AI Service - Integration Guide

Guide for integrating the AI Service with other Mission Engadi microservices.

Overview

The AI Service integrates with:
- **Auth Service** (port 8001): Authentication and authorization
- **Content Service** (port 8002): Content publishing and management
- **Social Media Service** (port 8009): Social media posting
- **Abacus.AI Platform**: AI model inference

---

1. Authentication Integration

JWT Token Validation

All AI Service endpoints require JWT tokens from the Auth Service.

Flow:
1. Client authenticates with Auth Service
2. Auth Service returns JWT token
3. Client includes token in AI Service requests
4. AI Service validates token

Implementation:

```python
import httpx

# Get token from Auth Service
async def get_auth_token(username: str, password: str) -> str:
    async with httpx.AsyncClient() as client:
        response = await client.post(
            "http://auth-service:8001/api/v1/auth/login",
            json={"username": username, "password": password}
        )
        data = response.json()
        return data["access_token"]

# Use token with AI Service
async def generate_content(token: str, request_data: dict):
    headers = {"Authorization": f"Bearer {token}"}
    async with httpx.AsyncClient() as client:
        response = await client.post(
            "http://ai-service:8010/api/v1/content/generate/social",
            json=request_data,
            headers=headers
        )
        return response.json()
```

---

2. Content Service Integration

Publishing Generated Content

The AI Service can publish generated content directly to the Content Service.

Endpoint: `POST /api/v1/generated/{content_id}/publish`

Example:

```python
import httpx

class ContentServiceClient:
    def __init__(self, base_url: str, auth_token: str):
        self.base_url = base_url
        self.headers = {"Authorization": f"Bearer {auth_token}"}

    async def publish_content(self, content_data: dict) -> dict:
        """Publish content to Content Service."""
        async with httpx.AsyncClient() as client:
            response = await client.post(
                f"{self.base_url}/api/v1/content",
                json=content_data,
                headers=self.headers
            )
            return response.json()

# Usage
client = ContentServiceClient(
    base_url="http://content-service:8002",
    auth_token="your-jwt-token"
)

content = await client.publish_content({
    "title": "Mission Update",
    "content": "Generated content...",
    "category": "mission_updates",
    "language": "en",
    "status": "published"
})
```

---

3. Social Media Service Integration

Posting to Social Media

Publish AI-generated social posts to various platforms.

Flow:
1. Generate social post with AI Service
2. Review/approve content
3. Publish to Social Media Service
4. Social Media Service posts to platforms

Example:

```python
import httpx

class SocialMediaClient:
    def __init__(self, base_url: str, auth_token: str):
        self.base_url = base_url
        self.headers = {"Authorization": f"Bearer {auth_token}"}

    async def post_to_platform(self, post_data: dict) -> dict:
        """Post content to social media platform."""
        async with httpx.AsyncClient() as client:
            response = await client.post(
                f"{self.base_url}/api/v1/posts",
                json=post_data,
                headers=self.headers
            )
            return response.json()

# Complete workflow
async def generate_and_post():
    # 1. Generate content
    ai_response = await ai_service.generate_social_post({
        "topic": "Mission Update",
        "platform": "twitter",
        "tone": "professional",
        "language": "en"
    })

    # 2. Review content (manual or automated)
    if review_content(ai_response["content"]):
        # 3. Post to social media
        social_client = SocialMediaClient(
            base_url="http://social-media-service:8009",
            auth_token="your-jwt-token"
        )

        result = await social_client.post_to_platform({
            "platform": "twitter",
            "content": ai_response["content"],
            "scheduled_time": None  # Post immediately
        })

        return result
```

---

4. Abacus.AI Integration

Configuration

The AI Service uses Abacus.AI for all AI operations.

Environment Variables:

```
ABACUS_API_KEY=your-abacus-api-key
ABACUS_BASE_URL=https://api.abacus.ai
ABACUS_PROJECT_ID=your-project-id
ABACUS_DEPLOYMENT_ID=your-deployment-id
```

Custom AI Models

You can use custom Abacus.AI models:

```python
from app.core.abacus_client import AbacusAIClient

client = AbacusAIClient()

# Use custom model
result = await client.generate_text(
    prompt="Generate mission update",
    model_id="custom-model-id",
    deployment_id="custom-deployment-id"
)
```

5. Service-to-Service Communication

Using Service Integration Client

The AI Service provides a built-in client for service communication:

```python
from app.services.service_integration import ServiceIntegrationClient

client = ServiceIntegrationClient(auth_token="your-jwt-token")

# Publish to Content Service
result = await client.publish_to_content_service({
    "title": "New Article",
    "content": "Generated content..."
})

# Post to Social Media Service
result = await client.post_to_social_media({
    "platform": "twitter",
    "content": "Social post..."
})
```

6. Automation Workflows

End-to-End Automation

Create automated workflows that span multiple services:

Example: Daily Social Media Posts

```python
from app.services.automation_service import AutomationService

automation = AutomationService(db)

# Create workflow
workflow = await automation.create_workflow(
    name="Daily Social Posts",
    workflow_type="content_generation",
    configuration={
        "schedule": "0 9 * * *",  # Daily at 9 AM
        "steps": [
            {
                "action": "generate_social_post",
                "params": {
                    "topic": "daily_mission_update",
                    "platform": "twitter",
                    "language": "en"
                }
            },
            {
                "action": "auto_approve",
                "condition": "quality_score > 0.8"
            },
            {
                "action": "publish_to_social_media",
                "params": {
                    "platforms": ["twitter", "facebook"]
                }
            }
        ]
    },
    user_id="system"
)
```

---

7. Event-Driven Integration

Webhooks

Configure webhooks to receive AI Service events:

Supported Events:
- `content.generated`
- `translation.completed`
- `image.generated`
- `task.approved`
- `workflow.completed`

Configuration:

```python
# In your service
@app.post("/webhooks/ai-service")
async def handle_ai_webhook(event: dict):
    event_type = event["type"]

    if event_type == "content.generated":
        content_id = event["data"]["content_id"]
        # Process generated content
        await process_generated_content(content_id)

    elif event_type == "translation.completed":
        translation_id = event["data"]["translation_id"]
        # Handle completed translation
        await handle_translation(translation_id)

    return {"status": "received"}
```

---

8. Error Handling

Retry Logic

Implement retry logic for resilient integration:

```python
import httpx
import asyncio
from tenacity import retry, stop_after_attempt, wait_exponential

@retry(
    stop=stop_after_attempt(3),
    wait=wait_exponential(multiplier=1, min=2, max=10)
)
async def call_ai_service_with_retry(endpoint: str, data: dict):
    async with httpx.AsyncClient() as client:
        response = await client.post(
            f"http://ai-service:8010{endpoint}",
            json=data,
            headers={"Authorization": f"Bearer {token}"},
            timeout=30.0
        )
        response.raise_for_status()
        return response.json()
```

Circuit Breaker

Implement circuit breaker pattern:

```python
from circuitbreaker import circuit

@circuit(failure_threshold=5, recovery_timeout=60)
async def call_ai_service(endpoint: str, data: dict):
    async with httpx.AsyncClient() as client:
        response = await client.post(
            f"http://ai-service:8010{endpoint}",
            json=data,
            headers={"Authorization": f"Bearer {token}"}
        )
        return response.json()
```

---

9. Testing Integration

Integration Tests

```python
import pytest
from httpx import AsyncClient

@pytest.mark.asyncio
async def test_generate_and_publish_workflow():
    # 1. Authenticate
    auth_token = await get_auth_token("test_user", "password")

    # 2. Generate content
    async with AsyncClient() as client:
        response = await client.post(
            "http://localhost:8010/api/v1/content/generate/social",
            json={
                "topic": "Test Update",
                "platform": "twitter",
                "language": "en"
            },
            headers={"Authorization": f"Bearer {auth_token}"}
        )
        assert response.status_code == 200
        content = response.json()

    # 3. Publish to Content Service
    async with AsyncClient() as client:
        response = await client.post(
            "http://localhost:8002/api/v1/content",
            json={
                "title": "Test",
                "content": content["content"]
            },
            headers={"Authorization": f"Bearer {auth_token}"}
        )
        assert response.status_code == 201
```

---

10. Best Practices

1. Token Management

- Cache tokens and refresh before expiry
- Handle token refresh automatically
- Use refresh tokens for long-running processes

2. Rate Limiting

- Respect rate limits (100 requests/minute)
- Implement exponential backoff
- Use batch endpoints when possible

3. Error Handling

- Always handle timeout errors
- Implement retry logic
- Log all integration errors

4. Monitoring

- Track integration success/failure rates
- Monitor latency between services
- Set up alerts for integration failures

5. Security

- Never log JWT tokens
- Use HTTPS in production
- Validate all responses
- Implement request signing for webhooks

Service URLs (Development)

```
Auth Service:         http://localhost:8001
Content Service:      http://localhost:8002
AI Service:           http://localhost:8010
Social Media Service: http://localhost:8009
```

Service URLs (Production)

```
Auth Service:         https://auth.mission-engadi.org
Content Service:      https://content.mission-engadi.org
AI Service:           https://ai.mission-engadi.org
Social Media Service: https://social.mission-engadi.org
```

Last Updated: December 2024
Version: 1.0.0