agentize.eu PoC — Teljes Technikai Specifikáció

Verzió: 1.0 **Dátum:** 2026-02-26 **Cél:** Vibe-coding reference — minden információ egy helyen a PoC implementálásához **Időkeret:** 10 munkanap

---

1. PROJEKT ÁTTEKINTÉS

1.1 Mi ez?

Enterprise AI platform PoC, amely Microsoft Teams-be (és Telegram-ba) integrált AI ügynököt biztosít, az ügyfél Azure előfizetésében futva, VNET izolációval, EU Data Zone Standard telepítéssel.

1.2 PoC Scope — KRISTÁLYTISZTA

BENNE VAN:

- Azure infrastruktúra Bicep template-ből (1 gombnyomás deploy)
- FastAPI backend + LangGraph agent orchestráció
- Teams Bot + Telegram Bot (Bot Framework, multi-channel)
- Adaptive Cards interakció (többpontos jóváhagyás)
- PDF generálás (az elsődleges output formátum)
- Web app PDF letöltéshez (ha SharePoint integráció nem fér bele)
- Cosmos DB state management + LangGraph checkpointing
- Entra ID app registration + SSO
- EU AI Act átláthatósági jelölés minden AI output-on
- Basic VNET izoláció

NINCS BENNE:

- Azure AI Search / RAG — NEM dokumentumokból dolgozunk, hanem user input-ból
- Hallucináció framework (confidence scoring, golden dataset) — a többpontos jóváhagyási workflow kezeli
- Private Link full hardening (7 pontos checklist)
- TISAX dokumentáció
- React Tab szerkesztő (Adaptive Card elég)
- SharePoint mentés (web app-ból letölthető PDF)
- Multi-tenancy
- Metered billing
- Managed Application wrapper (sima resource group + RBAC)

1.3 Kulcsdöntések (Péter feedback alapján)

1. **Nincs AI Search** — az agent user input-ból dolgozik, nem dokumentum-RAG-ból. Ha később kell RAG, az "bővített csomag".
2. **Nincs külön hallucináció framework** — folyamatos interakció + többpontos jóváhagyás van beépítve a workflow-ba.
3. **PDF kell** — a jelenlegi outputok PDF-ben készülnek, ez a PoC-ban is kell.
4. **Telegram támogatás** — a Bot Framework natívan kezeli, +1 channel bekapcsolás.
5. **Nem Managed App** — PoC-ban sima resource group + explicit RBAC. IP védelem szerződéses.
6. **Token költség → agent vendor** — a platform infra fix, az LLM fogyasztás az agent vendor dolga.

---

2. ARCHITEKTÚRA

2.1 High-Level Architecture

```
┌─────────────────────────────────────────────────────────────┐
│                  AZURE VNET (PoC)                           │
│                 Sweden Central Region                       │
│                                                             │
│  ┌───────────┐    ┌──────────────┐    ┌──────────────┐      │
│  │ Azure     │    │ Container    │    │ Azure AI     │      │
│  │ Bot       │───▶│ App          │───▶│ Foundry      │      │
│  │ Service   │    │ (FastAPI +   │    │ (Mistral/    │      │
│  │           │    │ LangGraph)   │    │ GPT-4o)      │      │
│  └───────────┘    └──────┬───────┘    └──────────────┘      │
│       ▲           │      │                                  │
│       │           ▼      │                                  │
│  ┌────┴──────┐    ┌──────────────┐    ┌──────────────┐      │
│  │ Teams /   │    │ Cosmos DB    │    │ Blob Storage │      │
│  │ Telegram  │    │ (MongoDB     │    │ (PDF output) │      │
│  │ Channels  │    │ API)         │    │              │      │
│  └───────────┘    └──────────────┘    └──────────────┘      │
│                                                             │
│  ┌────────────────┐    ┌────────────────┐                   │
│  │ Key Vault      │    │ App Insights   │                   │
│  │                │    │                │                   │
│  └────────────────┘    └────────────────┘                   │
│                                                             │
│  ┌─────────────────────────────────────────────────────┐    │
│  │             Entra ID (SSO / OBO)                    │    │
│  └─────────────────────────────────────────────────────┘    │
└─────────────────────────────────────────────────────────────┘
```

2.2 Request Flow

```
User (Teams/Telegram)
  │
  ▼
Azure Bot Service (channel routing)
  │
  ▼
FastAPI Backend (Container App)
  │
  ├── Auth: Entra ID token validation
  ├── Request parsing + context extraction
  │
  ▼
LangGraph Agent Graph
  │
  ├── 1. Intent Recognition Node
  │     └── Mi a cél? (generálás, szerkesztés, kérdés)
  │
  ├── 2. Input Processing Node
  │     └── User input strukturálás, validálás
  │
  ├── 3. Generation Node
  │     ├── LLM hívás (AI Foundry → Mistral Large / GPT-4o)
  │     └── "AI által generált tartalom" jelölés hozzáadása
  │
  ├── 4. Review Checkpoint (Human-in-the-loop #1)
  │     ├── Adaptive Card: vázlat megjelenítés
  │     ├── Opciók: Jóváhagyom / Szerkesztés kérem / Elvetés
  │     └── INTERRUPT — vár a user válaszra
  │
  ├── 5. [Ha szerkesztés]: Revision Node
  │     ├── User feedback feldolgozás
  │     ├── Újragenerálás módosításokkal
  │     └── Vissza a Review Checkpoint-hoz
  │
  ├── 6. Final Approval Checkpoint (Human-in-the-loop #2)
  │     ├── Adaptive Card: "Ellenőriztem és jóváhagyom"
  │     └── INTERRUPT — kötelező jóváhagyás
  │
  ├── 7. Output Generation Node
  │     ├── PDF generálás (Jinja2 template → HTML → PDF)
  │     └── Blob Storage-ba mentés
  │
  └── 8. Audit Log Node
```

```
        ├── Cosmos DB: request, LLM version, approval timestamp, user ID
        └── Notification: "A dokumentum elkészült: [letöltés link]"
```

2.3 Data Flow — Részletes

```
1. User → Teams/Telegram: "Készíts egy TWI utasítást a CNC-01 gép beállításáról"

2. Bot Service → FastAPI: POST /api/messages
   {
    "type": "message",
    "text": "Készíts egy TWI utasítást...",
    "from": { "id": "user-entra-id", "name": "Kovács János" },
    "channelId": "msteams" | "telegram",
    "conversation": { "id": "conv-123" }
   }

3. FastAPI → LangGraph: invoke graph with state
   {
    "user_id": "user-entra-id",
    "tenant_id": "tenant-123",
    "channel": "msteams",
    "message": "Készíts egy TWI utasítást...",
    "conversation_id": "conv-123",
    "history": [...previous messages...]
   }

4. LangGraph Intent Node → "generate_twi"

5. LangGraph Generation Node → AI Foundry API:
   POST https://<endpoint>.swedencentral.inference.ai.azure.com/chat/completions
   {
    "model": "mistral-large-latest",
    "messages": [
      {"role": "system", "content": "<TWI system prompt>"},
      {"role": "user", "content": "Készíts egy TWI utasítást..."}
    ],
    "temperature": 0.3,
    "max_tokens": 4000
   }

6. LangGraph Review Checkpoint → INTERRUPT
   Bot sends Adaptive Card to user with draft

7. User approves → LangGraph resumes

8. LangGraph Output Node → PDF generation → Blob Storage
```

```
   Bot sends download link to user

9. Audit log → Cosmos DB
```

---

3. INFRASTRUKTÚRA — BICEP SPECIFIKÁCIÓ

3.1 Resource Group Struktúra (PoC)

```
Resource Group: rg-agentize-poc-swedencentral
├── Virtual Network: vnet-agentize-poc
│   ├── Subnet: snet-container-apps (/23, min 256 IPs)
│   ├── Subnet: snet-private-endpoints (/24)
│   └── NSG: nsg-agentize-poc
├── Container App Environment: cae-agentize-poc
│   └── Container App: ca-agentize-backend (FastAPI + LangGraph)
├── Azure AI Foundry: ai-agentize-poc
│   └── Model Deployment: mistral-large (Data Zone Standard)
├── Cosmos DB Account: cosmos-agentize-poc (MongoDB API)
│   └── Database: agentize-poc-db
│       ├── Collection: conversations
│       ├── Collection: agent_state (LangGraph checkpoints)
│       ├── Collection: audit_log
│       └── Collection: generated_documents
├── Storage Account: stagentizepoc
│   └── Container: pdf-output
├── Key Vault: kv-agentize-poc
├── Application Insights: ai-agentize-poc-insights
├── Azure Bot Service: bot-agentize-poc
│   ├── Channel: Microsoft Teams
│   └── Channel: Telegram
└── Entra ID App Registration: app-agentize-poc
```

3.2 Bicep Paraméterek

```
bicep
```

```
// main.bicep - Paraméterek
@description('Azure region - KIZÁRÓLAG Sweden Central')
param location string = 'swedencentral'

@description('Projekt prefix')
param projectPrefix string = 'agentize-poc'

@description('AI Foundry model')
@allowed(['mistral-large-latest', 'gpt-4o'])
param aiModel string = 'mistral-large-latest'

@description('Container App min replicas (1 = no cold start)')
param minReplicas int = 1

@description('Cosmos DB throughput mode')
@allowed(['serverless', 'provisioned'])
param cosmosThroughputMode string = 'serverless'

@description('Bot Microsoft App ID (Entra ID-ból)')
param botAppId string

@description('Bot Microsoft App Password (Key Vault-ból)')
@secure()
param botAppPassword string

@description('Telegram Bot Token (opcionális)')
@secure()
param telegramBotToken string = ''
```

3.3 VNET Konfiguráció

```
bicep
```

```
resource vnet 'Microsoft.Network/virtualNetworks@2023-09-01' = {
  name: 'vnet-${projectPrefix}'
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: ['10.0.0.0/16']
    }
    subnets: [
      {
        name: 'snet-container-apps'
        properties: {
          addressPrefix: '10.0.0.0/23'  // Min /23 a Container Apps-hez (256 IP)
          delegations: [
            {
              name: 'Microsoft.App.environments'
              properties: {
                serviceName: 'Microsoft.App/environments'
              }
            }
          ]
        }
      }
      {
        name: 'snet-private-endpoints'
        properties: {
          addressPrefix: '10.0.2.0/24'
        }
      }
    ]
  }
}
```

3.4 Container App

```
bicep
```

```
resource containerAppEnv 'Microsoft.App/managedEnvironments@2023-05-01' = {
  name: 'cae-${projectPrefix}'
  location: location
  properties: {
    vnetConfiguration: {
      infrastructureSubnetId: vnet.properties.subnets[0].id
      internal: true  // Csak VNET-en belülről elérhető
    }
    appLogsConfiguration: {
      destination: 'azure-monitor'
    }
  }
}

resource backendApp 'Microsoft.App/containerApps@2023-05-01' = {
  name: 'ca-${projectPrefix}-backend'
  location: location
  properties: {
    managedEnvironmentId: containerAppEnv.id
    configuration: {
      ingress: {
        external: true  // Bot Service-nek el kell érnie
        targetPort: 8000
        transport: 'http'
      }
      secrets: [
        { name: 'ai-foundry-key', keyVaultUrl: '${keyVault.properties.vaultUri}secrets/ai-foundry-key' }
        { name: 'cosmos-connection', keyVaultUrl: '${keyVault.properties.vaultUri}secrets/cosmos-connection' }
        { name: 'bot-app-password', keyVaultUrl: '${keyVault.properties.vaultUri}secrets/bot-app-password' }
        { name: 'blob-connection', keyVaultUrl: '${keyVault.properties.vaultUri}secrets/blob-connection' }
      ]
    }
    template: {
      containers: [
        {
          name: 'backend'
          image: 'ghcr.io/agentize-eu/poc-backend:latest'
          resources: {
            cpu: json('1.0')
            memory: '2Gi'
          }
          env: [
            { name: 'AI_FOUNDRY_ENDPOINT', value: '<AI Foundry endpoint URL>' }
            { name: 'AI_FOUNDRY_KEY', secretRef: 'ai-foundry-key' }
            { name: 'COSMOS_CONNECTION', secretRef: 'cosmos-connection' }
            { name: 'BOT_APP_ID', value: botAppId }
            { name: 'BOT_APP_PASSWORD', secretRef: 'bot-app-password' }
            { name: 'BLOB_CONNECTION', secretRef: 'blob-connection' }
            { name: 'APPLICATIONINSIGHTS_CONNECTION_STRING', value: appInsights.properties.ConnectionString }
          ]
        }
      ]
      scale: {
        minReplicas: minReplicas  // 1 = nincs cold start (~$10/hó)
        maxReplicas: 5
        rules: [
          {
            name: 'http-scaling'
            http: { metadata: { concurrentRequests: '20' } }
          }
        ]
      }
    }
  }
}
```

3.5 Azure AI Foundry

```
bicep
```

```
// Azure AI Foundry (korábban AI Model Catalog)
// KRITIKUS: Data Zone Standard, Sweden Central
resource aiFoundry 'Microsoft.CognitiveServices/accounts@2024-04-01-preview' = {
  name: 'ai-${projectPrefix}'
  location: location  // swedencentral
  kind: 'AIServices'
  sku: { name: 'S0' }
  properties: {
    customSubDomainName: 'ai-${projectPrefix}'
    publicNetworkAccess: 'Enabled'  // PoC-ban OK, MVP-ben Disabled + Private Endpoint
    // Data Zone Standard — EU adatrezidencia garancia
    // FONTOS: Csak Sweden Central és Germany West Central ad szerződéses EU garanciát
  }
}

resource aiDeployment 'Microsoft.CognitiveServices/accounts/deployments@2024-04-01-preview' = {
  parent: aiFoundry
  name: aiModel
  sku: {
    name: 'DataZoneStandard'  // NEM Global Standard!
    capacity: 10  // TPM (tokens per minute) in thousands
  }
  properties: {
    model: {
      format: 'MistralAI'  // vagy 'OpenAI' ha GPT-4o
      name: aiModel
      version: 'latest'
    }
  }
}
```

3.6 Cosmos DB

```
bicep
```

```
resource cosmosAccount 'Microsoft.DocumentDB/databaseAccounts@2023-11-15' = {
  name: 'cosmos-${projectPrefix}'
  location: location
  kind: 'MongoDB'
  properties: {
    databaseAccountOfferType: 'Standard'
    locations: [{ locationName: location, failoverPriority: 0 }]
    capabilities: cosmosThroughputMode == 'serverless'
      ? [{ name: 'EnableServerless' }, { name: 'EnableMongo' }]
      : [{ name: 'EnableMongo' }]
    publicNetworkAccess: 'Enabled'  // PoC-ban OK
    // MongoDB API — kompatibilis a meglévő kódbázissal
  }
}

resource cosmosDb 'Microsoft.DocumentDB/databaseAccounts/mongodbDatabases@2023-11-15' = {
  parent: cosmosAccount
  name: '${projectPrefix}-db'
  properties: {
    resource: { id: '${projectPrefix}-db' }
  }
}

// Collections — lásd a 6. szekciót a sémákhoz
```

3.7 Blob Storage + Key Vault + App Insights

```
bicep
```

```
resource storageAccount 'Microsoft.Storage/storageAccounts@2023-01-01' = {
  name: replace('st${projectPrefix}', '-', '')
  location: location
  sku: { name: 'Standard_LRS' }
  kind: 'StorageV2'
  properties: {
    publicNetworkAccess: 'Enabled'  // PoC-ban OK
    allowBlobPublicAccess: false
    minimumTlsVersion: 'TLS1_2'
  }
}

resource blobContainer 'Microsoft.Storage/storageAccounts/blobServices/containers@2023-01-01' = {
  name: '${storageAccount.name}/default/pdf-output'
  properties: { publicAccess: 'None' }
}

resource keyVault 'Microsoft.KeyVault/vaults@2023-07-01' = {
  name: 'kv-${projectPrefix}'
  location: location
  properties: {
    sku: { family: 'A', name: 'standard' }
    tenantId: subscription().tenantId
    enableRbacAuthorization: true
    publicNetworkAccess: 'Enabled'  // PoC-ban OK
  }
}

resource appInsights 'Microsoft.Insights/components@2020-02-02' = {
  name: 'ai-${projectPrefix}-insights'
  location: location
  kind: 'web'
  properties: {
    Application_Type: 'web'
    RetentionInDays: 30
  }
}
```

3.8 Azure Bot Service

```
bicep
```

```
resource botService 'Microsoft.BotService/botServices@2022-09-15' = {
  name: 'bot-${projectPrefix}'
  location: 'global'  // Bot Service mindig global
  kind: 'azurebot'
  sku: { name: 'S1' }
  properties: {
    displayName: 'agentize.eu PoC Bot'
    description: 'Enterprise AI Platform PoC'
    endpoint: 'https://${backendApp.properties.configuration.ingress.fqdn}/api/messages'
    msaAppId: botAppId
    msaAppType: 'SingleTenant'
    msaAppTenantId: subscription().tenantId
  }
}

// Teams channel
resource teamsChannel 'Microsoft.BotService/botServices/channels@2022-09-15' = {
  parent: botService
  name: 'MsTeamsChannel'
  location: 'global'
  properties: {
    channelName: 'MsTeamsChannel'
    properties: { isEnabled: true }
  }
}

// Telegram channel (opcionális)
resource telegramChannel 'Microsoft.BotService/botServices/channels@2022-09-15' = if (telegramBotToken != '') {
  parent: botService
  name: 'TelegramChannel'
  location: 'global'
  properties: {
    channelName: 'TelegramChannel'
    properties: {
      accessToken: telegramBotToken
      isEnabled: true
    }
  }
}
```

4. BACKEND — FASTAPI + LANGGRAPH

4.1 Projekt Struktúra

```
poc-backend/
├── Dockerfile
├── requirements.txt
├── pyproject.toml
├── .env.example
├── .devcontainer/
│   └── devcontainer.json
│
├── app/
│   ├── __init__.py
│   ├── main.py               # FastAPI app + Bot endpoint
│   ├── config.py             # Environment config (pydantic-settings)
│   │
│   ├── bot/
│   │   ├── __init__.py
│   │   ├── bot_handler.py        # Bot Framework message handler
│   │   ├── adaptive_cards.py     # Adaptive Card JSON templates
│   │   └── teams_helpers.py      # Teams/Telegram specifikus logika
│   │
│   ├── agent/
│   │   ├── __init__.py
│   │   ├── graph.py             # LangGraph graph definition (A FŐ LOGIKA)
│   │   ├── nodes/
│   │   │   ├── __init__.py
│   │   │   ├── intent.py          # Intent recognition node
│   │   │   ├── process_input.py   # User input processing node
│   │   │   ├── generate.py        # LLM generation node
│   │   │   ├── review.py          # Human-in-the-loop review checkpoint
│   │   │   ├── revise.py          # Revision node (user feedback)
│   │   │   ├── approve.py         # Final approval checkpoint
│   │   │   ├── output.py          # PDF generation + storage
│   │   │   └── audit.py           # Audit logging node
│   │   ├── state.py             # LangGraph State definition
│   │   ├── prompts/
│   │   │   ├── twi_system.txt     # TWI system prompt
│   │   │   ├── twi_generate.txt   # TWI generation prompt template
│   │   │   └── intent_classify.txt # Intent classification prompt
│   │   └── tools/
│   │       ├── __init__.py
│   │       └── pdf_generator.py   # PDF generation tool
│   │
│   ├── services/
│   │   ├── __init__.py
│   │   ├── ai_foundry.py         # Azure AI Foundry client
│   │   ├── cosmos_db.py          # Cosmos DB MongoDB client
│   │   ├── blob_storage.py       # Blob Storage client (PDF upload)
│   │   └── key_vault.py          # Key Vault secret client
│   │
│   ├── models/
│   │   ├── __init__.py
│   │   ├── conversation.py       # Conversation data model
│   │   ├── twi_document.py       # TWI document model
│   │   └── audit_entry.py        # Audit log entry model
│   │
│   └── templates/
│       ├── twi_template.html     # Jinja2 HTML template → PDF
│       └── twi_style.css         # PDF styling
│
├── tests/
│   ├── test_graph.py
│   ├── test_generation.py
│   └── test_pdf.py
│
└── infra/
    ├── main.bicep
    ├── parameters.json
    └── deploy.sh
```

4.2 Requirements

```
txt
```

```
# requirements.txt

# Web framework
fastapi==0.115.*
uvicorn[standard]==0.32.*

# Bot Framework
botbuilder-core==4.16.*
botbuilder-integration-aiohttp==4.16.*
aiohttp==3.10.*

# LangGraph + LangChain (CSAK LangGraph-ot használunk)
langgraph==0.3.*
langchain-core==0.3.*
langsmith==0.2.*

# Azure services
azure-identity==1.19.*
azure-keyvault-secrets==4.9.*
azure-storage-blob==12.23.*
azure-ai-inference==1.0.*       # AI Foundry client
pymongo==4.10.*                 # Cosmos DB MongoDB API
motor==3.6.*                    # Async MongoDB driver

# PDF generation
jinja2==3.1.*
weasyprint==63.*                # HTML → PDF

# Config & utils
pydantic==2.10.*
pydantic-settings==2.7.*
python-dotenv==1.0.*

# Observability
opentelemetry-api==1.29.*
opentelemetry-sdk==1.29.*
azure-monitor-opentelemetry==1.6.*

# Dev/test
pytest==8.3.*
pytest-asyncio==0.25.*
httpx==0.28.*                   # Async HTTP test client
```

4.3 Config

```
python
```

```python
# app/config.py
from pydantic_settings import BaseSettings

class Settings(BaseSettings):
    # Azure AI Foundry
    ai_foundry_endpoint: str
    ai_foundry_key: str
    ai_model: str = "mistral-large-latest"
    ai_temperature: float = 0.3
    ai_max_tokens: int = 4000

    # Cosmos DB
    cosmos_connection: str
    cosmos_database: str = "agentize-poc-db"

    # Blob Storage
    blob_connection: str
    blob_container: str = "pdf-output"

    # Bot Framework
    bot_app_id: str
    bot_app_password: str

    # Application Insights
    applicationinsights_connection_string: str = ""

    # App config
    environment: str = "poc"
    log_level: str = "INFO"

    class Config:
        env_file = ".env"
        case_sensitive = False

settings = Settings()
```

4.4 FastAPI Main

```
python
```

```python
# app/main.py
import logging
from fastapi import FastAPI, Request, Response
from botbuilder.core import BotFrameworkAdapter, BotFrameworkAdapterSettings
from botbuilder.schema import Activity

from app.config import settings
from app.bot.bot_handler import AgentizeBotHandler

# Logging
logging.basicConfig(level=settings.log_level)
logger = logging.getLogger(__name__)

# FastAPI app
app = FastAPI(
    title="agentize.eu PoC Backend",
    version="0.1.0",
    docs_url="/docs" if settings.environment == "poc" else None,
)

# Bot Framework adapter
adapter_settings = BotFrameworkAdapterSettings(
    app_id=settings.bot_app_id,
    app_password=settings.bot_app_password,
)
adapter = BotFrameworkAdapter(adapter_settings)

# Bot handler
bot = AgentizeBotHandler()

# Error handler
async def on_error(context, error):
    logger.error(f"Bot error: {error}", exc_info=True)
    await context.send_activity("Hiba történt. Kérlek próbáld újra.")

adapter.on_turn_error = on_error


@app.post("/api/messages")
async def messages(request: Request):
    """Bot Framework messaging endpoint"""
    body = await request.json()
    activity = Activity().deserialize(body)
    auth_header = request.headers.get("Authorization", "")

    response = await adapter.process_activity(
        activity, auth_header, bot.on_turn
    )

    if response:
        return Response(
            content=response.body,
            status_code=response.status,
            headers=dict(response.headers) if response.headers else {},
        )
    return Response(status_code=200)


@app.get("/health")
async def health():
    return {"status": "healthy", "environment": settings.environment}


@app.get("/")
async def root():
    return {"service": "agentize.eu PoC Backend", "version": "0.1.0"}
```

4.5 Bot Handler

```
python
```

```python
# app/bot/bot_handler.py
import logging
from botbuilder.core import ActivityHandler, TurnContext, CardFactory
from botbuilder.schema import ActivityTypes, Attachment

from app.agent.graph import create_agent_graph, run_agent
from app.bot.adaptive_cards import (
    create_review_card,
    create_approval_card,
    create_result_card,
    create_welcome_card,
)
from app.services.cosmos_db import ConversationStore

logger = logging.getLogger(__name__)


class AgentizeBotHandler(ActivityHandler):
    def __init__(self):
        self.graph = create_agent_graph()
        self.conversation_store = ConversationStore()

    async def on_message_activity(self, turn_context: TurnContext):
        """Bejövő üzenet feldolgozás"""
        user_id = turn_context.activity.from_property.id
        conversation_id = turn_context.activity.conversation.id
        channel_id = turn_context.activity.channel_id
        text = turn_context.activity.text or ""
        value = turn_context.activity.value  # Adaptive Card submit value

        logger.info(f"Message from {user_id} on {channel_id}: {text[:50]}...")

        # Adaptive Card válasz (jóváhagyás / szerkesztés kérés / elvetés)
        if value:
            await self._handle_card_action(turn_context, value, conversation_id, user_id)
            return

        # Normál szöveges üzenet → Agent graph futtatás
        await self._handle_text_message(turn_context, text, conversation_id, user_id, channel_id)

    async def _handle_text_message(self, turn_context, text, conversation_id, user_id, channel_id):
        """Szöveges üzenet → LangGraph agent"""
        # "Gondolkozom..." jelzés
        await turn_context.send_activity("⏳ Feldolgozom a kérésedet...")

        # Agent futtatás
        result = await run_agent(
            graph=self.graph,
            message=text,
            user_id=user_id,
            conversation_id=conversation_id,
            channel=channel_id,
        )

        # Eredmény kezelése
        if result["status"] == "review_needed":
            # Vázlat kész → review Adaptive Card küldése
            card = create_review_card(
                draft=result["draft"],
                metadata=result["metadata"],
            )
            await turn_context.send_activity(
                Activity(
                    type=ActivityTypes.message,
                    attachments=[CardFactory.adaptive_card(card)],
                )
            )

        elif result["status"] == "clarification_needed":
            # Nem egyértelmű input → kérdés vissza
            await turn_context.send_activity(result["message"])

        elif result["status"] == "error":
            await turn_context.send_activity(f"❌ Hiba: {result['message']}")

    async def _handle_card_action(self, turn_context, value, conversation_id, user_id):
        """Adaptive Card válasz feldolgozás"""
        action = value.get("action")

        if action == "approve_draft":
            # Review #1 — vázlat jóváhagyva → Final approval card
            card = create_approval_card(
                draft=value.get("draft"),
                metadata=value.get("metadata"),
            )
            await turn_context.send_activity(
                Activity(
                    type=ActivityTypes.message,
                    attachments=[CardFactory.adaptive_card(card)],
                )
            )

        elif action == "request_edit":
```

```
        # Szerkesztés kérés → Revision node
        feedback = value.get("feedback", "")
        await turn_context.send_activity("⏳ Módosítom a szerkesztési kérésed alapján...")

        result = await run_agent(
            graph=self.graph,
            message=feedback,
            user_id=user_id,
            conversation_id=conversation_id,
            resume_from="revision",
            context=value,
        )

        card = create_review_card(
            draft=result["draft"],
            metadata=result["metadata"],
        )
        await turn_context.send_activity(
            Activity(
                type=ActivityTypes.message,
                attachments=[CardFactory.adaptive_card(card)],
            )
        )

    elif action == "final_approve":
        # Final approval → PDF generálás
        await turn_context.send_activity("⏳ PDF generálás folyamatban...")

        result = await run_agent(
            graph=self.graph,
            message="",
            user_id=user_id,
            conversation_id=conversation_id,
            resume_from="output",
            context=value,
        )

        card = create_result_card(
            pdf_url=result["pdf_url"],
            document_title=result["title"],
            metadata=result["metadata"],
        )
        await turn_context.send_activity(
            Activity(
                type=ActivityTypes.message,
                attachments=[CardFactory.adaptive_card(card)],
            )
        )

    elif action == "reject":
        await turn_context.send_activity("🗑️ Elvettem a vázlatot. Új kéréssel indíthatsz újat.")

async def on_members_added_activity(self, members_added, turn_context: TurnContext):
    """Üdvözlő üzenet új member-nek"""
    for member in members_added:
        if member.id != turn_context.activity.recipient.id:
            card = create_welcome_card()
            await turn_context.send_activity(
                Activity(
                    type=ActivityTypes.message,
                    attachments=[CardFactory.adaptive_card(card)],
                )
            )
```

4.6 LangGraph Agent Graph

```
python
```

```python
# app/agent/state.py
from typing import TypedDict, Literal, Optional, List, Any
from langgraph.graph import MessagesState


class AgentState(TypedDict):
    """LangGraph agent state"""
    # Input
    user_id: str
    tenant_id: str
    conversation_id: str
    channel: str
    message: str

    # Processing
    intent: Optional[str]  # "generate_twi", "edit_twi", "question", "unknown"
    processed_input: Optional[dict]
    draft: Optional[str]
    draft_metadata: Optional[dict]

    # Revision loop
    revision_feedback: Optional[str]
    revision_count: int

    # Output
    status: str  # "review_needed", "approved", "completed", "error"
    pdf_url: Optional[str]
    pdf_blob_name: Optional[str]

    # Audit
    llm_model: Optional[str]
    llm_tokens_used: Optional[int]
    approval_timestamp: Optional[str]

    # History
    messages: List[Any]
```

python

```python
# app/agent/graph.py
import logging
from langgraph.graph import StateGraph, END
from langgraph.checkpoint.memory import MemorySaver  # PoC-ban in-memory, prod-ban Cosmos DB

from app.agent.state import AgentState
from app.agent.nodes.intent import intent_node
from app.agent.nodes.process_input import process_input_node
from app.agent.nodes.generate import generate_node
from app.agent.nodes.review import review_node
from app.agent.nodes.revise import revise_node
from app.agent.nodes.approve import approve_node
from app.agent.nodes.output import output_node
from app.agent.nodes.audit import audit_node

logger = logging.getLogger(__name__)


def should_generate(state: AgentState) -> str:
    """Intent alapján routing"""
    intent = state.get("intent", "unknown")
    if intent in ("generate_twi", "edit_twi"):
        return "process_input"
    elif intent == "question":
        return "generate"  # Egyszerű Q&A, nincs input processing
    else:
        return "clarify"   # Nem egyértelmű → kérdezzünk vissza


def after_review(state: AgentState) -> str:
    """Review után routing"""
    status = state.get("status", "")
    if status == "approved":
        return "approve"
    elif status == "revision_requested":
        return "revise"
    else:
        return "end"  # rejected


def after_revision(state: AgentState) -> str:
    """Revision után routing — max 3 kör"""
    if state.get("revision_count", 0) >= 3:
        return "approve"  # Forced approval after 3 rounds
    return "review"
```

```python
def create_agent_graph() -> StateGraph:
    """LangGraph agent gráf felépítés"""

    # Gráf definíció
    builder = StateGraph(AgentState)

    # Node-ok hozzáadása
    builder.add_node("intent", intent_node)
    builder.add_node("process_input", process_input_node)
    builder.add_node("generate", generate_node)
    builder.add_node("review", review_node)       # Human-in-the-loop #1
    builder.add_node("revise", revise_node)
    builder.add_node("approve", approve_node)      # Human-in-the-loop #2
    builder.add_node("output", output_node)        # PDF generation
    builder.add_node("audit", audit_node)

    # Edge-ek (flow)
    builder.set_entry_point("intent")
    builder.add_conditional_edges("intent", should_generate)
    builder.add_edge("process_input", "generate")
    builder.add_edge("generate", "review")

    # Review → conditional
    builder.add_conditional_edges("review", after_review)

    # Revision loop
    builder.add_edge("revise", "generate")  # Újragenerálás a módosításokkal

    # Approve → Output → Audit → END
    builder.add_edge("approve", "output")
    builder.add_edge("output", "audit")
    builder.add_edge("audit", END)

    # Checkpointer (PoC: in-memory; Prod: Cosmos DB)
    checkpointer = MemorySaver()

    return builder.compile(
        checkpointer=checkpointer,
        interrupt_before=["review", "approve"],  # Human-in-the-loop breakpoints
    )


async def run_agent(
    graph,
    message: str,
    user_id: str,
    conversation_id: str,
    channel: str = "msteams",
    resume_from: str = None,
    context: dict = None,
) -> dict:
    """Agent futtatás vagy folytatás"""

    config = {"configurable": {"thread_id": conversation_id}}

    if resume_from:
        # Folytatás interrupt utánról
        state_update = _build_resume_state(resume_from, context)
        result = await graph.ainvoke(state_update, config)
    else:
        # Új futtatás
        initial_state = AgentState(
            user_id=user_id,
            tenant_id="poc-tenant",  # PoC-ban fix
            conversation_id=conversation_id,
            channel=channel,
            message=message,
            revision_count=0,
            status="processing",
            messages=[],
        )
        result = await graph.ainvoke(initial_state, config)

    return result


def _build_resume_state(resume_from: str, context: dict) -> dict:
    """Interrupt utáni state update"""
    if resume_from == "revision":
        return {
            "status": "revision_requested",
            "revision_feedback": context.get("feedback", ""),
        }
    elif resume_from == "output":
        return {
            "status": "approved",
            "approval_timestamp": context.get("timestamp"),
        }
    return {}
```

4.7 Agent Node-ok — Implementáció

python

```python
# app/agent/nodes/intent.py
from app.agent.state import AgentState
from app.services.ai_foundry import call_llm

INTENT_PROMPT = """Te az agentize.eu AI platform intent felismerő modulja vagy.
Osztályozd a felhasználó kérését az alábbi kategóriák egyikébe:

- generate_twi: Új TWI (Training Within Industry) utasítás generálása
- edit_twi: Meglévő TWI szerkesztése, módosítása
- question: Általános kérdés a rendszerről vagy a folyamatokról
- unknown: Nem egyértelmű, kérdezzünk vissza

VÁLASZOLJ KIZÁRÓLAG az intent nevével, semmi mással.

Felhasználó üzenete: {message}"""

async def intent_node(state: AgentState) -> AgentState:
    """Szándék felismerés"""
    response = await call_llm(
        prompt=INTENT_PROMPT.format(message=state["message"]),
        temperature=0.1,
        max_tokens=20,
    )
    intent = response.strip().lower()
    if intent not in ("generate_twi", "edit_twi", "question", "unknown"):
        intent = "unknown"

    return {**state, "intent": intent}
```

python

```python
# app/agent/nodes/generate.py
from app.agent.state import AgentState
from app.services.ai_foundry import call_llm

TWI_SYSTEM_PROMPT = """Te az agentize.eu TWI (Training Within Industry) generátor modulja vagy.

FELADATOD:
A felhasználó inputja alapján strukturált munkautasítást generálsz az alábbi formátumban:

1. CÍM: A munkautasítás rövid címe
2. CÉL: Mit ér el a dolgozó, ha követi az utasítást
3. SZÜKSÉGES ANYAGOK ÉS ESZKÖZÖK: Felsorolás
4. BIZTONSÁGI ELŐÍRÁSOK: Releváns biztonsági figyelmeztetések
5. LÉPÉSEK: Számozott lépések, mindegyikhez:
   - Főlépés: Mit csinálj
   - Kulcspontok: Hogyan csinálj (részletek, amik a minőséget biztosítják)
   - Indoklás: Miért fontos ez a lépés
6. MINŐSÉGI ELLENŐRZÉS: Hogyan ellenőrizhető a munka minősége

SZABÁLYOK:
- Minden output AUTOMATIKUSAN tartalmazza: "⚠️ AI által generált tartalom — emberi felülvizsgálat szükséges."
- Légy precíz és konkrét — gyártási környezetben használják
- Ha nem kapsz elég információt, KÉRDEZZ VISSZA — ne találj ki részleteket
- Magyar nyelven válaszolj, technikai szakkifejezések angolul is megadhatók zárójelben
"""

TWI_GENERATE_PROMPT = """A felhasználó kérése:
{message}

{revision_context}

Generáld a TWI utasítást a megadott formátumban."""

async def generate_node(state: AgentState) -> AgentState:
    """TWI generálás LLM-mel"""
    revision_context = ""
    if state.get("revision_feedback"):
        revision_context = f"""
KORÁBBI VÁZLAT:
{state.get('draft', '')}

FELHASZNÁLÓI VISSZAJELZÉS:
{state['revision_feedback']}

Módosítsd a vázlatot a visszajelzés alapján.
"""
```

```
response = await call_llm(
    system_prompt=TWI_SYSTEM_PROMPT,
    prompt=TWI_GENERATE_PROMPT.format(
        message=state["message"],
        revision_context=revision_context,
    ),
    temperature=0.3,
    max_tokens=4000,
)

# AI jelölés hozzáadása
draft = f"⚠️ AI által generált tartalom — emberi felülvizsgálat szükséges.\n\n{response}"

return {
    **state,
    "draft": draft,
    "draft_metadata": {
        "model": state.get("llm_model", "mistral-large"),
        "generated_at": _now_iso(),
        "revision": state.get("revision_count", 0),
    },
    "status": "review_needed",
}
```

python

```python
# app/agent/nodes/output.py
import uuid
from app.agent.state import AgentState
from app.agent.tools.pdf_generator import generate_twi_pdf
from app.services.blob_storage import upload_pdf

async def output_node(state: AgentState) -> AgentState:
    """PDF generálás és feltöltés Blob Storage-ba"""
    # PDF generálás
    pdf_bytes = await generate_twi_pdf(
        content=state["draft"],
        metadata=state["draft_metadata"],
        user_id=state["user_id"],
    )

    # Blob Storage feltöltés
    blob_name = f"twi/{state['conversation_id']}/{uuid.uuid4().hex}.pdf"
    pdf_url = await upload_pdf(pdf_bytes, blob_name)

    return {
        **state,
        "pdf_url": pdf_url,
        "pdf_blob_name": blob_name,
        "status": "completed",
    }
```

python

```python
# app/agent/nodes/audit.py
from datetime import datetime, timezone
from app.agent.state import AgentState
from app.services.cosmos_db import AuditStore

async def audit_node(state: AgentState) -> AgentState:
    """Audit log mentés Cosmos DB-be"""
    audit_store = AuditStore()
    await audit_store.log({
        "conversation_id": state["conversation_id"],
        "user_id": state["user_id"],
        "tenant_id": state["tenant_id"],
        "channel": state["channel"],
        "intent": state["intent"],
        "llm_model": state.get("llm_model"),
        "revision_count": state.get("revision_count", 0),
        "pdf_blob_name": state.get("pdf_blob_name"),
        "status": state["status"],
        "approval_timestamp": state.get("approval_timestamp"),
        "created_at": datetime.now(timezone.utc).isoformat(),
    })

    return state
```

---

5. ADAPTIVE CARDS

5.1 Review Card (Human-in-the-loop #1)

```
python
```

```python
# app/bot/adaptive_cards.py

def create_review_card(draft: str, metadata: dict) -> dict:
    """Vázlat review card — jóváhagyás / szerkesztés kérés / elvetés"""
    return {
        "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
        "type": "AdaptiveCard",
        "version": "1.4",
        "body": [
            {
                "type": "TextBlock",
                "text": "📋 TWI Vázlat — Felülvizsgálat szükséges",
                "weight": "bolder",
                "size": "large",
                "wrap": True,
            },
            {
                "type": "TextBlock",
                "text": f"⚠️ AI által generált tartalom | Modell: {metadata.get('model', 'N/A')} | "
                        f"Generálva: {metadata.get('generated_at', 'N/A')}",
                "size": "small",
                "color": "warning",
                "wrap": True,
            },
            {"type": "TextBlock", "text": "---", "separator": True},
            {
                "type": "TextBlock",
                "text": draft[:2000],  # Adaptive Card limit
                "wrap": True,
                "fontType": "default",
            },
            {
                "type": "TextBlock",
                "text": "---",
                "separator": True,
            },
            {
                "type": "TextBlock",
                "text": "Szerkesztési megjegyzés (opcionális):",
                "size": "small",
            },
            {
                "type": "Input.Text",
                "id": "feedback",
                "isMultiline": True,
                "placeholder": "Pl.: A 3. lépésben hiányzik a hőmérséklet beállítás...",
            },
        ],
        "actions": [
            {
                "type": "Action.Submit",
                "title": "✅ Jóváhagyom a vázlatot",
                "style": "positive",
                "data": {
                    "action": "approve_draft",
                    "draft": draft,
                    "metadata": metadata,
                },
            },
            {
                "type": "Action.Submit",
                "title": "✏️ Szerkesztés kérem",
                "data": {
                    "action": "request_edit",
                    "draft": draft,
                    "metadata": metadata,
                },
            },
            {
                "type": "Action.Submit",
                "title": "🗑️ Elvetés",
                "style": "destructive",
                "data": {"action": "reject"},
            },
        ],
    }


def create_approval_card(draft: str, metadata: dict) -> dict:
    """Végső jóváhagyás card (Human-in-the-loop #2) — KÖTELEZŐ"""
    return {
        "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
        "type": "AdaptiveCard",
        "version": "1.4",
        "body": [
            {
                "type": "TextBlock",
                "text": "🔒 Véglegesítés — Kötelező Jóváhagyás",
                "weight": "bolder",
                "size": "large",
                "color": "attention",
                "wrap": True,
            },
            {
                "type": "TextBlock",
                "text": "⚠️ Ez a dokumentum AI által generált tartalom. "
                        "Kérlek ellenőrizd a tartalmat, mielőtt véglegesíted. "
                        "Véglegesítés után PDF készül és archiválásra kerül.",
                "wrap": True,
                "color": "warning",
            },
            {
                "type": "TextBlock",
                "text": draft[:2000],
                "wrap": True,
            },
        ],
        "actions": [
            {
                "type": "Action.Submit",
                "title": "✅ Ellenőriztem és jóváhagyom",
                "style": "positive",
                "data": {
                    "action": "final_approve",
                    "draft": draft,
                    "metadata": metadata,
                    "timestamp": "__CURRENT_TIMESTAMP__",
                },
            },
            {
                "type": "Action.Submit",
                "title": "↩️ Vissza a szerkesztéshez",
                "data": {
                    "action": "request_edit",
                    "draft": draft,
                    "metadata": metadata,
                },
            },
        ],
    }


def create_result_card(pdf_url: str, document_title: str, metadata: dict) -> dict:
    """Eredmény card — PDF letöltés link"""
    return {
        "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
        "type": "AdaptiveCard",
        "version": "1.4",
        "body": [
            {
                "type": "TextBlock",
                "text": "✅ Dokumentum elkészült",
                "weight": "bolder",
                "size": "large",
                "color": "good",
            },
            {
                "type": "FactSet",
                "facts": [
                    {"title": "Cím:", "value": document_title},
                    {"title": "Formátum:", "value": "PDF"},
                    {"title": "Modell:", "value": metadata.get("model", "N/A")},
                    {"title": "Jóváhagyta:", "value": metadata.get("approved_by", "N/A")},
                ],
            },
        ],
        "actions": [
            {
                "type": "Action.OpenUrl",
                "title": "📥 PDF letöltés",
                "url": pdf_url,
            },
        ],
    }


def create_welcome_card() -> dict:
    """Üdvözlő card"""
    return {
        "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
        "type": "AdaptiveCard",
        "version": "1.4",
        "body": [
            {
                "type": "TextBlock",
                "text": "👋 Üdvözöllek! Én az agentize.eu AI asszisztens vagyok.",
                "weight": "bolder",
                "size": "medium",
                "wrap": True,
            },
            {
                "type": "TextBlock",
                "text": "Segíthetek TWI (Training Within Industry) munkautasítások "
                        "generálásában. Írd le, milyen utasításra van szükséged!",
                "wrap": True,
            },
            {
```

```
            "type": "TextBlock",
            "text": "Példa: \"Készíts egy TWI utasítást a CNC-01 gép beállításáról\"",
            "wrap": True,
            "isSubtle": True,
            "fontType": "monospace",
        },
    ],
}
```

---

6. ADATBÁZIS SÉMA — COSMOS DB (MongoDB API)

6.1 Collections

```
javascript
```

```
// Collection: conversations
{
  "_id": ObjectId,
  "conversation_id": "conv-123",          // Bot Framework conversation ID
  "user_id": "user-entra-id",
  "tenant_id": "poc-tenant",
  "channel": "msteams" | "telegram",
  "started_at": ISODate,
  "last_activity": ISODate,
  "message_count": 5,
  "status": "active" | "completed" | "expired"
}
// Index: { conversation_id: 1 } unique
// Index: { tenant_id: 1, user_id: 1 }
// TTL index: { last_activity: 1 }, expireAfterSeconds: 7776000  (90 nap)


// Collection: agent_state (LangGraph checkpoints)
// AUTOMATIKUSAN KEZELI a LangGraph checkpointer
// Partition key: thread_id (= conversation_id)
{
  "_id": ObjectId,
  "thread_id": "conv-123",
  "checkpoint_id": "cp-456",
  "parent_checkpoint_id": "cp-455" | null,
  "checkpoint": { /* LangGraph internal state */ },
  "metadata": {
    "step": 3,
    "node": "review",
    "writes": {...}
  },
  "created_at": ISODate
}


// Collection: generated_documents
{
  "_id": ObjectId,
  "document_id": "doc-uuid",
  "conversation_id": "conv-123",
  "user_id": "user-entra-id",
  "tenant_id": "poc-tenant",
  "title": "TWI — CNC-01 gép beállítása",
  "content_type": "twi",
  "draft_content": "...",                 // Utolsó jóváhagyott szöveg
  "pdf_blob_name": "twi/conv-123/abc.pdf",
  "pdf_url": "https://...",
  "llm_model": "mistral-large-latest",
  "revision_count": 1,
  "status": "approved" | "draft" | "rejected",
  "created_at": ISODate,
  "approved_at": ISODate,
  "approved_by": "user-entra-id"
}
// Index: { tenant_id: 1, created_at: -1 }
// Index: { conversation_id: 1 }


// Collection: audit_log
{
  "_id": ObjectId,
  "conversation_id": "conv-123",
  "user_id": "user-entra-id",
  "tenant_id": "poc-tenant",
  "channel": "msteams",
  "event_type": "twi_generated" | "twi_approved" | "twi_rejected" | "twi_revised",
  "intent": "generate_twi",
  "llm_model": "mistral-large-latest",
  "llm_tokens_input": 1250,
  "llm_tokens_output": 2800,
  "revision_count": 1,
  "pdf_blob_name": "twi/conv-123/abc.pdf",
  "status": "completed",
  "approval_timestamp": ISODate,
  "created_at": ISODate
}
// Index: { tenant_id: 1, created_at: -1 }
// Index: { event_type: 1 }
```

---

7. PDF GENERÁLÁS

7.1 Pipeline

```
TWI szöveg (markdown-szerű)
  → Jinja2 template rendering (HTML)
  → WeasyPrint (HTML → PDF)
  → Blob Storage upload
  → URL visszaadás
```

7.2 Jinja2 HTML Template

html

```html
<!-- app/templates/twi_template.html -->
<!DOCTYPE html>
<html lang="hu">
<head>
  <meta charset="UTF-8">
  <style>
    @page {
      size: A4;
      margin: 2cm;
      @bottom-center {
        content: "agentize.eu — AI által generált tartalom — " counter(page) "/" counter(pages);
        font-size: 8pt;
        color: #888;
      }
    }
    body {
      font-family: Arial, sans-serif;
      font-size: 11pt;
      line-height: 1.5;
      color: #2c3e50;
    }
    .header {
      border-bottom: 3px solid #1b4f72;
      padding-bottom: 10px;
      margin-bottom: 20px;
    }
    .header h1 { color: #1b4f72; margin: 0; font-size: 18pt; }
    .header .meta { color: #666; font-size: 9pt; margin-top: 5px; }
    .ai-warning {
      background: #fef5e7;
      border-left: 4px solid #e67e22;
      padding: 10px 15px;
      margin: 15px 0;
      font-size: 9pt;
      color: #856404;
    }
    h2 { color: #2e86c1; font-size: 14pt; border-bottom: 1px solid #ddd; padding-bottom: 5px; }
    h3 { color: #1b4f72; font-size: 12pt; }
    .step {
      background: #f8f9fa;
      border: 1px solid #dee2e6;
      border-radius: 4px;
      padding: 12px;
      margin: 10px 0;
    }
    .step-number { color: #1b4f72; font-weight: bold; font-size: 13pt; }
    .key-point { color: #2e86c1; font-style: italic; }
    .reason { color: #666; font-size: 10pt; }
    .approval-box {
      border: 2px solid #27ae60;
      background: #eafaf1;
      padding: 15px;
      margin-top: 30px;
    }
    .approval-box .label { font-weight: bold; color: #27ae60; }
  </style>
</head>
<body>
  <div class="header">
    <h1>{{ title }}</h1>
    <div class="meta">
      Generálva: {{ generated_at }} | Modell: {{ model }} | Verzió: {{ revision }}
    </div>
  </div>

  <div class="ai-warning">
    ⚠️ AI által generált tartalom — emberi felülvizsgálat szükséges.
    Ez a dokumentum az agentize.eu AI platform segítségével készült.
  </div>

  {{ content_html }}

  {% if approved %}
  <div class="approval-box">
    <div class="label">✅ Jóváhagyva</div>
    <div>{{ approved_by }} — {{ approved_at }}</div>
  </div>
  {% endif %}
</body>
</html>
```

7.3 PDF Generator Tool

```
python
```

```
# app/agent/tools/pdf_generator.py
import io
from datetime import datetime, timezone
from jinja2 import Environment, FileSystemLoader
from weasyprint import HTML
import markdown

template_env = Environment(loader=FileSystemLoader("app/templates"))

async def generate_twi_pdf(content: str, metadata: dict, user_id: str) -> bytes:
    """TWI tartalom → PDF bytes"""
    # Markdown → HTML konverzió
    content_html = markdown.markdown(content, extensions=["tables", "fenced_code"])

    # Jinja2 template rendering
    template = template_env.get_template("twi_template.html")
    html_content = template.render(
        title=_extract_title(content),
        generated_at=metadata.get("generated_at", "N/A"),
        model=metadata.get("model", "N/A"),
        revision=metadata.get("revision", 0),
        content_html=content_html,
        approved=True,
        approved_by=user_id,
        approved_at=datetime.now(timezone.utc).strftime("%Y-%m-%d %H:%M UTC"),
    )

    # HTML → PDF
    pdf_bytes = HTML(string=html_content).write_pdf()
    return pdf_bytes

def _extract_title(content: str) -> str:
    """Első sor kinyerés címként"""
    for line in content.split("\n"):
        line = line.strip().lstrip("#").strip()
        if line and not line.startswith("⚠️"):
            return line[:100]
    return "TWI Munkautasítás"
```

---

8. AZURE SZOLGÁLTATÁS KLIENSEK

8.1 AI Foundry Client

```
python
```

```python
# app/services/ai_foundry.py
import logging
from azure.ai.inference import ChatCompletionsClient
from azure.core.credentials import AzureKeyCredential

from app.config import settings

logger = logging.getLogger(__name__)

_client = None

def _get_client():
    global _client
    if _client is None:
        _client = ChatCompletionsClient(
            endpoint=settings.ai_foundry_endpoint,
            credential=AzureKeyCredential(settings.ai_foundry_key),
        )
    return _client


async def call_llm(
    prompt: str,
    system_prompt: str = None,
    temperature: float = None,
    max_tokens: int = None,
) -> str:
    """LLM hívás az Azure AI Foundry-n keresztül"""
    client = _get_client()

    messages = []
    if system_prompt:
        messages.append({"role": "system", "content": system_prompt})
    messages.append({"role": "user", "content": prompt})

    response = client.complete(
        messages=messages,
        model=settings.ai_model,
        temperature=temperature or settings.ai_temperature,
        max_tokens=max_tokens or settings.ai_max_tokens,
    )

    result = response.choices[0].message.content

    # Token tracking (audit-hoz)
    usage = response.usage
    logger.info(
        f"LLM call: model={settings.ai_model}, "
        f"input_tokens={usage.prompt_tokens}, "
        f"output_tokens={usage.completion_tokens}"
    )

    return result
```

8.2 Cosmos DB Client

```
python
```

```python
# app/services/cosmos_db.py
from motor.motor_asyncio import AsyncIOMotorClient
from datetime import datetime, timezone

from app.config import settings

_client = None
_db = None

def _get_db():
    global _client, _db
    if _db is None:
        _client = AsyncIOMotorClient(settings.cosmos_connection)
        _db = _client[settings.cosmos_database]
    return _db


class ConversationStore:
    def __init__(self):
        self.collection = _get_db()["conversations"]

    async def get_or_create(self, conversation_id: str, user_id: str, channel: str, tenant_id: str = "poc-tenant"):
        doc = await self.collection.find_one({"conversation_id": conversation_id})
        if doc:
            await self.collection.update_one(
                {"conversation_id": conversation_id},
                {"$set": {"last_activity": datetime.now(timezone.utc)}, "$inc": {"message_count": 1}}
            )
            return doc

        new_doc = {
            "conversation_id": conversation_id,
            "user_id": user_id,
            "tenant_id": tenant_id,
            "channel": channel,
            "started_at": datetime.now(timezone.utc),
            "last_activity": datetime.now(timezone.utc),
            "message_count": 1,
            "status": "active",
        }
        await self.collection.insert_one(new_doc)
        return new_doc


class AuditStore:
    def __init__(self):
        self.collection = _get_db()["audit_log"]

    async def log(self, entry: dict):
        entry["created_at"] = datetime.now(timezone.utc)
        await self.collection.insert_one(entry)


class DocumentStore:
    def __init__(self):
        self.collection = _get_db()["generated_documents"]

    async def save(self, doc: dict):
        doc["created_at"] = datetime.now(timezone.utc)
        await self.collection.insert_one(doc)
        return doc
```

8.3 Blob Storage Client

```
python
```

```
# app/services/blob_storage.py
from azure.storage.blob import BlobServiceClient, generate_blob_sas, BlobSasPermissions
from datetime import datetime, timezone, timedelta

from app.config import settings

_client = None

def _get_client():
    global _client
    if _client is None:
        _client = BlobServiceClient.from_connection_string(settings.blob_connection)
    return _client


async def upload_pdf(pdf_bytes: bytes, blob_name: str) -> str:
    """PDF feltöltés Blob Storage-ba, SAS URL visszaadás"""
    client = _get_client()
    container_client = client.get_container_client(settings.blob_container)
    blob_client = container_client.get_blob_client(blob_name)

    blob_client.upload_blob(pdf_bytes, content_settings={"content_type": "application/pdf"})

    # SAS token generálás (24 órás érvényesség)
    sas_token = generate_blob_sas(
        account_name=client.account_name,
        container_name=settings.blob_container,
        blob_name=blob_name,
        account_key=client.credential.account_key,
        permission=BlobSasPermissions(read=True),
        expiry=datetime.now(timezone.utc) + timedelta(hours=24),
    )

    return f"{blob_client.url}?{sas_token}"
```

---

9. DOCKER + CI/CD

9.1 Dockerfile

```
dockerfile
```

```
FROM python:3.12-slim

# WeasyPrint rendszer-függőségek
RUN apt-get update && apt-get install -y --no-install-recommends \
    libpango-1.0-0 \
    libpangoft2-1.0-0 \
    libharfbuzz0b \
    libffi-dev \
    libgdk-pixbuf2.0-0 \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY app/ ./app/

EXPOSE 8000

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000", "--workers", "2"]
```

9.2 Dev Container

```
json
```

```
// .devcontainer/devcontainer.json
{
  "name": "agentize-poc",
  "image": "mcr.microsoft.com/devcontainers/python:3.12",
  "features": {
    "ghcr.io/devcontainers/features/azure-cli:1": {},
    "ghcr.io/devcontainers/features/docker-in-docker:2": {}
  },
  "postCreateCommand": "pip install -r requirements.txt",
  "customizations": {
    "vscode": {
      "extensions": [
        "ms-python.python",
        "ms-azuretools.vscode-bicep",
        "ms-azuretools.vscode-azureresourcegroups"
      ]
    }
  },
  "forwardPorts": [8000]
}
```

9.3 GitHub Actions (Minimal PoC)

yaml

```
# .github/workflows/deploy.yml
name: Deploy PoC Backend

on:
  push:
    branches: [main]

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Login to Azure
        uses: azure/login@v2
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}

      - name: Build and push to ACR
        run: |
          az acr build \
            --registry ${{ vars.ACR_NAME }} \
            --image poc-backend:${{ github.sha }} \
            --file Dockerfile .

      - name: Deploy to Container App
        run: |
          az containerapp update \
            --name ca-agentize-poc-backend \
            --resource-group rg-agentize-poc-swedencentral \
            --image ${{ vars.ACR_NAME }}.azurecr.io/poc-backend:${{ github.sha }}
```

---

10. TEAMS APP MANIFEST

```
json
```

```
// teams-app/manifest.json
{
  "$schema": "https://developer.microsoft.com/en-us/json-schemas/teams/v1.17/MicrosoftTeams.schema.json",
  "manifestVersion": "1.17",
  "version": "0.1.0",
  "id": "{{BOT_APP_ID}}",
  "developer": {
    "name": "agentize.eu",
    "websiteUrl": "https://agentize.eu",
    "privacyUrl": "https://agentize.eu/privacy",
    "termsOfUseUrl": "https://agentize.eu/terms"
  },
  "name": {
    "short": "agentize AI",
    "full": "agentize.eu Enterprise AI Platform"
  },
  "description": {
    "short": "AI munkautasítás generátor",
    "full": "Enterprise AI platform gyártási munkautasítások (TWI) generálásához. GDPR-konform, EU adatközpontban fut."
  },
  "icons": {
    "outline": "outline.png",
    "color": "color.png"
  },
  "accentColor": "#1B4F72",
  "bots": [
    {
      "botId": "{{BOT_APP_ID}}",
      "scopes": ["personal", "team", "groupChat"],
      "supportsFiles": false,
      "commandLists": [
        {
          "scopes": ["personal"],
          "commands": [
            {
              "title": "Új TWI",
              "description": "Új munkautasítás generálása"
            },
            {
              "title": "Segítség",
              "description": "Használati útmutató"
            }
          ]
        }
      ]
    }
  ],
  "permissions": ["identity", "messageTeamMembers"],
  "validDomains": ["{{BACKEND_FQDN}}"]
}
```

---

11. KÖRNYEZETI VÁLTOZÓK

bash

```bash
# .env.example

# Azure AI Foundry
AI_FOUNDRY_ENDPOINT=https://ai-agentize-poc.swedencentral.inference.ai.azure.com
AI_FOUNDRY_KEY=<from Key Vault>
AI_MODEL=mistral-large-latest
AI_TEMPERATURE=0.3
AI_MAX_TOKENS=4000

# Cosmos DB (MongoDB API)
COSMOS_CONNECTION=mongodb://<account>:<key>@<account>.mongo.cosmos.azure.com:10255/?ssl=true&replicaSe
COSMOS_DATABASE=agentize-poc-db

# Blob Storage
BLOB_CONNECTION=DefaultEndpointsProtocol=https;AccountName=stagentizepoc;AccountKey=<key>;EndpointSuffix
BLOB_CONTAINER=pdf-output

# Bot Framework
BOT_APP_ID=<Entra ID App Registration client ID>
BOT_APP_PASSWORD=<Entra ID App Registration client secret>

# Telegram (opcionális)
TELEGRAM_BOT_TOKEN=<from @BotFather>

# Application Insights
APPLICATIONINSIGHTS_CONNECTION_STRING=InstrumentationKey=<key>;IngestionEndpoint=https://swedencentra

# App
ENVIRONMENT=poc
LOG_LEVEL=INFO
```

---

12. IMPLEMENTÁCIÓS SORREND — 10 MUNKANAP

Nap 1-2: Infrastruktúra

- [ ] Bicep template megírás (main.bicep + parameters.json)
- [ ] `az deployment group create` — test deploy Sweden Central-ba
- [ ] AI Foundry model deployment (Mistral Large, Data Zone Standard)
- [ ] Entra ID App Registration (Bot Framework-höz)
- [ ] Key Vault secrets feltöltés
- [ ] Cosmos DB collections + indexek létrehozás
- [ ] Blob Storage container létrehozás
- [ ] **Validáció:** minden resource elérhető, AI Foundry válaszol

Nap 3-4: Backend Core

- [ ] Projekt struktúra létrehozás (lásd 4.1)
- [ ] `app/config.py` — Settings osztály
- [ ] `app/services/ai_foundry.py` — LLM client + test hívás
- [ ] `app/services/cosmos_db.py` — MongoDB client + test CRUD
- [ ] `app/agent/state.py` — AgentState definíció
- [ ] `app/agent/nodes/intent.py` — Intent recognition
- [ ] `app/agent/nodes/generate.py` — TWI generálás
- [ ] `app/agent/graph.py` — Alap gráf (intent → generate → END)
- [ ] **Validáció:** `pytest` — intent felismerés + generálás működik

Nap 5-6: Bot Framework + Adaptive Cards

- [ ] `app/main.py` — FastAPI + /api/messages endpoint
- [ ] `app/bot/bot_handler.py` — Message handler
- [ ] `app/bot/adaptive_cards.py` — Review + Approval + Result + Welcome cards
- [ ] Azure Bot Service resource regisztráció
- [ ] Teams channel + Telegram channel bekapcsolás
- [ ] Teams App Manifest (sideload-hoz)
- [ ] **Validáció:** Teams-ből üzenet → Bot válaszol Adaptive Card-dal

Nap 7-8: Human-in-the-loop + Revision Loop

- [ ] `app/agent/nodes/review.py` — Review checkpoint
- [ ] `app/agent/nodes/revise.py` — Revision node
- [ ] `app/agent/nodes/approve.py` — Final approval checkpoint
- [ ] Graph interrupt_before konfigurálás
- [ ] Bot handler: Adaptive Card submit → graph resume
- [ ] Revision loop tesztelés (max 3 kör)
- [ ] **Validáció:** Teljes flow: generálás → review → szerkesztés → jóváhagyás

Nap 9: PDF + Output

- [ ] `app/templates/twi_template.html` — Jinja2 template
- [ ] `app/agent/tools/pdf_generator.py` — PDF generálás (WeasyPrint)
- [ ] `app/services/blob_storage.py` — PDF upload + SAS URL
- [ ] `app/agent/nodes/output.py` — Output node
- [ ] `app/agent/nodes/audit.py` — Audit log node
- [ ] Dockerfile + local build + test
- [ ] **Validáció:** PDF generálódik, letölthető URL-ről, audit log Cosmos DB-ben

Nap 10: Integration Test + Demo Prep

- [ ] End-to-end teszt: Teams → Bot → Agent → PDF → letöltés
- [ ] End-to-end teszt: Telegram → Bot → Agent → PDF → letöltés
- [ ] Container App deployment (GitHub Actions vagy manuális)
- [ ] Demo script megírás (3 perces happy path)
- [ ] Known issues dokumentálás
- [ ] **Validáció:** Demo futtatás valós Teams + Telegram környezetben

---

13. DEMÓ SCRIPT (3 perc)

```
1. [Teams-ben] "Szia! Készíts egy TWI utasítást a CNC-01 gép napi karbantartásáról."
   → Bot: "⏳ Feldolgozom..."
   → Bot: [Review Adaptive Card — vázlat megjelenik]

2. [Review Card-on] "✏️ Szerkesztés kérem" + "A 3. lépésben add hozzá a hőmérséklet ellenőrzést"
   → Bot: "⏳ Módosítom..."
   → Bot: [Módosított Review Adaptive Card]

3. [Review Card-on] "✅ Jóváhagyom a vázlatot"
   → Bot: [Final Approval Card]

4. [Approval Card-on] "✅ Ellenőriztem és jóváhagyom"
   → Bot: "⏳ PDF generálás..."
   → Bot: [Result Card — PDF letöltés link]

5. PDF megnyitás — formázott dokumentum, agentize.eu branding, AI jelölés, jóváhagyási info

6. [Opcionális — Telegram-ban ugyanez]
```

---

14. SIKERESSÉGI KRITÉRIUMOK

A PoC akkor kész, ha:

1. ✅ Bicep template egy gombnyomásra települ Sweden Central-ba
2. ✅ Teams-ből végigfut a teljes flow: kérés → generálás → review → szerkesztés → jóváhagyás → PDF
3. ✅ Telegram-ból ugyanez működik
4. ✅ PDF letölthető, formázott, tartalmazza az AI jelölést és jóváhagyási infót
5. ✅ Audit log Cosmos DB-ben: ki, mit, mikor, melyik modellel
6. ✅ EU AI Act jelölés minden AI output-on
7. ✅ Többpontos jóváhagyás működik (review + final approval)
8. ✅ Revision loop működik (szerkesztés kérés → módosítás → újra review)
9. ✅ 3 perces demó végigfuttatható hiba nélkül

---

agentize.eu — AI & Organizational Solutions PoC Technical Spec v1.0 — 2026-02-26