Zephyr - Modular Monolith Architecture

Why Modular Monolith for Solo Development

Perfect for your situation because:

- ✅ Single deployment and database
- ✅ Keep all the clean architectural boundaries
- ✅ Easy debugging and development
- ✅ Can extract to microservices later if needed
- ✅ Builds on your existing code structure

Architecture Overview

```
┌──────────────────────────────────────────────────────────────────────────────────────
│                               SINGLE FASTAPI APPLICATION
│
│
│
│
┌────────────────────────────────────────────────────────────────────────────────┐
│
│  │                               API LAYER (FastAPI)
│ │
│  │
┌─────────────┬─────────────┬─────────────┬─────────────┬──────────────────┐ │ │
│  │  │    Teams     │   Players    │  Champions   │  Analytics   │       Drafts
│ │ │
│  │  │    Routes    │   Routes     │   Routes     │   Routes     │       Routes
│ │ │
│  │  │              │              │              │              │
│ │ │
│  │  │  GET /teams  │GET /players  │GET /champs   │POST /score   │  POST /draft/pick
│ │ │
│  │  │POST /scout   │POST /verify  │GET /pools    │GET /predict  │  WebSocket /live
│ │ │
│  │
└─────────────┴─────────────┴─────────────┴─────────────┴──────────────────┘ │ │
│
└────────────────────────────────────────────────────────────────────────────────┘

│
│                                        │
│
│

┌────────────────────────────────────────────────────────────────────────────────┐
│
│  │                           SERVICE LAYER (Business Logic)
│ │
│  │
┌─────────────┬─────────────┬─────────────┬─────────────┬──────────────────┐ │ │
│  │  │     Team     │   Player     │  Champion    │ Analytics    │      Draft
│ │ │
│  │  │    Service   │  Service     │  Service     │  Service     │      Service
│ │ │
│  │  │              │              │              │              │
│ │ │
│  │  │ • Team CRUD  │• Player CRUD│• Pool Agg.   │• Rank Score  │  • Live Drafts
```

```
│ │ │
│  │  │ • Rosters   │• Multi-Acct │• Meta Track │• Predictions│ • Pick/Ban Track
│ │ │
│  │  │ • Analysis  │• Linking    │• Role Stats │• Statistics │ • Counter Picks
│ │ │
│  │
└──────────────┴──────────────┴──────────────┴──────────────┴────────────────────┘ │ │
│
└──────────────────────────────────────────────────────────────────────────────────┘
│
│                                          │
│
│
┌──────────────────────────────────────────────────────────────────────────────────┐
│
│  │                        ORCHESTRATION SERVICES
│ │
│  │
┌──────────────────────────────────┬──────────────────────────────────────────────┐ │
│
│  │  │        Report Service           │           Tournament Service
│ │ │
│  │  │                                 │
│ │ │
│  │  │ • Multi-service orchestration   │ • LEPL Logic
│ │ │
│  │  │ • Async report generation       │ • GCS Logic
│ │ │
│  │  │ • Cross-module coordination     │ • Tournament scoring
│ │ │
│  │
└──────────────────────────────────┴──────────────────────────────────────────────┘ │
│
│
└──────────────────────────────────────────────────────────────────────────────────┘
│
│                                          │
│
│
┌──────────────────────────────────────────────────────────────────────────────────┐
│
│  │                        REPOSITORY LAYER (Data Access)
│ │
│  │
```

```
┌──────────────┬──────────────┬──────────────┬──────────────┬──────────────────────┐ │ │
│  │  │    Team      │   Player     │  Champion    │    Rank      │       Draft
│ │ │
│  │  │ Repository   │ Repository   │ Repository   │ Repository   │    Repository
│ │ │
│  │  │              │              │              │              │
│ │ │
│  │  │ • Team CRUD  │• Player CRUD│• Champ Data  │• Rank Data   │ • Draft Sessions
│ │ │
│  │  │ • SQL Logic  │• Account DB  │• SQL Agg.    │• SQL Queries│ • Pick/Ban History
│ │ │
│  │
└──────────────┴──────────────┴──────────────┴──────────────┴──────────────────────┘ │ │
│
└──────────────────────────────────────────────────────────────────────────────────────┘
│
│                                        │
│
│
┌──────────────────────────────────────────────────────────────────────────────────────┐
│
│  │                        EXTERNAL INTEGRATION LAYER
│ │
│  │
┌──────────────┬──────────────┬──────────────┬──────────────┬──────────────────────┐ │ │
│  │  │ Riot API     │   Scrapers   │   Sheets     │    Cache     │    Background
│ │ │
│  │  │   Client     │   Manager    │   Client     │   Manager    │      Tasks
│ │ │
│  │  │              │              │              │              │
│ │ │
│  │  │ • API Calls  │• OP.GG       │• Sheet CRUD  │• Redis       │ • Celery Tasks
│ │ │
│  │  │ • Rate Limit│• LOG         │• Formatting  │• Cache Logic│ • Background Jobs
│ │ │
│  │  │ • Retry      │• Rewind.lol  │• Hyperlinks  │• Invalidate  │ • Async Processing
│ │ │
│  │
└──────────────┴──────────────┴──────────────┴──────────────┴──────────────────────┘ │ │
│
└──────────────────────────────────────────────────────────────────────────────────────┘
│
│                                        │
│
```

```
│
┌──────────────────────────────────────────────────────────────────────────────────┐
│
│   │                              DATABASE LAYER
│ │
│   │                            Single PostgreSQL Database
│ │
│   │
┌──────────────┬──────────────┬──────────────┬──────────────┬──────────────────────┐ │ │
│   │  │   Teams     │   Players   │  Champion   │    Ranks    │      Drafts
│ │ │
│   │  │   Tables    │   Tables    │   Tables    │   Tables    │      Tables
│ │ │
│   │
└──────────────┴──────────────┴──────────────┴──────────────┴──────────────────────┘ │ │
│
└──────────────────────────────────────────────────────────────────────────────────┘
│
└──────────────────────────────────────────────────────────────────────────────────────

┌──────────────────────────────────────────────────────────────────────────────────────
│                            EXTERNAL RESOURCES
│
├──────────────────┬──────────────────┬──────────────────┬──────────────────────────────
│   Redis Cache    │   Celery Queue   │  External APIs   │     File Storage
│
│   (Single)       │   (Single)       │  (Rate Limited)  │     (Processed Data)
│
└──────────────────┴──────────────────┴──────────────────┴──────────────────────────────
```

Project Structure (Building on Your Existing Code)

python

```python
zephyr/
├── app.py                          # FastAPI application entry point
├── config/                         # Configuration (your existing)
├── models/                         # Domain models (your existing DTOs)
│   ├── domain/                     # Rich domain models
│   │   ├── team.py
│   │   ├── player.py
│   │   └── champion_pool.py
│   └── database/                   # Database models (SQLAlchemy)
│       └── tables.py
├── services/                       # Business logic layer (NEW)
│   ├── team_service.py
│   ├── player_service.py
│   ├── champion_service.py
│   ├── analytics_service.py
│   ├── draft_service.py
│   ├── report_service.py           # Orchestrates your gcs_scout_main.py logic
│   └── tournament_service.py       # LEPL/GCS logic
├── repositories/                   # Data access layer (NEW)
│   ├── team_repository.py
│   ├── player_repository.py
│   └── champion_repository.py
├── api/                           # FastAPI routes (NEW)
│   ├── teams.py
│   ├── players.py
│   ├── champions.py
│   ├── analytics.py
│   ├── drafts.py
│   └── reports.py
├── integrations/                  # External services (REFACTORED from modules/)
│   ├── riot_api/                  # Your existing riot_client/
│   ├── scrapers/                  # Your existing scrapers/
│   ├── sheets/                    # Your existing google_client/
│   └── cache/                     # Redis wrapper
├── tasks/                         # Background tasks (NEW)
│   ├── scraping_tasks.py          # Async scraping jobs
│   ├── report_tasks.py            # Async report generation
│   └── data_refresh_tasks.py      # Background data updates
└── migrations/                    # Database migrations
    └── versions/
```

Implementation Strategy (Keeps Your Existing Code)

1. Keep Your Existing Logic, Add Layers

```python
# services/team_service.py (NEW - wraps your existing logic)
from integrations.riot_api.services.account_v1 import ACCOUNT_V1
from repositories.team_repository import TeamRepository

class TeamService:
    def __init__(self, team_repo: TeamRepository, riot_client: ACCOUNT_V1):
        self.team_repo = team_repo
        self.riot_client = riot_client

    async def create_team_roster(self, team_id: str, player_riot_ids: List[str]):
        """
        This wraps your craft_team_rosters.py logic but adds:
        - Database persistence
        - Error handling
        - Return structured data instead of writing to JSON
        """
        team = await self.team_repo.get_by_id(team_id)

        for riot_id in player_riot_ids:
            # Your existing Riot API logic
            game_name, tag_line = riot_id.split("#")
            status_code, account_data = self.riot_client.get_account_by_riot_id(game_n

            # But now store in database instead of JSON
            player = await self.create_or_update_player(account_data)
            await self.team_repo.add_player_to_roster(team_id, player.id)

        return await self.team_repo.get_with_roster(team_id)
```

2. Single FastAPI App with Module Routes

```python
# app.py (NEW)
from fastapi import FastAPI
from api import teams, players, champions, drafts, reports

app = FastAPI(title="Zephyr League Scouting")

# Include all route modules
app.include_router(teams.router, prefix="/api/teams", tags=["teams"])
app.include_router(players.router, prefix="/api/players", tags=["players"])
app.include_router(champions.router, prefix="/api/champions", tags=["champions"])
app.include_router(drafts.router, prefix="/api/drafts", tags=["drafts"])
app.include_router(reports.router, prefix="/api/reports", tags=["reports"])

# Your existing Google Sheets endpoints
@app.post("/api/reports/scouting/{target_team_id}")
async def generate_scouting_report(target_team_id: str, requesting_team_id: str):
    """
    This replaces manually running gcs_scout_main.py
    Same logic, but triggered via API and runs in background
    """
    from services.report_service import ReportService
    from tasks.report_tasks import generate_team_scouting_report

    # Queue background task (your existing logic)
    task = generate_team_scouting_report.delay(target_team_id, requesting_team_id)

    return {
        "task_id": task.id,
        "status": "queued",
        "estimated_completion": "2-3 minutes",
        "check_status_url": f"/api/reports/tasks/{task.id}"
    }
```

3. Background Tasks (Your Long-Running Scripts)

python

```python
# tasks/report_tasks.py (converts your gcs_scout_main.py to Celery task)
from celery import Celery
from services.report_service import ReportService

celery_app = Celery('zephyr')

@celery_app.task
def generate_team_scouting_report(target_team_id: str, requesting_team_id: str):
    """
    This IS your gcs_scout_main.py logic, just wrapped as a Celery task
    Same scraping, same Google Sheets generation, same everything
    """
    report_service = ReportService()

    # Your existing logic but orchestrated through services
    team_data = report_service.get_team_tournament_data(target_team_id)

    # Your existing OP.GG/LOG scraping logic
    scraped_data = report_service.scrape_team_ranks(team_data)

    # Your existing Google Sheets generation
    sheet_url = report_service.generate_google_sheet(target_team_id, scraped_data)

    return {
        "status": "completed",
        "sheet_url": sheet_url,
        "target_team": target_team_id
    }
```

4. Repository Layer (Clean Database Access)

```python
# repositories/team_repository.py (NEW - replaces your JSON files)
from sqlalchemy.ext.asyncio import AsyncSession
from models.database.tables import Team, Player

class TeamRepository:
    def __init__(self, db: AsyncSession):
        self.db = db

    async def get_by_id(self, team_id: str) -> Team:
        # Replace your JSON file loading with database query
        result = await self.db.execute(
            select(Team).where(Team.id == team_id).options(selectinload(Team.players))
        )
        return result.scalar_one_or_none()

    async def save_roster_data(self, team_id: str, roster_data: dict):
        # Replace your JSON file writing with database persistence
        # Your data is preserved, just stored differently
        pass
```

Migration Strategy (Low Risk)

Phase 1: Add Database Layer (Week 1)

- Keep all existing scripts working
- Add PostgreSQL database
- Create repository layer
- Migrate CSV data to database
- **Your scripts still work, just read from DB too**

Phase 2: Add Service Layer (Week 2)

- Extract business logic from scripts into services
- Keep scripts as thin wrappers around services
- **Scripts still work, just call services internally**

Phase 3: Add API Layer (Week 3)

- Add FastAPI with endpoints
- Endpoints call the same services your scripts do

- **Now you have both script access AND web API access**

Phase 4: Add Background Tasks (Week 4)

- Convert long-running scripts to Celery tasks
- **Instead of running gcs_scout_main.py manually, trigger via API**

Phase 5: Add Web UI (Optional - Week 5+)

- React frontend that calls your APIs
- **Scripts, APIs, and Web UI all work together**

Key Benefits Over Microservices

✅ **Single Deployment**: One FastAPI app, easy to deploy ✅ **Database Transactions**: ACID guarantees across all operations
✅ **Easy Debugging**: Everything in one process ✅ **Simple Development**: No network complexity ✅ **Future Proof**: Can extract services later if needed ✅ **Builds on Existing Code**: Minimal rewrites needed

What You Keep vs What Changes

✅ **KEEP (No Changes Needed)**:

- All your Riot API logic
- All your scraping logic
- All your Google Sheets logic
- All your data processing algorithms
- All your League of Legends domain knowledge

🔄 **REFACTOR (Structural Changes)**:

- CSV files → Database tables
- Manual script execution → API endpoints + background tasks
- Direct function calls → Service layer calls
- Scattered code → Organized by domain

You get 90% of the architectural benefits with 10% of the operational complexity. Perfect for solo development!