JIRA Duplicate Detection Agent  
Complete Documentation

Generated on: 2025-10-13 12:43:52

Table of Contents

JIRA Duplicate Detection Agent - Complete Documentation

📋 Table of Contents

1. [Project Overview](#project-overview)

2. [Architecture](#architecture)

3. [API Reference](#api-reference)

4. [User Guide](#user-guide)

5. [Development Guide](#development-guide)

6. [Deployment Guide](#deployment-guide)

7. [Troubleshooting](#troubleshooting)

8. [Contributing](#contributing)

---

🎯 Project Overview

What is this project?

The \*\*JIRA Duplicate Detection Agent\*\* is an intelligent Azure AI-powered system that automatically identifies similar and duplicate issues in JIRA projects. It uses advanced machine learning algorithms and Azure AI Foundry to provide comprehensive similarity analysis with tiered scoring and actionable insights.

Key Features

\*\*🔄 Automatic OAuth 2.0 Authentication\*\*: Seamless JIRA authentication with persistent token management

\*\*🎯 Tiered Similarity Detection\*\*: Three-level system (High ≥80%, Medium 50-79%, Low 30-49%)

\*\*📊 Percentage-Based Scoring\*\*: Easy-to-understand similarity percentages

\*\*🧠 Multi-Algorithm AI\*\*: Combines TF-IDF, sequence matching, word overlap, and n-gram analysis

\*\*🔍 Advanced Text Processing\*\*: Intelligent normalization and word mapping

\*\*🤖 Azure AI Integration\*\*: Leverages Azure AI Foundry for enhanced analysis and insights

\*\*📋 Professional Reporting\*\*: Comprehensive markdown reports with JIRA links

\*\*💾 Persistent Authentication\*\*: Automatic token refresh for long-term use

\*\*🧪 Test Mode\*\*: Development and demonstration with sample data

Use Cases

1. \*\*Project Management\*\*: Identify duplicate work items to prevent redundant effort

2. \*\*Quality Assurance\*\*: Find similar bugs or issues that might be related

3. \*\*Resource Optimization\*\*: Consolidate similar tasks to improve efficiency

4. \*\*Team Coordination\*\*: Identify related work that requires coordination

5. \*\*Backlog Cleanup\*\*: Clean up project backlogs by merging duplicates

Business Value

\*\*Time Savings\*\*: Reduces manual effort in identifying duplicate issues

\*\*Cost Reduction\*\*: Prevents duplicate work and redundant development

\*\*Improved Quality\*\*: Ensures comprehensive issue tracking

\*\*Better Planning\*\*: Provides insights for project planning and resource allocation

\*\*Team Efficiency\*\*: Enables better coordination between team members

---

🏗️ Architecture

System Overview

The system follows a modular architecture with clear separation of concerns:

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐  
│ Main Entry │ │ Azure AI Agent │ │ JIRA Client │  
│ (main.py) │───▶│ (azure\_ai\_agent)│───▶│ (jira\_client) │  
└─────────────────┘ └─────────────────┘ └─────────────────┘  
 │  
 ▼  
┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐  
│ Similarity │ │ JIRA Auth │ │ Configuration │  
│ Detector │◀───│ (jira\_auth) │ │ (config) │  
└─────────────────┘ └─────────────────┘ └─────────────────┘

Core Components

1. Azure AI Agent (`azure\_ai\_agent.py`)

\*\*Purpose\*\*: Main orchestrator that coordinates the entire duplicate detection process

\*\*Responsibilities\*\*:

Initialize and validate configuration

Manage JIRA authentication

Coordinate issue fetching and analysis

Generate comprehensive reports

Handle error scenarios

2. JIRA Authentication (`jira\_auth.py`)

\*\*Purpose\*\*: Handles OAuth 2.0 authentication with JIRA

\*\*Key Features\*\*:

Automatic browser-based authentication

Local server callback capture

Token persistence and refresh

Secure token storage

3. JIRA Client (`jira\_client.py`)

\*\*Purpose\*\*: Interfaces with JIRA REST API

\*\*Capabilities\*\*:

Fetch issues using JQL queries

Handle pagination for large datasets

Retrieve detailed issue information

Support for Atlassian Cloud API

4. Similarity Detector (`similarity\_detector.py`)

\*\*Purpose\*\*: Core AI engine for similarity analysis

\*\*Algorithms\*\*:

TF-IDF Cosine Similarity (30%)

Sequence Matching (30%)

Word Overlap Analysis (20%)

N-gram Analysis (20%)

\*\*Features\*\*:

Text normalization and cleaning

Intelligent word mapping

Tiered similarity grouping

5. Configuration Management (`config.py`)

\*\*Purpose\*\*: Centralized configuration management

\*\*Features\*\*:

Environment variable handling

Configuration validation

Default value management

Data Flow

1. \*\*Authentication\*\*: User runs the application → OAuth flow → Token storage

2. \*\*Data Retrieval\*\*: Fetch issues from JIRA → Parse and normalize

3. \*\*Analysis\*\*: Calculate similarities → Group by similarity levels

4. \*\*AI Enhancement\*\*: Generate insights using Azure AI Foundry

5. \*\*Reporting\*\*: Create markdown report → Save JSON data

Technology Stack

\*\*Language\*\*: Python 3.8+

\*\*AI/ML\*\*: Azure AI Foundry, scikit-learn, numpy

\*\*Authentication\*\*: OAuth 2.0, Atlassian Cloud API

\*\*Data Processing\*\*: pandas, TF-IDF vectorization

\*\*Web Framework\*\*: Built-in HTTP server for OAuth callback

\*\*Configuration\*\*: python-dotenv

---

📚 API Reference

AzureAIAgent Class

Constructor

AzureAIAgent()

Initializes the agent with configuration validation and component setup.

Methods

`authenticate\_jira() -> bool`

Authenticates with JIRA using OAuth 2.0 with persistent tokens.

\*\*Returns\*\*: `bool` - True if authentication successful, False otherwise

\*\*Example\*\*:

agent = AzureAIAgent()  
if agent.authenticate\_jira():  
 print("Authentication successful!")

`fetch\_issues() -> List[Dict]`

Fetches all issues from the configured JIRA project.

\*\*Returns\*\*: `List[Dict]` - List of issue dictionaries

\*\*Raises\*\*: `ValueError` if JIRA client not initialized

`analyze\_duplicates(issues: List[Dict]) -> Dict`

Analyzes issues for duplicates using AI.

\*\*Parameters\*\*:

`issues`: List of issue dictionaries

\*\*Returns\*\*: `Dict` containing:

`similar\_groups`: Dictionary of similarity groups

`insights`: AI-generated insights

`total\_issues\_analyzed`: Number of issues processed

`duplicate\_groups\_found`: Number of similarity groups

`generate\_report(analysis\_results: Dict) -> str`

Generates a comprehensive markdown report.

\*\*Parameters\*\*:

`analysis\_results`: Results from `analyze\_duplicates()`

\*\*Returns\*\*: `str` - Markdown-formatted report

`run\_analysis() -> Dict`

Runs the complete duplicate detection analysis.

\*\*Returns\*\*: `Dict` containing:

`success`: Boolean indicating success

`analysis\_results`: Analysis results

`report`: Generated report

`error`: Error message if failed

JIRAAuth Class

Methods

`get\_authorization\_url() -> str`

Generates the OAuth 2.0 authorization URL.

\*\*Returns\*\*: `str` - Authorization URL

`authenticate\_interactive() -> Dict[str, str]`

Performs interactive authentication with automatic callback handling.

\*\*Returns\*\*: `Dict[str, str]` - Token dictionary

`get\_valid\_tokens() -> Optional[Dict[str, str]]`

Gets valid tokens, either from file or by refreshing.

\*\*Returns\*\*: `Optional[Dict[str, str]]` - Valid tokens or None

JIRAClient Class

Constructor

JIRAClient(access\_token: str)

Methods

`get\_issues(project\_key: str, issue\_type: str, max\_results: int = 1000) -> List[Dict]`

Fetches issues from a JIRA project.

\*\*Parameters\*\*:

`project\_key`: JIRA project key

`issue\_type`: Type of issues to fetch

`max\_results`: Maximum number of results

\*\*Returns\*\*: `List[Dict]` - List of issue dictionaries

SimilarityDetector Class

Methods

`find\_similar\_issues(issues: List[Dict]) -> Dict[str, Dict]`

Finds similar issues using tiered similarity detection.

\*\*Parameters\*\*:

`issues`: List of issue dictionaries

\*\*Returns\*\*: `Dict[str, Dict]` - Dictionary of similarity groups

`calculate\_text\_similarity(text1: str, text2: str) -> float`

Calculates similarity between two texts using multiple methods.

\*\*Parameters\*\*:

`text1`: First text

`text2`: Second text

\*\*Returns\*\*: `float` - Similarity score (0.0 to 1.0)

`normalize\_text(text: str) -> str`

Normalizes text for better similarity detection.

\*\*Parameters\*\*:

`text`: Input text

\*\*Returns\*\*: `str` - Normalized text

Config Class

Class Variables

`AZURE\_AI\_FOUNDRY\_ENDPOINT`: Azure AI Foundry endpoint URL

`AZURE\_AI\_FOUNDRY\_API\_KEY`: Azure AI Foundry API key

`AZURE\_AI\_FOUNDRY\_DEPLOYMENT\_NAME`: Deployment name for AI model

`JIRA\_SERVER\_URL`: JIRA server URL

`JIRA\_CLIENT\_ID`: OAuth client ID

`JIRA\_CLIENT\_SECRET`: OAuth client secret

`JIRA\_PROJECT\_KEY`: Target project key

`ISSUE\_TYPE`: Type of issues to analyze

`SIMILARITY\_THRESHOLD`: Base similarity threshold

Methods

`validate() -> bool`

Validates that all required configuration is present.

\*\*Returns\*\*: `bool` - True if valid, raises ValueError if invalid

---

👥 User Guide

Prerequisites

Before using the JIRA Duplicate Detection Agent, ensure you have:

1. \*\*Python 3.8+\*\* installed on your system

2. \*\*Azure AI Foundry account\*\* with API access

3. \*\*JIRA Cloud account\*\* with administrative access

4. \*\*Atlassian developer account\*\* for OAuth app creation

Installation

Step 1: Clone the Repository

git clone <repository-url>  
cd Backlog-AiFoundry

Step 2: Create Virtual Environment

python -m venv venv  
source venv/bin/activate # On Windows: venv\Scripts\activate

Step 3: Install Dependencies

pip install -r requirements.txt

Step 4: Configure Environment

cp env.example .env  
# Edit .env with your credentials

Configuration Setup

Azure AI Foundry Setup

1. \*\*Create Azure AI Foundry Resource\*\*:

Go to Azure Portal

Create a new AI Foundry resource

Note the endpoint URL and API key

2. \*\*Deploy AI Model\*\*:

Deploy a GPT-4 model (e.g., `gpt-4o`)

Note the deployment name

3. \*\*Update Configuration\*\*:

AZURE\_AI\_FOUNDRY\_ENDPOINT=https://your-resource.cognitiveservices.azure.com  
 AZURE\_AI\_FOUNDRY\_API\_KEY=your\_api\_key  
 AZURE\_AI\_FOUNDRY\_DEPLOYMENT\_NAME=gpt-4o  
 AZURE\_AI\_FOUNDRY\_EMBEDDING\_MODEL=gpt-4o

JIRA OAuth Setup

1. \*\*Create OAuth App\*\*:

Go to [Atlassian Developer Console](https://developer.atlassian.com/console/myapps/)

Create a new OAuth 2.0 app

Set callback URL to: `http://localhost:8080/callback`

2. \*\*Get Credentials\*\*:

Note the client ID and client secret

Get your cloud ID from JIRA URL

3. \*\*Update Configuration\*\*:

JIRA\_SERVER\_URL=https://your-domain.atlassian.net  
 JIRA\_CLIENT\_ID=your\_client\_id  
 JIRA\_CLIENT\_SECRET=your\_client\_secret  
 JIRA\_CLOUD\_ID=your\_cloud\_id  
 JIRA\_REDIRECT\_URI=http://localhost:8080/callback  
 JIRA\_PROJECT\_KEY=YOUR\_PROJECT\_KEY  
 ISSUE\_TYPE=Story  
 SIMILARITY\_THRESHOLD=0.3

Usage

Quick Start

1. \*\*Run the Agent\*\*:

source venv/bin/activate  
 python main.py

2. \*\*Test Mode\*\* (no authentication required):

source venv/bin/activate  
 python test\_mode\_agent.py

Authentication Flow

The application handles OAuth 2.0 authentication automatically:

1. \*\*First Run\*\*: Browser opens for authorization

2. \*\*Automatic Callback\*\*: Local server captures authorization code

3. \*\*Token Storage\*\*: Tokens saved for future use

4. \*\*Subsequent Runs\*\*: Automatic token refresh

Understanding the Output

The agent generates two types of output:

1. \*\*Markdown Report\*\* (`duplicate\_analysis\_report\_{timestamp}.md`):

Executive summary with similarity tier counts

All issues analyzed with complete details

Tiered similarity groups with percentage scores

AI-powered insights and recommendations

Direct JIRA links for easy navigation

2. \*\*JSON Data\*\* (`duplicate\_analysis\_results\_{timestamp}.json`):

Raw analysis data

Similarity scores and metadata

Issue information for further processing

Similarity Tiers Explained

\*\*🔥 High Similarity (≥80%)\*\*: Likely duplicates requiring immediate consolidation

\*\*🟡 Medium Similarity (50-79%)\*\*: Potential duplicates requiring manual review

\*\*🟠 Low Similarity (30-49%)\*\*: Related issues that might benefit from coordination

Configuration Options

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Description** | **Default** | **Range** |
| `SIMILARITY\_THRESHOLD` | Base similarity threshold | 0.3 | 0.0-1.0 |
| `JIRA\_PROJECT\_KEY` | JIRA project to analyze | KAN | Any project key |
| `ISSUE\_TYPE` | Type of issues to analyze | Story | Any issue type |
| `MAX\_RESULTS` | Maximum issues to fetch | 1000 | 1-1000 |

---

🛠️ Development Guide

Project Structure

Backlog-AiFoundry/  
├── main.py # Main entry point  
├── azure\_ai\_agent.py # Core orchestrator  
├── jira\_auth.py # OAuth authentication  
├── jira\_client.py # JIRA API client  
├── similarity\_detector.py # AI similarity analysis  
├── config.py # Configuration management  
├── test\_mode\_agent.py # Test mode with sample data  
├── requirements.txt # Python dependencies  
├── env.example # Environment template  
├── .gitignore # Git ignore rules  
├── README.md # Project README  
└── DOCUMENTATION.md # This documentation

Development Setup

1. \*\*Clone and Setup\*\*:

git clone <repository-url>  
 cd Backlog-AiFoundry  
 python -m venv venv  
 source venv/bin/activate  
 pip install -r requirements.txt

2. \*\*Environment Configuration\*\*:

cp env.example .env  
 # Configure your environment variables

3. \*\*Run Tests\*\*:

python test\_mode\_agent.py

Code Architecture

Design Patterns

\*\*Singleton Pattern\*\*: Configuration management

\*\*Factory Pattern\*\*: Component initialization

\*\*Strategy Pattern\*\*: Multiple similarity algorithms

\*\*Observer Pattern\*\*: Progress reporting

Error Handling

The system implements comprehensive error handling:

\*\*Configuration Validation\*\*: Validates all required settings

\*\*Authentication Errors\*\*: Handles OAuth failures gracefully

\*\*API Errors\*\*: Manages JIRA API failures

\*\*AI Errors\*\*: Fallback mechanisms for AI service failures

Logging and Debugging

Enable debug mode:

export DEBUG=1  
python main.py

Testing

Test Mode

The `test\_mode\_agent.py` provides a testing environment:

Uses sample data instead of real JIRA issues

No authentication required

Demonstrates all functionality

Generates test reports

Unit Testing

Create unit tests for individual components:

import unittest  
from similarity\_detector import SimilarityDetector  
  
class TestSimilarityDetector(unittest.TestCase):  
 def setUp(self):  
 self.detector = SimilarityDetector()  
   
 def test\_text\_similarity(self):  
 text1 = "Login error"  
 text2 = "Authentication error"  
 similarity = self.detector.calculate\_text\_similarity(text1, text2)  
 self.assertGreater(similarity, 0.0)  
 self.assertLessEqual(similarity, 1.0)

Extending the System

Adding New Similarity Algorithms

1. \*\*Create Algorithm Function\*\*:

def new\_similarity\_algorithm(self, text1: str, text2: str) -> float:  
 # Implementation  
 return similarity\_score

2. \*\*Integrate into Main Algorithm\*\*:

def calculate\_text\_similarity(self, text1: str, text2: str) -> float:  
 # Add new algorithm to weighted combination  
 final\_similarity = (  
 0.25 \* seq\_similarity +  
 0.25 \* tfidf\_similarity +  
 0.25 \* overlap\_similarity +  
 0.25 \* new\_algorithm\_similarity  
 )

Adding New Report Formats

1. \*\*Create Report Generator\*\*:

def generate\_json\_report(self, analysis\_results: Dict) -> str:  
 # Implementation  
 return json\_string

2. \*\*Integrate into Main Agent\*\*:

def generate\_report(self, analysis\_results: Dict) -> str:  
 # Add new format option  
 if self.config.REPORT\_FORMAT == 'json':  
 return self.generate\_json\_report(analysis\_results)

Performance Optimization

Caching

Implement caching for frequently accessed data:

from functools import lru\_cache  
  
@lru\_cache(maxsize=128)  
def get\_embedding(self, text: str) -> List[float]:  
 # Implementation with caching

Parallel Processing

Use multiprocessing for large datasets:

from multiprocessing import Pool  
  
def process\_issues\_parallel(issues):  
 with Pool() as pool:  
 results = pool.map(process\_single\_issue, issues)  
 return results

---

🚀 Deployment Guide

Production Deployment

Environment Setup

1. \*\*Server Requirements\*\*:

Python 3.8+

4GB RAM minimum

10GB disk space

Network access to Azure and JIRA

2. \*\*Security Considerations\*\*:

Use environment variables for secrets

Implement proper firewall rules

Use HTTPS for all communications

Regular security updates

Docker Deployment

Create `Dockerfile`:

FROM python:3.9-slim  
  
WORKDIR /app  
COPY requirements.txt .  
RUN pip install -r requirements.txt  
  
COPY . .  
CMD ["python", "main.py"]

Create `docker-compose.yml`:

version: '3.8'  
services:  
 jira-agent:  
 build: .  
 environment:  
 - AZURE\_AI\_FOUNDRY\_ENDPOINT=${AZURE\_AI\_FOUNDRY\_ENDPOINT}  
 - AZURE\_AI\_FOUNDRY\_API\_KEY=${AZURE\_AI\_FOUNDRY\_API\_KEY}  
 - JIRA\_SERVER\_URL=${JIRA\_SERVER\_URL}  
 - JIRA\_CLIENT\_ID=${JIRA\_CLIENT\_ID}  
 - JIRA\_CLIENT\_SECRET=${JIRA\_CLIENT\_SECRET}  
 volumes:  
 - ./reports:/app/reports

Cloud Deployment

Azure Container Instances

1. \*\*Build and Push Image\*\*:

docker build -t jira-agent .  
 docker tag jira-agent your-registry.azurecr.io/jira-agent  
 docker push your-registry.azurecr.io/jira-agent

2. \*\*Deploy Container\*\*:

az container create \  
 --resource-group myResourceGroup \  
 --name jira-agent \  
 --image your-registry.azurecr.io/jira-agent \  
 --environment-variables \  
 AZURE\_AI\_FOUNDRY\_ENDPOINT=$AZURE\_ENDPOINT \  
 AZURE\_AI\_FOUNDRY\_API\_KEY=$AZURE\_KEY

AWS ECS

1. \*\*Create Task Definition\*\*:

{  
 "family": "jira-agent",  
 "networkMode": "awsvpc",  
 "requiresCompatibilities": ["FARGATE"],  
 "cpu": "512",  
 "memory": "1024",  
 "containerDefinitions": [  
 {  
 "name": "jira-agent",  
 "image": "your-account.dkr.ecr.region.amazonaws.com/jira-agent",  
 "environment": [  
 {"name": "AZURE\_AI\_FOUNDRY\_ENDPOINT", "value": "your-endpoint"}  
 ]  
 }  
 ]  
 }

Monitoring and Maintenance

Logging

Implement structured logging:

import logging  
import json  
  
logging.basicConfig(  
 level=logging.INFO,  
 format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'  
)  
  
logger = logging.getLogger(\_\_name\_\_)  
  
def log\_analysis\_results(results):  
 logger.info(json.dumps({  
 "event": "analysis\_complete",  
 "issues\_analyzed": results["total\_issues\_analyzed"],  
 "groups\_found": results["duplicate\_groups\_found"]  
 }))

Health Checks

Create health check endpoint:

from flask import Flask, jsonify  
  
app = Flask(\_\_name\_\_)  
  
@app.route('/health')  
def health\_check():  
 return jsonify({  
 "status": "healthy",  
 "timestamp": datetime.now().isoformat()  
 })

Scheduled Execution

Use cron for regular execution:

# Run daily at 9 AM  
0 9 \* \* \* /path/to/venv/bin/python /path/to/main.py

Backup and Recovery

Data Backup

1. \*\*Configuration Backup\*\*:

cp .env .env.backup  
 cp jira\_tokens.json jira\_tokens.json.backup

2. \*\*Report Backup\*\*:

tar -czf reports\_backup\_$(date +%Y%m%d).tar.gz duplicate\_analysis\_report\_\*.md

Disaster Recovery

1. \*\*Restore Configuration\*\*:

cp .env.backup .env  
 cp jira\_tokens.json.backup jira\_tokens.json

2. \*\*Re-authentication\*\*:

rm jira\_tokens.json  
 python main.py # Will trigger re-authentication

---

🔧 Troubleshooting

Common Issues

Authentication Problems

\*\*Issue\*\*: OAuth authentication fails

\*\*Solutions\*\*:

1. Verify callback URL is `http://localhost:8080/callback`

2. Check OAuth app configuration in Atlassian Developer Console

3. Ensure client ID and secret are correct

4. Check if firewall is blocking port 8080

\*\*Issue\*\*: Token refresh fails

\*\*Solutions\*\*:

1. Delete `jira\_tokens.json` and re-authenticate

2. Check if refresh token is expired

3. Verify client secret hasn't changed

API Connection Issues

\*\*Issue\*\*: Azure AI Foundry connection fails

\*\*Solutions\*\*:

1. Verify endpoint URL and API key

2. Check if deployment name is correct

3. Ensure API quota hasn't been exceeded

4. Verify network connectivity

\*\*Issue\*\*: JIRA API connection fails

\*\*Solutions\*\*:

1. Check JIRA server URL and cloud ID

2. Verify access token is valid

3. Check if project key exists

4. Ensure user has access to the project

Similarity Detection Issues

\*\*Issue\*\*: No similarities found

\*\*Solutions\*\*:

1. Lower `SIMILARITY\_THRESHOLD` in configuration

2. Check if issues have sufficient text content

3. Verify text normalization is working

4. Review similarity algorithm weights

\*\*Issue\*\*: Too many false positives

\*\*Solutions\*\*:

1. Increase `SIMILARITY\_THRESHOLD`

2. Improve text normalization

3. Add domain-specific word mappings

4. Adjust algorithm weights

Debug Mode

Enable verbose logging:

export DEBUG=1  
export LOG\_LEVEL=DEBUG  
python main.py

Performance Issues

Memory Usage

\*\*Issue\*\*: High memory consumption

\*\*Solutions\*\*:

1. Process issues in batches

2. Implement streaming for large datasets

3. Use more efficient data structures

4. Add memory monitoring

Processing Speed

\*\*Issue\*\*: Slow similarity calculation

\*\*Solutions\*\*:

1. Use parallel processing

2. Implement caching for embeddings

3. Optimize text preprocessing

4. Use more efficient algorithms

Error Codes

|  |  |  |
| --- | --- | --- |
| **Code** | **Description** | **Solution** |
| AUTH\_001 | OAuth authentication failed | Check credentials and callback URL |
| API\_001 | Azure AI API error | Verify endpoint and API key |
| API\_002 | JIRA API error | Check server URL and access token |
| CONFIG\_001 | Missing configuration | Verify all required environment variables |
| SIM\_001 | Similarity calculation error | Check text content and algorithms |

Getting Help

1. \*\*Check Logs\*\*: Review application logs for detailed error information

2. \*\*Test Mode\*\*: Use `test\_mode\_agent.py` to isolate issues

3. \*\*Configuration\*\*: Verify all environment variables are set correctly

4. \*\*Documentation\*\*: Review this documentation for configuration options

5. \*\*Community\*\*: Check GitHub issues for similar problems

---

🤝 Contributing

Development Workflow

1. \*\*Fork the Repository\*\*:

git clone https://github.com/your-username/Backlog-AiFoundry.git  
 cd Backlog-AiFoundry

2. \*\*Create Feature Branch\*\*:

git checkout -b feature/amazing-feature

3. \*\*Make Changes\*\*:

Follow the existing code style

Add tests for new functionality

Update documentation as needed

4. \*\*Test Changes\*\*:

python test\_mode\_agent.py  
 python -m pytest tests/

5. \*\*Commit Changes\*\*:

git commit -m 'Add amazing feature'

6. \*\*Push to Branch\*\*:

git push origin feature/amazing-feature

7. \*\*Create Pull Request\*\*:

Open a pull request on GitHub

Provide detailed description of changes

Include test results and screenshots

Code Standards

Python Style

Follow PEP 8 guidelines

Use type hints for function parameters and return values

Write docstrings for all public functions and classes

Use meaningful variable and function names

Documentation

Update this documentation for any API changes

Include examples for new features

Document any breaking changes

Update the README if needed

Testing

Write unit tests for new functionality

Ensure all tests pass before submitting

Add integration tests for complex features

Test with both real and mock data

Issue Reporting

When reporting issues, please include:

1. \*\*Environment Information\*\*:

Python version

Operating system

Package versions

2. \*\*Configuration\*\*:

Relevant environment variables (without secrets)

JIRA project details

Azure AI Foundry configuration

3. \*\*Error Details\*\*:

Complete error message

Stack trace

Steps to reproduce

4. \*\*Logs\*\*:

Application logs

Debug output if available

Feature Requests

When requesting features, please provide:

1. \*\*Use Case\*\*: Describe the problem you're trying to solve

2. \*\*Proposed Solution\*\*: How you envision the feature working

3. \*\*Alternatives\*\*: Other approaches you've considered

4. \*\*Impact\*\*: Who would benefit from this feature

---

📄 License

This project is licensed under the MIT License - see the [LICENSELICENSE](LICENSE) file for details.

🆘 Support

For issues and questions:

1. \*\*Check Documentation\*\*: Review this documentation and the README

2. \*\*Search Issues\*\*: Look for similar issues on GitHub

3. \*\*Create Issue\*\*: Open a new issue with detailed information

4. \*\*Community\*\*: Join discussions in GitHub Discussions

🔄 Version History

\*\*v2.0.0\*\*: Tiered similarity detection with percentage scoring

\*\*v1.5.0\*\*: Automatic OAuth callback capture

\*\*v1.0.0\*\*: Initial release with OAuth 2.0 and Azure AI integration

🎯 Roadmap

Planned Features

\*\*Web Interface\*\*: Browser-based UI for easier interaction

\*\*Real-time Monitoring\*\*: Live dashboard for analysis progress

\*\*Advanced Analytics\*\*: Trend analysis and historical reporting

\*\*Integration APIs\*\*: REST API for external system integration

\*\*Machine Learning\*\*: Continuous learning from user feedback

\*\*Multi-language Support\*\*: Support for non-English JIRA instances

Future Enhancements

\*\*Custom Algorithms\*\*: User-defined similarity algorithms

\*\*Batch Processing\*\*: Handle larger datasets efficiently

\*\*Cloud Storage\*\*: Integration with cloud storage providers

\*\*Notification System\*\*: Email/Slack notifications for results

\*\*Workflow Integration\*\*: Direct JIRA workflow integration

---

\*This documentation is maintained alongside the codebase. Please contribute improvements and report any inaccuracies.\*

1. Project Overview

2. Architecture

3. API Reference

4. User Guide

5. Development Guide

6. Deployment Guide

7. Troubleshooting

8. Contributing