()

### 📄 src\scripts\main\_pipeline.py

import argparse  
import logging  
import sys  
import os  
from datetime import datetime  
  
from dotenv import load\_dotenv  
load\_dotenv()  
import src.utils.config as config  
  
from src.db.db\_session import get\_session  
from src.services.ingestion\_orchestrator\_service import (  
 run\_full\_ingestion\_and\_enrichment,  
 ingest\_specific\_activity,  
 ingest\_between\_dates,  
)  
from src.services.token\_service import refresh\_token\_if\_expired  
from src.db.dao.athlete\_dao import get\_all\_athletes  
from src.db.dao.token\_dao import get\_tokens\_sa  
from src.scripts import oauth\_cli  
  
logging.basicConfig(level=logging.INFO)  
logger = logging.getLogger(\_\_name\_\_)  
  
def parse\_date(date\_str):  
 if not date\_str:  
 return None  
 try:  
 return datetime.fromisoformat(date\_str)  
 except ValueError:  
 raise argparse.ArgumentTypeError(f"Invalid date format: {date\_str}. Use YYYY-MM-DD.")  
  
def run\_for\_athlete(session, athlete\_id, args):  
 try:  
 tokens = get\_tokens\_sa(session, athlete\_id)  
 except Exception:  
 tokens = None  
  
 if not tokens:  
 logger.info(f"🔐 No token found for athlete {athlete\_id}. Launching OAuth flow...")  
 oauth\_cli.main(athlete\_id\_override=athlete\_id)  
  
 if args.activity\_id:  
 ingest\_specific\_activity(session, athlete\_id, args.activity\_id)  
 elif args.start\_date and args.end\_date:  
 ingest\_between\_dates(  
 session,  
 athlete\_id,  
 args.start\_date,  
 args.end\_date,  
 batch\_size=args.batch\_size,  
 max\_activities=args.max\_activities,  
 per\_page=args.per\_page  
 )  
 elif args.start\_date or args.end\_date:  
 raise ValueError("Both --start\_date and --end\_date must be provided together.")  
 else:  
 refresh\_token\_if\_expired(session, athlete\_id)  
 run\_full\_ingestion\_and\_enrichment(  
 session,  
 athlete\_id,  
 lookback\_days=args.lookback\_days,  
 batch\_size=args.batch\_size,  
 max\_activities=args.max\_activities,  
 per\_page=args.per\_page  
 )  
  
def main():  
 print("⚙️ FLASK\_ENV =", os.getenv("FLASK\_ENV"))  
 print("⚙️ config.DATABASE\_URL =", config.DATABASE\_URL)  
 parser = argparse.ArgumentParser(description="Orchestrate sync + enrichment")  
  
 group = parser.add\_mutually\_exclusive\_group(required=True)  
 group.add\_argument("--athlete\_id", type=int, help="Run for one athlete")  
 group.add\_argument("--all", action="store\_true", help="Run for all athletes")  
  
 parser.add\_argument("--lookback\_days", type=int, default=None, help="Lookback window in days. If omitted, pulls latest N activities by --max\_activities.")  
 parser.add\_argument("--max\_activities", type=int, default=10, help="Maximum number of activities to ingest")  
 parser.add\_argument("--batch\_size", type=int, default=10, help="Number of activities to enrich per batch")  
 parser.add\_argument("--activity\_id", type=int, help="Specific activity ID to sync")  
 parser.add\_argument("--start\_date", type=parse\_date, help="Start date YYYY-MM-DD")  
 parser.add\_argument("--end\_date", type=parse\_date, help="End date YYYY-MM-DD")  
 parser.add\_argument("--per\_page", type=int, default=200, help="Number of results per API page")  
  
 args = parser.parse\_args()  
 session = get\_session()  
  
 try:  
 if args.all:  
 athletes = get\_all\_athletes(session)  
 if not athletes:  
 logger.warning("No athletes found.")  
 return  
 for athlete in athletes:  
 try:  
 logger.info(f"🔄 Syncing athlete {athlete.strava\_athlete\_id}")  
 run\_for\_athlete(session, athlete.strava\_athlete\_id, args)  
 session.commit()  
 except Exception as e:  
 logger.exception(f"❌ Error for athlete {athlete.strava\_athlete\_id}: {e}")  
 session.rollback()  
 finally:  
 session.expire\_all()  
 else:  
 run\_for\_athlete(session, args.athlete\_id, args)  
 session.commit()  
 except Exception as e:  
 logger.exception(f"❌ Pipeline failed: {e}")  
 sys.exit(1)  
 finally:  
 session.close()  
  
 sys.exit(0)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

### 📄 src\scripts\manual\_verify.py

import os  
import sys  
from pathlib import Path  
from dotenv import load\_dotenv  
  
# ✅ Adjust path to include `app/` where `src/` resides  
current\_file = Path(\_\_file\_\_).resolve()  
src\_root = current\_file.parents[2] # .../app/  
sys.path.insert(0, str(src\_root))  
  
# ✅ Load environment  
load\_dotenv(dotenv\_path=src\_root / ".env", override=True)  
  
# ✅ Confirm environment loaded  
print(f"[DEBUG] DATABASE\_URL: {os.getenv('DATABASE\_URL')}")  
  
# ✅ Import logic  
from src.db.db\_session import get\_session  
from src.services.ingestion\_orchestrator\_service import run\_full\_ingestion\_and\_enrichment  
  
# ✅ Main entry  
if \_\_name\_\_ == "\_\_main\_\_":  
 athlete\_id = int(input("Enter athlete\_id: "))  
 session = get\_session()  
 result = run\_full\_ingestion\_and\_enrichment(session, athlete\_id)  
 print(result)  
 session.close()

### 📄 src\scripts\oauth\_cli.py

import os  
import time  
import webbrowser  
import requests  
from urllib.parse import urlencode  
from datetime import datetime  
  
from dotenv import load\_dotenv  
load\_dotenv()   
  
from src.db.db\_session import get\_session  
from src.db.models.tokens import Token  
from src.db.dao.token\_dao import get\_tokens\_sa  
  
# Load from environment  
CLIENT\_ID = os.getenv("STRAVA\_CLIENT\_ID")  
CLIENT\_SECRET = os.getenv("STRAVA\_CLIENT\_SECRET")  
REDIRECT\_URI = os.getenv("STRAVA\_REDIRECT\_URI")  
AUTH\_URL = "https://www.strava.com/oauth/authorize"  
STATUS\_URL = os.getenv("STRAVA\_TOKEN\_STATUS\_URL") # Must point to `/auth/token\_status?athlete\_id=...`  
  
def generate\_auth\_url():  
 params = {  
 "client\_id": CLIENT\_ID,  
 "response\_type": "code",  
 "redirect\_uri": REDIRECT\_URI,  
 "approval\_prompt": "force",  
 "scope": "activity:read\_all,profile:read\_all"  
 }  
 return f"{AUTH\_URL}?{urlencode(params)}"  
  
  
def wait\_for\_token(athlete\_id, timeout=120, poll\_interval=5):  
 print(f"⏳ Waiting for token in DB for athlete {athlete\_id}...")  
 session = get\_session()  
 start\_time = time.time()  
 try:  
 while time.time() - start\_time < timeout:  
 tokens = get\_tokens\_sa(session, athlete\_id)  
 if tokens:  
 print("✅ Token found in database.")  
 return True  
 time.sleep(poll\_interval)  
 finally:  
 session.close()  
 raise TimeoutError(f"❌ Timed out waiting for token for athlete {athlete\_id}")  
  
  
  
def authorize\_and\_wait(athlete\_id):  
 print(f"➡️ Opening browser for Strava authorization (athlete\_id={athlete\_id})...")  
 webbrowser.open(generate\_auth\_url())  
  
 if not wait\_for\_token(athlete\_id):  
 raise RuntimeError("❌ Token not received after authorization.")  
  
 print(f"🎯 OAuth flow completed for athlete {athlete\_id}")  
  
# Reusable entrypoint for programmatic use  
def main(athlete\_id\_override=None):  
 athlete\_id = athlete\_id\_override or int(input("Enter athlete\_id: "))  
 authorize\_and\_wait(athlete\_id)  
  
# Only used for manual testing  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

### 📄 src\scripts\scheduled\_ingest.py

from src.db.db\_session import get\_session  
from src.services.ingestion\_orchestrator\_service import run\_full\_ingestion\_and\_enrichment  
from src.services.token\_service import refresh\_access\_token, refresh\_token\_if\_expired  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 session = get\_session()  
 athlete\_id = 347085  
  
 # 🔁 Force refresh (ensures valid token)  
 refresh\_access\_token(session, athlete\_id)  
 session.commit()  
 session.expire\_all()  
  
 # ✅ Optional: double-check expiry fallback  
 refresh\_token\_if\_expired(session, athlete\_id)  
  
 run\_full\_ingestion\_and\_enrichment(session, athlete\_id, lookback\_days=1)  
  
 session.close()

### 📄 src\services\\_\_init\_\_.py

### 📄 src\services\activity\_service.py

"""  
Activity service for ingestion and enrichment logic.  
"""  
  
import time  
import json  
import logging  
import os  
from datetime import datetime, timedelta  
from sqlalchemy import text  
  
from src.services.token\_service import get\_valid\_token  
from src.db.dao.split\_dao import upsert\_splits  
from src.db.dao.activity\_dao import ActivityDAO  
from src.services.strava\_access\_service import StravaClient  
from src.utils.logger import get\_logger  
from src.utils.conversions import convert\_metrics  
from src.db.models.activities import Activity  
  
log = get\_logger(\_\_name\_\_)  
log.setLevel(logging.INFO)  
  
def log\_strava\_payload(activity\_id, activity\_json, zones\_data, streams):  
 """  
 Write debug payload to file.  
 """  
 try:  
 os.makedirs("debug\_dumps", exist\_ok=True)  
 with open(  
 f"debug\_dumps/strava\_debug\_{activity\_id}.json",  
 "w",  
 encoding="utf-8"  
 ) as f:  
 json.dump({  
 "activity": activity\_json,  
 "zones": zones\_data,  
 "streams": streams  
 }, f, indent=2)  
 except Exception as e: # pylint: disable=broad-exception-caught  
 log.warning("⚠️ Could not write debug payload for %s: %s", activity\_id, e)  
  
def get\_activities\_to\_enrich(session, athlete\_id, limit):  
 """  
 Get recent unenriched activities.  
 """  
 result = session.execute(  
 text("""  
 SELECT activity\_id FROM activities  
 WHERE athlete\_id = :athlete\_id AND type = 'Run'  
 ORDER BY start\_date DESC  
 LIMIT :limit  
 """),  
 {"athlete\_id": athlete\_id, "limit": limit}  
 )  
 return [row.activity\_id for row in result.fetchall()]  
  
def enrich\_one\_activity(session, access\_token, activity\_id):  
 """  
 Enrich a single activity with streams, splits, zones.  
 """  
 try:  
 client = StravaClient(access\_token)  
 retries = 3  
 required\_fields = ["distance", "moving\_time", "average\_speed", "name"]  
 soft\_fields = ["average\_heartrate", "suffer\_score", "max\_speed", "calories"]  
  
 for attempt in range(retries):  
 activity\_json = client.get\_activity(activity\_id)  
 zones\_data = client.get\_hr\_zones(activity\_id)  
 streams = client.get\_streams(  
 activity\_id,  
 keys=["distance", "time", "velocity\_smooth", "heartrate"]  
 )  
  
 if all(activity\_json.get(field) for field in required\_fields):  
 break  
  
 log.warning(  
 "⚠️ Missing required fields for activity %s, retry %d/%d...",  
 activity\_id, attempt + 1, retries  
 )  
 time.sleep(1)  
 else:  
 raise ValueError(  
 f"❌ Critical data missing after retries for activity {activity\_id}: "  
 f"{[(field, activity\_json.get(field)) for field in required\_fields]}"  
 )  
  
 log\_strava\_payload(activity\_id, activity\_json, zones\_data, streams)  
  
 missing\_soft = [f for f in soft\_fields if activity\_json.get(f) is None]  
 if missing\_soft:  
 log.warning(  
 "⚠️ Partial enrichment for activity %s — missing: %s",  
 activity\_id, missing\_soft  
 )  
  
 log.info(  
 "➡️ Enriching activity %s — %s",  
 activity\_id, activity\_json.get("name")  
 )  
  
 hr\_zone\_pcts = extract\_hr\_zone\_percentages(zones\_data) or [0.0] \* 5  
 update\_activity\_enrichment(session, activity\_id, activity\_json, hr\_zone\_pcts)  
  
 splits = build\_mile\_splits(activity\_id, streams)  
 if splits:  
 upsert\_splits(session, splits)  
 log.info("✅ Synced %d splits for activity %s", len(splits), activity\_id)  
  
 return True  
 except Exception as e: # pylint: disable=broad-exception-caught  
 log.error("🔥 Exception while enriching %s: %s", activity\_id, e)  
 raise  
  
def enrich\_one\_activity\_with\_refresh(session, athlete\_id, activity\_id, max\_retries=2):  
 """  
 Attempt enrichment with token refresh and retries.  
 """  
 for attempt in range(1, max\_retries + 1):  
 try:  
 access\_token = get\_valid\_token(session, athlete\_id)  
 enrich\_one\_activity(session, access\_token, activity\_id)  
 session.expire\_all()  
  
 enriched = session.query(Activity).filter(  
 Activity.activity\_id == activity\_id,  
 Activity.average\_speed.isnot(None),  
 Activity.suffer\_score.isnot(None),  
 Activity.average\_heartrate.isnot(None),  
 Activity.max\_speed.isnot(None),  
 Activity.calories.isnot(None)  
 ).first()  
  
 if enriched:  
 log.info(  
 "✅ Enrichment succeeded on attempt %d for activity %s",  
 attempt, activity\_id  
 )  
 return True  
  
 log.warning(  
 "⚠️ Enrichment fields missing on attempt %d for %s. Retrying in 5s...",  
 attempt, activity\_id  
 )  
 time.sleep(1)  
  
 except Exception as e: # pylint: disable=broad-exception-caught  
 log.error(  
 "🔥 Enrichment error on attempt %d for %s: %s",  
 attempt, activity\_id, e  
 )  
 time.sleep(1)  
  
 log.error("❌ All retries failed — Activity %s has incomplete enrichment.", activity\_id)  
 raise RuntimeError(f"Enrichment failed for activity {activity\_id}")  
  
  
def update\_activity\_enrichment(session, activity\_id, activity\_json, hr\_zone\_pcts):  
 """  
 Update enriched fields on activity.  
 """  
 conv = convert\_metrics({  
 "distance": activity\_json.get("distance"),  
 "elevation": activity\_json.get("total\_elevation\_gain"),  
 "average\_speed": activity\_json.get("average\_speed"),  
 "max\_speed": activity\_json.get("max\_speed"),  
 "moving\_time": activity\_json.get("moving\_time"),  
 "elapsed\_time": activity\_json.get("elapsed\_time")  
 }, ["distance", "elevation", "average\_speed", "max\_speed", "moving\_time", "elapsed\_time"])  
  
 for key in ["average\_heartrate", "max\_speed", "suffer\_score", "calories"]:  
 if activity\_json.get(key) is None:  
 log.warning("⚠️ %s missing from activity %s", key, activity\_id)  
  
 params = {  
 "activity\_id": activity\_id,  
 "name": activity\_json.get("name"),  
 "distance": activity\_json.get("distance"),  
 "moving\_time": activity\_json.get("moving\_time"),  
 "elapsed\_time": activity\_json.get("elapsed\_time"),  
 "elevation": activity\_json.get("total\_elevation\_gain"),  
 "type": activity\_json.get("type"),  
 "avg\_speed": activity\_json.get("average\_speed"),  
 "max\_speed": activity\_json.get("max\_speed"),  
 "suffer\_score": activity\_json.get("suffer\_score"),  
 "average\_heartrate": activity\_json.get("average\_heartrate"),  
 "max\_heartrate": activity\_json.get("max\_heartrate"),  
 "calories": activity\_json.get("calories"),  
 "hr\_zone\_1": hr\_zone\_pcts[0],  
 "hr\_zone\_2": hr\_zone\_pcts[1],  
 "hr\_zone\_3": hr\_zone\_pcts[2],  
 "hr\_zone\_4": hr\_zone\_pcts[3],  
 "hr\_zone\_5": hr\_zone\_pcts[4],  
 \*\*conv  
 }  
  
 session.execute(  
 text("""  
 UPDATE activities SET  
 name = :name,  
 distance = :distance,  
 moving\_time = :moving\_time,  
 elapsed\_time = :elapsed\_time,  
 total\_elevation\_gain = :elevation,  
 type = :type,  
 average\_speed = :avg\_speed,  
 max\_speed = :max\_speed,  
 suffer\_score = :suffer\_score,  
 average\_heartrate = :average\_heartrate,  
 max\_heartrate = :max\_heartrate,  
 calories = :calories,  
 conv\_distance = :conv\_distance,  
 conv\_elevation\_feet = :conv\_elevation\_feet,  
 conv\_avg\_speed = :conv\_avg\_speed,  
 conv\_max\_speed = :conv\_max\_speed,  
 conv\_moving\_time = :conv\_moving\_time,  
 conv\_elapsed\_time = :conv\_elapsed\_time,  
 hr\_zone\_1 = :hr\_zone\_1,  
 hr\_zone\_2 = :hr\_zone\_2,  
 hr\_zone\_3 = :hr\_zone\_3,  
 hr\_zone\_4 = :hr\_zone\_4,  
 hr\_zone\_5 = :hr\_zone\_5  
 WHERE activity\_id = :activity\_id  
 """),  
 params  
 )  
 session.commit()  
  
def extract\_hr\_zone\_percentages(zones\_data):  
 """  
 Compute HR zone percentages.  
 """  
 try:  
 for zone\_group in zones\_data:  
 if zone\_group.get("type") == "heartrate":  
 buckets = zone\_group.get("distribution\_buckets", [])  
 times = [b.get("time", 0.0) for b in buckets[:5]]  
 total\_time = sum(times)  
 if total\_time > 0:  
 return [round((t / total\_time) \* 100, 2) for t in times]  
 except Exception as e: # pylint: disable=broad-exception-caught  
 log.warning("⚠️ HR zone extraction failed: %s", e)  
 return [0.0] \* 5  
  
def build\_mile\_splits(activity\_id, streams):  
 """  
 Build mile splits from stream data.  
 """  
 distances = streams.get("distance", [])  
 times = streams.get("time", [])  
 paces = streams.get("velocity\_smooth", [])  
 hrs = streams.get("heartrate", [])  
  
 splits = []  
 mile\_threshold = 1609.344  
 mile\_index = 1  
 start\_index = 0  
 speed\_threshold = 0.5  
  
 for i, d in enumerate(distances):  
 if float(d) < mile\_index \* mile\_threshold - 1e-6:  
 continue  
  
 segment\_distance = float(d) - float(distances[start\_index])  
 elapsed\_time = float(times[i]) - float(times[start\_index])  
 moving\_time = sum(  
 float(times[j]) - float(times[j - 1])  
 for j in range(start\_index + 1, i + 1)  
 if j < len(paces) and float(paces[j]) > speed\_threshold  
 )  
  
 avg\_speed = sum(paces[start\_index:i + 1]) / (i + 1 - start\_index) if paces else 0  
 avg\_hr = sum(hrs[start\_index:i + 1]) / (i + 1 - start\_index) if hrs else None  
  
 segment\_distance = round(segment\_distance, 2)  
 avg\_speed = round(avg\_speed, 2)  
 max\_speed\_val = round(max(paces[start\_index:i + 1]), 2) if paces else None  
 avg\_hr = round(avg\_hr, 2) if avg\_hr else None  
  
 conv\_data = convert\_metrics({  
 "distance": segment\_distance,  
 "average\_speed": avg\_speed,  
 "moving\_time": moving\_time,  
 "elapsed\_time": elapsed\_time  
 }, ["distance", "average\_speed", "moving\_time", "elapsed\_time"])  
  
 splits.append({  
 "activity\_id": activity\_id,  
 "lap\_index": mile\_index,  
 "distance": segment\_distance,  
 "elapsed\_time": elapsed\_time,  
 "moving\_time": moving\_time,  
 "average\_speed": avg\_speed,  
 "max\_speed": max\_speed\_val,  
 "start\_index": start\_index,  
 "end\_index": i,  
 "split": mile\_index,  
 "average\_heartrate": avg\_hr,  
 "pace\_zone": None,  
...truncated...

### 📄 src\services\ingestion\_orchestrator\_service.py

import sys  
import os  
import time  
from pathlib import Path  
from datetime import datetime, timedelta  
from sqlalchemy import exists  
  
sys.path.append(str(Path(\_\_file\_\_).resolve().parents[2])) # adds project root  
  
from src.db.dao.activity\_dao import ActivityDAO  
from src.utils.logger import get\_logger  
from src.services.activity\_service import (  
 ActivityIngestionService,  
 enrich\_one\_activity\_with\_refresh,  
 run\_enrichment\_batch,  
)  
from src.db.dao.token\_dao import get\_tokens\_sa  
from src.services.token\_service import get\_valid\_token  
from src.db.models.activities import Activity  
from src.db.models.tokens import Token  
from src.utils.seeder import seed\_sample\_activity  
  
logger = get\_logger(\_\_name\_\_)  
  
def run\_full\_ingestion\_and\_enrichment(  
 session,  
 athlete\_id,  
 lookback\_days=None,  
 max\_activities=10,  
 batch\_size=10,  
 per\_page=200  
):  
 logger.info(f"[CRON SYNC] ✅ Sync job started at {datetime.utcnow().isoformat()}")  
 logger.info(f"🚀 Starting run\_full\_ingestion\_and\_enrichment for athlete {athlete\_id}")  
  
 tokens = get\_tokens\_sa(session, athlete\_id)  
 if not tokens:  
 logger.warning(f"⚠️ No tokens found for athlete {athlete\_id}. Attempting to seed from .env...")  
  
 access\_token = os.getenv("STRAVA\_ACCESS\_TOKEN")  
 refresh\_token = os.getenv("STRAVA\_REFRESH\_TOKEN")  
 expires\_at = int(os.getenv("STRAVA\_EXPIRES\_AT", time.time() + 3600))  
  
 if access\_token and refresh\_token:  
 token = Token(  
 athlete\_id=athlete\_id,  
 access\_token=access\_token,  
 refresh\_token=refresh\_token,  
 expires\_at=expires\_at,  
 )  
 session.merge(token)  
 session.commit()  
 logger.info(f"✅ Seeded Strava token from .env for athlete {athlete\_id}")  
 else:  
 logger.warning("⚠️ .env credentials not found. Using fallback seeding for mock activity")  
 seed\_sample\_activity(session, athlete\_id)  
 session.commit()  
 logger.info(f"✅ Seeded mock activity for athlete {athlete\_id}")  
 return {"synced": 1, "enriched": 0}  
  
 access\_token = get\_valid\_token(session, athlete\_id)  
 logger.info(f"🟢 Retrieved valid access token for athlete {athlete\_id}")  
  
 logger.info(f"🗖️ Fetching recent activities from Strava...")  
 service = ActivityIngestionService(session, athlete\_id)  
  
 # ✅ Apply lookback\_days dynamically if provided  
 after\_ts = None  
 if lookback\_days:  
 after\_ts = int((datetime.utcnow() - timedelta(days=lookback\_days)).timestamp())  
  
 all\_fetched = service.client.get\_activities(  
 after=after\_ts,  
 per\_page=per\_page,  
 limit=max\_activities  
 )  
  
 all\_fetched = [a for a in all\_fetched if a.get("type") == "Run"]  
  
 if not all\_fetched:  
 logger.info("📬 No activities returned from Strava.")  
 return {"synced": 0, "enriched": 0}  
  
 fetched\_ids = [a["id"] for a in all\_fetched]  
 existing\_ids = {  
 r[0]  
 for r in session.query(Activity.activity\_id).filter(Activity.activity\_id.in\_(fetched\_ids)).all()  
 }  
 new\_activities = [a for a in all\_fetched if a["id"] not in existing\_ids]  
  
 if not new\_activities:  
 logger.info("✅ All activities from Strava already exist in the database.")  
 return {"synced": 0, "enriched": 0}  
  
 logger.info(f"⬇️ Ingesting {len(new\_activities)} new activities...")  
 ActivityDAO.upsert\_activities(session, athlete\_id, new\_activities)  
 logger.info(f"✅ Synced {len(new\_activities)} activities")  
  
 enriched = run\_enrichment\_batch(session, athlete\_id, batch\_size=batch\_size)  
 logger.info(f"✅ Enriched {enriched} activities")  
  
 logger.info(f"🎯 Ingestion + enrichment complete for athlete {athlete\_id}")  
 return {"synced": len(new\_activities), "enriched": enriched}  
  
def ingest\_specific\_activity(session, athlete\_id, activity\_id):  
 logger.info(f"⏳ Ingesting specific activity {activity\_id} for athlete {athlete\_id}")  
 service = ActivityIngestionService(session, athlete\_id)  
 activity\_data = service.client.get\_activity(activity\_id)  
 if not activity\_data:  
 logger.warning(f"Activity {activity\_id} not found for athlete {athlete\_id}")  
 return 0  
  
 ActivityDAO.upsert\_activities(session, athlete\_id, [activity\_data])  
 logger.info(f"✅ Activity {activity\_id} upserted")  
  
 try:  
 enrich\_one\_activity\_with\_refresh(session, athlete\_id, activity\_id)  
 logger.info(f"✅ Activity {activity\_id} enriched")  
 except Exception as e:  
 logger.error(f"❌ Skipping enrichment for activity {activity\_id} due to error: {e}")  
  
 return 1  
  
def ingest\_between\_dates(session, athlete\_id, start\_date: datetime, end\_date: datetime, batch\_size=10, max\_activities=None, per\_page=200):  
 logger.info(f"⏳ Ingesting activities for athlete {athlete\_id} between {start\_date} and {end\_date}")  
 service = ActivityIngestionService(session, athlete\_id)  
 activities = service.client.get\_activities(  
 after=int(start\_date.timestamp()),  
 before=int(end\_date.timestamp()),  
 per\_page=per\_page,  
 limit=max\_activities  
 )  
  
 activities = [a for a in activities if a.get("type") == "Run"]  
  
 if not activities:  
 logger.warning(f"No Run activities found between dates for athlete {athlete\_id}")  
 return 0  
  
 ActivityDAO.upsert\_activities(session, athlete\_id, activities)  
 logger.info(f"✅ Upserted {len(activities)} activities")  
  
 count = 0  
 for act in activities:  
 try:  
 enrich\_one\_activity\_with\_refresh(session, athlete\_id, act["id"])  
 count += 1  
 if count % batch\_size == 0:  
 logger.info(f"Processed {count} activities for enrichment")  
 except Exception:  
 continue  
  
 logger.info(f"✅ Enriched {count} activities")  
 return count  
  
def ingest\_today(session, athlete\_id):  
 today = datetime.utcnow()  
 start = datetime(today.year, today.month, today.day)  
 end = start + timedelta(days=1)  
 return ingest\_between\_dates(session, athlete\_id, start\_date=start, end\_date=end)

### 📄 src\services\strava\_access\_service.py

import requests  
import time  
from src.utils.config import STRAVA\_API\_BASE\_URL  
  
  
class StravaClient:  
 def \_\_init\_\_(self, access\_token):  
 self.access\_token = access\_token  
  
 def \_request\_with\_backoff(self, method, url, \*\*kwargs):  
 max\_retries = 5  
 backoff = 10 # Start with 10 sec backoff  
  
 headers = {"Authorization": f"Bearer {self.access\_token}"}  
  
 print(f"📤 Strava Request: {method} {url}")  
 print(f"📤 Headers: {headers}")  
 if "params" in kwargs:  
 print(f"📤 Params: {kwargs['params']}")  
  
 for attempt in range(max\_retries):  
 response = requests.request(  
 method,  
 url,  
 headers=headers,  
 \*\*kwargs  
 )  
  
 if response.status\_code == 429:  
 print(f"⚠️ Rate limit hit (429). Backing off {backoff} seconds...")  
 time.sleep(backoff)  
 backoff \*= 2  
 continue  
  
 if response.status\_code == 401:  
 print(f"❌ Unauthorized! Token: {self.access\_token}")  
  
 response.raise\_for\_status()  
 return response.json()  
  
 raise RuntimeError("Exceeded max retries due to repeated 429 errors")  
  
  
 def get\_activities(self, after=None, before=None, limit=None, per\_page=200):  
 url = f"{STRAVA\_API\_BASE\_URL}/athlete/activities"  
 all\_activities = []  
 page = 1  
  
 while True:  
 params = {  
 "page": page,  
 "per\_page": per\_page  
 }  
 if after:  
 params["after"] = after  
 if before:  
 params["before"] = before  
  
 batch = self.\_request\_with\_backoff("GET", url, params=params)  
  
 if not batch:  
 break  
  
 all\_activities.extend(batch)  
  
 if limit and len(all\_activities) >= limit:  
 return all\_activities[:limit]  
  
 page += 1  
  
 return all\_activities  
  
 def get\_activity(self, activity\_id):  
 url = f"{STRAVA\_API\_BASE\_URL}/activities/{activity\_id}"  
 return self.\_request\_with\_backoff("GET", url)  
  
 def get\_hr\_zones(self, activity\_id):  
 url = f"{STRAVA\_API\_BASE\_URL}/activities/{activity\_id}/zones"  
 try:  
 return self.\_request\_with\_backoff("GET", url)  
 except requests.exceptions.HTTPError as e:  
 if e.response.status\_code == 404:  
 return None  
 raise  
  
 def get\_splits(self, activity\_id):  
 url = f"{STRAVA\_API\_BASE\_URL}/activities/{activity\_id}/laps"  
 try:  
 return self.\_request\_with\_backoff("GET", url)  
 except requests.exceptions.HTTPError as e:  
 if e.response.status\_code == 404:  
 return []  
 raise  
  
 def get\_streams(self, activity\_id, keys):  
 url = f"{STRAVA\_API\_BASE\_URL}/activities/{activity\_id}/streams"  
 resp = self.\_request\_with\_backoff("GET", url, params={"keys": ",".join(keys), "key\_by\_type": "true"})  
  
 streams = {}  
 for key in keys:  
 raw = resp.get(key)  
 if isinstance(raw, dict) and "data" in raw:  
 try:  
 streams[key] = [  
 float(x) for x in raw["data"]  
 if isinstance(x, (int, float, str)) and str(x).replace('.', '', 1).isdigit()  
 ]  
 except Exception as e:  
 print(f"Failed to convert stream {key}: {e}")  
 streams[key] = []  
 else:  
 streams[key] = []  
 return streams

### 📄 src\services\token\_service.py

import logging  
import requests  
from datetime import datetime  
import jwt  
  
import src.utils.config as config  
from src.db.db\_session import get\_session as db\_get\_session  
from src.db.dao.token\_dao import get\_tokens\_sa, insert\_token\_sa  
from src.db.dao.athlete\_dao import insert\_athlete  
from src.db.models.tokens import Token  
  
  
logger = logging.getLogger(\_\_name\_\_)  
  
def get\_session():  
 return db\_get\_session()  
  
  
def is\_expired(expires\_at):  
 return expires\_at <= int(datetime.utcnow().timestamp())  
  
  
def get\_valid\_token(session, athlete\_id):  
 token\_data = get\_tokens\_sa(session, athlete\_id)  
 if not token\_data:  
 raise RuntimeError(f"No tokens found for athlete {athlete\_id}")  
  
 if is\_expired(token\_data["expires\_at"]):  
 return refresh\_access\_token(session, athlete\_id)["access\_token"]  
  
 return token\_data["access\_token"]  
  
  
def refresh\_access\_token(session, athlete\_id):  
 token\_data = get\_tokens\_sa(session, athlete\_id)  
 if not token\_data:  
 raise RuntimeError(f"No refresh token available for athlete {athlete\_id}")  
  
 tokens = refresh\_token\_static(token\_data["refresh\_token"])  
 print("🔁 Refreshed token:", tokens, flush=True)  
 insert\_token\_sa(  
 session=session,  
 athlete\_id=athlete\_id,  
 access\_token=tokens["access\_token"],  
 refresh\_token=tokens["refresh\_token"],  
 expires\_at=tokens["expires\_at"]  
 )  
 return tokens  
  
  
def refresh\_token\_static(refresh\_token):  
 response = requests.post(  
 "https://www.strava.com/api/v3/oauth/token",  
 data={  
 "client\_id": config.STRAVA\_CLIENT\_ID,  
 "client\_secret": config.STRAVA\_CLIENT\_SECRET,  
 "grant\_type": "refresh\_token",  
 "refresh\_token": refresh\_token,  
 },  
 )  
 response.raise\_for\_status()  
 return response.json()  
  
  
def refresh\_token\_if\_expired(session, athlete\_id):  
 token = session.query(Token).filter\_by(athlete\_id=athlete\_id).first()  
 if not token:  
 raise ValueError(f"No token found for athlete ID {athlete\_id}")  
  
 now = datetime.utcnow().timestamp()  
 if token.expires\_at <= now:  
 refreshed = refresh\_token\_static(token.refresh\_token)  
 token.access\_token = refreshed["access\_token"]  
 token.refresh\_token = refreshed["refresh\_token"]  
 token.expires\_at = refreshed["expires\_at"]  
 session.commit()  
 return True  
 return False  
  
  
def delete\_athlete\_tokens(session, athlete\_id):  
 deleted = session.query(Token).filter\_by(athlete\_id=athlete\_id).delete()  
 session.commit()  
 return deleted  
  
  
def store\_tokens\_from\_callback(code, session):  
 print(f"🔁 Exchanging code for tokens: {code}", flush=True)  
 response = requests.post(  
 "https://www.strava.com/api/v3/oauth/token",  
 data={  
 "client\_id": config.STRAVA\_CLIENT\_ID,  
 "client\_secret": config.STRAVA\_CLIENT\_SECRET,  
 "code": code,  
 "grant\_type": "authorization\_code"  
 },  
 )  
 response.raise\_for\_status()  
 token\_data = response.json()  
  
 athlete = token\_data.get("athlete")  
 if not athlete or "id" not in athlete:  
 raise KeyError("❌ Strava callback response missing athlete ID")  
  
 athlete\_id = athlete["id"]  
  
 # ✅ Ensure athlete is inserted   
 insert\_athlete(  
 session=session,  
 strava\_athlete\_id=athlete\_id,  
 name=f"{athlete.get('firstname', '')} {athlete.get('lastname', '')}".strip(),  
 email=athlete.get("email")  
 )  
  
  
  
  
 insert\_token\_sa(  
 session=session,  
 athlete\_id=athlete\_id,  
 access\_token=token\_data["access\_token"],  
 refresh\_token=token\_data["refresh\_token"],  
 expires\_at=token\_data["expires\_at"]  
 )  
  
 print(f"✅ Token stored for athlete: {athlete\_id}", flush=True)  
 return athlete\_id  
  
  
def logout\_user(token):  
 print(f"[LOGOUT] Token logged out: {token}")  
  
  
def login\_user(data):  
 if data["username"] != config.ADMIN\_USER or data["password"] != config.ADMIN\_PASS:  
 raise PermissionError("Invalid credentials")  
  
 session = get\_session()  
 token\_payload = {"sub": "admin"}  
 access\_token = jwt.encode(token\_payload, config.SECRET\_KEY, algorithm="HS256")  
 refresh\_token = jwt.encode({"sub": "admin", "type": "refresh"}, config.JWT\_SECRET, algorithm="HS256")  
  
 insert\_token\_sa(  
 session=session,  
 athlete\_id=0,  
 access\_token=access\_token,  
 refresh\_token=refresh\_token,  
 expires\_at=int(datetime.utcnow().timestamp()) + 3600,  
 )  
  
 return access\_token, refresh\_token  
  
  
def refresh\_token(encoded\_refresh\_token):  
 try:  
 payload = jwt.decode(encoded\_refresh\_token, config.SECRET\_KEY, algorithms=["HS256"])  
 if payload.get("type") != "refresh":  
 raise PermissionError("Invalid token type")  
 except (jwt.ExpiredSignatureError, jwt.InvalidTokenError):  
 raise PermissionError("Invalid or expired refresh token")  
  
 session = get\_session()  
 token\_data = get\_tokens\_sa(session, config.ADMIN\_ATHLETE\_ID)  
 if not token\_data:  
 raise PermissionError("No refresh token found")  
  
 new\_tokens = refresh\_token\_static(token\_data["refresh\_token"])  
 insert\_token\_sa(  
 session=session,  
 athlete\_id=config.ADMIN\_ATHLETE\_ID,  
 access\_token=new\_tokens["access\_token"],  
 refresh\_token=new\_tokens["refresh\_token"],  
 expires\_at=new\_tokens["expires\_at"]  
 )  
 return new\_tokens["access\_token"]  
  
  
def exchange\_code\_for\_token(code):  
 response = requests.post(  
 "https://www.strava.com/api/v3/oauth/token",  
 data={  
 "client\_id": config.STRAVA\_CLIENT\_ID,  
 "client\_secret": config.STRAVA\_CLIENT\_SECRET,  
 "code": code,  
 "grant\_type": "authorization\_code"  
 },  
 )  
 response.raise\_for\_status()  
 return response.json()  
  
  
def get\_authorization\_url():  
 redirect\_uri = config.STRAVA\_REDIRECT\_URI  
 client\_id = config.STRAVA\_CLIENT\_ID  
 url = (  
 f"https://www.strava.com/oauth/authorize"  
 f"?client\_id={client\_id}"  
 f"&response\_type=code"  
 f"&redirect\_uri={redirect\_uri}"  
 f"&scope=read,activity:read\_all"  
 )  
 return url

### 📄 src\utils\config.py

import os  
  
# ❌ REMOVE this block entirely:  
# from dotenv import load\_dotenv  
# env\_mode = os.getenv("FLASK\_ENV") or os.getenv("RAILWAY\_ENVIRONMENT") or "development"  
# env\_file = ...  
# load\_dotenv(env\_file, override=True)  
  
# ✅ config.py should only read, never load  
# Assume env already set up by app.py  
  
# ----- OAuth / Strava -----  
STRAVA\_CLIENT\_ID = os.getenv("STRAVA\_CLIENT\_ID")  
STRAVA\_CLIENT\_SECRET = os.getenv("STRAVA\_CLIENT\_SECRET")  
STRAVA\_REDIRECT\_URI = os.getenv("STRAVA\_REDIRECT\_URI") or os.getenv("REDIRECT\_URI")  
STRAVA\_API\_BASE\_URL = "https://www.strava.com/api/v3"  
  
# ----- Token Expiry -----  
ACCESS\_TOKEN\_EXP = int(os.getenv("ACCESS\_TOKEN\_EXP", 900)) # 15 min  
REFRESH\_TOKEN\_EXP = int(os.getenv("REFRESH\_TOKEN\_EXP", 604800)) # 7 days  
  
# ----- JWT / Auth -----  
SECRET\_KEY = os.getenv("SECRET\_KEY", "dev")  
ADMIN\_USER = os.getenv("ADMIN\_USER")  
ADMIN\_PASS = os.getenv("ADMIN\_PASS")  
ADMIN\_ATHLETE\_ID = 0  
  
# ----- Database -----  
DATABASE\_URL = os.getenv("DATABASE\_URL")  
  
# ----- Internal API / Jobs -----  
CRON\_SECRET\_KEY = os.getenv("CRON\_SECRET\_KEY")  
INTERNAL\_API\_KEY = os.getenv("INTERNAL\_API\_KEY")  
  
# ----- Misc -----  
PORT = int(os.getenv("PORT", 5000))  
IS\_LOCAL = os.getenv("IS\_LOCAL", "false").lower() == "true"  
  
JWT\_SECRET = SECRET\_KEY

### 📄 src\utils\conversions.py

def meters\_to\_miles(meters):  
 return round(meters / 1609.344, 2) if meters is not None else None  
  
def meters\_to\_feet(meters):  
 return round(meters \* 3.28084, 1) if meters is not None else None  
  
def mps\_to\_min\_per\_mile(mps):  
 return round(26.8224 / mps, 2) if mps and mps > 0 else None  
  
def format\_seconds\_to\_hms(seconds):  
 if seconds is None:  
 return None  
 try:  
 seconds = int(seconds)  
 except (ValueError, TypeError):  
 return None  
 minutes, sec = divmod(seconds, 60)  
 hours, minutes = divmod(minutes, 60)  
 return f"{hours}:{minutes:02}:{sec:02}" if hours > 0 else f"{minutes}:{sec:02}"  
  
def safe\_float(val):  
 try:  
 return float(val)  
 except (ValueError, TypeError):  
 return None  
  
def safe\_int(val):  
 try:  
 return int(val)  
 except (ValueError, TypeError):  
 return None  
  
def convert\_metrics(data: dict, fields: list[str]) -> dict:  
 conversions = {}  
 if "distance" in fields:  
 conversions["conv\_distance"] = meters\_to\_miles(safe\_float(data.get("distance")))  
 if "elevation" in fields:  
 conversions["conv\_elevation\_feet"] = meters\_to\_feet(safe\_float(data.get("elevation")))  
 if "average\_speed" in fields:  
 conversions["conv\_avg\_speed"] = mps\_to\_min\_per\_mile(safe\_float(data.get("average\_speed")))  
 if "max\_speed" in fields:  
 conversions["conv\_max\_speed"] = mps\_to\_min\_per\_mile(safe\_float(data.get("max\_speed")))  
 if "moving\_time" in fields:  
 conversions["conv\_moving\_time"] = format\_seconds\_to\_hms(safe\_int(data.get("moving\_time")))  
 if "elapsed\_time" in fields:  
 conversions["conv\_elapsed\_time"] = format\_seconds\_to\_hms(safe\_int(data.get("elapsed\_time")))  
 return conversions

### 📄 src\utils\db\_check.py

# db\_check.py  
  
import os  
from sqlalchemy import create\_engine, text  
from dotenv import load\_dotenv  
  
# Load environment variables  
load\_dotenv()  
  
DATABASE\_URL = os.getenv("DATABASE\_URL")  
print("📡 Testing DATABASE\_URL:", DATABASE\_URL)  
  
# Create engine and attempt connection  
engine = create\_engine(DATABASE\_URL)  
  
try:  
 with engine.connect() as conn:  
 result = conn.execute(text("SELECT 1")) # ✅ Use text() for SQLAlchemy 2.0+  
 print("✅ DB connection succeeded:", result.fetchone())  
except Exception as e:  
 print("❌ DB connection failed:", str(e))

### 📄 src\utils\generate\_gpt\_handoff\_summary.py

from docx import Document  
from docx.shared import Pt  
  
OUTPUT\_DOCX = "SmartCoach\_GPT\_Handoff\_Summary\_CLEAN.docx"  
  
def add\_heading(doc, text, level):  
 doc.add\_heading(text, level=level)  
  
def add\_paragraph(doc, text):  
 doc.add\_paragraph(text)  
  
def generate\_gpt\_handoff\_summary():  
 doc = Document()  
  
 add\_heading(doc, "✅ SmartCoach – GPT Handoff Summary", level=1)  
 add\_paragraph(doc, "This document was generated to bootstrap a Custom GPT session with complete project context.")  
  
 # Last Known Good State  
 add\_heading(doc, "🏁 Last known good state:", level=2)  
 add\_paragraph(doc, "• Tag: v1.0.0")  
 add\_paragraph(doc, "• Deployment: Railway (web-production-c4329.up.railway.app)")  
 add\_paragraph(doc, "• OAuth tested: ✅")  
 add\_paragraph(doc, "• DB Connected: ✅")  
  
 # Environment  
 add\_heading(doc, "🔐 Environment Settings:", level=2)  
 add\_paragraph(doc, "• ADMIN\_USER=admin")  
 add\_paragraph(doc, "• REDIRECT\_URI=https://web-production-c4329.up.railway.app/oauth/callback")  
 add\_paragraph(doc, "• DATABASE\_URL=(set in Railway)")  
 add\_paragraph(doc, "• SECRET\_KEY, INTERNAL\_API\_KEY, CRON\_SECRET\_KEY defined")  
  
 # Architecture  
 add\_heading(doc, "🔧 Architecture Notes:", level=2)  
 add\_paragraph(doc, "• Flask API deployed on Railway")  
 add\_paragraph(doc, "• PostgreSQL managed via Railway plugin")  
 add\_paragraph(doc, "• JWT-based authentication and user sessions")  
 add\_paragraph(doc, "• Strava OAuth 2.0 integration and webhook registration planned")  
  
 # Coaching Intelligence Objective  
 add\_heading(doc, "🎯 Coaching Intelligence Objectives:", level=2)  
 add\_paragraph(doc, "• Build personalized weekly training plans")  
 add\_paragraph(doc, "• Track Strava-recorded performance")  
 add\_paragraph(doc, "• Monitor deviations and adapt based on user behavior")  
 add\_paragraph(doc, "• Support natural conversation for advising and adjusting")  
  
 # System Components  
 add\_heading(doc, "🧱 System Components:", level=2)  
 add\_paragraph(doc, "• Custom GPT logic: planning, feedback, and conversation")  
 add\_paragraph(doc, "• PostgreSQL DB storing plan and activity data")  
 add\_paragraph(doc, "• Strava Sync service")  
 add\_paragraph(doc, "• Planned: Training plan generator, weekly comparator")  
  
 # Known Issues  
 add\_heading(doc, "⚠️ Known Issues:", level=2)  
 add\_paragraph(doc, "• None at last deploy. Strava activity fetch untested.")  
  
 # Tasks  
 add\_heading(doc, "📌 Next Tasks:", level=2)  
 add\_paragraph(doc, "1. Test full Strava flow: code -> token -> activity fetch")  
 add\_paragraph(doc, "2. Implement auto-sync")  
 add\_paragraph(doc, "3. Add athlete-to-user ID mapping")  
 add\_paragraph(doc, "4. Connect activity analysis")  
 add\_paragraph(doc, "5. Ship Phase 2 readiness")  
  
 # GPT Boot Prompt  
 add\_heading(doc, "🧠 GPT Boot Prompt", level=2)  
 add\_paragraph(doc, "Create a Custom GPT with the following context:")  
 add\_paragraph(doc, "- Name: SmartCoachDev")  
 add\_paragraph(doc, "- Role: Flask engineer + PM for Smart Marathon Coach API")  
 add\_paragraph(doc, "- Accesses: JWT auth, Strava OAuth, Railway deploy logs, pg DB")  
 add\_paragraph(doc, "- Code base starts in: /src, /templates, /scripts")  
 add\_paragraph(doc, "- Current live env: https://web-production-c4329.up.railway.app")  
 add\_paragraph(doc, "- Start by reviewing recent work, then continue with task #1: Test full Strava activity flow.")  
 add\_paragraph(doc, "- Reference: ‘Smart Coach - Project Reference.docx’ for architecture, history, and schema.")  
  
  
 from docx import Document as DocxDocument  
  
 def append\_docx\_content(target\_doc, source\_path):  
 source\_doc = DocxDocument(source\_path)  
 for element in source\_doc.element.body:  
 target\_doc.element.body.append(element)  
  
 # At the end of generate\_gpt\_handoff\_summary():  
 try:  
 append\_docx\_content(doc, "final\_project\_map.docx")  
 print("✅ Appended folder/code map from final\_project\_map.docx")  
 except Exception as e:  
 print(f"⚠️ Could not append code map: {e}")  
  
  
  
  
  
  
 doc.save(OUTPUT\_DOCX)  
 print(f"✅ Summary written to {OUTPUT\_DOCX}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 generate\_gpt\_handoff\_summary()

### 📄 src\utils\gpt\_ops.py

import openai  
import os  
from datetime import datetime  
  
openai.api\_key = os.getenv("OPENAI\_API\_KEY")  
  
  
def parse\_date\_safe(date\_str: str) -> datetime:  
 """  
 Parses a date string that may or may not include time.  
 Accepts formats like '2025-06-24' or '2025-06-24 13:45:00'  
 """  
 for fmt in ("%Y-%m-%d", "%Y-%m-%d %H:%M:%S"):  
 try:  
 return datetime.strptime(date\_str, fmt)  
 except ValueError:  
 continue  
 raise ValueError(f"Unsupported date format: {date\_str}")  
  
  
def format\_prompt(user\_question: str, activities: list[dict]) -> str:  
 """  
 Format a prompt string for GPT using the user's question and a list of activity records.  
 Always returns structured prompt for coaching assistant, even if empty input.  
 """  
 prompt = "You are a smart coaching assistant helping a runner improve.\n\n"  
 prompt += "ACTIVITIES:\n"  
  
 if not activities:  
 prompt += "[No activities available]\n"  
 else:  
 for i, a in enumerate(activities, start=1):  
 prompt += f"[{i}] date: {a['date']}, distance\_km: {a['distance\_km']}, duration\_min: {a['duration\_min']}\n"  
  
 prompt += "\nUSER QUESTION:\n"  
 prompt += user\_question.strip()  
  
 return prompt  
  
  
def get\_gpt\_response(prompt: str) -> str:  
 try:  
 response = openai.ChatCompletion.create(  
 model="gpt-4o",  
 messages=[  
 {"role": "system", "content": "You are a helpful fitness assistant."},  
 {"role": "user", "content": prompt}  
 ],  
 temperature=0.7,  
 max\_tokens=2000  
 )  
 return response['choices'][0]['message']['content'].strip()  
 except Exception as e:  
 import traceback  
 print("GPT API call failed:", e)  
 traceback.print\_exc()  
 return f"❌ GPT error: {e}"

### 📄 src\utils\jwt\_utils.py

import jwt  
from functools import wraps  
from flask import request, jsonify, current\_app  
  
import src.utils.config as config  
  
  
def require\_auth(f):  
 @wraps(f)  
 def decorated(\*args, \*\*kwargs):  
 # ✅ Internal service key override  
 internal\_key = request.headers.get("X-Internal-Key")  
  
 if internal\_key and config.INTERNAL\_API\_KEY and internal\_key == config.INTERNAL\_API\_KEY:  
 request.user = {  
 "user\_id": "internal",  
 "is\_internal": True # ✅ Enable admin privileges  
 }  
 return f(\*args, \*\*kwargs)  
  
 # 🔐 Fallback to regular Bearer token auth  
 auth\_header = request.headers.get("Authorization")  
 if not auth\_header or not auth\_header.lower().startswith("bearer "):  
 return jsonify({"error": "Authorization header missing"}), 401  
  
 token = auth\_header.split(" ")[1]  
 try:  
 payload = jwt.decode(token, config.SECRET\_KEY, algorithms=["HS256"])  
 user\_id = payload.get("sub")  
 if not user\_id:  
 return jsonify({"error": "Token missing subject (sub)"}), 401  
  
 request.user = {  
 "user\_id": user\_id,  
 "is\_internal": user\_id == "internal"  
 }  
 except jwt.ExpiredSignatureError:  
 return jsonify({"error": "Token expired"}), 401  
 except jwt.InvalidTokenError:  
 return jsonify({"error": "Invalid token"}), 401  
  
 return f(\*args, \*\*kwargs)  
 return decorated  
  
  
def decode\_token(token: str) -> dict:  
 """Decode JWT without expiration check (for internal inspection)."""  
 try:  
 return jwt.decode(token, config.SECRET\_KEY, algorithms=["HS256"], options={"verify\_exp": False})  
 except jwt.DecodeError:  
 raise ValueError("Invalid token format")

### 📄 src\utils\logger.py

import logging  
import sys  
  
def get\_logger(name=\_\_name\_\_):  
 logger = logging.getLogger(name)  
 logger.setLevel(logging.INFO)  
  
 if not logger.handlers:  
 # Console handler with UTF-8 support  
 console\_handler = logging.StreamHandler(sys.stdout)  
 console\_handler.setLevel(logging.INFO)  
 console\_formatter = logging.Formatter("%(asctime)s %(levelname)s %(message)s")  
 console\_handler.setFormatter(console\_formatter)  
 logger.addHandler(console\_handler)  
  
 # File handler with UTF-8 encoding  
 file\_handler = logging.FileHandler("activity\_ingestion.log", encoding="utf-8")  
 file\_handler.setLevel(logging.ERROR)  
 file\_formatter = logging.Formatter("%(asctime)s %(levelname)s %(message)s")  
 file\_handler.setFormatter(file\_formatter)  
 logger.addHandler(file\_handler)  
  
 return logger

### 📄 src\utils\map\_and\_extract\_Railway.py

import os  
from docx import Document  
from docx.shared import Pt, RGBColor  
from docx.oxml.ns import qn  
  
ROOT\_DIR = r"C:\Users\andre\projects\railway-pg-test"  
OUTPUT\_DOCX = "final\_project\_map.docx"  
  
EXCLUDE\_DIRS = {'venv', '\_\_pycache\_\_', '.git', '.mypy\_cache', '.pytest\_cache', 'node\_modules', 'dist', 'build', 'logs'}  
EXCLUDE\_SUFFIXES = {'.dist-info', '.egg-info'}  
EXCLUDE\_FILES = {'.env', '.env.test', '.env.prod', '.env.dev'}  
EXCLUDE\_EXTENSIONS = {'.json', '.html'} # Added .html here  
  
INCLUDE\_EXTENSIONS = {'.py', '.md', '.txt', '.css', '.js'} # Removed .html here  
  
MAX\_FILE\_SIZE\_BYTES = 50 \* 1024 # 50 KB  
MAX\_LINES\_PER\_FILE = 300  
  
def should\_exclude\_dir(name):  
 return name in EXCLUDE\_DIRS or any(name.endswith(suffix) for suffix in EXCLUDE\_SUFFIXES)  
  
def should\_include\_file(entry, full\_path):  
 # Exclude hidden files except allowed ones  
 if entry.startswith('.') and entry not in EXCLUDE\_FILES:  
 return False  
  
 ext = os.path.splitext(entry)[1].lower()  
 if ext in EXCLUDE\_EXTENSIONS:  
 return False  
 if ext not in INCLUDE\_EXTENSIONS:  
 return False  
 if entry in EXCLUDE\_FILES:  
 return False  
  
 try:  
 size = os.path.getsize(full\_path)  
 if size > MAX\_FILE\_SIZE\_BYTES:  
 return False  
 except Exception:  
 return False  
  
 return True  
  
def icon\_for(entry, is\_dir):  
 if is\_dir:  
 return "📁"  
 elif entry.endswith(".py"):  
 return "🐍"  
 else:  
 return "📄"  
  
def read\_file\_content(filepath):  
 try:  
 with open(filepath, "r", encoding="utf-8") as f:  
 lines = []  
 for i, line in enumerate(f):  
 if i >= MAX\_LINES\_PER\_FILE:  
 lines.append("...truncated...\n")  
 break  
 lines.append(line)  
 return lines  
 except Exception:  
 return None  
  
def add\_code\_block(doc, lines):  
 para = doc.add\_paragraph()  
 para.paragraph\_format.line\_spacing = 1.0  
 para.paragraph\_format.space\_before = Pt(0)  
 para.paragraph\_format.space\_after = Pt(0)  
  
 for line in lines:  
 run = para.add\_run(line if line.endswith('\n') else line + '\n')  
 run.font.name = 'Consolas'  
 run.font.size = Pt(9)  
 run.font.color.rgb = RGBColor(51, 51, 51)  
 run.\_element.rPr.rFonts.set(qn('w:eastAsia'), 'Consolas')  
  
def apply\_folder\_style(paragraph):  
 if not paragraph.runs:  
 return  
 run = paragraph.runs[0]  
 run.font.name = 'Consolas'  
 run.font.size = Pt(9)  
 run.font.color.rgb = RGBColor(51, 51, 51)  
 run.\_element.rPr.rFonts.set(qn('w:eastAsia'), 'Consolas')  
 paragraph.paragraph\_format.space\_before = Pt(0)  
 paragraph.paragraph\_format.space\_after = Pt(0)  
 paragraph.paragraph\_format.line\_spacing = 1.0  
  
def write\_project\_map(root\_dir, output\_docx):  
 doc = Document()  
 doc.add\_heading("📁 Folder & File Structure", level=1)  
  
 folder\_map\_lines = []  
 code\_blocks = []  
  
 def scan\_structure(path, level=0):  
 try:  
 entries = sorted(os.listdir(path))  
 except PermissionError:  
 return  
 for entry in entries:  
 full\_path = os.path.join(path, entry)  
 is\_dir = os.path.isdir(full\_path)  
  
 # Skip symlinks or non-regular files if needed:  
 if not is\_dir and not os.path.isfile(full\_path):  
 continue  
  
 # Exclude directories we don't want  
 if should\_exclude\_dir(entry):  
 continue  
   
 # Exclude files by extension from folder tree as well  
 ext = os.path.splitext(entry)[1].lower()  
 if not is\_dir and ext in EXCLUDE\_EXTENSIONS:  
 continue  
  
 indent = " " \* level  
 icon = icon\_for(entry, is\_dir)  
 line = f"{indent}{icon} {entry}/" if is\_dir else f"{indent}{icon} {entry}"  
 folder\_map\_lines.append(line)  
  
 if not is\_dir and should\_include\_file(entry, full\_path):  
 content = read\_file\_content(full\_path)  
 if content:  
 rel\_path = os.path.relpath(full\_path, root\_dir)  
 code\_blocks.append((rel\_path, content))  
  
 if is\_dir:  
 scan\_structure(full\_path, level + 1)  
  
 folder\_map\_lines.append(f"{os.path.basename(root\_dir)}/")  
 scan\_structure(root\_dir, level=1)  
  
 for line in folder\_map\_lines:  
 p = doc.add\_paragraph(line)  
 apply\_folder\_style(p)  
  
 doc.add\_paragraph() # spacer  
  
 doc.add\_heading("🧠 Code & Content", level=1)  
 for rel\_path, lines in code\_blocks:  
 doc.add\_paragraph(f"📄 {rel\_path}", style='Heading3')  
 add\_code\_block(doc, lines)  
  
 doc.save(output\_docx)  
 print(f"✅ Saved clean layout to: {output\_docx}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 write\_project\_map(ROOT\_DIR, OUTPUT\_DOCX)

### 📄 src\utils\sample\_data.py

# utils/sample\_data.py  
  
SAMPLE\_ACTIVITY\_JSON = {  
 "id": 99999,  
 "activity\_id": 99999,  
 "name": "Mock Run",  
 "type": "Run",  
 "distance": 5000.0,  
 "moving\_time": 1500,  
 "elapsed\_time": 1600,  
 "total\_elevation\_gain": 50.0,  
 "average\_speed": 3.5,  
 "max\_speed": 4.0,  
 "suffer\_score": 30,  
 "average\_heartrate": 150,  
 "max\_heartrate": 170,  
 "calories": 400,  
 "splits\_metric": [  
 {  
 "lap\_index": 1,  
 "distance": 1000,  
 "elapsed\_time": 300,  
 "moving\_time": 295,  
 "average\_speed": 3.33,  
 "max\_speed": 3.5,  
 "start\_index": 0,  
 "end\_index": 299,  
 "split": 1,  
 "average\_heartrate": 145,  
 "pace\_zone": 2  
 }  
 ]  
}  
  
SAMPLE\_HR\_ZONE\_RESPONSE = {  
 "heart\_rate": {  
 "custom\_zones": [  
 {"score": 0.1},  
 {"score": 0.2},  
 {"score": 0.3},  
 {"score": 0.25},  
 {"score": 0.15}  
 ]  
 }  
}  
  
  
SAMPLE\_STREAMS\_RESPONSE = {  
 "distance": {"data": [0.0, 500.0, 1000.0]},  
 "time": {"data": [0, 150, 300]},  
 "velocity\_smooth": {"data": [3.2, 3.4, 3.3]},  
 "heartrate": {"data": [130, 140, 150]}  
}

### 📄 src\utils\seeder.py

# src/utils/seeder.py  
  
from src.db.dao.activity\_dao import ActivityDAO  
from src.utils.sample\_data import SAMPLE\_ACTIVITY\_JSON  
  
  
def seed\_sample\_activity(session, athlete\_id: int):  
 mock = SAMPLE\_ACTIVITY\_JSON.copy()  
 mock["athlete\_id"] = athlete\_id  
 mock["id"] = mock["activity\_id"] = 99999 + athlete\_id # ensure unique PK  
 ActivityDAO.upsert\_activities(session, athlete\_id, [mock])

### 📄 static\assets\index-Dtn62Xmo.css

:root{font-family:system-ui,Avenir,Helvetica,Arial,sans-serif;line-height:1.5;font-weight:400;color-scheme:light dark;color:#ffffffde;background-color:#242424;font-synthesis:none;text-rendering:optimizeLegibility;-webkit-font-smoothing:antialiased;-moz-osx-font-smoothing:grayscale}a{font-weight:500;color:#646cff;text-decoration:inherit}a:hover{color:#535bf2}body{margin:0;display:flex;place-items:center;min-width:320px;min-height:100vh}h1{font-size:3.2em;line-height:1.1}button{border-radius:8px;border:1px solid transparent;padding:.6em 1.2em;font-size:1em;font-weight:500;font-family:inherit;background-color:#1a1a1a;cursor:pointer;transition:border-color .25s}button:hover{border-color:#646cff}button:focus,button:focus-visible{outline:4px auto -webkit-focus-ring-color}@media (prefers-color-scheme: light){:root{color:#213547;background-color:#fff}a:hover{color:#747bff}button{background-color:#f9f9f9}}

### 📄 test\_enrichment\_sync.py

from src.db.db\_session import get\_session  
from src.services.activity\_service import run\_enrichment\_batch  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 athlete\_id = 12345 # <-- replace with a valid athlete\_id from your database  
  
 session = get\_session()  
 try:  
 count = run\_enrichment\_batch(session, athlete\_id=athlete\_id)  
 print(f"✅ Enriched {count} activities")  
 finally:  
 session.close()

### 📄 tests\conftest.py

import os  
import sys  
import pytest  
from pathlib import Path  
from sqlalchemy.orm import sessionmaker  
from dotenv import load\_dotenv  
from unittest.mock import patch  
from datetime import datetime, timedelta  
  
  
  
# -------------------------  
# 🔧 Environment & Path Setup  
# -------------------------  
  
# Add project root to PYTHONPATH  
PROJECT\_ROOT = Path(\_\_file\_\_).resolve().parents[1]  
sys.path.insert(0, str(PROJECT\_ROOT))  
  
# Load test-specific environment variables  
load\_dotenv(dotenv\_path=PROJECT\_ROOT / ".env.test", override=True)  
  
# -------------------------  
# 🔌 Flask App Fixtures  
# -------------------------  
  
from src.app import create\_app  
from src.db.db\_session import get\_engine  
from src.db.models.tokens import Token  
from src.db.models.athletes import Athlete  
from src.db.models.activities import Activity  
from tests.test\_data.sample\_activities import SAMPLE\_ACTIVITY\_JSON  
  
  
@pytest.fixture(scope="session")  
def shared\_engine():  
 database\_url = os.getenv("DATABASE\_URL")  
 print(f"[TEST] Using DATABASE\_URL = {database\_url}")  
 return get\_engine(database\_url)  
  
  
@pytest.fixture(scope="function")  
def app(shared\_engine):  
 test\_config = {  
 "TESTING": True,  
 "DATABASE\_URL": os.getenv("DATABASE\_URL")  
 }  
 yield create\_app(test\_config)  
  
  
@pytest.fixture(scope="function")  
def client(app):  
 return app.test\_client()  
  
  
# -------------------------  
# 🧪 Database Fixtures  
# -------------------------  
  
@pytest.fixture(scope="function")  
def sqlalchemy\_session(shared\_engine):  
 connection = shared\_engine.connect()  
 transaction = connection.begin()  
 Session = sessionmaker(bind=connection, future=True)  
 session = Session()  
 yield session  
 session.close()  
 transaction.rollback()  
 connection.close()  
  
  
@pytest.fixture(scope="function")  
def test\_db\_session(sqlalchemy\_session):  
 return sqlalchemy\_session  
  
  
  
  
@pytest.fixture(scope="function")  
def seed\_test\_data(test\_db\_session):  
 if not test\_db\_session.query(Athlete).filter\_by(athlete\_id=1).first():  
 test\_db\_session.add(Athlete(athlete\_id=1, first\_name="Test", last\_name="Athlete"))  
  
 if not test\_db\_session.query(Token).filter\_by(athlete\_id=1).first():  
 test\_db\_session.add(Token(  
 athlete\_id=1,  
 access\_token="test\_access\_token",  
 refresh\_token="test\_refresh\_token",  
 expires\_at=int((datetime.utcnow() + timedelta(days=1)).timestamp())  
 ))  
  
 if not test\_db\_session.query(Activity).filter\_by(activity\_id=SAMPLE\_ACTIVITY\_JSON["activity\_id"]).first():  
 test\_db\_session.add(Activity(  
 activity\_id=SAMPLE\_ACTIVITY\_JSON["activity\_id"],  
 athlete\_id=1,  
 name=SAMPLE\_ACTIVITY\_JSON["name"],  
 type=SAMPLE\_ACTIVITY\_JSON["type"],  
 distance=SAMPLE\_ACTIVITY\_JSON["distance"],  
 moving\_time=SAMPLE\_ACTIVITY\_JSON["moving\_time"],  
 elapsed\_time=SAMPLE\_ACTIVITY\_JSON["elapsed\_time"],  
 total\_elevation\_gain=SAMPLE\_ACTIVITY\_JSON["total\_elevation\_gain"],  
 average\_speed=SAMPLE\_ACTIVITY\_JSON["average\_speed"],  
 max\_speed=SAMPLE\_ACTIVITY\_JSON["max\_speed"],  
 suffer\_score=SAMPLE\_ACTIVITY\_JSON["suffer\_score"],  
 average\_heartrate=SAMPLE\_ACTIVITY\_JSON["average\_heartrate"],  
 max\_heartrate=SAMPLE\_ACTIVITY\_JSON["max\_heartrate"],  
 calories=SAMPLE\_ACTIVITY\_JSON["calories"]  
 ))  
  
 test\_db\_session.commit()  
  
  
# -------------------------  
# 🔁 Patched App Fixtures  
# -------------------------  
  
@pytest.fixture(scope="function")  
def patched\_app(monkeypatch):  
 monkeypatch.setenv("CRON\_SECRET\_KEY", "devkey123")  
  
 with patch("src.routes.sync\_routes.sync\_recent") as mock\_sync\_recent:  
 mock\_sync\_recent.return\_value = 10  
 app = create\_app({"TESTING": True, "DATABASE\_URL": os.getenv("DATABASE\_URL")})  
 yield app  
  
  
@pytest.fixture(scope="function")  
def patched\_client(patched\_app):  
 return patched\_app.test\_client()  
  
  
# -------------------------  
# 🔐 Token Mock Fixtures  
# -------------------------  
  
@pytest.fixture(scope="function")  
def patched\_token\_mocks():  
 with patch("src.services.token\_service.get\_valid\_token") as mock\_valid, \  
 patch("src.db.dao.token\_dao.get\_tokens\_sa") as mock\_tokens:  
  
 mock\_tokens.return\_value = [  
 Token(  
 athlete\_id=1,  
 access\_token="mock\_access\_token",  
 refresh\_token="mock\_refresh\_token",  
 expires\_at=int((datetime.utcnow() + timedelta(hours=1)).timestamp())  
 )  
 ]  
 mock\_valid.return\_value = "mock\_access\_token"  
  
 yield mock\_tokens, mock\_valid

### 📄 tests\test\_activity\_dao.py

import pytest  
from unittest.mock import MagicMock, patch  
from datetime import datetime, timedelta  
from src.db.models.activities import Activity  
from src.db.dao.activity\_stats\_dao import ActivityStatsDAO  
from src.db.dao.activity\_dao import ActivityDAO  
  
  
def test\_get\_by\_id\_returns\_activity():  
 mock\_session = MagicMock()  
  
 mock\_query = mock\_session.query.return\_value  
 mock\_filter\_by = mock\_query.filter\_by.return\_value  
 expected\_activity = Activity(activity\_id=123)  
 mock\_filter\_by.first.return\_value = expected\_activity  
  
 result = ActivityDAO.get\_by\_id(mock\_session, 123)  
 assert result == expected\_activity  
  
 mock\_session.query.assert\_called\_once\_with(Activity)  
 mock\_query.filter\_by.assert\_called\_once\_with(activity\_id=123)  
 mock\_filter\_by.first.assert\_called\_once()  
  
  
def test\_get\_by\_id\_returns\_none():  
 mock\_session = MagicMock()  
 mock\_query = mock\_session.query.return\_value  
 mock\_filter\_by = mock\_query.filter\_by.return\_value  
 mock\_filter\_by.first.return\_value = None  
  
 result = ActivityDAO.get\_by\_id(mock\_session, 999)  
 assert result is None  
  
 mock\_session.query.assert\_called\_once\_with(Activity)  
 mock\_query.filter\_by.assert\_called\_once\_with(activity\_id=999)  
 mock\_filter\_by.first.assert\_called\_once()  
  
  
@patch("src.db.dao.activity\_dao.convert\_metrics")  
def test\_upsert\_activities\_empty\_list\_returns\_zero(mock\_convert):  
 mock\_session = MagicMock()  
 count = ActivityDAO.upsert\_activities(mock\_session, athlete\_id=1, activities=[])  
 assert count == 0  
 mock\_convert.assert\_not\_called()  
 mock\_session.execute.assert\_not\_called()  
 mock\_session.commit.assert\_not\_called()  
  
  
@patch("src.db.dao.activity\_dao.convert\_metrics")  
def test\_upsert\_activities\_single\_activity(mock\_convert):  
 mock\_session = MagicMock()  
 mock\_convert.return\_value = {  
 "conv\_distance": 100,  
 "conv\_elevation\_feet": 50,  
 "conv\_avg\_speed": 5,  
 "conv\_max\_speed": 10,  
 "conv\_moving\_time": 60,  
 "conv\_elapsed\_time": 65,  
 }  
  
 activities = [{  
 "id": 101,  
 "name": "Run",  
 "type": "Run",  
 "start\_date": "2023-01-01T00:00:00Z",  
 "distance": 1000,  
 "elapsed\_time": 65,  
 "moving\_time": 60,  
 "total\_elevation\_gain": 15,  
 "external\_id": "ext-101",  
 "timezone": "UTC",  
 "hr\_zone\_1": 10,  
 "hr\_zone\_2": 20,  
 "hr\_zone\_3": 30,  
 "hr\_zone\_4": 40,  
 "hr\_zone\_5": 50  
 }]  
  
 mock\_result = MagicMock()  
 mock\_result.rowcount = 1  
 mock\_session.execute.return\_value = mock\_result  
  
 count = ActivityDAO.upsert\_activities(mock\_session, athlete\_id=42, activities=activities)  
  
 assert count == 1  
 mock\_convert.assert\_called\_once()  
 mock\_session.execute.assert\_called\_once()  
 mock\_session.commit.assert\_called\_once()  
  
  
@patch("src.db.dao.activity\_dao.convert\_metrics")  
def test\_upsert\_activities\_multiple\_activities(mock\_convert):  
 mock\_session = MagicMock()  
 mock\_convert.return\_value = {  
 "conv\_distance": 100,  
 "conv\_elevation\_feet": 50,  
 "conv\_avg\_speed": 5,  
 "conv\_max\_speed": 10,  
 "conv\_moving\_time": 60,  
 "conv\_elapsed\_time": 65,  
 }  
  
 activities = [  
 {"id": 201, "name": "Morning Run", "type": "Run", "start\_date": "2023-01-01T06:00:00Z", "distance": 1000, "elapsed\_time": 65, "moving\_time": 60, "total\_elevation\_gain": 15, "external\_id": "ext-201"},  
 {"id": 202, "name": "Treadmill Run", "type": "Run", "start\_date": "2023-01-02T07:00:00Z", "distance": 500, "elapsed\_time": 35, "moving\_time": 30, "total\_elevation\_gain": 5, "external\_id": "ext-202"},  
 ]  
  
 mock\_result = MagicMock()  
 mock\_result.rowcount = 2  
 mock\_session.execute.return\_value = mock\_result  
  
 count = ActivityDAO.upsert\_activities(mock\_session, athlete\_id=99, activities=activities)  
  
 assert count == 2  
 assert mock\_convert.call\_count == 2  
 mock\_session.execute.assert\_called\_once()  
 mock\_session.commit.assert\_called\_once()  
  
  
def test\_get\_recent\_activities():  
 mock\_session = MagicMock()  
 fake\_activity = MagicMock()  
 fake\_activity.activity\_id = 999999  
 fake\_activity.athlete\_id = 123  
 fake\_activity.name = "Test Run"  
 mock\_session.scalars.return\_value.all.return\_value = [fake\_activity]  
  
 result = ActivityStatsDAO.get\_recent\_activities(mock\_session, athlete\_id=123, days=7)  
  
 assert isinstance(result, list)  
 assert result[0].activity\_id == 999999  
 mock\_session.scalars.assert\_called\_once()

### 📄 tests\test\_activity\_routes.py

import pytest  
from unittest.mock import patch, MagicMock  
from src.routes.activity\_routes import activity\_bp  
  
@pytest.fixture  
def client():  
 from flask import Flask  
 app = Flask(\_\_name\_\_)  
 app.register\_blueprint(activity\_bp)  
 app.config['TESTING'] = True  
 with app.test\_client() as client:  
 yield client  
  
def test\_enrich\_status(client):  
 resp = client.get("/enrich/status")  
 assert resp.status\_code == 200  
 assert resp.json == {"enrich": "ok"}  
  
@patch("src.routes.activity\_routes.get\_session")  
@patch("src.routes.activity\_routes.ActivityIngestionService")  
def test\_enrich\_single\_activity\_success(mock\_service\_cls, mock\_get\_session, client):  
 mock\_session = MagicMock()  
 mock\_get\_session.return\_value = mock\_session  
  
 # Simulate DB query returning athlete\_id  
 mock\_session.execute.return\_value.fetchone.return\_value = MagicMock(athlete\_id=123)  
 mock\_service = mock\_service\_cls.return\_value  
 mock\_service.enrich\_single\_activity.return\_value = True  
  
 resp = client.post("/enrich/activity/456")  
  
 assert resp.status\_code == 200  
 assert "enriched" in resp.json.get("status", "").lower()  
  
 mock\_get\_session.assert\_called\_once()  
 mock\_service\_cls.assert\_called\_once\_with(mock\_session, 123)  
 mock\_service.enrich\_single\_activity.assert\_called\_once\_with(456)  
 mock\_session.close.assert\_called\_once()  
  
@patch("src.routes.activity\_routes.get\_session")  
def test\_enrich\_single\_activity\_not\_found(mock\_get\_session, client):  
 mock\_session = MagicMock()  
 mock\_get\_session.return\_value = mock\_session  
 mock\_session.execute.return\_value.fetchone.return\_value = None  
  
 resp = client.post("/enrich/activity/999")  
  
 assert resp.status\_code == 404  
 assert "not found" in resp.json.get("error", "").lower()  
 mock\_session.close.assert\_called\_once()  
  
@patch("src.routes.activity\_routes.get\_session")  
@patch("src.routes.activity\_routes.run\_enrichment\_batch")  
def test\_enrich\_batch\_success(mock\_run\_batch, mock\_get\_session, client):  
 mock\_session = MagicMock()  
 mock\_get\_session.return\_value = mock\_session  
 mock\_run\_batch.return\_value = 5  
  
 resp = client.post("/enrich/batch?athlete\_id=123&batch=10")  
  
 assert resp.status\_code == 200  
 assert resp.json.get("count") == 5  
  
 mock\_run\_batch.assert\_called\_once\_with(mock\_session, 123, batch\_size=10)  
 mock\_session.close.assert\_called\_once()  
  
@patch("src.routes.activity\_routes.get\_session")  
def test\_enrich\_batch\_missing\_athlete\_id(mock\_get\_session, client):  
 resp = client.post("/enrich/batch")  
  
 assert resp.status\_code == 400  
 assert "missing athlete\_id" in resp.json.get("error", "").lower()  
 mock\_get\_session.assert\_not\_called()  
  
@patch("src.routes.activity\_routes.get\_session")  
@patch("src.routes.activity\_routes.ActivityIngestionService")  
def test\_sync\_strava\_to\_db\_success(mock\_service\_cls, mock\_get\_session, client):  
 mock\_session = MagicMock()  
 mock\_get\_session.return\_value = mock\_session  
  
 mock\_service = mock\_service\_cls.return\_value  
 mock\_service.ingest\_recent.return\_value = 7  
  
 # Set the correct CRON\_SECRET\_KEY in config  
 import src.utils.config as config  
 old\_key = config.CRON\_SECRET\_KEY  
 config.CRON\_SECRET\_KEY = "secret"  
  
 url = "/sync/123?key=secret&lookback=15&limit=5"  
 resp = client.get(url)  
  
 assert resp.status\_code == 200  
 assert resp.json.get("inserted") == 7  
  
 mock\_service\_cls.assert\_called\_once\_with(mock\_session, 123)  
 mock\_service.ingest\_recent.assert\_called\_once\_with(lookback\_days=15, max\_activities=5)  
 mock\_session.close.assert\_called\_once()  
  
 # Restore config  
 config.CRON\_SECRET\_KEY = old\_key  
  
def test\_sync\_strava\_to\_db\_unauthorized(client):  
 resp = client.get("/sync/123?key=wrongkey")  
 assert resp.status\_code == 401  
 assert "unauthorized" in resp.json.get("error", "").lower()

### 📄 tests\test\_activity\_service.py

import pytest  
from unittest.mock import MagicMock, patch, call  
from datetime import datetime, timedelta  
from src.services import activity\_service as svc  
  
@pytest.fixture  
def mock\_session():  
 return MagicMock()  
  
@pytest.fixture  
def athlete\_id():  
 return 42  
  
@pytest.fixture  
def dummy\_activity\_json():  
 return {  
 "id": 123,  
 "name": "Test Activity",  
 "distance": 5000,  
 "total\_elevation\_gain": 100,  
 "average\_speed": 3.5,  
 "max\_speed": 5.0,  
 "moving\_time": 1800,  
 "elapsed\_time": 2000,  
 "type": "Run",  
 "suffer\_score": 50,  
 "average\_heartrate": 140,  
 "max\_heartrate": 170,  
 "calories": 400,  
 "hr\_zone\_1": 10,  
 "hr\_zone\_2": 20,  
 "hr\_zone\_3": 30,  
 "hr\_zone\_4": 25,  
 "hr\_zone\_5": 15  
 }  
  
@pytest.fixture  
def dummy\_zones\_data():  
 return [  
 {  
 "type": "heartrate",  
 "distribution\_buckets": [  
 {"time": 300},  
 {"time": 300},  
 {"time": 200},  
 {"time": 100},  
 {"time": 100},  
 ]  
 }  
 ]  
  
@pytest.fixture  
def dummy\_streams():  
 return {  
 "distance": [0, 1609.344, 3218.688, 4000],  
 "time": [0, 600, 1200, 1500],  
 "velocity\_smooth": [3.0, 3.5, 4.0, 3.8],  
 "heartrate": [130, 135, 140, 145],  
 }  
  
def test\_get\_activities\_to\_enrich\_returns\_ids(mock\_session, athlete\_id):  
 # Setup mock execute().fetchall()  
 mock\_session.execute.return\_value.fetchall.return\_value = [  
 MagicMock(activity\_id=101),  
 MagicMock(activity\_id=102),  
 MagicMock(activity\_id=103),  
 ]  
 result = svc.get\_activities\_to\_enrich(mock\_session, athlete\_id, limit=3)  
 assert result == [101, 102, 103]  
 mock\_session.execute.assert\_called\_once()  
  
@patch("src.services.activity\_service.StravaClient")  
@patch("src.services.activity\_service.extract\_hr\_zone\_percentages", return\_value=[10,20,30,25,15])  
@patch("src.services.activity\_service.upsert\_splits")  
def test\_enrich\_one\_activity\_success(mock\_upsert, mock\_extract\_zones, MockClient, mock\_session, dummy\_activity\_json, dummy\_zones\_data, dummy\_streams):  
 mock\_client = MockClient.return\_value  
 mock\_client.get\_activity.return\_value = dummy\_activity\_json  
 mock\_client.get\_hr\_zones.return\_value = dummy\_zones\_data  
 mock\_client.get\_streams.return\_value = dummy\_streams  
  
 result = svc.enrich\_one\_activity(mock\_session, "fake-token", 123)  
 assert result is True  
 mock\_client.get\_activity.assert\_called\_once\_with(123)  
 mock\_extract\_zones.assert\_called\_once\_with(dummy\_zones\_data)  
 mock\_upsert.assert\_called\_once()  
 mock\_session.execute.assert\_called()  
 mock\_session.commit.assert\_called()  
  
@patch("src.services.activity\_service.get\_valid\_token", return\_value="fake-token")  
@patch("src.services.activity\_service.enrich\_one\_activity", return\_value=True)  
def test\_enrich\_one\_activity\_with\_refresh\_calls\_enrich(mock\_enrich, mock\_token, mock\_session, athlete\_id):  
 result = svc.enrich\_one\_activity\_with\_refresh(mock\_session, athlete\_id, 456)  
 assert result is True  
 mock\_token.assert\_called\_once\_with(mock\_session, athlete\_id)  
 mock\_enrich.assert\_called\_once\_with(mock\_session, "fake-token", 456)  
  
def test\_update\_activity\_enrichment\_executes\_sql(mock\_session, dummy\_activity\_json):  
 hr\_zones = [10, 20, 30, 25, 15]  
 svc.update\_activity\_enrichment(mock\_session, 123, dummy\_activity\_json, hr\_zones)  
 mock\_session.execute.assert\_called\_once()  
 mock\_session.commit.assert\_called\_once()  
  
def test\_extract\_hr\_zone\_percentages\_returns\_correct\_percentages():  
 zones\_data = [  
 {  
 "type": "heartrate",  
 "distribution\_buckets": [  
 {"time": 300},  
 {"time": 300},  
 {"time": 200},  
 {"time": 100},  
 {"time": 100},  
 ]  
 }  
 ]  
 result = svc.extract\_hr\_zone\_percentages(zones\_data)  
 assert sum(result) == 100.0  
 assert len(result) == 5  
  
def test\_build\_mile\_splits\_correctness():  
 streams = {  
 "distance": [0, 1609.344, 3218.688, 4000],  
 "time": [0, 600, 1200, 1500],  
 "velocity\_smooth": [3.0, 3.5, 4.0, 3.8],  
 "heartrate": [130, 135, 140, 145],  
 }  
 splits = svc.build\_mile\_splits(1, streams)  
 assert isinstance(splits, list)  
 assert all("activity\_id" in split for split in splits)  
 assert splits[0]["lap\_index"] == 1  
 assert splits[-1]["lap\_index"] == len(splits)  
  
@patch("src.services.activity\_service.ActivityDAO.upsert\_activities")  
@patch("src.services.activity\_service.StravaClient.get\_activities")  
@patch("src.services.activity\_service.get\_valid\_token", return\_value="fake-token")  
def test\_activity\_ingestion\_service\_methods(mock\_token, mock\_get\_activities, mock\_upsert, mock\_session, athlete\_id):  
 # Setup mock activities  
 # Setup mock activities with "Run" type to pass filtering logic  
 mock\_get\_activities.return\_value = [{"id": 1, "type": "Run"}, {"id": 2, "type": "Run"}]   
  
  
 service = svc.ActivityIngestionService(mock\_session, athlete\_id)  
  
 # ingest\_recent calls DAO upsert  
 service.ingest\_recent(lookback\_days=10, max\_activities=5)  
 mock\_get\_activities.assert\_called()  
 mock\_upsert.assert\_called\_once()  
  
 # ingest\_full\_history delegates to ingest\_recent  
 mock\_upsert.reset\_mock()  
 service.ingest\_full\_history(lookback\_days=365, max\_activities=10)  
 mock\_upsert.assert\_called\_once()  
  
 # ingest\_between calls DAO upsert  
 start = datetime.utcnow() - timedelta(days=5)  
 end = datetime.utcnow()  
 mock\_upsert.reset\_mock()  
 service.ingest\_between(start, end, max\_activities=10)  
 mock\_upsert.assert\_called\_once()  
  
@patch("src.services.activity\_service.get\_activities\_to\_enrich")  
@patch("src.services.activity\_service.enrich\_one\_activity\_with\_refresh")  
def test\_run\_enrichment\_batch\_calls\_all(mock\_enrich, mock\_get\_activities, mock\_session, athlete\_id):  
 mock\_get\_activities.return\_value = [1, 2, 3]  
 svc.run\_enrichment\_batch(mock\_session, athlete\_id, batch\_size=3)  
 assert mock\_enrich.call\_count == 3  
 mock\_enrich.assert\_has\_calls([call(mock\_session, athlete\_id, 1), call(mock\_session, athlete\_id, 2), call(mock\_session, athlete\_id, 3)])

### 📄 tests\test\_app.py

import pytest  
from unittest.mock import patch, MagicMock  
from src.app import create\_app  
  
@pytest.fixture  
def client():  
 app = create\_app({"TESTING": True})  
 with app.test\_client() as client:  
 yield client  
  
def test\_ping(client):  
 resp = client.get("/ping")  
 assert resp.status\_code == 200  
 assert resp.data == b"pong"  
  
@patch("sqlalchemy.inspect")  
@patch("sqlalchemy.create\_engine")  
def test\_db\_check\_success(mock\_create\_engine, mock\_inspect, client):  
 mock\_engine = MagicMock()  
 mock\_create\_engine.return\_value = mock\_engine  
 mock\_insp = MagicMock()  
 mock\_inspect.return\_value = mock\_insp  
 mock\_insp.get\_columns.return\_value = [  
 {"name": "id", "type": "Integer", "nullable": False},  
 {"name": "split", "type": "Integer", "nullable": True}  
 ]  
  
 resp = client.get("/db-check")  
 assert resp.status\_code == 200  
 json\_data = resp.get\_json()  
 assert json\_data["status"] == "ok"  
 assert json\_data["split\_column"]["name"] == "split"  
 assert json\_data["split\_column"]["nullable"] is True  
  
@patch("sqlalchemy.create\_engine", side\_effect=Exception("DB failure"))  
def test\_db\_check\_failure(mock\_create\_engine, client):  
 resp = client.get("/db-check")  
 assert resp.status\_code == 500  
 json\_data = resp.get\_json()  
 assert json\_data["status"] == "fail"  
 assert "DB failure" in json\_data["error"]  
  
def test\_startup(client):  
 resp = client.get("/startup")  
 assert resp.status\_code == 200  
 json\_data = resp.get\_json()  
 assert "status" in json\_data and json\_data["status"] == "started"  
 assert "cwd" in json\_data  
 assert "files" in json\_data  
  
def test\_blueprints\_registered(client):  
 # Test that known blueprint routes exist  
 resp = client.get("/ping")  
 assert resp.status\_code == 200  
  
 resp = client.get("/auth/login")  
 # This route exists but may redirect or 405 if not POST; just check not 404  
 assert resp.status\_code != 404  
  
 resp = client.get("/sync/enrich/status")  
 assert resp.status\_code == 200

### 📄 tests\test\_ask\_routes.py

import pytest  
from flask import Flask  
from src.routes.ask\_routes import ask\_bp  
  
@pytest.fixture  
def client():  
 app = Flask(\_\_name\_\_)  
 app.register\_blueprint(ask\_bp)  
 app.config['TESTING'] = True  
 return app.test\_client()  
  
def test\_valid\_request(client):  
 response = client.post('/ask', json={"question": " How far did I run today? ", "athlete\_id": 123})  
 assert response.status\_code in (200, 302)  
 data = response.get\_json()  
 assert data['athlete\_id'] == 123  
 assert data['question'] == "How far did I run today?"  
  
def test\_missing\_content\_type(client):  
 response = client.post('/ask', data="invalid", content\_type='text/plain')  
 assert response.status\_code == 400  
 assert "content-type" in response.get\_json()['error']  
  
def test\_missing\_payload(client):  
 response = client.post('/ask', json=None)  
 assert response.status\_code == 400  
 assert "content-type must be application/json" in response.get\_json()['error'].lower()  
  
  
def test\_invalid\_question(client):  
 response = client.post('/ask', json={"question": " ", "athlete\_id": 1})  
 assert response.status\_code == 400  
 assert "question" in response.get\_json()['error']  
  
def test\_missing\_athlete\_id(client):  
 response = client.post('/ask', json={"question": "What's my progress?"})  
 assert response.status\_code == 400  
 assert "athlete\_id" in response.get\_json()['error']  
  
def test\_invalid\_athlete\_id(client):  
 response = client.post('/ask', json={"question": "Status?", "athlete\_id": -5})  
 assert response.status\_code == 400  
 assert "athlete\_id" in response.get\_json()['error']

### 📄 tests\test\_athlete\_dao.py

# tests/test\_athlete\_dao.py  
  
import pytest  
from sqlalchemy.orm import Session  
from src.db.dao.athlete\_dao import (  
 insert\_athlete,  
 get\_athlete\_by\_strava\_id,  
 get\_athlete\_id\_from\_strava\_id,  
)  
from src.db.models.athletes import Athlete  
  
def test\_insert\_and\_get\_athlete(test\_db\_session: Session):  
 strava\_id = 123456789  
 name = "Test User"  
 email = "test@example.com"  
  
 # Insert athlete  
 athlete\_id = insert\_athlete(test\_db\_session, strava\_id, name, email)  
 assert isinstance(athlete\_id, int)  
  
 # Get full athlete  
 athlete = get\_athlete\_by\_strava\_id(test\_db\_session, strava\_id)  
 assert athlete is not None  
 assert athlete.name == name  
 assert athlete.email == email  
  
 # Get ID by strava ID  
 fetched\_id = get\_athlete\_id\_from\_strava\_id(test\_db\_session, strava\_id)  
 assert fetched\_id == athlete\_id

### 📄 tests\test\_auth.py

import os  
import jwt  
from datetime import datetime, timedelta  
from unittest.mock import patch  
import src.utils.config as config # Import config for admin credentials  
  
  
@patch("src.services.token\_service.refresh\_token\_static")  
def test\_login\_refresh\_logout(mock\_refresh, client):  
 """Test successful login, token refresh using Authorization header, and logout."""  
 mock\_refresh.return\_value = {  
 "access\_token": "mocked\_access",  
 "refresh\_token": "mocked\_refresh",  
 "expires\_at": int((datetime.utcnow() + timedelta(hours=1)).timestamp())  
 }  
  
 # Step 1: Login with correct credentials from config  
 resp = client.post("/auth/login", json={"username": config.ADMIN\_USER, "password": config.ADMIN\_PASS})  
 assert resp.status\_code == 200  
  
 tokens = resp.get\_json()  
 access\_token = tokens["access\_token"]  
 refresh\_token = tokens["refresh\_token"]  
  
 # 🔧 Inject mock token record into DB for athlete\_id=0  
 from src.db.models.tokens import Token  
 from src.db.db\_session import get\_session  
 session = get\_session()  
  
 with session as db\_session:  
 db\_session.add(Token(  
 athlete\_id=0,  
 access\_token="old\_access",  
 refresh\_token="mocked\_refresh",  
 expires\_at=int((datetime.utcnow() - timedelta(hours=1)).timestamp()) # expired to trigger refresh  
 ))  
 db\_session.commit()  
  
 # Step 2: Refresh (using Authorization header)  
 headers = {"Authorization": f"Bearer {refresh\_token}"}  
 resp = client.post("/auth/refresh/0", headers=headers)  
 assert resp.status\_code == 200  
  
 # Step 3: Logout  
 resp = client.post("/auth/logout/0", headers=headers)  
 assert resp.status\_code == 200  
  
  
def test\_invalid\_login\_rejected(client):  
 """Test that invalid credentials are rejected."""  
 resp = client.post("/auth/login", json={"username": "wrong", "password": "bad"})  
 assert resp.status\_code == 401  
  
  
def test\_invalid\_refresh\_token(client):  
 """Test that an invalid refresh token is rejected."""  
 headers = {"Authorization": "Bearer not.a.real.token"}  
 resp = client.post("/auth/refresh/0", headers=headers)  
 assert resp.status\_code == 401  
  
  
def test\_expired\_refresh\_token(client):  
 """Test refresh fails with an expired token."""  
 secret = os.environ.get("SECRET\_KEY", "testsecret")  
  
 expired\_token = jwt.encode(  
 {  
 "sub": "admin",  
 "exp": datetime.utcnow() - timedelta(seconds=1)  
 },  
 secret,  
 algorithm="HS256"  
 )  
  
 resp = client.post("/auth/refresh/0", headers={"Authorization": f"Bearer {expired\_token}"})  
 print(f"⏰ Expired refresh status: {resp.status\_code}, Body: {resp.data.decode()}")  
 assert resp.status\_code == 401

### 📄 tests\test\_auth\_routes.py

import pytest  
from unittest.mock import patch, MagicMock  
from flask import Flask  
import jwt  
from datetime import datetime, timedelta  
import src.utils.config as config  
from src.routes.auth\_routes import auth\_bp  
  
# ----------------------  
# Test Setup  
# ----------------------  
  
@pytest.fixture  
def client():  
 app = Flask(\_\_name\_\_)  
 app.secret\_key = "test-secret-key" # Required for flask.session usage  
 app.register\_blueprint(auth\_bp, url\_prefix="/auth")  
 app.config['TESTING'] = True  
 with app.test\_client() as client:  
 yield client  
  
def create\_jwt\_token(sub="admin", exp=None):  
 if exp is None:  
 exp = datetime.utcnow() + timedelta(seconds=config.ACCESS\_TOKEN\_EXP)  
 return jwt.encode({"sub": sub, "exp": exp}, config.SECRET\_KEY, algorithm="HS256")  
  
# ----------------------  
# /auth/login POST  
# ----------------------  
  
def test\_admin\_login\_success(client):  
 data = {"username": config.ADMIN\_USER, "password": config.ADMIN\_PASS}  
 response = client.post("/auth/login", json=data)  
 assert response.status\_code == 200  
 assert "access\_token" in response.json  
 assert "refresh\_token" in response.json  
  
def test\_admin\_login\_invalid\_credentials(client):  
 data = {"username": "wrong", "password": "wrong"}  
 response = client.post("/auth/login", json=data)  
 assert response.status\_code == 401  
 assert response.json["error"] == "Unauthorized"  
  
def test\_admin\_login\_missing\_json(client):  
 response = client.post("/auth/login")  
 assert response.status\_code in (400, 500)  
  
# ----------------------  
# /auth/login GET (Strava OAuth)  
# ----------------------  
  
def test\_strava\_login\_redirect(client, monkeypatch):  
 monkeypatch.setattr("os.getenv", lambda k: {  
 "STRAVA\_CLIENT\_ID": "123",  
 "STRAVA\_REDIRECT\_URI": "http://redirect"  
 }.get(k))  
 response = client.get("/auth/login")  
 assert response.status\_code == 302  
 assert "strava.com/oauth/authorize" in response.location  
  
# ----------------------  
# /auth/callback GET  
# ----------------------  
  
@patch("src.services.token\_service.store\_tokens\_from\_callback")  
@patch("src.routes.auth\_routes.get\_session")  
def test\_callback\_success(mock\_get\_session, mock\_store\_tokens, client):  
 mock\_store\_tokens.return\_value = 123  
 mock\_get\_session.return\_value = MagicMock()  
 response = client.get("/auth/callback?code=fakecode")  
 assert response.status\_code == 200  
 assert "Token stored for athlete\_id: 123" in response.get\_data(as\_text=True)  
  
def test\_callback\_missing\_code(client):  
 response = client.get("/auth/callback")  
 assert response.status\_code == 400  
 assert "Missing OAuth code" in response.get\_data(as\_text=True)  
  
@patch("src.services.token\_service.store\_tokens\_from\_callback", side\_effect=Exception("fail"))  
@patch("src.routes.auth\_routes.get\_session")  
def test\_callback\_exception(mock\_get\_session, mock\_store\_tokens, client):  
 mock\_get\_session.return\_value = MagicMock()  
 response = client.get("/auth/callback?code=code")  
 assert response.status\_code == 500  
 assert "Callback error" in response.get\_data(as\_text=True)  
  
# ----------------------  
# /auth/refresh/<athlete\_id> POST  
# ----------------------  
  
@patch("src.routes.auth\_routes.get\_session")  
@patch("src.routes.auth\_routes.refresh\_token\_if\_expired")  
def test\_refresh\_token\_success(mock\_refresh, mock\_get\_session, client):  
 mock\_refresh.return\_value = True  
 mock\_get\_session.return\_value = MagicMock()  
 token = create\_jwt\_token()  
 headers = {"Authorization": f"Bearer {token}"}  
 response = client.post("/auth/refresh/1", headers=headers)  
 assert response.status\_code == 200  
 assert response.json == {"refreshed": True}  
  
def test\_refresh\_token\_missing\_auth\_header(client):  
 response = client.post("/auth/refresh/1")  
 assert response.status\_code == 401  
 assert "Missing or invalid Authorization header" in response.json["error"]  
  
def test\_refresh\_token\_invalid\_token(client):  
 headers = {"Authorization": "Bearer invalidtoken"}  
 response = client.post("/auth/refresh/1", headers=headers)  
 assert response.status\_code == 401  
 assert response.json["error"] == "Invalid token"  
  
# ----------------------  
# /auth/logout/<athlete\_id> POST  
# ----------------------  
  
@patch("src.routes.auth\_routes.get\_session")  
@patch("src.routes.auth\_routes.delete\_athlete\_tokens")  
def test\_logout\_success(mock\_delete, mock\_get\_session, client):  
 mock\_delete.return\_value = True  
 mock\_get\_session.return\_value = MagicMock()  
 response = client.post("/auth/logout/1")  
 assert response.status\_code == 200  
 assert response.json == {"deleted": True}  
  
# ----------------------  
# /auth/monitor-tokens GET  
# ----------------------  
  
@patch("src.routes.auth\_routes.get\_session")  
def test\_monitor\_tokens\_success(mock\_get\_session, client):  
 mock\_session = MagicMock()  
 mock\_session.execute.return\_value.fetchall.return\_value = [  
 MagicMock(athlete\_id=1, expires\_at=123456789),  
 MagicMock(athlete\_id=2, expires\_at=987654321)  
 ]  
 mock\_get\_session.return\_value = mock\_session  
  
 response = client.get("/auth/monitor-tokens")  
 assert response.status\_code == 200  
 assert isinstance(response.json, list)  
 assert response.json[0]["athlete\_id"] == 1  
  
@patch("src.routes.auth\_routes.get\_session")  
def test\_monitor\_tokens\_exception(mock\_get\_session, client):  
 mock\_session = MagicMock()  
 mock\_session.execute.side\_effect = Exception("DB fail")  
 mock\_get\_session.return\_value = mock\_session  
  
 response = client.get("/auth/monitor-tokens")  
 assert response.status\_code == 500  
 assert "error" in response.json

### 📄 tests\test\_data\sample\_activities.py

# tests/test\_data/sample\_activities.py  
  
SAMPLE\_ACTIVITY\_JSON = {  
 "id": 99999,  
 "activity\_id": 99999,  
 "external\_id": "external\_99999", # required field  
 "name": "Mock Run",  
 "type": "Run",  
 "distance": 5000.0,  
 "moving\_time": 1500,  
 "elapsed\_time": 1600,  
 "total\_elevation\_gain": 50.0,  
 "average\_speed": 3.5,  
 "max\_speed": 4.0,  
 "suffer\_score": 30,  
 "average\_heartrate": 150,  
 "max\_heartrate": 170,  
 "calories": 400,  
 "start\_date": "2025-06-01T08:00:00Z", # <--- ADD THIS LINE  
 "splits\_metric": [  
 {  
 "lap\_index": 1,  
 "distance": 1000,  
 "elapsed\_time": 300,  
 "moving\_time": 295,  
 "average\_speed": 3.33,  
 "max\_speed": 3.5,  
 "start\_index": 0,  
 "end\_index": 299,  
 "split": 1,  
 "average\_heartrate": 145,  
 "pace\_zone": 2  
 }  
 ]  
}  
  
SAMPLE\_HR\_ZONE\_RESPONSE = {  
 "heart\_rate": {  
 "custom\_zones": [  
 {"score": 0.1},  
 {"score": 0.2},  
 {"score": 0.3},  
 {"score": 0.25},  
 {"score": 0.15}  
 ]  
 }  
}  
  
SAMPLE\_STREAMS\_RESPONSE = {  
 "distance": {"data": [0.0, 500.0, 1000.0]},  
 "time": {"data": [0, 150, 300]},  
 "velocity\_smooth": {"data": [3.2, 3.4, 3.3]},  
 "heartrate": {"data": [130, 140, 150]}  
}

### 📄 tests\test\_enrichment\_with\_splits.py

# tests/test\_enrichment\_with\_splits.py  
  
import pytest  
from unittest.mock import patch  
from datetime import datetime, timedelta  
import random  
  
from src.db.models.activities import Activity  
from src.db.models.splits import Split  
from src.db.models.tokens import Token  
from src.services.activity\_service import enrich\_one\_activity\_with\_refresh  
from tests.test\_data.sample\_activities import SAMPLE\_ACTIVITY\_JSON  
  
SAMPLE\_HR\_ZONE\_RESPONSE = {  
 "type": "heartrate",  
 "distribution\_buckets": [  
 {"time": 300},  
 {"time": 300},  
 {"time": 200},  
 {"time": 100},  
 {"time": 100}  
 ]  
}  
  
@pytest.fixture  
def sqlalchemy\_token(sqlalchemy\_session):  
 token = Token(  
 athlete\_id=42,  
 access\_token="mock\_access",  
 refresh\_token="mock\_refresh",  
 expires\_at=int((datetime.utcnow() + timedelta(hours=1)).timestamp())  
 )  
 sqlalchemy\_session.add(token)  
 sqlalchemy\_session.commit()  
 return token  
  
@pytest.fixture  
def seed\_activity(sqlalchemy\_session):  
 activity\_id = random.randint(100000, 999999)  
 activity = Activity(  
 activity\_id=activity\_id,  
 athlete\_id=42,  
 start\_date=datetime.utcnow()  
 )  
 sqlalchemy\_session.add(activity)  
 sqlalchemy\_session.commit()  
 return activity  
  
@patch("src.services.activity\_service.get\_valid\_token")  
@patch("src.services.strava\_access\_service.StravaClient.get\_activity")  
@patch("src.services.strava\_access\_service.StravaClient.get\_hr\_zones")  
@patch("src.services.strava\_access\_service.StravaClient.get\_streams")  
@patch("src.services.strava\_access\_service.StravaClient.get\_splits")  
def test\_enrich\_one\_activity\_with\_splits(  
 mock\_get\_splits,  
 mock\_get\_streams,  
 mock\_get\_hr\_zones,  
 mock\_get\_activity,  
 mock\_get\_token,  
 sqlalchemy\_session,  
 sqlalchemy\_token,  
 seed\_activity  
):  
 mock\_get\_token.return\_value = sqlalchemy\_token  
 mock\_get\_activity.return\_value = SAMPLE\_ACTIVITY\_JSON  
 mock\_get\_hr\_zones.return\_value = SAMPLE\_HR\_ZONE\_RESPONSE  
 mock\_get\_streams.return\_value = {  
 "distance": [0.0, 800.0, 1609.34, 1700.0], # 1609.34 is exactly 1 mile  
 "time": [0, 200, 400, 420],  
 "velocity\_smooth": [3.1, 3.3, 3.4, 3.2],  
 "heartrate": [138, 140, 142, 144]  
 }  
 mock\_get\_splits.return\_value = [  
 {"elapsed\_time": 300, "distance": 1700.0, "average\_speed": 3.2, "split": 1, "lap\_index": 1}  
 ]  
  
 activity\_id = seed\_activity.activity\_id  
  
 result = enrich\_one\_activity\_with\_refresh(  
 sqlalchemy\_session, seed\_activity.athlete\_id, activity\_id=activity\_id  
 )  
 assert result is True  
  
 splits = sqlalchemy\_session.query(Split).filter\_by(activity\_id=activity\_id).all()  
 assert len(splits) == 1  
 assert splits[0].lap\_index == 1  
 assert splits[0].distance == 1700.0  
 assert splits[0].elapsed\_time == 420  
 assert isinstance(splits[0].split, int)  
  
 activity = sqlalchemy\_session.query(Activity).filter\_by(activity\_id=activity\_id).one()  
 assert activity.hr\_zone\_1 is not None  
 assert activity.hr\_zone\_5 is not None

### 📄 tests\test\_extract\_hr\_zone\_percentages.py

import pytest  
from src.services.activity\_service import extract\_hr\_zone\_percentages  
  
def test\_extract\_hr\_zone\_percentages\_normal():  
 zones\_data = [  
 {  
 "type": "heartrate",  
 "distribution\_buckets": [  
 {"time": 100},  
 {"time": 200},  
 {"time": 300},  
 {"time": 400},  
 {"time": 0},  
 ],  
 }  
 ]  
 expected = [10.0, 20.0, 30.0, 40.0, 0.0]  
 result = extract\_hr\_zone\_percentages(zones\_data)  
 assert result == expected  
  
def test\_extract\_hr\_zone\_percentages\_zero\_total():  
 zones\_data = [  
 {  
 "type": "heartrate",  
 "distribution\_buckets": [  
 {"time": 0},  
 {"time": 0},  
 {"time": 0},  
 {"time": 0},  
 {"time": 0},  
 ],  
 }  
 ]  
 result = extract\_hr\_zone\_percentages(zones\_data)  
 assert result == [0.0, 0.0, 0.0, 0.0, 0.0]  
  
def test\_extract\_hr\_zone\_percentages\_no\_heartrate():  
 zones\_data = [{"type": "something\_else", "distribution\_buckets": []}]  
 result = extract\_hr\_zone\_percentages(zones\_data)  
 assert result == [0.0, 0.0, 0.0, 0.0, 0.0]  
  
def test\_extract\_hr\_zone\_percentages\_malformed\_input():  
 result = extract\_hr\_zone\_percentages(None)  
 assert result == [0.0, 0.0, 0.0, 0.0, 0.0]

### 📄 tests\test\_format\_prompt.py

import pytest  
from src.utils.gpt\_ops import format\_prompt  
  
def test\_format\_prompt\_with\_valid\_data():  
 user\_question = "How far did I run this week?"  
 activities = [  
 {"date": "2025-06-24", "distance\_km": 5.2, "duration\_min": 28},  
 {"date": "2025-06-25", "distance\_km": 10.0, "duration\_min": 54},  
 ]  
  
 result = format\_prompt(user\_question, activities)  
  
 assert "You are a smart coaching assistant" in result  
 assert "ACTIVITIES" in result  
 assert "[1] date: 2025-06-24, distance\_km: 5.2, duration\_min: 28" in result  
 assert "[2] date: 2025-06-25, distance\_km: 10.0, duration\_min: 54" in result  
 assert "USER QUESTION" in result  
 assert "How far did I run this week?" in result  
  
def test\_format\_prompt\_with\_empty\_activities():  
 result = format\_prompt("What's my performance?", [])  
 assert "ACTIVITIES" in result  
 assert "USER QUESTION" in result  
 assert "What's my performance?" in result

### 📄 tests\test\_full\_ingestion\_flow.py

import pytest  
from unittest.mock import patch  
from datetime import datetime, timedelta  
  
from src.db.db\_session import get\_session  
from src.db.models.tokens import Token  
from src.db.models.activities import Activity  
from src.db.models.splits import Split  
from src.scripts.main\_pipeline import run\_full\_ingestion\_and\_enrichment  
  
# Import the sample activity JSON with external\_id included  
from tests.test\_data.sample\_activities import SAMPLE\_ACTIVITY\_JSON  
  
@pytest.fixture(scope="module")  
def test\_session():  
 session = get\_session()  
 yield session  
 session.rollback()  
 session.close()  
  
@pytest.fixture(scope="function")  
def seeded\_token(test\_session):  
 token = Token(  
 athlete\_id=1,  
 access\_token="mock\_access",  
 refresh\_token="mock\_refresh",  
 expires\_at=int((datetime.utcnow() + timedelta(hours=1)).timestamp())  
 )  
 test\_session.add(token)  
 test\_session.commit()  
 yield token  
 test\_session.delete(token)  
 test\_session.commit()  
  
@patch("src.services.strava\_access\_service.StravaClient.get\_streams")  
@patch("src.services.strava\_access\_service.StravaClient.get\_activity")  
@patch("src.services.strava\_access\_service.StravaClient.get\_activities")  
@patch("src.services.strava\_access\_service.StravaClient.get\_splits")  
@patch("src.services.strava\_access\_service.StravaClient.get\_hr\_zones")  
def test\_run\_full\_ingestion\_flow(  
 mock\_zones, mock\_splits, mock\_activities, mock\_activity, mock\_streams,  
 test\_session, seeded\_token  
):  
 athlete\_id = 1  
 mock\_activity\_id = SAMPLE\_ACTIVITY\_JSON["id"]  
  
 # Ensure no conflicting data  
 test\_session.query(Split).filter\_by(activity\_id=mock\_activity\_id).delete()  
 test\_session.query(Activity).filter\_by(activity\_id=mock\_activity\_id).delete()  
 test\_session.commit()  
  
 # Use SAMPLE\_ACTIVITY\_JSON and ensure it has external\_id  
 mock\_activity\_data = SAMPLE\_ACTIVITY\_JSON.copy()  
 mock\_activity\_data["external\_id"] = f"external\_{mock\_activity\_id}" # Ensure external\_id exists  
  
 mock\_activities.return\_value = [mock\_activity\_data]  
 mock\_activity.return\_value = mock\_activity\_data  
  
 mock\_splits.return\_value = [  
 {"elapsed\_time": 600, "distance": 1609, "average\_speed": 3.3, "split": 1},  
 {"elapsed\_time": 620, "distance": 1609, "average\_speed": 3.2, "split": 2}  
 ]  
  
 mock\_zones.return\_value = [{  
 "type": "heartrate",  
 "distribution\_buckets": [  
 {"time": 300}, {"time": 300}, {"time": 200}, {"time": 100}, {"time": 100}  
 ]  
 }]  
  
 mock\_streams.return\_value = {  
 "distance": [1609.344, 3219], # Slightly more than 2 \* 1609.344  
 "time": [600, 1220],  
 "velocity\_smooth": [3.3, 3.2],  
 "heartrate": [140, 145]  
 }  
  
 result = run\_full\_ingestion\_and\_enrichment(test\_session, athlete\_id)  
  
 assert result["synced"] >= 1, "Expected at least one activity to be inserted"  
  
 activity = test\_session.query(Activity).filter\_by(activity\_id=mock\_activity\_id).first()  
 assert activity is not None, "Activity not found in DB"  
 assert activity.name == mock\_activity\_data["name"]  
 assert activity.start\_date.isoformat().startswith("2025-06-01")  
 assert activity.distance == mock\_activity\_data["distance"]  
  
 splits = test\_session.query(Split).filter\_by(activity\_id=mock\_activity\_id).all()  
 assert len(splits) == 2, "Expected 2 splits to be inserted"

### 📄 tests\test\_health.py

# tests/test\_health.py  
def test\_ping(client):  
 resp = client.get("/ping")  
 assert resp.status\_code == 200  
 assert resp.data == b"pong"

### 📄 tests\test\_hr\_zone\_api.py

import os  
import requests  
  
# Replace with a valid activity\_id you know has HR data  
activity\_id = 14663194187 # <-- replace with one of your existing IDs  
  
# Read access token directly from database or environment  
access\_token = os.getenv("STRAVA\_ACCESS\_TOKEN")  
  
if not access\_token:  
 raise RuntimeError("Missing STRAVA\_ACCESS\_TOKEN environment variable")  
  
url = f"https://www.strava.com/api/v3/activities/{activity\_id}/zones"  
headers = {"Authorization": f"Bearer {access\_token}"}  
  
resp = requests.get(url, headers=headers, timeout=10)  
print(f"HTTP Status: {resp.status\_code}")  
  
if resp.status\_code == 200:  
 data = resp.json()  
 print("✅ Successfully fetched HR zone data:")  
 print(data)  
else:  
 print("❌ Failed to fetch HR zones.")  
 print(resp.text)

### 📄 tests\test\_ingestion\_orchestrator.py

# tests/test\_ingestion\_orchestrator.py  
  
import pytest  
from unittest.mock import patch, MagicMock  
from datetime import datetime  
from src.services.ingestion\_orchestrator\_service import ingest\_specific\_activity, ingest\_between\_dates  
  
  
@pytest.fixture  
def session():  
 """Mocked DB session fixture"""  
 return MagicMock()  
  
  
@patch("src.services.ingestion\_orchestrator\_service.enrich\_one\_activity\_with\_refresh")  
@patch("src.services.ingestion\_orchestrator\_service.ActivityDAO.upsert\_activities")  
@patch("src.services.ingestion\_orchestrator\_service.ActivityIngestionService")  
def test\_ingest\_specific\_activity\_success(mock\_service, mock\_upsert, mock\_enrich, session):  
 athlete\_id = 123  
 activity\_id = 456  
  
 mock\_service\_instance = mock\_service.return\_value  
 mock\_activity = {"id": activity\_id, "name": "Test Activity"}  
 mock\_service\_instance.client.get\_activity.return\_value = mock\_activity  
  
 mock\_upsert.return\_value = 1  
 mock\_enrich.return\_value = None  
  
 result = ingest\_specific\_activity(session, athlete\_id, activity\_id)  
  
 mock\_service\_instance.client.get\_activity.assert\_called\_once\_with(activity\_id)  
 mock\_upsert.assert\_called\_once\_with(session, athlete\_id, [mock\_activity])  
 mock\_enrich.assert\_called\_once\_with(session, athlete\_id, activity\_id)  
 assert result == 1  
  
  
@patch("src.services.ingestion\_orchestrator\_service.ActivityIngestionService")  
def test\_ingest\_specific\_activity\_not\_found(mock\_service, session):  
 athlete\_id = 123  
 activity\_id = 456  
  
 mock\_service\_instance = mock\_service.return\_value  
 mock\_service\_instance.client.get\_activity.return\_value = None  
  
 result = ingest\_specific\_activity(session, athlete\_id, activity\_id)  
  
 mock\_service\_instance.client.get\_activity.assert\_called\_once\_with(activity\_id)  
 assert result == 0  
  
  
@patch("src.services.ingestion\_orchestrator\_service.enrich\_one\_activity\_with\_refresh")  
@patch("src.services.ingestion\_orchestrator\_service.ActivityDAO.upsert\_activities")  
@patch("src.services.ingestion\_orchestrator\_service.ActivityIngestionService")  
def test\_ingest\_between\_dates\_success(mock\_service, mock\_upsert, mock\_enrich, session):  
 athlete\_id = 123  
 start\_date = datetime(2025, 1, 1)  
 end\_date = datetime(2025, 1, 3)  
  
 mock\_service\_instance = mock\_service.return\_value  
 mock\_activities = [  
 {"id": 1, "name": "A1", "type": "Run"},  
 {"id": 2, "name": "A2", "type": "Run"}  
 ]  
 mock\_service\_instance.client.get\_activities.return\_value = mock\_activities  
  
 mock\_upsert.return\_value = 2  
 mock\_enrich.return\_value = None  
  
 result = ingest\_between\_dates(session, athlete\_id, start\_date, end\_date, batch\_size=1)  
  
 mock\_service\_instance.client.get\_activities.assert\_called\_once()  
 mock\_upsert.assert\_called\_once\_with(session, athlete\_id, mock\_activities)  
 assert mock\_enrich.call\_count == 2  
 assert result == 2  
  
  
@patch("src.services.ingestion\_orchestrator\_service.ActivityIngestionService")  
def test\_ingest\_between\_dates\_no\_activities(mock\_service, session):  
 athlete\_id = 123  
 start\_date = datetime(2025, 1, 1)  
 end\_date = datetime(2025, 1, 3)  
  
 mock\_service\_instance = mock\_service.return\_value  
 mock\_service\_instance.client.get\_activities.return\_value = []  
  
 result = ingest\_between\_dates(session, athlete\_id, start\_date, end\_date)  
  
 mock\_service\_instance.client.get\_activities.assert\_called\_once()  
 assert result == 0  
  
  
@patch("src.services.ingestion\_orchestrator\_service.enrich\_one\_activity\_with\_refresh")  
@patch("src.services.ingestion\_orchestrator\_service.ActivityDAO.upsert\_activities")  
@patch("src.services.ingestion\_orchestrator\_service.ActivityIngestionService")  
def test\_ingest\_between\_dates\_enrichment\_failure(mock\_service, mock\_upsert, mock\_enrich, session):  
 athlete\_id = 123  
 start\_date = datetime(2025, 1, 1)  
 end\_date = datetime(2025, 1, 3)  
  
 mock\_service\_instance = mock\_service.return\_value  
 activities = [  
 {"id": 1, "name": "A1", "type": "Run"},  
 {"id": 2, "name": "A2", "type": "Run"}  
 ]  
 mock\_service\_instance.client.get\_activities.return\_value = activities  
  
 mock\_upsert.return\_value = 2  
  
 def enrich\_side\_effect(sess, ath\_id, act\_id):  
 if act\_id == 2:  
 raise Exception("Enrich error")  
 return None  
  
 mock\_enrich.side\_effect = enrich\_side\_effect  
  
 # Run ingestion; enrichment errors are handled internally, so no exception expected  
 result = ingest\_between\_dates(session, athlete\_id, start\_date, end\_date, batch\_size=1)  
  
 assert mock\_enrich.call\_count == 2 # Both enrichment attempts made  
 assert result == 1 # Upsert count remains 2

### 📄 tests\test\_integration\_ask\_prompt.py

from src.routes.ask\_routes import ask\_bp  
from flask import Flask  
import pytest  
  
@pytest.fixture  
def client():  
 app = Flask(\_\_name\_\_)  
 app.register\_blueprint(ask\_bp)  
 app.config["TESTING"] = True  
 return app.test\_client()  
  
def test\_ask\_endpoint\_formats\_prompt(monkeypatch, client):  
 # Fake activity data  
 mock\_activities = [  
 {"date": "2025-06-26", "distance\_km": 7, "duration\_min": 38},  
 {"date": "2025-06-27", "distance\_km": 5, "duration\_min": 30}  
 ]  
  
 # Patch downstream prompt formatter if wired  
 from src.utils import gpt\_ops  
 monkeypatch.setattr(gpt\_ops, "format\_prompt", lambda q, a: f"MOCK\_PROMPT: {q} / {len(a)} activities")  
  
 response = client.post('/ask', json={  
 "question": "How much did I run?",  
 "athlete\_id": 10  
 })  
  
 assert response.status\_code == 200  
 data = response.get\_json()  
 assert "question" in data  
 assert data["question"].startswith("How much did I run?")

### 📄 tests\test\_integration\_ingestion.py

# tests/test\_integration\_ingestion.py  
  
import pytest  
from datetime import datetime, timedelta  
from unittest.mock import patch  
from src.db.db\_session import get\_session  
from src.services.ingestion\_orchestrator\_service import ingest\_specific\_activity  
from src.db.models.activities import Activity  
from src.db.models.tokens import Token  
  
@pytest.fixture(scope="module")  
def test\_session():  
 session = get\_session()  
 yield session  
 session.rollback()  
 session.close()  
  
@pytest.fixture(scope="function")  
def token\_fixture(test\_session):  
 token = Token(  
 athlete\_id=1,  
 access\_token="test\_token",  
 refresh\_token="test\_refresh",  
 expires\_at=int((datetime.utcnow() + timedelta(hours=1)).timestamp()),  
 )  
 test\_session.add(token)  
 test\_session.commit()  
 yield token  
 test\_session.delete(token)  
 test\_session.commit()  
  
  
@patch("src.services.strava\_access\_service.StravaClient.\_request\_with\_backoff")  
def test\_ingest\_specific\_activity\_integration(mock\_strava\_call, test\_session, token\_fixture):  
 athlete\_id = 1  
 activity\_id = 123456  
  
 mock\_strava\_call.return\_value = {  
 "id": activity\_id,  
 "external\_id": "mock123.fit", # ✅ Required field  
 "name": "Mocked Run",  
 "distance": 5000,  
 "moving\_time": 1800,  
 "elapsed\_time": 1900,  
 "total\_elevation\_gain": 100,  
 "type": "Run",  
 "average\_speed": 2.8,  
 "max\_speed": 3.5,  
 "suffer\_score": 30,  
 "average\_heartrate": 145,  
 "max\_heartrate": 160,  
 "calories": 300,  
 "start\_date": "2025-06-01T08:00:00Z"  
 }  
  
 result = ingest\_specific\_activity(test\_session, athlete\_id, activity\_id)  
  
 assert result == 1  
 activity = test\_session.query(Activity).filter(Activity.activity\_id == activity\_id).first()  
 assert activity is not None  
 assert activity.activity\_id == activity\_id

### 📄 tests\test\_jwt\_utils.py

import pytest  
from unittest.mock import patch, MagicMock  
from flask import Flask, jsonify  
import jwt  
  
import src.utils.jwt\_utils as jwt\_utils  
import src.utils.config as config  
  
  
@pytest.fixture  
def app():  
 app = Flask(\_\_name\_\_)  
 app.config['TESTING'] = True  
 return app  
  
  
def create\_token(payload, expired=False):  
 key = config.SECRET\_KEY  
 if expired:  
 payload['exp'] = 0 # expired  
 else:  
 payload['exp'] = 9999999999  
 return jwt.encode(payload, key, algorithm="HS256")  
  
  
def test\_decode\_token\_valid():  
 token = create\_token({"sub": "user1"})  
 decoded = jwt\_utils.decode\_token(token)  
 assert decoded["sub"] == "user1"  
  
  
def test\_decode\_token\_invalid():  
 with pytest.raises(ValueError):  
 jwt\_utils.decode\_token("not-a-token")  
  
  
def test\_require\_auth\_internal\_key(client, app):  
 app.route("/protected")(jwt\_utils.require\_auth(lambda: jsonify(success=True)))  
  
 # Internal key present and matches config  
 with app.test\_client() as client:  
 headers = {"X-Internal-Key": config.INTERNAL\_API\_KEY}  
 resp = client.get("/protected", headers=headers)  
 assert resp.status\_code == 200  
 assert resp.json == {"success": True}  
  
  
def test\_require\_auth\_missing\_auth\_header(client, app):  
 app.route("/protected")(jwt\_utils.require\_auth(lambda: jsonify(success=True)))  
  
 with app.test\_client() as client:  
 resp = client.get("/protected")  
 assert resp.status\_code == 401  
 assert resp.json["error"] == "Authorization header missing"  
  
  
def test\_require\_auth\_invalid\_auth\_header(client, app):  
 app.route("/protected")(jwt\_utils.require\_auth(lambda: jsonify(success=True)))  
  
 with app.test\_client() as client:  
 headers = {"Authorization": "InvalidToken abc"}  
 resp = client.get("/protected", headers=headers)  
 assert resp.status\_code == 401  
 assert resp.json["error"] == "Authorization header missing"  
  
  
def test\_require\_auth\_expired\_token(client, app):  
 app.route("/protected")(jwt\_utils.require\_auth(lambda: jsonify(success=True)))  
  
 expired\_token = create\_token({"sub": "user1"}, expired=True)  
 auth\_header = f"Bearer {expired\_token}"  
  
 with app.test\_client() as client:  
 resp = client.get("/protected", headers={"Authorization": auth\_header})  
 assert resp.status\_code == 401  
 assert resp.json["error"] == "Token expired"  
  
  
def test\_require\_auth\_invalid\_token(client, app):  
 app.route("/protected")(jwt\_utils.require\_auth(lambda: jsonify(success=True)))  
  
 invalid\_token = "abc.def.ghi"  
 auth\_header = f"Bearer {invalid\_token}"  
  
 with app.test\_client() as client:  
 resp = client.get("/protected", headers={"Authorization": auth\_header})  
 assert resp.status\_code == 401  
 assert resp.json["error"] == "Invalid token"  
  
  
def test\_require\_auth\_valid\_token(client, app):  
 app.route("/protected")(jwt\_utils.require\_auth(lambda: jsonify(success=True)))  
  
 token = create\_token({"sub": "user1"})  
 auth\_header = f"Bearer {token}"  
  
 with app.test\_client() as client:  
 resp = client.get("/protected", headers={"Authorization": auth\_header})  
 assert resp.status\_code == 200  
 assert resp.json == {"success": True}

### 📄 tests\test\_main\_pipeline.py

# tests/test\_main\_pipeline.py  
  
import types  
import pytest  
from unittest.mock import patch, MagicMock  
import src.scripts.main\_pipeline as main\_pipeline  
  
  
def test\_main\_pipeline\_calls\_full\_ingestion(monkeypatch):  
 test\_args = ["main\_pipeline.py", "--athlete\_id", "123", "--lookback\_days", "30"]  
 monkeypatch.setattr("sys.argv", test\_args)  
  
 mock\_token = types.SimpleNamespace(expires\_at=9999999999)  
  
 mock\_filter = MagicMock()  
 mock\_filter.first.return\_value = mock\_token  
 mock\_query = MagicMock()  
 mock\_query.filter.return\_value = mock\_filter  
  
 mock\_session = MagicMock()  
 mock\_session.query.return\_value = mock\_query  
  
 patch\_get\_session = patch("src.scripts.main\_pipeline.get\_session", return\_value=mock\_session)  
 patch\_get\_tokens\_sa = patch("src.scripts.main\_pipeline.get\_tokens\_sa", return\_value=mock\_token)  
 patch\_ingestion = patch("src.scripts.main\_pipeline.run\_full\_ingestion\_and\_enrichment")  
 patch\_refresh\_token = patch("src.scripts.main\_pipeline.refresh\_token\_if\_expired", return\_value=None)  
  
 mock\_get\_session = patch\_get\_session.start()  
 mock\_get\_tokens = patch\_get\_tokens\_sa.start()  
 mock\_ingest = patch\_ingestion.start()  
 patch\_refresh\_token.start()  
  
 try:  
 with pytest.raises(SystemExit):  
 main\_pipeline.main()  
  
 mock\_ingest.assert\_called\_once()  
 args, kwargs = mock\_ingest.call\_args  
 assert args[0] == mock\_session  
 assert args[1] == 123  
 assert kwargs["lookback\_days"] == 30  
 assert kwargs["batch\_size"] == 10  
 finally:  
 patch.stopall()  
  
  
def test\_main\_pipeline\_calls\_specific\_activity(monkeypatch):  
 test\_args = ["main\_pipeline.py", "--athlete\_id", "123", "--activity\_id", "456"]  
 monkeypatch.setattr("sys.argv", test\_args)  
  
 session\_mock = MagicMock()  
 patch\_get\_session = patch("src.scripts.main\_pipeline.get\_session", return\_value=session\_mock)  
 patch\_specific = patch("src.scripts.main\_pipeline.ingest\_specific\_activity")  
  
 mock\_get\_session = patch\_get\_session.start()  
 mock\_ingest = patch\_specific.start()  
  
 try:  
 with pytest.raises(SystemExit):  
 main\_pipeline.main()  
  
 mock\_ingest.assert\_called\_once()  
 args = mock\_ingest.call\_args[0]  
 assert args[0] == session\_mock  
 assert args[1] == 123  
 assert args[2] == 456  
 finally:  
 patch.stopall()  
  
  
def test\_main\_pipeline\_calls\_between\_dates(monkeypatch):  
 test\_args = [  
 "main\_pipeline.py", "--athlete\_id", "123",  
 "--start\_date", "2025-01-01", "--end\_date", "2025-01-05"  
 ]  
 monkeypatch.setattr("sys.argv", test\_args)  
  
 session\_mock = MagicMock()  
 patch\_get\_session = patch("src.scripts.main\_pipeline.get\_session", return\_value=session\_mock)  
 patch\_between = patch("src.scripts.main\_pipeline.ingest\_between\_dates")  
  
 mock\_get\_session = patch\_get\_session.start()  
 mock\_ingest = patch\_between.start()  
  
 try:  
 with pytest.raises(SystemExit):  
 main\_pipeline.main()  
  
 args = mock\_ingest.call\_args[0]  
 assert args[0] == session\_mock  
 assert args[1] == 123  
 assert str(args[2].date()) == "2025-01-01"  
 assert str(args[3].date()) == "2025-01-05"  
 finally:  
 patch.stopall()  
  
  
def test\_main\_pipeline\_handles\_exception(monkeypatch):  
 test\_args = ["main\_pipeline.py", "--athlete\_id", "123"]  
 monkeypatch.setattr("sys.argv", test\_args)  
  
 patch\_get\_session = patch("src.scripts.main\_pipeline.get\_session", return\_value=MagicMock())  
 patch\_ingest = patch("src.scripts.main\_pipeline.run\_full\_ingestion\_and\_enrichment", side\_effect=Exception("fail"))  
  
 patch\_get\_session.start()  
 patch\_ingest.start()  
  
 try:  
 with pytest.raises(SystemExit):  
 main\_pipeline.main()  
 finally:  
 patch.stopall()

### 📄 tests\test\_oauth\_flow.py

import os  
import requests  
from unittest.mock import patch, Mock  
  
  
def test\_oauth\_callback\_missing\_code(client):  
 resp = client.get("/auth/callback")  
 assert resp.status\_code == 400  
  
  
@patch("requests.post")  
def test\_oauth\_callback\_strava\_http\_error(mock\_post, client):  
 mock\_response = Mock()  
 mock\_response.raise\_for\_status.side\_effect = requests.exceptions.HTTPError("Strava error")  
 mock\_response.text = "Bad Request"  
 mock\_post.return\_value = mock\_response  
  
 resp = client.get("/auth/callback?code=badcode")  
 assert resp.status\_code == 502  
  
  
@patch("requests.post")  
def test\_oauth\_callback\_incomplete\_response(mock\_post, client):  
 mock\_response = Mock()  
 mock\_response.raise\_for\_status.return\_value = None  
 mock\_response.json.return\_value = {"athlete": {}} # Simulates missing athlete\_id  
 mock\_post.return\_value = mock\_response  
  
 resp = client.get("/auth/callback?code=incomplete")  
 assert resp.status\_code == 500   
  
  
def test\_oauth\_callback\_missing\_env(monkeypatch, client):  
 monkeypatch.delenv("STRAVA\_CLIENT\_ID", raising=False)  
 monkeypatch.delenv("STRAVA\_CLIENT\_SECRET", raising=False)  
 monkeypatch.delenv("REDIRECT\_URI", raising=False)  
  
 resp = client.get("/auth/callback?code=fakecode")  
 assert resp.status\_code == 502

### 📄 tests\test\_split\_dao.py

# tests/test\_split\_dao.py  
  
import pytest  
from src.db.dao.split\_dao import upsert\_splits  
from src.db.models.splits import Split  
from src.db.models.activities import Activity # ✅ Import Activity to insert FK parent  
  
def test\_upsert\_splits\_basic(sqlalchemy\_session):  
 # ✅ Insert parent Activity row to satisfy ForeignKey constraint  
 sqlalchemy\_session.add(Activity(activity\_id=123, athlete\_id=1))  
 sqlalchemy\_session.commit()  
  
 splits = [  
 {  
 "activity\_id": 123,  
 "lap\_index": 1,  
 "distance": 1000.0,  
 "elapsed\_time": 300,  
 "moving\_time": None,  
 "average\_speed": 3.33,  
 "max\_speed": None,  
 "start\_index": None,  
 "end\_index": None,  
 "split": 1 # Ensure it's stored as INTEGER  
 },  
 {  
 "activity\_id": 123,  
 "lap\_index": 2,  
 "distance": 1000.0,  
 "elapsed\_time": 320,  
 "moving\_time": None,  
 "average\_speed": 3.12,  
 "max\_speed": None,  
 "start\_index": None,  
 "end\_index": None,  
 "split": 2 # Ensure it's stored as INTEGER  
 }  
 ]  
  
 # ✅ Perform the upsert  
 inserted = upsert\_splits(sqlalchemy\_session, splits)  
 assert inserted == 2  
  
 # ✅ Verify inserted rows  
 rows = sqlalchemy\_session.query(Split).filter\_by(activity\_id=123).order\_by(Split.lap\_index).all()  
 assert len(rows) == 2  
 assert rows[0].lap\_index == 1  
 assert rows[0].distance == 1000.0  
 assert rows[0].elapsed\_time == 300  
 assert rows[0].average\_speed == 3.33  
 assert isinstance(rows[0].split, int)

### 📄 tests\test\_split\_upsert\_idempotency.py

# tests/test\_split\_upsert\_idempotency.py  
  
from src.db.models.activities import Activity  
from src.db.dao.split\_dao import upsert\_splits  
from src.db.models.splits import Split  
  
def test\_upsert\_splits\_idempotency(sqlalchemy\_session):  
 # Insert parent activity  
 activity\_id = 55555  
 sqlalchemy\_session.add(Activity(activity\_id=activity\_id, athlete\_id=1))  
 sqlalchemy\_session.commit()  
  
 splits = [  
 {  
 "activity\_id": activity\_id,  
 "lap\_index": 1,  
 "distance": 1000.0,  
 "elapsed\_time": 300,  
 "moving\_time": 290,  
 "average\_speed": 3.3,  
 "max\_speed": 3.5,  
 "start\_index": 0,  
 "end\_index": 299,  
 "split": 1  
 }  
 ]  
  
 # First insert  
 inserted = upsert\_splits(sqlalchemy\_session, splits)  
 assert inserted == 1  
  
 # Second insert (should conflict-update, not duplicate)  
 inserted\_again = upsert\_splits(sqlalchemy\_session, splits)  
 assert inserted\_again == 1  
  
 # Verify only 1 row exists  
 rows = sqlalchemy\_session.query(Split).filter\_by(activity\_id=activity\_id).all()  
 assert len(rows) == 1  
 assert isinstance(rows[0].split, int)

### 📄 tests\test\_splits.py

import pytest  
from sqlalchemy.exc import IntegrityError  
from src.db.db\_session import get\_session  
from src.db.models.splits import Split, upsert\_splits  
from src.db.models.activities import Activity  
  
  
  
@pytest.fixture  
def session():  
 session = get\_session()  
 yield session  
 session.rollback()  
 session.close()  
  
def test\_model\_instantiation():  
 split = Split(  
 activity\_id=123,  
 lap\_index=1,  
 distance=1000.0,  
 elapsed\_time=600,  
 moving\_time=590,  
 average\_speed=3.4,  
 max\_speed=5.0,  
 split=1,  
 average\_heartrate=150.0,  
 pace\_zone=2,  
 conv\_distance=0.62,  
 conv\_avg\_speed=7.6,  
 conv\_moving\_time="9:50",  
 conv\_elapsed\_time="10:00",  
 )  
 assert split.activity\_id == 123  
 assert split.lap\_index == 1  
 assert split.conv\_moving\_time == "9:50"  
  
def test\_upsert\_inserts\_and\_updates(session):  
 # Delete existing activity if present to avoid PK conflict  
 existing = session.query(Activity).filter\_by(activity\_id=1).first()  
 if existing:  
 session.delete(existing)  
 session.commit()  
  
 dummy\_activity = Activity(activity\_id=1, athlete\_id=1, name="Test Activity")  
 session.add(dummy\_activity)  
 session.commit()  
  
 splits\_data = [  
 {"activity\_id": 1, "lap\_index": 1, "distance": 1609.34},  
 {"activity\_id": 1, "lap\_index": 2, "distance": 1609.34},  
 ]  
 upsert\_splits(session, splits\_data)  
  
 result = session.query(Split).filter(Split.activity\_id == 1).all()  
 assert len(result) == 2  
  
def test\_upsert\_empty\_list(session):  
 # Should not raise or commit anything  
 upsert\_splits(session, [])  
 assert True # Passed if no exception  
  
def test\_unique\_constraint(session):  
 existing = session.query(Activity).filter\_by(activity\_id=2).first()  
 if existing:  
 session.delete(existing)  
 session.commit()  
  
 dummy\_activity = Activity(activity\_id=2, athlete\_id=1, name="Another Activity")  
 session.add(dummy\_activity)  
 session.commit()  
  
 split\_data = [{"activity\_id": 2, "lap\_index": 1, "distance": 1000.0}]  
 upsert\_splits(session, split\_data)  
  
 split\_data\_updated = [{"activity\_id": 2, "lap\_index": 1, "distance": 1100.0}]  
 upsert\_splits(session, split\_data\_updated)  
  
 split = session.query(Split).filter\_by(activity\_id=2, lap\_index=1).one()  
 assert split.distance == 1100.0

### 📄 tests\test\_strava\_access\_service.py

import pytest  
from unittest.mock import patch, MagicMock  
from requests.exceptions import HTTPError  
from src.services.strava\_access\_service import StravaClient  
  
@pytest.fixture  
def client():  
 return StravaClient(access\_token="fake-token")  
  
@patch("src.services.strava\_access\_service.requests.request")  
def test\_request\_with\_backoff\_success(mock\_request, client):  
 mock\_resp = MagicMock()  
 mock\_resp.status\_code = 200  
 mock\_resp.json.return\_value = {"data": "ok"}  
 mock\_request.return\_value = mock\_resp  
  
 result = client.\_request\_with\_backoff("GET", "http://test-url")  
 assert result == {"data": "ok"}  
 mock\_request.assert\_called\_once()  
  
@patch("src.services.strava\_access\_service.requests.request")  
@patch("time.sleep", return\_value=None)  
def test\_request\_with\_backoff\_rate\_limit\_retries(mock\_sleep, mock\_request, client):  
 resp\_429 = MagicMock(status\_code=429)  
 resp\_429.raise\_for\_status.side\_effect = None  
 resp\_429.json.return\_value = {}  
  
 resp\_200 = MagicMock(status\_code=200)  
 resp\_200.json.return\_value = {"success": True}  
  
 mock\_request.side\_effect = [resp\_429, resp\_429, resp\_200]  
  
 result = client.\_request\_with\_backoff("GET", "http://test-url")  
 assert result == {"success": True}  
 assert mock\_request.call\_count == 3  
 assert mock\_sleep.call\_count == 2  
  
@patch("src.services.strava\_access\_service.requests.request")  
@patch("time.sleep", return\_value=None)  
def test\_request\_with\_backoff\_max\_retries\_exceeded(mock\_sleep, mock\_request, client):  
 resp\_429 = MagicMock(status\_code=429)  
 resp\_429.raise\_for\_status.side\_effect = None  
 resp\_429.json.return\_value = {}  
  
 mock\_request.return\_value = resp\_429  
  
 with pytest.raises(RuntimeError, match="Exceeded max retries"):  
 client.\_request\_with\_backoff("GET", "http://test-url")  
  
@patch("src.services.strava\_access\_service.requests.request")  
def test\_get\_activities\_pagination\_and\_limit(mock\_request, client):  
 batch1 = [{"id": 1}, {"id": 2}]  
 batch2 = [{"id": 3}]  
 batch3 = []  
  
 mock\_request.side\_effect = [  
 MagicMock(status\_code=200, json=lambda: batch1),  
 MagicMock(status\_code=200, json=lambda: batch2),  
 MagicMock(status\_code=200, json=lambda: batch3),  
 ]  
  
 activities = client.get\_activities(limit=3, per\_page=2)  
 assert len(activities) == 3  
 assert activities[0]["id"] == 1  
 assert mock\_request.call\_count == 2 # Corrected here  
  
@patch("src.services.strava\_access\_service.requests.request")  
def test\_get\_activity\_success(mock\_request, client):  
 expected = {"id": 123}  
 mock\_request.return\_value = MagicMock(status\_code=200, json=lambda: expected)  
  
 activity = client.get\_activity(123)  
 assert activity == expected  
 mock\_request.assert\_called\_once()  
  
@patch("src.services.strava\_access\_service.requests.request")  
def test\_get\_hr\_zones\_success\_and\_404(mock\_request, client):  
 mock\_request.return\_value = MagicMock(status\_code=200, json=lambda: {"zones": []})  
 result = client.get\_hr\_zones(1)  
 assert "zones" in result  
  
 # Simulate 404 HTTPError  
 mock\_resp = MagicMock(status\_code=404)  
 mock\_error = HTTPError(response=mock\_resp)  
 mock\_request.side\_effect = mock\_error  
 result = client.get\_hr\_zones(999)  
 assert result is None  
  
@patch("src.services.strava\_access\_service.requests.request")  
def test\_get\_splits\_success\_and\_404(mock\_request, client):  
 mock\_request.return\_value = MagicMock(status\_code=200, json=lambda: [{"lap": 1}])  
 splits = client.get\_splits(1)  
 assert len(splits) == 1  
  
 mock\_resp = MagicMock(status\_code=404)  
 mock\_error = HTTPError(response=mock\_resp)  
 mock\_request.side\_effect = mock\_error  
 splits = client.get\_splits(999)  
 assert splits == []  
  
@patch("src.services.strava\_access\_service.requests.request")  
def test\_get\_streams\_parsing\_and\_empty(mock\_request, client):  
 resp\_json = {  
 "heartrate": {"data": [100, 101, "102", "abc"]},  
 "cadence": {"data": [80, 81]},  
 "watts": None  
 }  
 mock\_request.return\_value = MagicMock(status\_code=200, json=lambda: resp\_json)  
  
 streams = client.get\_streams(1, ["heartrate", "cadence", "watts"])  
 assert streams["heartrate"] == [100.0, 101.0, 102.0]  
 assert streams["cadence"] == [80.0, 81.0]  
 assert streams["watts"] == []  
  
@patch("src.services.strava\_access\_service.requests.request")  
def test\_get\_streams\_handles\_bad\_data(mock\_request, client):  
 # Simulate bad data causing exception in float conversion  
 resp\_json = {  
 "heartrate": {"data": ["bad", "data", 123]},  
 }  
 mock\_request.return\_value = MagicMock(status\_code=200, json=lambda: resp\_json)  
  
 streams = client.get\_streams(1, ["heartrate"])  
 assert streams["heartrate"] == [123.0]

### 📄 tests\test\_sync.py

from unittest.mock import patch, MagicMock  
import pytest  
import os  
  
@pytest.fixture(scope="session", autouse=True)  
def set\_env\_vars():  
 os.environ["CRON\_SECRET\_KEY"] = "devkey123"  
  
  
  
@patch("src.services.activity\_service.StravaClient")  
@patch("src.services.activity\_service.get\_valid\_token", return\_value="fake-token")  
@patch("src.services.activity\_service.run\_enrichment\_batch", return\_value=1)  
@patch("src.routes.activity\_routes.ActivityIngestionService")  
def test\_sync\_success(mock\_ingestor, mock\_enrich, mock\_token, mock\_strava, client):  
 with patch("src.utils.config.CRON\_SECRET\_KEY", "devkey123"):  
 instance = mock\_ingestor.return\_value  
 instance.ingest\_recent.return\_value = 5  
 mock\_strava.return\_value.get\_activities.return\_value = []  
  
 resp = client.get("/sync/sync/123?key=devkey123")  
 assert resp.status\_code == 200  
 assert resp.json == {"inserted": 5}

### 📄 tests\test\_token\_service.py

import pytest  
from unittest.mock import patch, MagicMock  
from datetime import datetime, timedelta  
import jwt  
from requests.models import Response  
from src.services import token\_service  
from src.db.models.tokens import Token  
  
  
def test\_is\_expired():  
 past = int((datetime.utcnow() - timedelta(seconds=10)).timestamp())  
 future = int((datetime.utcnow() + timedelta(seconds=10)).timestamp())  
 assert token\_service.is\_expired(past) is True  
 assert token\_service.is\_expired(future) is False  
  
  
@patch("src.services.token\_service.get\_tokens\_sa")  
@patch("src.services.token\_service.is\_expired")  
def test\_get\_valid\_token\_success(mock\_is\_expired, mock\_get\_tokens):  
 mock\_session = MagicMock()  
 athlete\_id = 123  
 mock\_get\_tokens.return\_value = {"access\_token": "abc", "expires\_at": 99999999999}  
 mock\_is\_expired.return\_value = False  
  
 token = token\_service.get\_valid\_token(mock\_session, athlete\_id)  
 assert token == "abc"  
  
  
@patch("src.services.token\_service.get\_tokens\_sa", return\_value=None)  
def test\_get\_valid\_token\_no\_tokens(mock\_get\_tokens):  
 mock\_session = MagicMock()  
 with pytest.raises(RuntimeError):  
 token\_service.get\_valid\_token(mock\_session, 123)  
  
  
@patch("src.services.token\_service.get\_tokens\_sa")  
@patch("src.services.token\_service.refresh\_token\_static")  
@patch("src.services.token\_service.insert\_token\_sa")  
def test\_refresh\_access\_token\_success(mock\_insert, mock\_refresh\_static, mock\_get\_tokens):  
 mock\_session = MagicMock()  
 mock\_get\_tokens.return\_value = {"refresh\_token": "old\_refresh"}  
 mock\_refresh\_static.return\_value = {  
 "access\_token": "new\_access",  
 "refresh\_token": "new\_refresh",  
 "expires\_at": 1234567890  
 }  
  
 result = token\_service.refresh\_access\_token(mock\_session, 123)  
 assert result["access\_token"] == "new\_access"  
 mock\_insert.assert\_called\_once()  
  
  
@patch("src.services.token\_service.get\_tokens\_sa", return\_value=None)  
def test\_refresh\_access\_token\_no\_tokens(mock\_get\_tokens):  
 mock\_session = MagicMock()  
 with pytest.raises(RuntimeError):  
 token\_service.refresh\_access\_token(mock\_session, 123)  
  
  
@patch("src.services.token\_service.requests.post")  
def test\_refresh\_token\_static\_success(mock\_post):  
 mock\_resp = MagicMock(spec=Response)  
 mock\_resp.status\_code = 200  
 mock\_resp.json.return\_value = {"access\_token": "access", "refresh\_token": "refresh", "expires\_at": 12345}  
 mock\_post.return\_value = mock\_resp  
  
 tokens = token\_service.refresh\_token\_static("dummy\_refresh")  
 assert "access\_token" in tokens  
  
  
@patch("src.services.token\_service.requests.post")  
def test\_exchange\_code\_for\_token\_success(mock\_post):  
 mock\_resp = MagicMock(spec=Response)  
 mock\_resp.status\_code = 200  
 mock\_resp.json.return\_value = {"access\_token": "access", "refresh\_token": "refresh", "expires\_at": 12345}  
 mock\_post.return\_value = mock\_resp  
  
 tokens = token\_service.exchange\_code\_for\_token("dummy\_code")  
 assert "access\_token" in tokens  
  
  
def test\_refresh\_token\_if\_expired\_true():  
 mock\_session = MagicMock()  
 expired\_token = Token(  
 athlete\_id=123,  
 access\_token="old",  
 refresh\_token="old\_refresh",  
 expires\_at=int((datetime.utcnow() - timedelta(hours=1)).timestamp())  
 )  
 mock\_session.query.return\_value.filter\_by.return\_value.first.return\_value = expired\_token  
  
 with patch("src.services.token\_service.refresh\_token\_static") as mock\_refresh\_static:  
 mock\_refresh\_static.return\_value = {  
 "access\_token": "new\_access",  
 "refresh\_token": "new\_refresh",  
 "expires\_at": int((datetime.utcnow() + timedelta(hours=1)).timestamp())  
 }  
 result = token\_service.refresh\_token\_if\_expired(mock\_session, 123)  
 assert result is True  
  
  
def test\_refresh\_token\_if\_expired\_false():  
 mock\_session = MagicMock()  
 valid\_token = Token(  
 athlete\_id=123,  
 access\_token="abc",  
 refresh\_token="ref",  
 expires\_at=int((datetime.utcnow() + timedelta(hours=1)).timestamp())  
 )  
 mock\_session.query.return\_value.filter\_by.return\_value.first.return\_value = valid\_token  
  
 result = token\_service.refresh\_token\_if\_expired(mock\_session, 123)  
 assert result is False  
  
  
def test\_refresh\_token\_if\_expired\_no\_tokens():  
 mock\_session = MagicMock()  
 mock\_session.query.return\_value.filter\_by.return\_value.first.return\_value = None  
  
 with pytest.raises(ValueError):  
 token\_service.refresh\_token\_if\_expired(mock\_session, 123)  
  
  
def test\_get\_authorization\_url\_valid():  
 url = token\_service.get\_authorization\_url()  
 assert "strava.com/oauth/authorize" in url  
 assert f"client\_id={token\_service.config.STRAVA\_CLIENT\_ID}" in url  
  
  
@patch("src.services.token\_service.insert\_athlete")  
@patch("src.services.token\_service.requests.post")  
@patch("src.services.token\_service.insert\_token\_sa")  
def test\_store\_tokens\_from\_callback(mock\_insert\_token, mock\_post, mock\_insert\_athlete):  
 mock\_session = MagicMock()  
 mock\_response = MagicMock(spec=Response)  
 mock\_response.status\_code = 200  
 mock\_response.json.return\_value = {  
 "athlete": {"id": 123, "firstname": "John", "lastname": "Doe", "email": "john@example.com"},  
 "access\_token": "access", "refresh\_token": "refresh", "expires\_at": 1234567890  
 }  
 mock\_post.return\_value = mock\_response  
  
 athlete\_id = token\_service.store\_tokens\_from\_callback("dummy\_code", mock\_session)  
 assert athlete\_id == 123  
 mock\_insert\_token.assert\_called\_once()  
 mock\_insert\_athlete.assert\_called\_once()  
  
  
  
@patch("src.services.token\_service.get\_session")  
@patch("src.services.token\_service.insert\_token\_sa")  
def test\_login\_user\_success(mock\_insert\_token, mock\_get\_session):  
 mock\_get\_session.return\_value = MagicMock()  
 data = {"username": token\_service.config.ADMIN\_USER, "password": token\_service.config.ADMIN\_PASS}  
  
 access\_token, refresh\_token = token\_service.login\_user(data)  
 assert access\_token and refresh\_token  
 mock\_insert\_token.assert\_called\_once()  
  
  
def test\_login\_user\_invalid\_credentials():  
 with pytest.raises(PermissionError):  
 token\_service.login\_user({"username": "bad", "password": "bad"})  
  
  
@patch("src.services.token\_service.refresh\_token\_static")  
@patch("src.services.token\_service.get\_session")  
@patch("src.services.token\_service.get\_tokens\_sa")  
@patch("src.services.token\_service.jwt.decode")  
def test\_refresh\_token\_success\_basic(mock\_jwt\_decode, mock\_get\_tokens, mock\_get\_session, mock\_refresh\_static):  
 mock\_get\_session.return\_value = MagicMock()  
 mock\_get\_tokens.return\_value = {"refresh\_token": "valid\_token"}  
 mock\_jwt\_decode.return\_value = {"sub": "admin", "type": "refresh"}  
 mock\_refresh\_static.return\_value = {  
 "access\_token": "new\_access",  
 "refresh\_token": "new\_refresh",  
 "expires\_at": 1234567890  
 }  
  
 token = token\_service.refresh\_token("valid\_token")  
 assert token  
  
  
@patch("src.services.token\_service.jwt.decode", side\_effect=jwt.ExpiredSignatureError)  
def test\_refresh\_token\_expired\_signature(mock\_jwt\_decode):  
 with pytest.raises(PermissionError):  
 token\_service.refresh\_token("token")  
  
  
@patch("src.services.token\_service.jwt.decode", side\_effect=jwt.InvalidTokenError)  
def test\_refresh\_token\_invalid\_token(mock\_jwt\_decode):  
 with pytest.raises(PermissionError):  
 token\_service.refresh\_token("token")  
  
  
def test\_delete\_athlete\_tokens():  
 mock\_session = MagicMock()  
 mock\_session.query().filter\_by().delete.return\_value = True  
  
 result = token\_service.delete\_athlete\_tokens(mock\_session, 123)  
 assert result is True  
 mock\_session.commit.assert\_called\_once()  
  
  
def test\_logout\_user\_noop():  
 token\_service.logout\_user("token")

### 📄 tests\utils.py

import jwt  
import datetime  
  
def generate\_test\_token(user\_id, secret\_key, expires\_in=3600):  
 payload = {  
 "sub": user\_id,  
 "exp": datetime.datetime.utcnow() + datetime.timedelta(seconds=expires\_in)  
 }  
 return jwt.encode(payload, secret\_key, algorithm="HS256")

### 📄 validate\_envs.py

# validate\_envs.py  
  
import os  
from dotenv import load\_dotenv  
import psycopg2  
  
REQUIRED\_KEYS = [  
 "SECRET\_KEY", "DATABASE\_URL", "STRAVA\_CLIENT\_ID", "STRAVA\_CLIENT\_SECRET",  
 "STRAVA\_REDIRECT\_URI", "ADMIN\_USER", "ADMIN\_PASS", "CRON\_SECRET\_KEY",  
 "INTERNAL\_API\_KEY", "ACCESS\_TOKEN\_EXP", "REFRESH\_TOKEN\_EXP", "ATHLETE\_ID"  
]  
  
ENV\_FILES = {  
 "development": ".env",  
 "test": ".env.test",  
 "production": ".env.prod"  
}  
  
def validate\_env(env\_name, env\_file):  
 print(f"\nValidating {env\_name.upper()} ({env\_file})")  
 load\_dotenv(dotenv\_path=env\_file, override=True)  
  
 # Check keys  
 missing = [key for key in REQUIRED\_KEYS if not os.getenv(key)]  
 if missing:  
 print(f"❌ Missing keys: {missing}")  
 else:  
 print("✅ All required keys present")  
  
 # DB Connection Test  
 db\_url = os.getenv("DATABASE\_URL")  
  
 # Sanitize for psycopg2  
 if db\_url.startswith("postgresql+psycopg2://"):  
 db\_url = db\_url.replace("postgresql+psycopg2://", "postgresql://")  
  
 try:  
 conn = psycopg2.connect(db\_url)  
 cur = conn.cursor()  
 cur.execute("SELECT 1;")  
 print("✅ Database connection successful")  
 cur.close()  
 conn.close()  
 except Exception as e:  
 print(f"❌ Database connection failed: {e}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 for name, path in ENV\_FILES.items():  
 validate\_env(name, path)

### 📄 wsgi.py

from src.app import create\_app  
app = create\_app()