Tooltip Service API 테스트 케이스 설계서

📖 개요

서비스명: Tooltip Service
테스트 대상: AnalysisController API
테스트 환경:

- **개발**: `http://localhost:8086`
- **통합**: Gateway를 통한 테스트
  테스트 도구: JUnit 5, MockMvc, TestContainers (Redis)

🧪 테스트 전략

테스트 레벨

1. **단위 테스트**: 컨트롤러 로직
2. **통합 테스트**: 서비스 + DB + Redis
3. **계약 테스트**: API 스펙 준수
4. **성능 테스트**: 응답 시간 및 처리량

테스트 범위

- ☑ 정상 케이스 (Happy Path)
- ☑ 경계값 테스트 (Boundary)
- ☑ 예외 상황 (Exception)
- ☑ 캐시 동작 (Cache Behavior)

🔬 API 1: 뉴스 본문 분석 및 마크업

테스트 클래스: `ProcessContentApiTest`

TC-001: 정상적인 뉴스 본문 분석

```
테스트 ID: TC-001
테스트 명: 정상적인 뉴스 본문 분석 및 마크업 적용
우선순위: High
카테고리: 정상 케이스

전제조건:
  - 토큰용 DB에 "예산" 단어가 등록되어 있음
  - Redis 캐시가 비어있음

테스트 데이터:
  newsId: 12345
  originalContent: '정부가 내년 예산을 편성했다.'
```

```
실행 단계: 1. POST /api/news/analysis/process 요청
  2. 요청 본문에 테스트 데이터 포함

예상 결과:
  - 응답 코드: 200 OK
  - processedContent에 <span> 태그가 포함됨
  - Redis에 결과가 캐싱됨

검증:
  - 응답 JSON 구조 확인
  - 마크업 태그 정확성 확인
  - 캐시 저장 여부 확인
```

TC-002: 캐시 히트 시나리오

```
테스트 ID: TC-002
테스트 명: 동일한 뉴스 ID로 두 번째 요청 시 캐시 반환
우선순위: High
카테고리: 캐시 테스트

전제조건:
  - TC-001 실행 완료로 캐시에 데이터 존재

테스트 데이터:
  newsId: 12345 (동일)
  originalContent: "정부가 내년 예산을 편성했다." (동일)

실행 단계:
  1. POST /api/news/analysis/process 재요청
  2. 응답 시간 측정

예상 결과:
  - 응답 코드: 200 OK
  - 응답 시간: 첫 번째 요청의 1/10 이하
  - 동일한 processedContent 반환

검증:
  - 캐시에서 조회했는지 로그 확인
  - 응답 시간 성능 확인
```

TC-003: 어려운 단어가 없는 본문

```
테스트 ID: TC-003
테스트 명: 어려운 단어가 포함되지 않은 일반 텍스트
우선순위: Medium
카테고리: 경계값 테스트
```

```
테스트 데이터:
  newsId: 99999
  originalContent: '오늘 날씨가 좋다.'

실행 단계: 1. POST /api/news/analysis/process 요청

예상 결과:
  - 응답 코드: 200 OK
  - processedContent가 originalContent와 동일
  - <span> 태그가 없음

검증:
  - 원본 텍스트 그대로 반환 확인
  - 불필요한 마크업이 없음 확인
```

TC-004: 빈 문자열 입력

```
테스트 ID: TC-004
테스트 명: 빈 문자열 또는 null 입력 처리
우선순위: Medium
카테고리: 예외 케이스

테스트 데이터:
  Case A: newsId: 1, originalContent: ""
  Case B: newsId: 2, originalContent: null

실행 단계:
  1. 각 케이스별로 POST 요청

예상 결과:
  - 응답 코드: 200 OK
  - 빈 문자열 또는 null 그대로 반환

검증:
  - 에러 없이 처리됨
  - 빈 결과 정상 반환
```

TC-005: 대용량 텍스트 처리

```
테스트 ID: TC-005
테스트 명: 10,000자 이상의 긴 뉴스 본문 처리
우선순위: Medium
카테고리: 성능 테스트

테스트 데이터:
  newsId: 55555
  originalContent: [10, 000자 이상의 텍스트]
```

```
실행 단계: 1. POST /api/news/analysis/process 요청
  2. 응답 시간 측정

예상 결과:
  - 응답 코드: 200 OK
  - 응답 시간: 5초 이내
  - 메모리 사용량 임계치 이내

검증:
  - 시간 초과 없이 처리
  - 마크업 정확성 샘플 확인
```

TC-006: 잘못된 JSON 형식

```
테스트 ID: TC-006
테스트 명: 잘못된 JSON 요청 본문
우선순위: Low
카테고리: 입력 검증

테스트 데이터:
  잘못된 JSON: '{"newsId": "abc", "originalContent"'

실행 단계: 1. 잘못된 JSON으로 POST 요청

예상 결과:
  - 응답 코드: 400 Bad Request
  - 에러 메시지 포함

검증:
  - Spring의 기본 JSON 파싱 에러 처리
```

🔍 API 2: 단어 정의 조회

테스트 클래스: `TermDefinitionApiTest`

TC-101: 존재하는 단어 정의 조회

```
테스트 ID: TC-101
테스트 명: DB에 존재하는 단어의 정의 조회
우선순위: High
카테고리: 정상 케이스

전제조건:
  - DB에 "예산" 단어와 2개 정의 등록됨
  - Redis 캐시가 비어있음

테스트 데이터:
```

```
  term: '예산'

실행 단계: 1. GET /api/news/analysis/definition/예산

예상 결과:
  - 응답 코드: 200 OK
  - term: '예산 (豫算)'
  - definitions 배열에 2개 항목
  - displayOrder 순서로 정렬됨

검증:
  - JSON 구조 정확성
  - 정의 개수 확인
  - 정렬 순서 확인
  - 캐시 저장 확인
```

TC-102: 한자 포함 단어 조회

```
테스트 ID: TC-102
테스트 명: 한자가 포함된 단어로 조회
우선순위: High
카테고리: 정상 케이스

테스트 데이터:
  term: "예산 (豫算)" (전체 형태)

실행 단계:
  1. GET /api/news/analysis/definition/예산%20(豫算)

예상 결과:
  - 응답 코드: 200 OK
  - 정확한 매칭으로 결과 반환

검증:
  - URL 인코딩 처리 확인
  - 한자 포함 검색 정상 동작
```

TC-103: 부분 일치 검색

```
테스트 ID: TC-103
테스트 명: 정확 일치가 없을 때 부분 일치로 검색
우선순위: Medium
카테고리: 검색 로직

전제조건:
  - DB에 "예산계획서" 단어 존재
  - "예산계획"이라는 정확한 단어는 없음
```

```
테스트 데이터:
  term: "예산계획"

실행 단계:
  1. GET /api/news/analysis/definition/예산계획

예상 결과:
  - 응답 코드: 200 OK
  - "예산계획서" 단어의 정의 반환

검증:
  - 부분 일치 검색 동작 확인
  - 가장 짧은 매칭 단어 우선 반환
```

TC-104: 존재하지 않는 단어

```
테스트 ID: TC-104
테스트 명: DB에 없는 단어 조회
우선순위: High
카테고리: 예외 케이스

테스트 데이터:
  term: '존재하지않는단어'

실행 단계: 1. GET /api/news/analysis/definition/존재하지않는단어

예상 결과:
  - 응답 코드: 404 Not Found
  - 응답 본문 없음

검증:
  - 404 상태 코드 정확히 반환
  - NoSuchElementException 정상 처리
```

TC-105: 캐시된 단어 정의 조회

```
테스트 ID: TC-105
테스트 명: 두 번째 요청 시 캐시에서 조회
우선순위: High
카테고리: 캐시 테스트

전제조건:
  - TC-101 실행으로 캐시 데이터 존재

테스트 데이터:
  term: "예산" (동일)

실행 단계:
```

```
  1. GET /api/news/analysis/definition/예산 재요청
  2. 응답 시간 측정

예상 결과:
  - 응답 코드: 200 OK
  - 빠른 응답 시간 (< 50ms)
  - 동일한 결과 반환

검증:
  - 캐시 히트 로그 확인
  - DB 쿼리 실행되지 않음 확인
```

TC-106: 특수문자 포함 단어

```
테스트 ID: TC-106
테스트 명: 특수문자가 포함된 단어 조회
우선순위: Low
카테고리: 경계값 테스트

테스트 데이터:
  term: 'COVID-19'

실행 단계: 1. GET /api/news/analysis/definition/COVID-19

예상 결과:
  - 응답 코드: 200 OK 또는 404 Not Found
  - URL 인코딩 정상 처리

검증:
  - 특수문자 인코딩/디코딩 확인
  - SQL Injection 공격 방어 확인
```

🔧 통합 테스트

테스트 클래스: `AnalysisControllerIntegrationTest`

TC-201: End-to-End 시나리오

```
테스트 ID: TC-201
테스트 명: 뉴스 분석 후 단어 정의 조회까지 전체 플로우
우선순위: High
카테고리: 통합 테스트

시나리오: 1. 뉴스 본문 분석 요청 → 마크업된 결과 수신
  2. 마크업된 단어 중 하나 선택
  3. 해당 단어의 정의 조회 → 정의 수신
```

```
검증:
  - 두 API 간 데이터 일관성
  - 캐시가 양쪽 API에서 정상 동작
  - 전체 플로우 응답 시간 < 2초
```

TC-202: 동시성 테스트

```
테스트 ID: TC-202
테스트 명: 동일한 뉴스 ID로 동시 요청 처리
우선순위: Medium
카테고리: 동시성 테스트

실행:
  - 동일한 newsId로 10개 스레드가 동시 요청
  - 캐시 경합 상황 시뮬레이션

검증:
  - 모든 요청이 성공적으로 처리
  - 캐시 무결성 유지
  - DB 쿼리는 1번만 실행됨
```

📊 성능 테스트

테스트 클래스: `AnalysisControllerPerformanceTest`

TC-301: 처리량 테스트

```
테스트 ID: TC-301
테스트 명: API 처리량 (TPS) 측정
목표: 분당 1000건 이상 처리

시나리오:
  - 1분간 연속 요청 발송
  - 다양한 뉴스 ID 사용 (캐시 미스 유도)

측정 지표:
  - TPS (Transactions Per Second)
  - 평균 응답 시간
  - P95 응답 시간
  - 에러율

성공 기준:
  - TPS ≥ 16.7 (1000/분)
  - P95 응답 시간 ≤ 500ms
  - 에러율 ≤ 1%
```

TC-302: 메모리 사용량 테스트

```
테스트 ID: TC-302
테스트 명: 대용량 처리 시 메모리 사용량
목표: 메모리 누수 없이 안정적 처리

시나리오:
  - 10,000자 이상 텍스트로 1000번 요청
  - GC 동작 모니터링

측정 지표:
  - 힙 메모리 사용량
  - GC 빈도 및 시간
  - 메모리 누수 여부

성공 기준:
  - 최대 힙 사용량 < 512MB
  - OutOfMemoryError 발생 없음
  - GC 시간 < 전체 처리 시간의 10%
```

🎯 테스트 자동화

CI/CD 파이프라인 통합

단계별 테스트 실행

```
1. Commit 단계:
  - 단위 테스트 (TC-001~006, TC-101~106)
  - 빌드 검증

2. Integration 단계:
  - 통합 테스트 (TC-201~202)
  - TestContainers로 Redis 테스트

3. Pre-Production 단계:
  - 성능 테스트 (TC-301~302)
  - 계약 테스트

4. Production 배포 후:
  - 헬스체크
  - 스모크 테스트
```

테스트 데이터 관리

테스트 픽스처

```
-- 테스트용 어려운 단어 데이터
INSERT INTO vocabulary_term (vocabulary_term_id, term) VALUES
(1, '예산 (豫算)'),
(2, '경제'),
(3, '정책'),
(4, 'COVID-19'),
(5, '예산계획서');

-- 테스트용 정의 데이터
INSERT INTO term_definition (term_definition_id, vocabulary_term_id, definition,
display_order) VALUES
(1, 1, '필요한 비용을 미리 계산해 둠. 또는 그 돈.', 1),
(2, 1, '진작부터 마음에 두어 작정을 함. 또는 그 작정.', 2),
(3, 2, '재화의 생산, 분배, 소비에 관한 사회 전체의 활동.', 1);
```

📈 테스트 커버리지 목표

코드 커버리지

- **라인 커버리지**: ≥ 90%
- **브랜치 커버리지**: ≥ 85%
- **메서드 커버리지**: 100%

기능 커버리지

- **정상 케이스**: 100%
- **예외 케이스**: ≥ 80%
- **경계값 테스트**: ≥ 70%

🔍 테스트 실행 가이드

로컬 환경에서 실행

```
# 전체 테스트 실행
./gradlew test

# 특정 테스트 클래스 실행
./gradlew test --tests ProcessContentApiTest

# 성능 테스트 실행
./gradlew test --tests "*PerformanceTest" -Dtest.performance=true

# 커버리지 리포트 생성
./gradlew jacocoTestReport
```

Docker 환경에서 실행

```
# TestContainers로 Redis 포함 테스트
docker-compose -f docker-compose.test.yml up -d
./gradlew integrationTest
docker-compose -f docker-compose.test.yml down
```

📋 테스트 체크리스트

테스트 실행 전 확인사항

- ☐ Redis 서버 실행
- ☐ 테스트 DB 초기화
- ☐ 테스트 데이터 삽입
- ☐ 환경변수 설정

테스트 완료 후 확인사항

- ☐ 모든 테스트 PASS
- ☐ 커버리지 목표 달성
- ☐ 성능 기준 만족
- ☐ 테스트 데이터 정리

문서 버전: v1.0
최종 수정일: 2025-08-29
작성자: AI Assistant