📖 KNOWLEDGE BASE & ONBOARDING - Agent Zero v1

🎯 Knowledge Base & Team Onboarding

Mission: Stworzyć centralne repozytorium wiedzy i efektywny system wdrażania nowych członków zespołu

Core Principles:

- **Self-Service**: Odpowiedzi na 80% pytań dostępne w KB
- **Living Documentation**: Aktualizowana z każdym nowym doswiadczeniem
- **Practical Focus**: Konkretne przykłady i playbooki

📋 TECHNICAL ONBOARDING CHECKLIST

🏁 Day 1: Environment Setup

✅ Pre-Onboarding (Before First Day)

- [ ] GitHub access granted to `HotelAiOS/agent-zero-v1`
- [ ] Notion workspace access configured
- [ ] Slack/Discord channels added
- [ ] Development machine prepared (see hardware requirements)
- [ ] VPN access configured (if remote)

✅ System Installation (2-4 hours)

- [ ] Follow Poradnik Deweloperski - Instalacja
- [ ] Clone repository successfully
- [ ] Docker environment running (all containers ✅)
- [ ] Python venv activated and dependencies installed

- [ ] All tests passing locally
- [ ] IDE configured with linting and formatting

✅ First Commits (1-2 hours)

- [ ] Create feature branch `onboarding/[name]-setup`
- [ ] Make small documentation update
- [ ] Submit first PR and get it reviewed
- [ ] Experience full git workflow

🎯 Day 2-3: Architecture Deep Dive

✅ Codebase Exploration

- [ ] Read Architecture Decisions (ADR)
- [ ] Review `shared/communication/` (Phase 1 code)
- [ ] Understand agent factory patterns
- [ ] Explore Neo4j knowledge integration
- [ ] Test LLM integration locally

✅ Technical Understanding

- [ ] Run `test_intelligent_agents.py` and understand output
- [ ] Experiment with Ollama LLM calls
- [ ] Connect to Neo4j browser and explore data
- [ ] Test RabbitMQ message flow
- [ ] Review API endpoints in `services/api-gateway/`

✅ Pair Programming Session

- [ ] 2-hour session with experienced team member
- [ ] Work on small bug fix together
- [ ] Learn debugging techniques

- [ ] Understand testing approach

📚 Week 1: Project Context & Processes

✅ Project Knowledge

- [ ] Read Roadmapa Projektu
- [ ] Understand Sprint Planning System
- [ ] Review Code Review & QA Process
- [ ] Study Communication & Escalation
- [ ] Participate in daily standups (observer mode)

✅ First Real Tasks

- [ ] Pick up 1-2 SP task from current sprint
- [ ] Complete end-to-end: development → testing → review → merge
- [ ] Add tests for your changes
- [ ] Update documentation as needed

🏆 End of Week 1: Onboarding Review

✅ Knowledge Assessment

- [ ] Can explain Agent Zero architecture to someone else
- [ ] Understands team workflow and tools
- [ ] Comfortable with development environment
- [ ] Knows how to find answers in Knowledge Base
- [ ] Comfortable asking for help when needed

✅ Feedback Session (30 minutes)

- [ ] What went well during onboarding?
- [ ] What was confusing or could be improved?
- [ ] What additional support is needed?

- [ ] Update onboarding process based on feedback

❓ FREQUENTLY ASKED QUESTIONS

🚀 Getting Started

Q: Co to jest Agent Zero v1?

A: To enterprise AI platforma z systemem wieloagentowym. Agents komunikują się przez RabbitMQ, przechowują wiedzę w Neo4j, i używają Ollama dla LLM inference. Cel: pełna automatyzacja software development workflow.

Q: Jaka jest aktualna architektura systemu?

A: Zobacz ADR-001 do ADR-007. Główne komponenty: FastAPI (API), React (UI), Neo4j (wiedza), RabbitMQ (komunikacja), Ollama (LLM), Docker (deployment).

Q: Które testy są najważniejsze do uruchomienia?

A:

1. `python test_simple.py` - basic system test
2. `shared/communication/test_intelligent_agents.py` - agent communication
3. `pytest shared/communication/ -v` - full communication suite

🔧 Development Workflow

Q: Jak stworzyć nowy branch dla feature?

A:

```
git checkout main
git pull origin main
git checkout -b feature/nazwa-funkcji
# lub fix/nazwa-naprawy
```

Q: Jakie są konwencje commit messages?

A: Używamy Conventional Commits:

- `feat: dodaj nową funkcję`

- `fix: napraw błąd w X`
- `docs: aktualizuj README`
- `test: dodaj testy dla Y`

Q: Jak dodać nowe testy?

A:

1. Unit tests: w tym samym folderze co kod, `test_*.py`
2. Integration tests: w `tests/integration/`
3. Użyj pytest fixtures dla setup
4. Mock external dependencies (Neo4j, Ollama)

🐛 Debugging & Troubleshooting

Q: Neo4j nie startuje w Docker?

A:

1. Check ports: `netstat -tulpn | grep 7474`
2. Check logs: `docker compose logs neo4j`
3. Reset data: `docker compose down -v && docker compose up -d`
4. Verify password: `neo4j/agent-pass`

Q: Ollama zwraca błędy connection?

A:

1. Check if running: `curl http://localhost:11434/api/version`
2. Install if needed: `curl https://ollama.ai/install.sh | sh`
3. Pull model: `ollama pull deepseek-coder:33b`
4. Test: `ollama run deepseek-coder:33b "Hello"`

Q: Testy są wolne lub flakey?

A:

1. Use pytest markers: `@pytest.mark.slow`

2. Mock external calls (LLM, database)
3. Use fixtures for test data
4. Run specific tests: `pytest -k test_name`

Q: Import errors w Python?

A:

1. Check venv: `which python` (should show venv path)
2. Reinstall: `pip install -r requirements.txt --force-reinstall`
3. Check PYTHONPATH: `export PYTHONPATH=.`
4. Use relative imports: `from shared.communication import agent_registry`

📄 Architecture & Design

Q: Kiedy stworzyć nowego agenta vs użyć istniejącego?

A: Nowy agent jeśli:

- Różne capabilities (backend vs frontend)
- Różne lifecycle patterns
- Niezależne skalowanie potrzebne

Użyj istniejącego jeśli tylko rozszerzasz funkcjonalność.

Q: Jak dodać nowy typ wiadomości między agentami?

A:

1. Define w `shared/communication/message_types.py`
2. Add handler w `IntelligentAgent.register_handler()`
3. Update routing logic if needed
4. Add tests for new message flow

Q: Gdzie przechować configuration?

A:

- **Env variables**: Secrets, database URLs

- **config.yaml**: Application settings
- **Code constants**: Business logic defaults
- **Neo4j**: Dynamic configuration data

📖 TECHNICAL PLAYBOOKS

🔧 Playbook 1: Jak debugować agent communication issues

Symptoms: Agents nie otrzymują messages, timeouts, connection errors

Debug Steps:

1. **Check RabbitMQ**

```
# Management UI
open http://localhost:15672
# User: agent, Pass: agent-pass

# CLI check
docker compose exec rabbitmq rabbitmqctl list_queues
```

2. **Check Agent Registry**

```
from shared.communication.agent_registry import agent_registry
stats = await agent_registry.get_stats()
print(f"Agents online: {stats.online}")
```

3. **Check Message Flow**

```
# Enable debug logging
import logging
logging.basicConfig(level=logging.DEBUG)

# Test simple message
await agent.send_to_agent("target_id", "test_message", {})
```

4. **Common Fixes**
   - Restart RabbitMQ: `docker compose restart rabbitmq`
   - Clear message queues
   - Check routing keys match expectations
   - Verify agent registration

🔧 Playbook 2: Jak dodać nowy LLM model

Goal: Add support for new model w Ollama

Steps:

1. **Install Model Locally**

```
ollama pull model-name:tag
ollama list  # verify installation
```

2. **Update LLM Client**

```
# shared/llm/ollama_client.py
SUPPORTED_MODELS = [
    "deepseek-coder:33b",
    "new-model:tag"  # Add here
]
```

3. **Add Model Configuration**

```
# config.yaml lub environment
LLM_MODELS:
  code_generation: "deepseek-coder:33b"
  chat: "new-model:tag"
```

4. **Test Integration**

```
from shared.llm.ollama_client import OllamaClient
client = OllamaClient()
```

```
response = await client.chat("new-model:tag", "Hello")
```

5. **Update Documentation**
   - Add to supported models list
   - Update installation instructions
   - Add performance benchmarks

🔧 Playbook 3: Jak optymalizować Neo4j queries

Symptoms: Slow database queries, high memory usage, timeouts

Optimization Steps:

1. **Identify Slow Queries**

```
# W Neo4j Browser
CALL dbms.listQueries() YIELD query, elapsedTimeMillis
WHERE elapsedTimeMillis > 1000
RETURN query, elapsedTimeMillis
ORDER BY elapsedTimeMillis DESC
```

2. **Add Indexes**

```
# For frequently queried properties
CREATE INDEX FOR (n:Agent) ON (n.agent_id)
CREATE INDEX FOR (n:Task) ON (n.status)
```

3. **Optimize Query Patterns**

```
# Bad: Scans all nodes
MATCH (n:Agent) WHERE n.capabilities CONTAINS 'python'

# Better: Use index
MATCH (n:Agent {status: 'active'})
WHERE n.capabilities CONTAINS 'python'
```

4. **Monitor Performance**

```
# Add query timing
start_time = time.time()
result = session.run(query)
elapsed = time.time() - start_time
logger.info(f"Query took {elapsed:.2f}s")
```

🔧 Playbook 4: Jak setup nowego microservice

Goal: Add new service w `services/` directory

Steps:

1. **Create Service Structure**

```
mkdir services/new-service
cd services/new-service

# Create files
touch Dockerfile
touch requirements.txt
touch main.py
mkdir tests
```

2. **Basic FastAPI Setup**

```
# main.py
from fastapi import FastAPI

app = FastAPI(title="New Service")

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

3. **Add to Docker Compose**

```
# docker-compose.yml
new-service:
  build: ./services/new-service
  ports: ["8003:8000"]
  depends_on: [postgres, rabbitmq]
```

4. **Setup CI/CD**

```
# .github/workflows/new-service.yml
# Add testing pipeline for new service
```

🎯 MENTORING PROGRAM

👥 Buddy System

For New Backend Developer:

- **Primary Mentor**: Current Backend Developer (Dev A)
- **Secondary Mentor**: Frontend Developer (Dev B) - for collaboration
- **Duration**: First 4 weeks
- **Format**: Weekly 1-hour sessions + ad-hoc support

For New Frontend Developer:

- **Primary Mentor**: Current Frontend Developer (Dev B)
- **Secondary Mentor**: Backend Developer (Dev A) - for API integration
- **Duration**: First 4 weeks
- **Format**: Weekly 1-hour sessions + ad-hoc support

📋 Mentoring Structure

Week 1: Foundation

- Environment setup assistance
- Architecture walkthrough

- First PR guidance
- Team process introduction

Week 2: Deep Dive

- Code review sessions
- Pair programming on real tasks
- Debugging techniques
- Testing best practices

Week 3: Independence

- Solo task completion with guidance
- Code quality improvement
- Process optimization ideas
- Team collaboration

Week 4: Integration

- Full team member responsibilities
- Mentoring feedback session
- Knowledge transfer to KB
- Future development planning

📝 Mentoring Sessions Template

```
## Mentoring Session - Week X

### Agenda (60 minutes)
1. **Check-in** (10min)
   - How are you feeling about progress?
   - Any immediate blockers?

2. **Technical Review** (30min)
   - Code review of recent work
```

```
   - Architecture questions
   - Best practices discussion

3. **Hands-on** (15min)
   - Pair programming lub debugging
   - Tool demonstration

4. **Next Steps** (5min)
   - Goals for next week
   - Resources to review
   - Scheduled follow-ups

### Notes
- Key learnings:
- Action items:
- Resources shared:

### Feedback
- What's working well?
- What needs improvement?
- Additional support needed?
```

📉 VIDEO RECORDINGS & RESOURCES

🎬 **Recorded Sessions** (To Be Created)

Architecture Overviews:

- [ ] "Agent Zero v1 - System Architecture" (30min)
- [ ] "Multi-Agent Communication Deep Dive" (45min)
- [ ] "Neo4j Integration Patterns" (20min)
- [ ] "LLM Integration with Ollama" (25min)

Development Workflows:

- [ ] "Complete Feature Development Workflow" (60min)

- [ ] "Debugging Agent Communication Issues" (30min)
- [ ] "Testing Strategy and Best Practices" (40min)
- [ ] "Code Review Process Walkthrough" (20min)

Tool-Specific Tutorials:

- [ ] "Neo4j Browser and Cypher Queries" (15min)
- [ ] "RabbitMQ Management and Troubleshooting" (20min)
- [ ] "Docker Compose Development Environment" (25min)
- [ ] "VS Code Setup for Agent Zero Development" (15min)

🔗 External Resources

Required Reading:

- FastAPI Documentation - API framework
- Neo4j Cypher Manual - Database queries
- RabbitMQ Tutorials - Message queuing
- React Documentation - Frontend framework

Recommended Learning:

- Python Type Hints
- Pytest Documentation - Testing
- Docker Compose Guide
- Git Workflow Best Practices

🔄 KNOWLEDGE BASE MAINTENANCE

📝 Content Update Process

Monthly Reviews (First Friday):

- [ ] Review FAQ for new questions from team
- [ ] Update technical playbooks based on recent issues

- [ ] Add new debugging scenarios encountered
- [ ] Update onboarding checklist based on feedback

Continuous Improvements:

- **After each onboarding**: Collect feedback and update process
- **After major architectural changes**: Update technical documentation
- **After process changes**: Update workflows and templates
- **After tool changes**: Update setup instructions

📈 Knowledge Base Metrics

Usage Tracking:

- Page views on key documentation
- Time to resolution for common issues
- Onboarding completion time
- Team satisfaction with KB

Quality Indicators:

- Accuracy of troubleshooting guides
- Completeness of onboarding process
- Relevance of FAQ answers
- Usefulness of technical playbooks

Knowledge Base Last Updated: 7 października 2025, 13:10 CEST

Next Review: 7 listopada 2025

Content Owner: Development Team