# YTEMPIRE Project Folder Architecture

### Version 1.0 - Enterprise-Grade Organization

---

## Table of Contents



---

## 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

```
ytempire/
├── .github/                    # GitHub specific files
│   ├── workflows/              # GitHub Actions CI/CD
│   ├── ISSUE_TEMPLATE/         # Issue templates
│   ├── PULL_REQUEST_TEMPLATE/  # PR templates
│   └── CODEOWNERS              # Code ownership rules
│
├── infrastructure/             # Infrastructure as Code
│   ├── terraform/              # Terraform configurations
│   ├── kubernetes/             # K8s manifests
│   ├── docker/                 # Docker configurations
│   ├── ansible/                # Configuration management
│   └── scripts/                # Infrastructure scripts
│
├── services/                   # Microservices
│   ├── orchestrator/           # Central orchestration service
│   ├── trend-analyzer/         # Trend analysis service
│   ├── content-generator/      # Content generation service
│   ├── media-processor/        # Media processing service
│   ├── publisher/              # Publishing service
│   ├── analytics/              # Analytics service
│   └── gateway/                # API Gateway
│
├── libraries/                  # Shared libraries
│   ├── python/                 # Python shared libs
│   ├── nodejs/                 # Node.js shared libs
│   └── proto/                  # Protocol definitions
│
├── frontend/                   # Frontend applications
│   ├── admin-dashboard/        # Admin web interface
│   ├── mobile-app/             # Mobile application
│   └── shared-components/      # Shared UI components
│
├── data/                       # Data layer
│   ├── migrations/             # Database migrations
│   ├── seeds/                  # Seed data
│   ├── schemas/                # Schema definitions
│   └── fixtures/               # Test fixtures
│
├── ml-models/                  # Machine Learning
│   ├── training/               # Training scripts
│   ├── models/                 # Trained models
│   ├── datasets/               # Training datasets
│   └── notebooks/              # Jupyter notebooks
│
├── workflows/                  # n8n Workflows
```

```
│   ├── production/          # Production workflows
│   ├── templates/           # Workflow templates
│   └── testing/             # Test workflows
│
├── monitoring/              # Monitoring configs
│   ├── prometheus/          # Prometheus configs
│   ├── grafana/             # Grafana dashboards
│   ├── alerts/              # Alert rules
│   └── logs/                # Log aggregation
│
├── security/                # Security configurations
│   ├── certificates/        # SSL certificates
│   ├── secrets/             # Encrypted secrets
│   ├── policies/            # Security policies
│   └── compliance/          # Compliance docs
│
├── tests/                   # Global test suites
│   ├── integration/         # Integration tests
│   ├── e2e/                 # End-to-end tests
│   ├── performance/         # Performance tests
│   └── security/            # Security tests
│
├── docs/                    # Documentation
│   ├── architecture/        # Architecture docs
│   ├── api/                 # API documentation
│   ├── guides/              # User guides
│   └── decisions/           # ADRs
│
├── tools/                   # Development tools
│   ├── cli/                 # CLI tools
│   ├── generators/          # Code generators
│   └── analyzers/           # Code analyzers
│
├── .env.example             # Environment template
├── .gitignore               # Git ignore rules
├── docker-compose.yml       # Local development
├── docker-compose.prod.yml  # Production config
├── Makefile                 # Build automation
├── README.md                # Project overview
└── LICENSE                  # License file
```

---

# Detailed Folder Breakdown

## Infrastructure Directory

```
infrastructure/
├── terraform/
│   ├── environments/
│   │   ├── dev/
│   │   │   ├── main.tf
│   │   │   ├── variables.tf
│   │   │   └── terraform.tfvars
│   │   ├── staging/
│   │   └── production/
│   ├── modules/
│   │   ├── networking/
│   │   ├── compute/
│   │   ├── database/
│   │   ├── storage/
│   │   └── security/
│   └── global/
│       ├── iam/
│       └── dns/
│
├── kubernetes/
│   ├── base/
│   │   ├── namespace.yaml
│   │   ├── rbac.yaml
│   │   └── network-policies.yaml
│   ├── services/
│   │   ├── orchestrator/
│   │   │   ├── deployment.yaml
│   │   │   ├── service.yaml
│   │   │   ├── configmap.yaml
│   │   │   └── hpa.yaml
│   │   └── [other-services]/
│   ├── monitoring/
│   └── ingress/
│
├── docker/
│   ├── base-images/
│   │   ├── python/
│   │   │   └── Dockerfile
│   │   └── node/
│   │       └── Dockerfile
│   └── compose/
│       ├── development/
│       └── testing/
│
├── ansible/
│   ├── playbooks/
```

```
│   ├── roles/
│   ├── inventory/
│   └── group_vars/
│
└── scripts/
    ├── setup/
    │   ├── init-cluster.sh
    │   ├── install-deps.sh
    │   └── configure-env.sh
    ├── deploy/
    │   ├── deploy.sh
    │   ├── rollback.sh
    │   └── health-check.sh
    └── maintenance/
        ├── backup.sh
        ├── restore.sh
        └── cleanup.sh
```

## Services Directory (Microservices)

```
services/
├── orchestrator/
│   ├── src/
│   │   ├── api/
│   │   │   ├── routes/
│   │   │   ├── middleware/
│   │   │   └── validators/
│   │   ├── core/
│   │   │   ├── config.py
│   │   │   ├── database.py
│   │   │   └── logging.py
│   │   ├── services/
│   │   │   ├── workflow_service.py
│   │   │   ├── scheduling_service.py
│   │   │   └── monitoring_service.py
│   │   ├── models/
│   │   │   ├── workflow.py
│   │   │   ├── task.py
│   │   │   └── schedule.py
│   │   ├── repositories/
│   │   ├── utils/
│   │   └── main.py
│   ├── tests/
│   │   ├── unit/
│   │   ├── integration/
│   │   └── fixtures/
│   ├── docker/
│   │   ├── Dockerfile
│   │   └── 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/
│   ├── src/
│   │   ├── analyzers/
│   │   │   ├── youtube_analyzer.py
```

```
│   │   │   ├── reddit_analyzer.py
│   │   │   └── twitter_analyzer.py
│   │   ├── ml/
│   │   │   ├── models/
│   │   │   ├── features/
│   │   │   └── predictors/
│   │   ├── scrapers/
│   │   └── api/
│   └── [similar structure]
│
└── [other-services]/
    └── [similar structure]
```

## Frontend Directory

```
frontend/
├── admin-dashboard/
│   ├── src/
│   │   ├── components/
│   │   │   ├── common/
│   │   │   ├── dashboard/
│   │   │   ├── analytics/
│   │   │   └── settings/
│   │   ├── pages/
│   │   │   ├── Dashboard/
│   │   │   ├── Channels/
│   │   │   ├── Videos/
│   │   │   └── Analytics/
│   │   ├── services/
│   │   │   ├── api/
│   │   │   └── auth/
│   │   ├── hooks/
│   │   ├── utils/
│   │   ├── store/
│   │   │   ├── slices/
│   │   │   └── index.ts
│   │   ├── styles/
│   │   ├── types/
│   │   └── App.tsx
│   ├── public/
│   ├── tests/
│   ├── package.json
│   ├── tsconfig.json
│   └── vite.config.ts
│
├── mobile-app/
│   ├── src/
│   ├── android/
│   ├── ios/
│   └── package.json
│
└── shared-components/
    ├── src/
    │   ├── Button/
    │   ├── Card/
    │   ├── Modal/
    │   └── index.ts
    └── package.json
```

## Data Directory

```
data/
├── migrations/
│   ├── core/
│   │   ├── 001_initial_schema.sql
│   │   ├── 002_add_channels_table.sql
│   │   └── 003_add_videos_table.sql
│   ├── analytics/
│   └── ml/
│
├── seeds/
│   ├── development/
│   │   ├── 01_users.sql
│   │   ├── 02_channels.sql
│   │   └── 03_sample_videos.sql
│   ├── staging/
│   └── test/
│
├── schemas/
│   ├── core/
│   │   ├── tables/
│   │   ├── views/
│   │   ├── functions/
│   │   └── triggers/
│   ├── analytics/
│   └── warehouse/
│
└── fixtures/
    ├── json/
    ├── csv/
    └── sql/
```

## ML Models Directory

```
ml-models/
├── training/
│   ├── trend_prediction/
│   │   ├── src/
│   │   │   ├── data_loader.py
│   │   │   ├── model.py
│   │   │   ├── train.py
│   │   │   └── evaluate.py
│   │   ├── configs/
│   │   └── requirements.txt
│   ├── content_quality/
│   └── thumbnail_generation/
│
├── models/
│   ├── production/
│   │   ├── trend_predictor_v1.0/
│   │   │   ├── model.pkl
│   │   │   ├── config.json
│   │   │   └── metrics.json
│   │   └── [other-models]/
│   ├── staging/
│   └── archive/
│
├── datasets/
│   ├── raw/
│   ├── processed/
│   └── features/
│
└── notebooks/
    ├── exploration/
    ├── experiments/
    └── reports/
```

## Monitoring Directory

```
monitoring/
├── prometheus/
│   ├── prometheus.yml
│   ├── rules/
│   │   ├── alerts.yml
│   │   ├── recording.yml
│   │   └── targets.yml
│   └── exporters/
│
├── grafana/
│   ├── dashboards/
│   │   ├── system-overview.json
│   │   ├── service-metrics.json
│   │   ├── business-kpis.json
│   │   └── ml-performance.json
│   ├── datasources/
│   └── provisioning/
│
├── alerts/
│   ├── pagerduty/
│   ├── slack/
│   └── email/
│
└── logs/
    ├── fluentd/
    │   └── fluent.conf
    ├── elasticsearch/
    └── kibana/
```

---

# Service-Specific Architecture

## Python Service Structure

```
service-name/
├── src/
│   ├── __init__.py
│   ├── api/
│   │   ├── __init__.py
│   │   ├── routes/
│   │   │   ├── __init__.py
│   │   │   ├── health.py
│   │   │   └── v1/
│   │   │       ├── __init__.py
│   │   │       └── endpoints.py
│   │   ├── middleware/
│   │   │   ├── __init__.py
│   │   │   ├── auth.py
│   │   │   ├── cors.py
│   │   │   └── rate_limit.py
│   │   └── dependencies.py
│   ├── core/
│   │   ├── __init__.py
│   │   ├── config.py
│   │   ├── exceptions.py
│   │   ├── logging.py
│   │   └── security.py
│   ├── db/
│   │   ├── __init__.py
│   │   ├── base.py
│   │   ├── session.py
│   │   └── models/
│   ├── schemas/
│   │   ├── __init__.py
│   │   └── v1/
│   ├── services/
│   │   ├── __init__.py
│   │   └── business_logic.py
│   ├── tasks/
│   │   ├── __init__.py
│   │   └── celery_tasks.py
│   └── utils/
│       ├── __init__.py
│       └── helpers.py
├── tests/
├── alembic/
├── requirements/
└── pyproject.toml
```

### Node.js Service Structure

```
service-name/
├── src/
│   ├── api/
│   │   ├── routes/
│   │   ├── middleware/
│   │   └── validators/
│   ├── config/
│   ├── models/
│   ├── services/
│   ├── utils/
│   └── index.ts
├── tests/
├── dist/
├── package.json
├── tsconfig.json
└── jest.config.js
```

---

# 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

```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

```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

```yaml
repository_structure:
  monorepo: true

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

  tooling:
    - lerna (for JavaScript)
    - poetry workspaces (for Python)
    - bazel (for multi-language)
```

## Branch Strategy

yaml

```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

```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.yml              # 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*