fastMCP Köln Presse

Ein robustes MCP-Server-Projekt, das RSS-Pressemitteilungen der Stadt Köln konsumiert und als MCP-Tools exponiert.

📋 Projektübersicht

Dieses Projekt implementiert einen vollständigen MCP (Model Context Protocol) Server, der:

- RSS-Feeds von der Stadt Köln abruft (`https://www.stadt-koeln.de/externe-dienste/rss/pressemeldungen.xml`)
- RSS-Inhalte robust parst und als strukturierte Daten bereitstellt
- Caching und Fehlerbehandlung implementiert
- Als MCP-Server für Local- und Cloud-Deployment konzipiert ist

✨ Features

MCP Tools

- **koeln.presse.latest(n=10)**: Neueste Pressemitteilungen abrufen
- **koeln.presse.search(query, limit=20)**: Pressemitteilungen durchsuchen
- **koeln.presse.get(id)**: Einzelnes Item per ID abrufen
- **koeln.presse.categories()**: Alle verfügbaren Kategorien auflisten

Technische Features

- ☑ Robustes XML-Parsing mit lxml
- ☑ HTTP-Client mit Timeout und Retry-Logic (3 Versuche, Exponential Backoff)
- ☑ In-Memory-Caching mit TTL (5 Minuten Standard)
- ☑ Pydantic-Datenmodelle mit vollständiger Typisierung
- ☑ JSON-Schema-konforme Responses
- ☑ Umfassende Fehlerbehandlung
- ☑ Docker-Support
- ☑ Health-Check-Endpoints
- ☑ Vollständige Test-Suite

🚀 Schnellstart

Voraussetzungen

- Python 3.11 oder höher
- pip (Python Package Installer)

Lokale Installation

1. **Repository klonen**:

```
git clone <repository-url>
cd fastmcp-koeln-presse
```

2. **Python-Umgebung erstellen**:

```
python -m venv .venv
# Windows:
.venv\\Scripts\\activate
# Linux/macOS:
source .venv/bin/activate
```

3. **Dependencies installieren**:

```
pip install -U pip
pip install fastmcp httpx lxml pydantic python-dateutil uvicorn pytest
```

4. **Umgebungsvariablen konfigurieren** (optional):

```
# Kopiere die Beispiel-Konfiguration
cp .env.example .env

# Bearbeite .env nach Bedarf:
CACHE_TTL=300          # Cache TTL in Sekunden
HTTP_TIMEOUT=8         # HTTP Timeout in Sekunden
HTTP_RETRIES=3         # Anzahl Retry-Versuche
```

5. **Server starten**:

```
# Standard: Python Module
python -m koeln_presse.server

# Alternative: Mit Uvicorn
uvicorn koeln_presse.server:app --host 0.0.0.0 --port 8000

# Oder mit dem Startup-Skript (Linux/macOS):
chmod +x run_local.sh
./run_local.sh
```

6. **Health Check**:

```
curl http://localhost:8000/health
```

API-Dokumentation

MCP Tool Schemas

1. koeln.presse.latest

Beschreibung: Ruft die neuesten Pressemitteilungen ab, sortiert nach Publikationsdatum.

Parameter:

```
{
  "n": {
    "type": "integer",
    "minimum": 1,
    "maximum": 100,
    "default": 10,
    "description": "Anzahl der zurückzugebenden Items"
  }
}
```

Response:

```
{
  "items": [
    {
      "id": "abc123",
      "title": "Stadt Köln informiert über...",
      "link": "https://www.stadt-koeln.de/pressemeldungen/123",
      "description": "<p>Köln, den 15. Oktober 2024</p>",
      "published_at": "2024-10-15T10:30:00+02:00",
      "categories": ["Politik", "Verkehr"],
      "source": "rss:stadt-koeln"
    }
  ]
}
```

2. koeln.presse.search

Beschreibung: Durchsucht Pressemitteilungen nach Titel, Beschreibung und Kategorien.

Parameter:

```
{
  "query": {
    "type": "string",
```

```
    "description": "Suchbegriff"
  },
  "limit": {
    "type": "integer",
    "minimum": 1,
    "maximum": 100,
    "default": 20,
    "description": "Maximale Anzahl Ergebnisse"
  }
}
```

Response:

```
{
  "items": [
    {
      "id": "def456",
      "title": "Baustellen in der Innenstadt",
      "link": "https://www.stadt-koeln.de/pressemeldungen/124",
      "description": "Aktuelle Baustellenübersicht",
      "published_at": "2024-10-14T14:15:00+02:00",
      "categories": ["Verkehr"],
      "source": "rss:stadt-koeln"
    }
  ]
}
```

Ranking-Logik:

- Titel-Matches: +3 Punkte
- Kategorie-Matches: +2 Punkte
- Beschreibung-Matches: +1 Punkt

3. koeln.presse.get

Beschreibung: Ruft ein einzelnes Pressemitteilung per ID ab.

Parameter:

```
{
  "id": {
    "type": "string",
    "description": "Press Item ID"
  }
}
```

Response:

```
{
  "id": "abc123",
  "title": "Stadt Köln informiert über...",
  "link": "https://www.stadt-koeln.de/pressemeldungen/123",
  "description": "Köln, den 15. Oktober 2024",
  "published_at": "2024-10-15T10:30:00+02:00",
  "categories": ["Politik"],
  "source": "rss:stadt-koeln"
}
```

Fehler: 404 wenn Item nicht gefunden

4. koeln.presse.categories

Beschreibung: Gibt alle verfügbaren Kategorien alphabetisch sortiert zurück.

Parameter: Keine

Response:

```
{
  "categories": ["Baustellen", "Kultur", "Politik", "Verkehr"]
}
```

REST API Endpoints

Endpoint	Methode	Beschreibung
`/health`	GET	Health Check
`/manifest`	GET	MCP Tool Manifest
`/tools/latest`	POST	Latest Tool
`/tools/search`	POST	Search Tool
`/tools/get`	POST	Get Tool
`/tools/categories`	POST	Categories Tool

🐳 Docker Deployment

Docker Image erstellen

```
# Image bauen
docker build -t fastmcp-koeln-presse .
```

```
# Container starten
docker run -d -p 8000:8000 --name koeln-presse fastmcp-koeln-presse
```

Docker Compose (optional)

Erstelle `docker-compose.yml`:

```
version: '3.8'
services:
  koeln-presse:
    build: .
    ports:
      - "8000:8000"
    environment:
      - CACHE_TTL=300
      - HTTP_TIMEOUT=8
      - HTTP_RETRIES=3
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8000/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s
```

Starten:

```
docker-compose up -d
```

☁️ fastMCP Cloud Deployment

Vorbereitung

1. **Repository auf GitHub pushen**:

```
git add .
git commit -m "Initial commit: fastMCP Köln Presse Server"
git push origin main
```

2. **fastMCP Cloud Projekt erstellen**:
   - Gehe zu fastMCP Cloud
   - Klicke auf "New Project"
   - Wähle "Python" als Runtime
   - Python Version: 3.11

Deployment-Konfiguration

Build Command:

```
pip install -e .
```

Start Command:

```
python -m koeln_presse.server
```

Alternative Start Command (falls benötigt):

```
uvicorn koeln_presse.server:app --host 0.0.0.0 --port 8000
```

Environment Variables

In der fastMCP Cloud Console:

```
CACHE_TTL=300
HTTP_TIMEOUT=8
HTTP_RETRIES=3
PORT=8000
```

Health Check

Nach dem Deployment:

1. Gehe zur Projekt-URL
2. Teste: `GET /health`
3. Verifiziere Tool-Manifest: `GET /manifest`

⚙ Konfiguration

Umgebungsvariablen

Variable	Standard	Beschreibung
`CACHE_TTL`	300	Cache TTL in Sekunden (5 Min)
`HTTP_TIMEOUT`	8	HTTP Request Timeout in Sekunden
`HTTP_RETRIES`	3	Anzahl Retry-Versuche
`MAX_CACHE_SIZE`	1000	Maximale Anzahl gecachter Items

Variable	Standard	Beschreibung
`HOST`	0.0.0.0	Server Host
`PORT`	8000	Server Port
`RELOAD`	false	Auto-Reload aktivieren (Dev)

Konfigurationsdatei

Erstelle `.env` Datei:

```
# Cache Konfiguration
CACHE_TTL=300
MAX_CACHE_SIZE=1000

# HTTP Konfiguration
HTTP_TIMEOUT=8
HTTP_RETRIES=3

# Server Konfiguration
HOST=0.0.0.0
PORT=8000
RELOAD=false
```

🧪 Testing

Tests ausführen

```
# Alle Tests
pytest -v

# Nur RSS Client Tests
pytest tests/test_rss_client.py -v

# Nur Tool Tests
pytest tests/test_tools.py -v

# Mit Coverage
pytest --cov=koeln_presse --cov-report=html

# Quick Tests
pytest -q
```

Test-Suite Übersicht

- **test_rss_client.py**: Tests für RSS Client Funktionalität

    - XML Parsing (valid/invalid)
    - HTTP Client (Mocked)
    - Caching-Verhalten
    - Suche und Ranking
    - Fehlerbehandlung

- **test_tools.py**: Tests für MCP Server Tools
    - Tool-Parameter Validierung
    - Response Format
    - HTTP Status Codes
    - Error Handling

🔍 Akzeptanzkriterien

☑ **Funktionale Anforderungen**:

- ☑ `koeln.presse.latest` liefert standardmäßig 10 Items, absteigend nach Datum
- ☑ `koeln.presse.search("Köln")` gibt relevante Treffer, Titel-Treffer ranken höher
- ☑ `koeln.presse.get(id)` liefert exakt das Item, 404/Fehler bei unbekannter ID
- ☑ `koeln.presse.categories` gibt deduplizierte, sortierte Kategorien

☑ **Technische Anforderungen**:

- ☑ Netzwerkfehler → klare Fehlermeldung, keine Abstürze
- ☑ Projekt startet lokal & via Docker
- ☑ Tests laufen erfolgreich (`pytest -q`)
- ☑ README beschreibt fastMCP Cloud Deploy

☑ **Code Qualität**:

- ☑ Vollständige Typisierung (Python 3.11+)
- ☑ Robuste Fehlerbehandlung
- ☑ Pydantic-Schema Validierung
- ☑ Structured Logging

🐛 Troubleshooting

Häufige Probleme

1. "ModuleNotFoundError: No module named 'koeln_presse'"

Lösung:

```
# Von Projekt-Root aus ausführen:
python -m koeln_presse.server

# Oder Python Path setzen:
export PYTHONPATH="${PYTHONPATH}:$(pwd)"
```

2. "Connection timeout" beim RSS-Fetch

Lösung:

```
# Timeout in .env erhöhen:
HTTP_TIMEOUT=15
```

3. "Permission denied" beim Ausführen von run_local.sh

Lösung (Linux/macOS):

```
chmod +x run_local.sh
./run_local.sh
```

4. Docker Container startet nicht

Lösung:

```
# Logs prüfen:
docker logs koeln-presse

# Mit Debug-Modus:
docker run -it --rm fastmcp-koeln-presse python -m koeln_presse.server
```

5. Tests schlagen fehl

Lösung:

```
# Dependencies prüfen:
pip install -e ".[dev]"

# Tests mit mehr Details:
pytest -v --tb=long

# Spezifischer Test:
pytest tests/test_rss_client.py::TestRssClient::test_parse_items -v
```

Logging

Server-Logs ansehen:

```
# Lokal (stdout)
python -m koeln_presse.server

# Docker Logs
docker logs -f koeln-presse

# In Production (falls Log-File konfiguriert)
tail -f /var/log/koeln-presse.log
```

Log-Level setzen:

```
export LOG_LEVEL=DEBUG
python -m koeln_presse.server
```

Performance

Cache-Hit Rate überprüfen:

```
# In Python Console:
from koeln_presse.rss_client import client
print(f"Cache Size: {client._cache.size()}")
print(f"Cache TTL: {client.cache_ttl}s")
```

Memory Usage überwachen:

```
# Docker Stats:
docker stats koeln-presse

# System Memory:
free -h
```

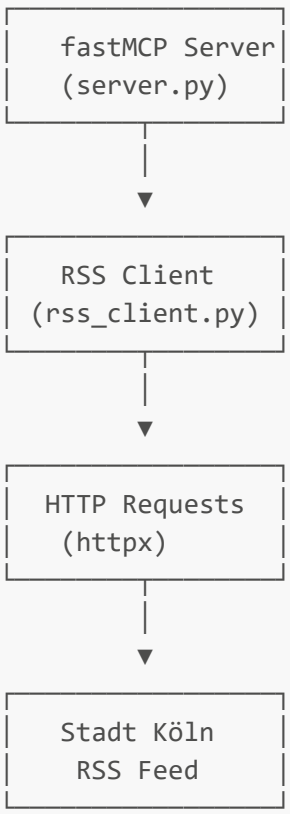
📖 Entwicklung

Projektstruktur

```
fastmcp-koeln-presse/
├── src/
│   └── koeln_presse/
│       ├── __init__.py          # Package Info
│       ├── models.py            # Pydantic Models
│       ├── rss_client.py        # RSS Client & Parser
│       ├── server.py            # fastMCP Server
```

```
│       └── utils.py              # Utility Functions
├── tests/
│   ├── test_rss_client.py       # RSS Client Tests
│   └── test_tools.py            # MCP Tools Tests
├── pyproject.toml               # Project Configuration
├── Dockerfile                   # Docker Image
├── run_local.sh                # Local Startup Script
├── .env.example                # Environment Template
├── .gitignore                  # Git Ignore Rules
└── README.md                   # This File
```

Architektur



Erweiterte Features (Optional)

1. Disk-Cache implementieren:

```python
# In rss_client.py ergänzen:
import os
import json
from pathlib import Path

class DiskCache:
    def __init__(self, cache_dir: str = ".cache"):
```

```
        self.cache_dir = Path(cache_dir)
        self.cache_dir.mkdir(exist_ok=True)
```

2. Fuzzy Search (Levenshtein):

```
# requirements.txt ergänzen:
python-Levenshtein>=0.20.0

# In utils.py:
from Levenshtein import ratio

def fuzzy_match(text: str, query: str, threshold: float = 0.7) -> bool:
    return ratio(text.lower(), query.lower()) >= threshold
```

3. Zusätzliche RSS-Feeds:

```
# In rss_client.py:
RSS_FEEDS = {
    "koeln": "https://www.stadt-koeln.de/externe-
dienste/rss/pressemeldungen.xml",
    "bonn": "https://www.bonn.de/rss/pressemeldungen.xml"
}
```

Lizenz

MIT License - siehe LICENSE Datei für Details.

Beitrag leisten

1. Fork das Repository
2. Feature Branch erstellen (`git checkout -b feature/neue-funktion`)
3. Changes committen (`git commit -am 'Neue Funktion hinzufügen'`)
4. Branch pushen (`git push origin feature/neue-funktion`)
5. Pull Request erstellen

Support

Bei Fragen oder Problemen:

1. GitHub Issues prüfen
2. README Troubleshooting-Sektion lesen
3. Tests ausführen zur Diagnose
4. Issue mit Logs und Environment-Details erstellen

Entwickelt von: Senior Python Engineer & MCP-Spezialist
Version: 1.0.0
Letztes Update: Oktober 2024