YTEMPIRE Project Folder Architecture

Version 1.0 - Enterprise-Grade Organization

Table of Contents

- 1. Overview
- 2. Root Directory Structure
- 3. Detailed Folder Breakdown
- 4. Service-Specific Architecture
- 5. Development Guidelines
- 6. File Naming Conventions
- 7. Git Structure and Branching

Overview

This document defines the optimal folder architecture for YTEMPIRE, designed to support a microservices-based, scalable content automation platform. The structure follows industry best practices for separation of concerns, maintainability, and team collaboration.

Key Principles:

- **Domain-Driven Design**: Organized by business domains and bounded contexts
- Microservices Architecture: Clear service boundaries and independent deployability
- Convention over Configuration: Predictable structure across all services
- Environment Separation: Clear distinction between development, staging, and production
- Documentation as Code: All documentation lives alongside the code

Root Directory Structure

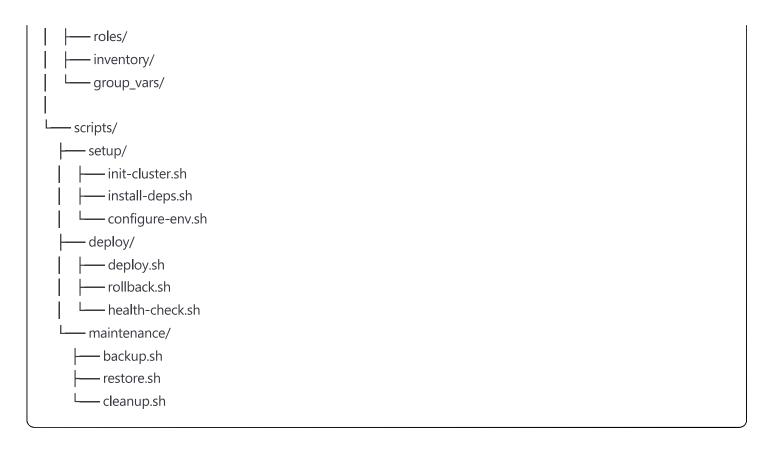




Detailed Folder Breakdown

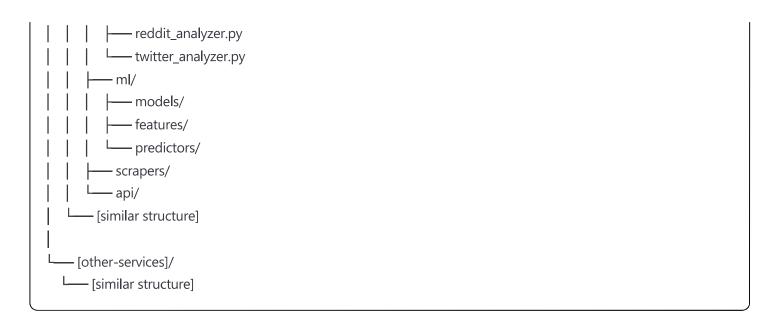
Infrastructure Directory



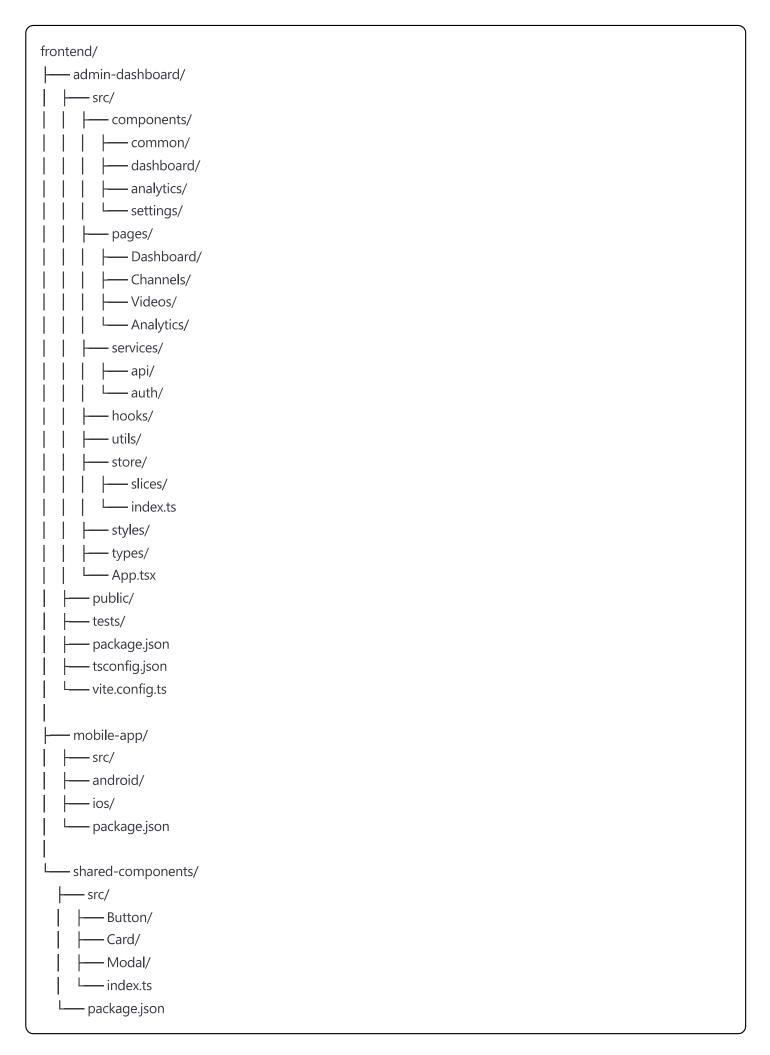


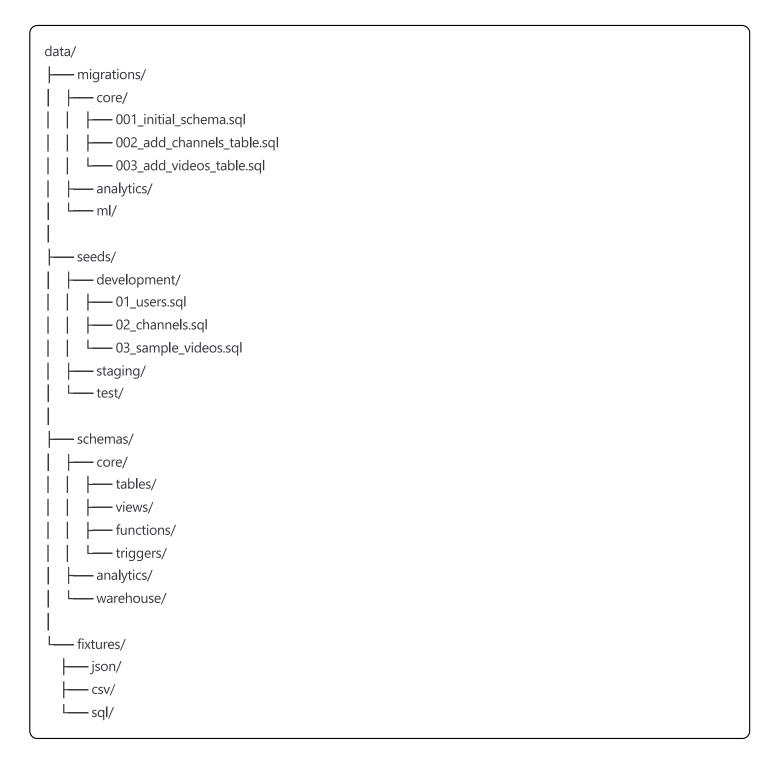
Services Directory (Microservices)

services/
— orchestrator/
middleware/
L—validators/
database.py
— workflow_service.py
scheduling_service.py
Land monitoring_service.py
— models/
workflow.py
│
schedule.py
repositories/
Land main.py
tests/
— integration/
Land fixtures/
— docker/
docker-compose.yml
— config/
— default.yaml
development.yaml
production.yaml
— requirements/
base.txt
dev.txt
prod.txt
env.example
Makefile
README.md
pyproject.toml
trend-analyzer/
analyzers/
— youtube_analyzer.py

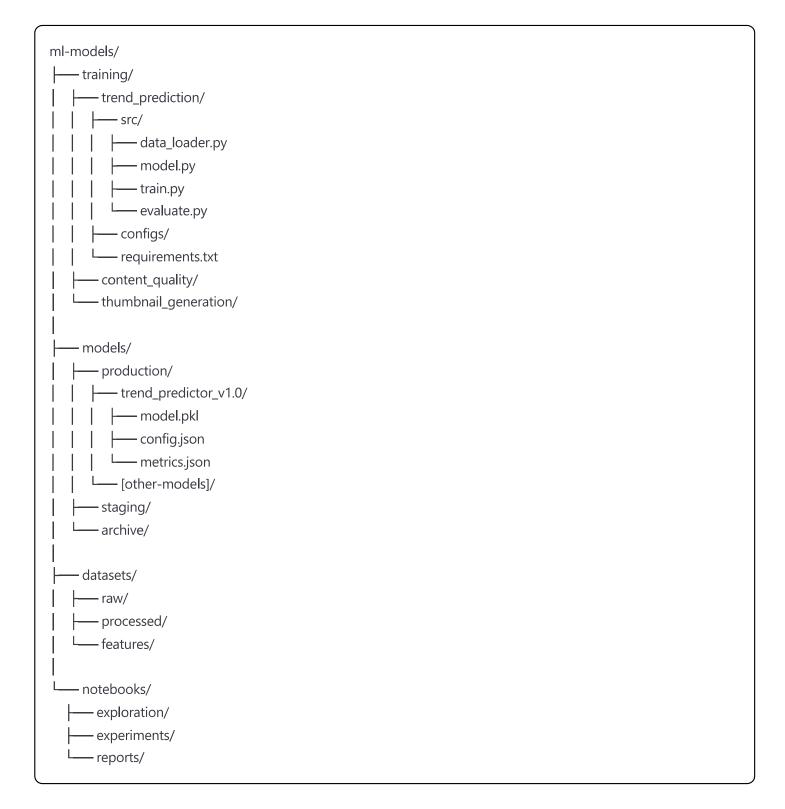


Frontend Directory

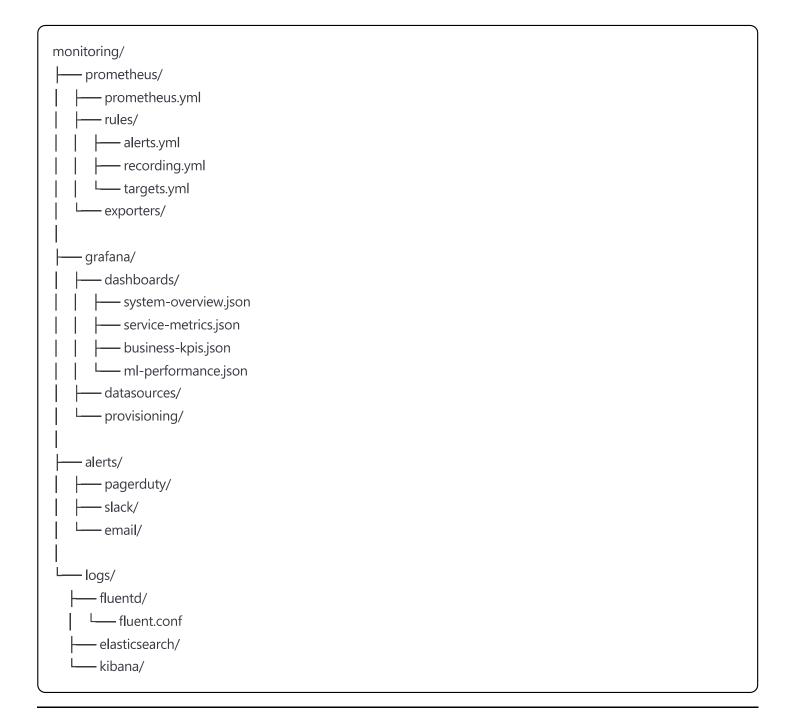




ML Models Directory

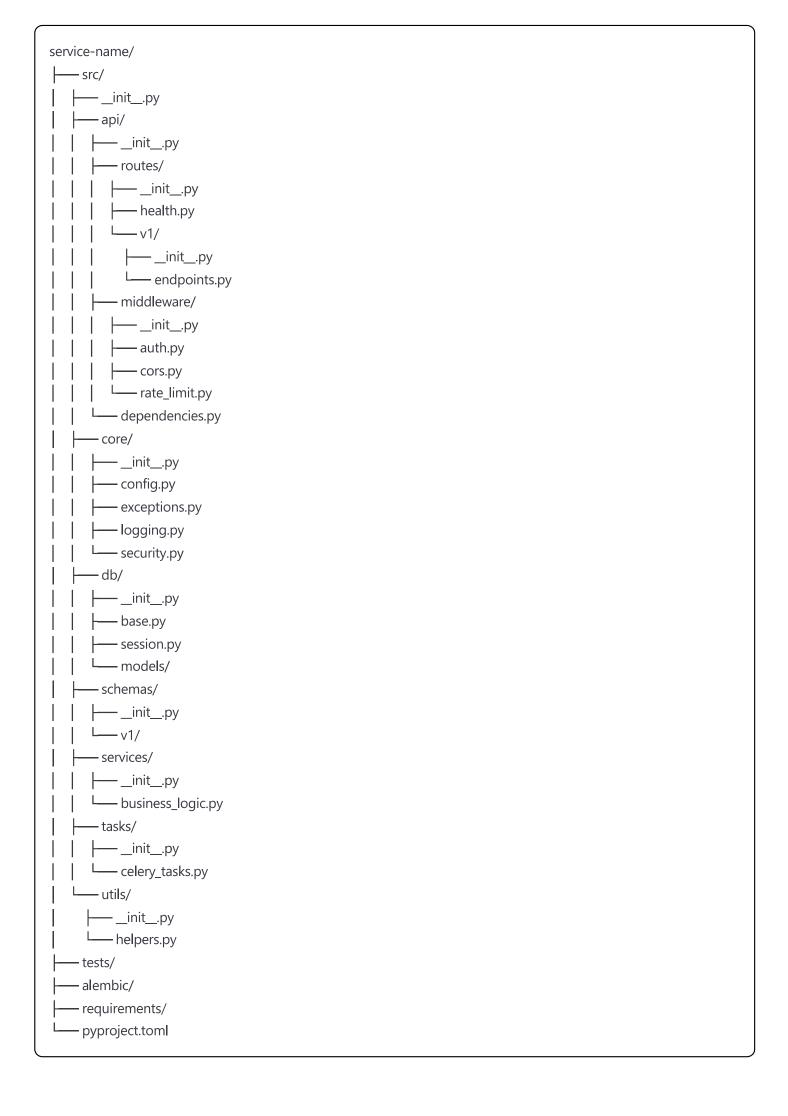


Monitoring Directory

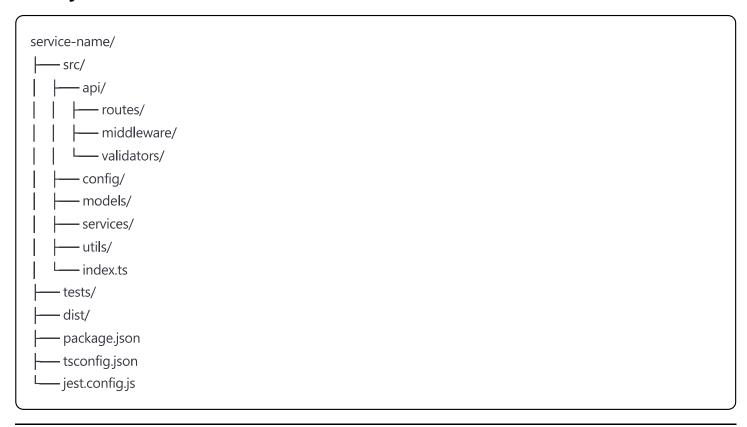


Service-Specific Architecture

Python Service Structure



Node.js Service Structure



Development Guidelines

Environment Configuration

```
# .env.example structure
# Application
APP_NAME=ytempire
APP_ENV=development
APP_PORT=8000
APP_HOST=0.0.0.0
# Database
DATABASE_URL=postgresql://user:pass@localhost:5432/ytempire
DATABASE_POOL_SIZE=20
DATABASE_MAX_OVERFLOW=0
# Redis
REDIS_URL=redis://localhost:6379/0
REDIS_PASSWORD=
# RabbitMQ
RABBITMQ_URL=amqp://user:pass@localhost:5672/
RABBITMQ_VHOST=ytempire
# API Keys
YOUTUBE_API_KEY=
OPENAI_API_KEY=
ANTHROPIC_API_KEY=
ELEVENLABS_API_KEY=
# AWS
AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_REGION=us-east-1
S3_BUCKET_NAME=ytempire-assets
# Monitoring
SENTRY_DSN=
PROMETHEUS PUSHGATEWAY=
# Feature Flags
FEATURE_AI_CONTENT_GEN=true
FEATURE_AUTO_PUBLISH=false
```

Makefile Commands

makefile

```
# Root Makefile example
.PHONY: help setup dev test build deploy clean
help:
 @echo "Available commands:"
 @echo " make setup - Initial project setup"
 @echo " make dev - Start development environment"
 @echo " make test - Run all tests"
 @echo " make build - Build all services"
 @echo " make deploy - Deploy to production"
 @echo " make clean - Clean build artifacts"
setup:
 @echo "Setting up YTEMPIRE development environment..."
 ./infrastructure/scripts/setup/install-deps.sh
 docker-compose pull
 make db-migrate
 make seed-dev
dev:
 docker-compose up -d
 @echo "Development environment started!"
 @echo "Admin Dashboard: http://localhost:3000"
 @echo "API Gateway: http://localhost:8000"
 @echo "Prometheus: http://localhost:9090"
 @echo "Grafana: http://localhost:3001"
test:
 @echo "Running tests..."
 make test-unit
 make test-integration
 make test-e2e
test-unit:
 @for service in services/*; do \
  echo "Testing $$service..."; \
  cd $$service && make test-unit; \
 done
build:
 @echo "Building all services..."
 docker-compose build
deploy:
 @echo "Deploying to production..."
 ./infrastructure/scripts/deploy/deploy.sh production
```

```
clean:
docker-compose down -v
find . -type d -name "__pycache__" -exec rm -rf {} +
find . -type d -name "node_modules" -exec rm -rf {} +
find . -type d -name "dist" -exec rm -rf {} +
```

File Naming Conventions

General Rules

```
yaml
naming_conventions:
 # Files
 python_files: snake_case.py
 typescript_files: camelCase.ts or PascalCase.tsx
 config_files: kebab-case.yaml or snake_case.json
 docker_files: Dockerfile or Dockerfile.service
 env_files: .env.environment
 # Directories
 source dirs: snake case/
 component_dirs: PascalCase/
 config_dirs: kebab-case/
 # Database
 migrations: XXX_description.sql (001_initial_schema.sql)
 models: singular_noun.py (user.py, video.py)
 tables: plural_nouns (users, videos)
 # Tests
 unit_tests: test_module_name.py
 integration_tests: test_integration_feature.py
 # Documentation
 markdown: UPPERCASE.md or kebab-case.md
 architecture: ADR-XXX-title.md
```

Service Naming

yaml

service_naming:

Service names

format: kebab-case

examples:

- trend-analyzer
- content-generator
- media-processor

Docker images

format: organization/service:tag

examples:

- ytempire/orchestrator:1.0.0
- ytempire/orchestrator:latest
- ytempire/orchestrator:dev

Kubernetes resources

deployment: service-name-deployment

service: service-name-service
configmap: service-name-config

secret: service-name-secret

Git Structure and Branching

Repository Structure

yaml

repository_structure:

monorepo: true

advantages:

- Unified versioning
- Simplified dependency management
- Atomic commits across services
- Easier refactoring

tooling:

- Ierna (for JavaScript)
- poetry workspaces (for Python)
- bazel (for multi-language)

Branch Strategy

yaml

git_flow:

main_branches:

main:

- Production-ready code
- Protected branch
- Requires PR and reviews

develop:

- Integration branch
- Next release features
- Base for feature branches

supporting_branches:

feature/*:

- New features
- Branch from: develop
- Merge to: develop
- Example: feature/youtube-analytics

release/*:

- Release preparation
- Branch from: develop
- Merge to: main and develop
- Example: release/1.2.0

hotfix/*:

- Emergency fixes
- Branch from: main
- Merge to: main and develop
- Example: hotfix/api-rate-limit

bugfix/*:

- Non-urgent fixes
- Branch from: develop
- Merge to: develop
- Example: bugfix/video-upload-timeout

Commit Convention

yaml

```
commit_convention:
 format: "<type>(<scope>): <subject>"
 types:
  - feat: New feature
  - fix: Bug fix
  - docs: Documentation
  - style: Code style
  - refactor: Refactoring
  - perf: Performance
  - test: Testing
  - build: Build system
  - ci: CI/CD
  - chore: Maintenance
 examples:
  - "feat(trend-analyzer): add Reddit integration"
  - "fix(publisher): handle YouTube API rate limits"
  - "docs(api): update authentication flow"
  - "perf(media-processor): optimize video encoding"
```

CI/CD Pipeline Structure

```
yaml
.github/workflows/
---- ci.ymI
                  # Continuous Integration
   — cd-staging.yml # Deploy to staging
---- cd-production.yml # Deploy to production
   security-scan.yml # Security scanning

    dependency-check.yml # Dependency updates

   – release.yml
                    # Release automation
pipeline_stages:
 - lint: Code quality checks
 - test: Unit and integration tests
 - build: Docker image building
 - scan: Security vulnerability scanning
 - deploy: Environment deployment
 - smoke: Post-deployment tests
```

Best Practices Summary

1. **Consistency**: Same structure across all services

2. **Isolation**: Clear boundaries between services

3. **Documentation**: README in every directory

4. **Testing**: Tests live alongside code

5. **Configuration**: Environment-specific configs

6. **Security**: Secrets never in code

7. **Monitoring**: Built-in from the start

8. **Automation**: Everything scriptable

This folder architecture provides a solid foundation for YTEMPIRE's growth from MVP to enterprise scale, ensuring maintainability, scalability, and team productivity.

Document Version: 1.0

Last Updated: [Current Date]

Author: Saad T. - Solution Architect