Integration Guide for NotebookLM

Fejlesztési és integrációs útmutató

Ez a dokumentum részletes útmutatót ad az API implementációhoz és integrációkhoz, NotebookLM segítségével automatizálható formában.

---

1. Projekt Setup

1.1 Projekt struktúra létrehozása

```
# Projekt struktúra
agentize-platform/
├── agents/
│   ├── meeting_assistant/
│   │   ├── __init__.py
│   │   ├── agent.py
│   │   ├── prompts.py
│   │   ├── processors.py
│   │   └── integrations.py
│   └── pmo_report_generator/
│       ├── __init__.py
│       ├── agent.py
│       ├── prompts.py
│       ├── data_collectors.py
│       └── report_builder.py
├── core/
│   ├── ai_client.py
│   ├── database.py
│   └── storage.py
├── api/
│   └── routes.py
├── tests/
│   ├── test_meeting_assistant.py
│   └── test_pmo_report_generator.py
├── requirements.txt
├── .env.example
└── README.md
```

1.2 Python környezet setup

```
# Virtual environment létrehozása
python -m venv venv

# Aktiválás
# Windows:
venv\Scripts\activate
# Linux/Mac:
source venv/bin/activate

# Függőségek telepítése
pip install -r requirements.txt
```

1.3 Requirements.txt

```
# Core dependencies
fastapi==0.104.1
```

```
uvicorn[standard]==0.24.0
pydantic==2.5.0
python-dotenv==1.0.0

# Database
sqlalchemy==2.0.23
psycopg2-binary==2.9.9
alembic==1.12.1

# AI & HTTP
httpx==0.25.1
openai==1.3.5

# Message Queue
redis==5.0.1
celery==5.3.4

# Storage
boto3==1.29.7
minio==7.2.0

# Integrations
msal==1.24.1  # Microsoft Teams
atlassian-python-api==3.41.5  # Jira

# Utilities
python-multipart==0.0.6
jinja2==3.1.2
reportlab==4.0.7  # PDF generation
openpyxl==3.1.2  # Excel handling
pandas==2.1.3

# Testing
pytest==7.4.3
pytest-asyncio==0.21.1
httpx==0.25.1  # For testing
```

---

2. OpenRouter Integráció

2.1 AI Client implementáció

Fájl: `core/ai_client.py`

```
import os
import httpx
import json
import re
from typing import Dict, Any, Optional
```

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

class OpenRouterClient:
    """
    OpenRouter API client wrapper
    Model: anthropic/claude-3.5-haiku
    """

    def __init__(self):
        self.api_key = os.getenv("OPENROUTER_API_KEY")
        if not self.api_key:
            raise ValueError("OPENROUTER_API_KEY environment variable not set")

        self.model = os.getenv("OPENROUTER_MODEL", "anthropic/claude-3.5-haiku")
        self.base_url = "https://openrouter.ai/api/v1"

        self.client = httpx.AsyncClient(
            base_url=self.base_url,
            headers={
                "Authorization": f"Bearer {self.api_key}",
                "HTTP-Referer": os.getenv("APP_URL", "http://localhost:8000"),
                "X-Title": "Agentize Platform"
            },
            timeout=60.0
        )

    @retry(
        stop=stop_after_attempt(3),
        wait=wait_exponential(multiplier=1, min=2, max=10)
    )
    async def chat_completion(
        self,
        system_prompt: str,
        user_prompt: str,
        temperature: float = 0.7,
        max_tokens: int = 4000
    ) -> Dict[str, Any]:
        """AI kérés küldése OpenRouter-on keresztül"""
        try:
            response = await self.client.post(
                "/chat/completions",
                json={
                    "model": self.model,
                    "messages": [
                        {"role": "system", "content": system_prompt},
                        {"role": "user", "content": user_prompt}
                    ],
                    "temperature": temperature,
                    "max_tokens": max_tokens
                }
            )
            response.raise_for_status()
            return response.json()
        except httpx.HTTPStatusError as e:
            if e.response.status_code == 429:  # Rate limit
                await asyncio.sleep(60)
```

```
                raise
            raise

    async def extract_json(self, response: Dict[str, Any]) -> Dict[str, Any]:
        """JSON kinyerése AI válaszból"""
        content = response["choices"][0]["message"]["content"]

        # JSON parsing (támogatja code block-ot is)
        json_match = re.search(r'```json\n(.*?)\n```', content, re.DOTALL)
        if json_match:
            return json.loads(json_match.group(1))

        # Próbáljuk meg közvetlenül JSON-ként
        try:
            return json.loads(content)
        except json.JSONDecodeError:
            # Ha nem JSON, próbáljuk meg kinyerni a JSON részt
            json_match = re.search(r'\{.*\}', content, re.DOTALL)
            if json_match:
                return json.loads(json_match.group(0))
            raise ValueError("Could not extract JSON from response")

    async def close(self):
        """Client lezárása"""
        await self.client.aclose()
```

---

3. Meeting Assistant Agent

3.1 Prompts

Fájl: agents/meeting_assistant/prompts.py

```
MEETING_ASSISTANT_SYSTEM_PROMPT = """
Te egy professzionális meeting jegyzőkönyv generáló asszisztens vagy.
Feladatod:
1. Átirat elemzése és strukturálása
2. Kulcsfontosságú pontok azonosítása
3. Action item-ek kiemelése (ki, mit, mikorra)
4. Következő lépések javaslása

Kimenet formátum (JSON):
{
  "summary": "Rövid összefoglaló (2-3 mondat)",
  "agenda_items": [
    {"topic": "...", "discussion": "...", "decisions": "..."}
  ],
  "key_points": ["...", "..."],
  "action_items": [
```

```
    {
      "id": "AI-001",
      "description": "...",
      "assignee": "név vagy email",
      "due_date": "YYYY-MM-DD",
      "priority": "high|medium|low"
    }
  ],
  "next_steps": ["...", "..."]
}

Fontos: Mindig JSON formátumban válaszolj, ne szöveges formátumban.
"""

def build_user_prompt(transcript: str, meeting_metadata: Dict[str, Any]) -> str:
    """User prompt építése meeting adatokból"""
    return f"""
Meeting átirat:
{transcript}

Meeting metaadatok:
- Dátum: {meeting_metadata.get('meeting_date', 'N/A')}
- Résztvevők: {', '.join(meeting_metadata.get('participants', []))}
- Cím: {meeting_metadata.get('meeting_title', 'N/A')}
- Időtartam: {meeting_metadata.get('duration_minutes', 'N/A')} perc

Generálj jegyzőkönyvet a fenti formátumban.
"""
```

3.2 Agent implementáció

Fájl: agents/meeting_assistant/agent.py

```
from core.ai_client import OpenRouterClient
from agents.meeting_assistant.prompts import (
    MEETING_ASSISTANT_SYSTEM_PROMPT,
    build_user_prompt
)
from typing import Dict, Any
import logging

logger = logging.getLogger(__name__)

class MeetingAssistantAgent:
    """Meeting Assistant AI Agent"""

    def __init__(self):
        self.ai_client = OpenRouterClient()

    async def process_meeting(
        self,
        transcript: str,
```

```
        meeting_metadata: Dict[str, Any]
    ) -> Dict[str, Any]:
        """Meeting feldolgozása és jegyzőkönyv generálása"""
        try:
            # 1. AI hívás
            user_prompt = build_user_prompt(transcript, meeting_metadata)
            response = await self.ai_client.chat_completion(
                system_prompt=MEETING_ASSISTANT_SYSTEM_PROMPT,
                user_prompt=user_prompt,
                temperature=0.7,
                max_tokens=4000
            )

            # 2. JSON kinyerése
            minutes_data = await self.ai_client.extract_json(response)

            # 3. Validáció
            self._validate_minutes(minutes_data)

            # 4. Action item-ek feldolgozása
            if meeting_metadata.get("create_jira_tickets"):
                await self._create_jira_tickets(minutes_data.get("action_items",
[]))

            return minutes_data

        except Exception as e:
            logger.error(f"Error processing meeting: {e}")
            raise

    def _validate_minutes(self, minutes_data: Dict[str, Any]):
        """Jegyzőkönyv adatok validálása"""
        required_fields = ["summary", "agenda_items", "key_points", "action_items"]
        for field in required_fields:
            if field not in minutes_data:
                raise ValueError(f"Missing required field: {field}")

    async def _create_jira_tickets(self, action_items: list):
        """Jira ticket-ek létrehozása action item-ekből"""
        # Implementáció Jira integrációval
        # Lásd: agents/meeting_assistant/integrations.py
        pass
```

---

4. PMO Report Generator Agent

4.1 Prompts

Fájl: agents/pmo_report_generator/prompts.py

```
PMO_REPORT_SYSTEM_PROMPT = """
Te egy PMO riport generáló asszisztens vagy.
Feladatod:
1. Projekt adatok elemzése (Jira + Excel)
2. Trend azonosítás (késések, túlköltések, erőforrás problémák)
3. Kockázatok prioritizálása
4. Vezetői összefoglaló írása (1-2 oldal, executive summary)

Kimenet formátum (JSON):
{
  "executive_summary": {
    "overview": "Rövid áttekintés (1 bekezdés)",
    "key_metrics": {
      "total_projects": 15,
      "on_track": 10,
      "at_risk": 3,
      "delayed": 2,
      "budget_variance": "+5%"
    },
    "critical_risks": [
      {
        "project": "Projekt neve",
        "risk": "Kockázat leírása",
        "impact": "high|medium|low",
        "mitigation": "Javasolt megoldás"
      }
    ],
    "recommendations": ["...", "..."]
  },
  "detailed_analysis": {
    "trends": ["...", "..."],
    "resource_allocation": {...},
    "budget_analysis": {...}
  }
}

Fontos: Mindig JSON formátumban válaszolj.
"""

def build_report_prompt(jira_data: Dict, budget_data: Dict, period: Dict) -> str:
    """Report prompt építése"""
    return f"""
Projekt adatok (Jira):
{json.dumps(jira_data, indent=2)}

Költségvetés adatok (Excel):
{json.dumps(budget_data, indent=2)}

Időszak: {period['start']} - {period['end']}

Generálj PMO riportot a fenti formátumban.
"""
```

5. API Endpoints

5.1 FastAPI Routes

Fájl: `api/routes.py`

```python
from fastapi import FastAPI, HTTPException, BackgroundTasks
from pydantic import BaseModel
from typing import List, Optional
from agents.meeting_assistant.agent import MeetingAssistantAgent
from agents.pmo_report_generator.agent import PMOReportGeneratorAgent
import uuid

app = FastAPI(title="Agentize Platform API")

class MeetingProcessRequest(BaseModel):
    meeting_id: str
    transcript: str
    participants: List[str]
    meeting_title: str
    meeting_date: str
    create_jira_tickets: bool = False

class ReportGenerateRequest(BaseModel):
    report_type: str  # weekly, monthly, quarterly
    period_start: str
    period_end: str
    jira_projects: List[str]
    excel_files: List[str]
    recipients: List[str]

@app.post("/api/v1/meetings/process")
async def process_meeting(request: MeetingProcessRequest):
    """Meeting feldolgozása"""
    try:
        agent = MeetingAssistantAgent()
        meeting_metadata = {
            "meeting_id": request.meeting_id,
            "participants": request.participants,
            "meeting_title": request.meeting_title,
            "meeting_date": request.meeting_date,
            "create_jira_tickets": request.create_jira_tickets
        }

        minutes_data = await agent.process_meeting(
            request.transcript,
            meeting_metadata
        )

        return {
            "meeting_id": request.meeting_id,
```

```
            "status": "completed",
            "minutes": minutes_data
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@app.post("/api/v1/reports/generate")
async def generate_report(request: ReportGenerateRequest):
    """PMO riport generálása"""
    try:
        agent = PMOReportGeneratorAgent()
        report_data = await agent.generate_report(
            report_type=request.report_type,
            period_start=request.period_start,
            period_end=request.period_end,
            jira_projects=request.jira_projects,
            excel_files=request.excel_files
        )

        report_id = str(uuid.uuid4())

        return {
            "report_id": report_id,
            "status": "completed",
            "report": report_data
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@app.get("/health")
async def health_check():
    """Health check endpoint"""
    return {"status": "healthy"}
```

---

6. Testing

6.1 Meeting Assistant Test

Fájl: `tests/test_meeting_assistant.py`

```
import pytest
from agents.meeting_assistant.agent import MeetingAssistantAgent

@pytest.mark.asyncio
async def test_meeting_assistant_basic():
    agent = MeetingAssistantAgent()
    transcript = """
    Meeting: Test Meeting
```

```
    Participant 1: We need to finish the project by next week.
    Participant 2: I'll handle the frontend part.
    Participant 1: Great, let's meet again on Friday.
    """
    metadata = {
        "meeting_title": "Test Meeting",
        "participants": ["user1@test.com", "user2@test.com"],
        "meeting_date": "2025-01-15T10:00:00Z"
    }

    result = await agent.process_meeting(transcript, metadata)

    assert "summary" in result
    assert "action_items" in result
    assert len(result["action_items"]) > 0
```

---

7. NotebookLM Automatizálás

7.1 Script futtatás NotebookLM-ben

1. **Töltsd be ezt a dokumentumot NotebookLM-be**
2. **Add meg a projekt könyvtárat**
3. **Futtasd a következő parancsokat**:

```
# 1. Projekt struktúra létrehozása
mkdir -p agents/meeting_assistant agents/pmo_report_generator core api tests

# 2. Requirements telepítése
pip install -r requirements.txt

# 3. Environment változók beállítása
cp .env.example .env
# Szerkeszd a .env fájlt a valós értékekkel

# 4. Teszt futtatása
pytest tests/
```

7.2 NotebookLM Prompt példák

"Hozd létre a Meeting Assistant agent.py fájlt a fenti specifikáció alapján"

"Generáld le a PMO Report Generator prompts.py fájlt"

"Készítsd el a FastAPI routes.py fájlt az API endpoint-okkal"

Ez a dokumentum NotebookLM-hez optimalizálva, könnyen feldolgozható és automatizálható formátumban.