뉴스레터 서비스 테스트 케이스 설계서

1. 테스트 개요

1.1 테스트 목적

- 뉴스레터 서비스의 기능 정상 동작 검증
- API 엔드포인트의 요청/응답 정확성 검증
- 에러 처리 및 예외 상황 대응 검증
- 성능 및 보안 요구사항 검증

1.2 테스트 범위

- 단위 테스트 (Unit Test)
- 통합 테스트 (Integration Test)
- API 테스트 (API Test)
- 성능 테스트 (Performance Test)

1.3 테스트 환경

- **개발 환경**: Local Development
- **테스트 환경**: Test Environment
- **스테이징 환경**: Staging Environment

2. 단위 테스트 (Unit Test)

2.1 NewsletterController 테스트

2.1.1 구독 관리 기능 테스트

TC-001: 뉴스레터 구독 성공

```
@Test
@DisplayName("유효한 구독 요청으로 뉴스레터 구독 성공")
void subscribe_Success() {
    // Given
    SubscriptionRequest request = SubscriptionRequest.builder()
        .email("test@example.com")
        .frequency(SubscriptionFrequency.DAILY)
        .preferredCategories(Arrays.asList(NewsCategory.POLITICS,
NewsCategory.ECONOMY))
        .keywords(Arrays.asList("AI", "블록체인"))
        .sendTime(9)
        .isPersonalized(true)
        .build();

    SubscriptionResponse expectedResponse = SubscriptionResponse.builder()
        .subscriptionId(1L)
```

```
            .userId(1L)
            .email("test@example.com")
            .frequency(SubscriptionFrequency.DAILY)
            .status("ACTIVE")
            .build();

        when(newsletterService.subscribe(any(SubscriptionRequest.class),
anyString()))
            .thenReturn(expectedResponse);

        // When
        ApiResponse<SubscriptionResponse> response =
newsletterController.subscribe(request, mockRequest);

        // Then
        assertThat(response.isSuccess()).isTrue();

assertThat(response.getData().getEmail()).isEqualTo("test@example.com");
        assertThat(response.getData().getStatus()).isEqualTo("ACTIVE");
    }
```

TC-002: 뉴스레터 구독 실패 - 잘못된 이메일

```
    @Test
    @DisplayName("잘못된 이메일 형식으로 구독 실패")
    void subscribe_Failure_InvalidEmail() {
        // Given
        SubscriptionRequest request = SubscriptionRequest.builder()
            .email("invalid-email")
            .frequency(SubscriptionFrequency.DAILY)
            .build();

        // When & Then
        assertThatThrownBy(() -> newsletterController.subscribe(request,
mockRequest))
            .isInstanceOf(MethodArgumentNotValidException.class);
    }
```

TC-003: 구독 정보 조회 성공

```
    @Test
    @DisplayName("존재하는 구독 ID로 구독 정보 조회 성공")
    void getSubscription_Success() {
        // Given
        Long subscriptionId = 1L;
        SubscriptionResponse expectedResponse = SubscriptionResponse.builder()
            .subscriptionId(subscriptionId)
            .userId(1L)
            .email("test@example.com")
```

```
            .status("ACTIVE")
            .build();

        when(newsletterService.getSubscription(subscriptionId, 1L))
            .thenReturn(expectedResponse);

        // When
        ResponseEntity<ApiResponse<SubscriptionResponse>> response =
            newsletterController.getSubscription(subscriptionId, mockRequest);

        // Then
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
        assertThat(response.getBody().isSuccess()).isTrue();

    assertThat(response.getBody().getData().getSubscriptionId()).isEqualTo(sub
    scriptionId);
    }
```

TC-004: 구독 정보 조회 실패 - 존재하지 않는 구독

```
    @Test
    @DisplayName("존재하지 않는 구독 ID로 조회 실패")
    void getSubscription_Failure_NotFound() {
        // Given
        Long subscriptionId = 999L;
        when(newsletterService.getSubscription(subscriptionId, 1L))
            .thenThrow(new NewsletterException("구독을 찾을 수 없습니다.",
    "SUBSCRIPTION_NOT_FOUND"));

        // When
        ResponseEntity<ApiResponse<SubscriptionResponse>> response =
            newsletterController.getSubscription(subscriptionId, mockRequest);

        // Then

    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.BAD_REQUEST);
        assertThat(response.getBody().isSuccess()).isFalse();

    assertThat(response.getBody().getErrorCode()).isEqualTo("SUBSCRIPTION_NOT_
    FOUND");
    }
```

2.1.2 콘텐츠 조회 기능 테스트

TC-005: 카테고리별 헤드라인 조회 성공

```
    @Test
    @DisplayName("카테고리별 헤드라인 조회 성공")
    void getCategoryHeadlines_Success() {
```

```
        // Given
        String category = "POLITICS";
        int limit = 5;
        List<NewsletterContent.Article> articles = Arrays.asList(
            NewsletterContent.Article.builder()
                .id(1L)
                .title("정치 뉴스 제목")
                .summary("정치 뉴스 요약")
                .category("POLITICS")
                .build()
        );

        when(newsletterService.getCategoryHeadlines(category, limit))
            .thenReturn(articles);

        // When
        ResponseEntity<ApiResponse<List<Map<String, Object>>>> response =
            newsletterController.getCategoryHeadlines(category, limit);

        // Then
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
        assertThat(response.getBody().isSuccess()).isTrue();
        assertThat(response.getBody().getData()).hasSize(1);
    }
```

TC-006: 트렌드 키워드 조회 성공

```
    @Test
    @DisplayName("트렌드 키워드 조회 성공")
    void getTrendingKeywords_Success() {
        // Given
        int limit = 10;
        List<String> keywords = Arrays.asList("AI", "블록체인", "메타버스");

        when(newsletterService.getTrendingKeywords(limit))
            .thenReturn(keywords);

        // When
        ResponseEntity<ApiResponse<List<String>>> response =
            newsletterController.getTrendingKeywords(limit);

        // Then
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
        assertThat(response.getBody().isSuccess()).isTrue();
        assertThat(response.getBody().getData()).hasSize(3);
    }
```

2.1.3 발송 관리 기능 테스트

TC-007: 뉴스레터 즉시 발송 성공

```
@Test
@DisplayName("뉴스레터 즉시 발송 성공")
void sendNewsletterNow_Success() {
    // Given
    NewsletterDeliveryRequest request =
NewsletterDeliveryRequest.builder()
        .newsletterId(1L)
        .targetUserIds(Arrays.asList(1L, 2L, 3L))
        .deliveryMethod(DeliveryMethod.EMAIL)
        .isPersonalized(true)
        .isScheduled(false)
        .build();

    DeliveryStats expectedStats = DeliveryStats.builder()
        .deliveryId(1L)
        .totalRecipients(3)
        .deliveredCount(3)
        .failedCount(0)
        .status("COMPLETED")
        .successRate(100.0)
        .build();


when(newsletterService.sendNewsletterNow(any(NewsletterDeliveryRequest.cla
ss), anyLong()))
        .thenReturn(expectedStats);

    // When
    ResponseEntity<ApiResponse<DeliveryStats>> response =
        newsletterController.sendNewsletterNow(request, mockRequest);

    // Then
    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
    assertThat(response.getBody().isSuccess()).isTrue();

assertThat(response.getBody().getData().getSuccessRate()).isEqualTo(100.0)
;
}
```

TC-008: 뉴스레터 예약 발송 성공

```
@Test
@DisplayName("뉴스레터 예약 발송 성공")
void scheduleNewsletter_Success() {
    // Given
    NewsletterDeliveryRequest request =
NewsletterDeliveryRequest.builder()
        .newsletterId(1L)
        .targetUserIds(Arrays.asList(1L, 2L, 3L))
        .deliveryMethod(DeliveryMethod.EMAIL)
```

```
            .isPersonalized(true)
            .isScheduled(true)
            .scheduledAt(LocalDateTime.now().plusHours(1))
            .build();

        DeliveryStats expectedStats = DeliveryStats.builder()
            .deliveryId(1L)
            .totalRecipients(3)
            .totalScheduled(3)
            .status("SCHEDULED")
            .build();


    when(newsletterService.scheduleNewsletter(any(NewsletterDeliveryRequest.cl
    ass), anyLong()))
            .thenReturn(expectedStats);

        // When
        ResponseEntity<ApiResponse<DeliveryStats>> response =
            newsletterController.scheduleNewsletter(request, mockRequest);

        // Then
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
        assertThat(response.getBody().isSuccess()).isTrue();

    assertThat(response.getBody().getData().getStatus()).isEqualTo("SCHEDULED"
    );
    }
```

2.2 NewsletterService 테스트

2.2.1 구독 관리 서비스 테스트

TC-009: 구독 생성 성공

```
    @Test
    @DisplayName("새로운 구독 생성 성공")
    void subscribe_Success() {
        // Given
        SubscriptionRequest request = SubscriptionRequest.builder()
            .email("new@example.com")
            .frequency(SubscriptionFrequency.WEEKLY)
            .preferredCategories(Arrays.asList(NewsCategory.SOCIETY))
            .build();

        String userId = "1";

        when(subscriptionRepository.save(any(Subscription.class)))
            .thenReturn(Subscription.builder()
                .id(1L)
                .userId(1L)
```

```
                .email("new@example.com")
                .frequency(SubscriptionFrequency.WEEKLY)
                .status(SubscriptionStatus.ACTIVE)
                .build());

        // When
        SubscriptionResponse response = newsletterService.subscribe(request,
userId);

        // Then
        assertThat(response.getEmail()).isEqualTo("new@example.com");

assertThat(response.getFrequency()).isEqualTo(SubscriptionFrequency.WEEKLY
);
        assertThat(response.getStatus()).isEqualTo("ACTIVE");
}
```

TC-010: 중복 구독 처리

```
@Test
@DisplayName("이미 존재하는 이메일로 구독 시도 시 중복 처리")
void subscribe_DuplicateEmail() {
    // Given
    SubscriptionRequest request = SubscriptionRequest.builder()
        .email("existing@example.com")
        .frequency(SubscriptionFrequency.DAILY)
        .build();

    String userId = "1";


when(subscriptionRepository.findByEmailAndUserId("existing@example.com",
1L))
        .thenReturn(Optional.of(Subscription.builder()
            .id(1L)
            .email("existing@example.com")
            .status(SubscriptionStatus.ACTIVE)
            .build()));

    // When & Then
    assertThatThrownBy(() -> newsletterService.subscribe(request, userId))
        .isInstanceOf(NewsletterException.class)
        .hasMessageContaining("이미 구독 중인 이메일입니다");
}
```

3. 통합 테스트 (Integration Test)

3.1 API 통합 테스트

3.1.1 구독 관리 API 통합 테스트

TC-011: 구독 생성부터 조회까지 전체 플로우

```
@Test
@DisplayName("구독 생성 -> 조회 -> 상태 변경 -> 해지 전체 플로우 테스트")
@Transactional
void subscriptionFullFlowTest() {
    // 1. 구독 생성
    SubscriptionRequest createRequest = SubscriptionRequest.builder()
        .email("flow@example.com")
        .frequency(SubscriptionFrequency.DAILY)
        .preferredCategories(Arrays.asList(NewsCategory.POLITICS))
        .build();

    ApiResponse<SubscriptionResponse> createResponse =
newsletterController
        .subscribe(createRequest, mockRequest);

    assertThat(createResponse.isSuccess()).isTrue();
    Long subscriptionId = createResponse.getData().getSubscriptionId();

    // 2. 구독 정보 조회
    ResponseEntity<ApiResponse<SubscriptionResponse>> getResponse =
newsletterController
        .getSubscription(subscriptionId, mockRequest);

    assertThat(getResponse.getStatusCode()).isEqualTo(HttpStatus.OK);

assertThat(getResponse.getBody().getData().getStatus()).isEqualTo("ACTIVE"
);

    // 3. 구독 상태 변경
    Map<String, String> statusRequest = Map.of("status", "PAUSED");
    ResponseEntity<ApiResponse<SubscriptionResponse>> statusResponse =
newsletterController
        .changeSubscriptionStatus(subscriptionId, statusRequest,
mockRequest);

    assertThat(statusResponse.getStatusCode()).isEqualTo(HttpStatus.OK);

assertThat(statusResponse.getBody().getData().getStatus()).isEqualTo("PAUS
ED");

    // 4. 구독 해지
    ResponseEntity<ApiResponse<String>> deleteResponse =
newsletterController
        .unsubscribe(subscriptionId, mockRequest);

    assertThat(deleteResponse.getStatusCode()).isEqualTo(HttpStatus.OK);
    assertThat(deleteResponse.getBody().isSuccess()).isTrue();
}
```

3.1.2 발송 관리 API 통합 테스트

TC-012: 뉴스레터 발송 전체 플로우

```
@Test
@DisplayName("뉴스레터 발송 전체 플로우 테스트")
@Transactional
void newsletterDeliveryFullFlowTest() {
    // 1. 구독 생성
    SubscriptionRequest subscriptionRequest =
SubscriptionRequest.builder()
        .email("delivery@example.com")
        .frequency(SubscriptionFrequency.DAILY)
        .build();

    ApiResponse<SubscriptionResponse> subscriptionResponse =
newsletterController
        .subscribe(subscriptionRequest, mockRequest);
    Long subscriptionId =
subscriptionResponse.getData().getSubscriptionId();

    // 2. 뉴스레터 즉시 발송
    NewsletterDeliveryRequest deliveryRequest =
NewsletterDeliveryRequest.builder()
        .newsletterId(1L)
        .targetUserIds(Arrays.asList(1L))
        .deliveryMethod(DeliveryMethod.EMAIL)
        .isPersonalized(true)
        .isScheduled(false)
        .build();

    ResponseEntity<ApiResponse<DeliveryStats>> deliveryResponse =
newsletterController
        .sendNewsletterNow(deliveryRequest, mockRequest);

    assertThat(deliveryResponse.getStatusCode()).isEqualTo(HttpStatus.OK);
    assertThat(deliveryResponse.getBody().isSuccess()).isTrue();

    Long deliveryId =
deliveryResponse.getBody().getData().getDeliveryId();

    // 3. 발송 취소
    ResponseEntity<ApiResponse<String>> cancelResponse =
newsletterController
        .cancelDelivery(deliveryId, mockRequest);

    assertThat(cancelResponse.getStatusCode()).isEqualTo(HttpStatus.OK);
    assertThat(cancelResponse.getBody().isSuccess()).isTrue();
}
```

3.2 데이터베이스 통합 테스트

TC-013: 구독 데이터 영속성 테스트

```
@Test
@DisplayName("구독 데이터가 데이터베이스에 정상적으로 저장되는지 확인")
@Transactional
void subscriptionPersistenceTest() {
    // Given
    Subscription subscription = Subscription.builder()
        .userId(1L)
        .email("persistence@example.com")
        .frequency(SubscriptionFrequency.WEEKLY)
        .status(SubscriptionStatus.ACTIVE)
        .preferredCategories(Arrays.asList(NewsCategory.ECONOMY))
        .build();

    // When
    Subscription savedSubscription =
subscriptionRepository.save(subscription);

    // Then
    assertThat(savedSubscription.getId()).isNotNull();

    Optional<Subscription> foundSubscription = subscriptionRepository
        .findByEmailAndUserId("persistence@example.com", 1L);

    assertThat(foundSubscription).isPresent();

assertThat(foundSubscription.get().getFrequency()).isEqualTo(SubscriptionF
requency.WEEKLY);

assertThat(foundSubscription.get().getStatus()).isEqualTo(SubscriptionStat
us.ACTIVE);
}
```

4. API 테스트 (API Test)

4.1 인증 테스트

TC-014: 인증 토큰 없이 접근 시 401 에러

```
@Test
@DisplayName("인증 토큰 없이 보호된 API 접근 시 401 에러 반환")
void unauthorizedAccessTest() {
    // Given
    SubscriptionRequest request = SubscriptionRequest.builder()
        .email("test@example.com")
        .frequency(SubscriptionFrequency.DAILY)
        .build();

    // When
```

```
    ResponseEntity<ApiResponse<SubscriptionResponse>> response =
        newsletterController.subscribe(request, mockRequestWithoutAuth);

    // Then

assertThat(response.getStatusCode()).isEqualTo(HttpStatus.UNAUTHORIZED);
}
```

TC-015: 잘못된 토큰으로 접근 시 401 에러

```
@Test
@DisplayName("잘못된 JWT 토큰으로 접근 시 401 에러 반환")
void invalidTokenTest() {
    // Given
    HttpServletRequest requestWithInvalidToken =
mock(HttpServletRequest.class);
    when(requestWithInvalidToken.getHeader("Authorization"))
        .thenReturn("Bearer invalid-token");

    SubscriptionRequest request = SubscriptionRequest.builder()
        .email("test@example.com")
        .frequency(SubscriptionFrequency.DAILY)
        .build();

    // When
    ResponseEntity<ApiResponse<SubscriptionResponse>> response =
        newsletterController.subscribe(request, requestWithInvalidToken);

    // Then

assertThat(response.getStatusCode()).isEqualTo(HttpStatus.UNAUTHORIZED);
}
```

4.2 입력 검증 테스트

TC-016: 필수 필드 누락 시 400 에러

```
@Test
@DisplayName("필수 필드 누락 시 400 에러 반환")
void missingRequiredFieldsTest() {
    // Given
    SubscriptionRequest request = SubscriptionRequest.builder()
        .frequency(SubscriptionFrequency.DAILY)
        .build(); // email 누락

    // When & Then
    assertThatThrownBy(() -> newsletterController.subscribe(request,
mockRequest))
```

```
        .isInstanceOf(MethodArgumentNotValidException.class);
}
```

TC-017: 잘못된 이메일 형식 시 400 에러

```
@Test
@DisplayName("잘못된 이메일 형식 시 400 에러 반환")
void invalidEmailFormatTest() {
    // Given
    SubscriptionRequest request = SubscriptionRequest.builder()
        .email("invalid-email-format")
        .frequency(SubscriptionFrequency.DAILY)
        .build();

    // When & Then
    assertThatThrownBy(() -> newsletterController.subscribe(request,
mockRequest))
        .isInstanceOf(MethodArgumentNotValidException.class);
}
```

TC-018: 잘못된 발송 시간 범위 시 400 에러

```
@Test
@DisplayName("발송 시간이 0-23 범위를 벗어날 때 400 에러 반환")
void invalidSendTimeRangeTest() {
    // Given
    SubscriptionRequest request = SubscriptionRequest.builder()
        .email("test@example.com")
        .frequency(SubscriptionFrequency.DAILY)
        .sendTime(25) // 0-23 범위 초과
        .build();

    // When & Then
    assertThatThrownBy(() -> newsletterController.subscribe(request,
mockRequest))
        .isInstanceOf(MethodArgumentNotValidException.class);
}
```

4.3 비즈니스 로직 테스트

TC-019: 존재하지 않는 구독 ID로 조회 시 404 에러

```
@Test
@DisplayName("존재하지 않는 구독 ID로 조회 시 적절한 에러 응답")
void nonExistentSubscriptionTest() {
    // Given
    Long nonExistentId = 999L;
```

```
        when(newsletterService.getSubscription(nonExistentId, 1L))
            .thenThrow(new NewsletterException("구독을 찾을 수 없습니다.",
"SUBSCRIPTION_NOT_FOUND"));

        // When
        ResponseEntity<ApiResponse<SubscriptionResponse>> response =
            newsletterController.getSubscription(nonExistentId, mockRequest);

        // Then

    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.BAD_REQUEST);
        assertThat(response.getBody().isSuccess()).isFalse();

    assertThat(response.getBody().getErrorCode()).isEqualTo("SUBSCRIPTION_NOT_
    FOUND");
    }
```

TC-020: 다른 사용자의 구독 정보 접근 시 403 에러

```
    @Test
    @DisplayName("다른 사용자의 구독 정보에 접근 시 403 에러 반환")
    void unauthorizedSubscriptionAccessTest() {
        // Given
        Long subscriptionId = 1L;
        when(newsletterService.getSubscription(subscriptionId, 1L))
            .thenThrow(new NewsletterException("접근 권한이 없습니다.",
    "ACCESS_DENIED"));

        // When
        ResponseEntity<ApiResponse<SubscriptionResponse>> response =
            newsletterController.getSubscription(subscriptionId, mockRequest);

        // Then

    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.BAD_REQUEST);

    assertThat(response.getBody().getErrorCode()).isEqualTo("ACCESS_DENIED");
    }
```

5. 성능 테스트 (Performance Test)

5.1 부하 테스트

TC-021: 동시 구독 요청 부하 테스트

```
    @Test
    @DisplayName("100개의 동시 구독 요청 처리 성능 테스트")
    void concurrentSubscriptionLoadTest() throws InterruptedException {
        // Given
```

```
    int concurrentUsers = 100;
    CountDownLatch latch = new CountDownLatch(concurrentUsers);
    ExecutorService executor =
Executors.newFixedThreadPool(concurrentUsers);
    List<Future<ApiResponse<SubscriptionResponse>>> futures = new
ArrayList<>();

    // When
    long startTime = System.currentTimeMillis();

    for (int i = 0; i < concurrentUsers; i++) {
        final int index = i;
        Future<ApiResponse<SubscriptionResponse>> future =
executor.submit(() -> {
            try {
                SubscriptionRequest request =
SubscriptionRequest.builder()
                    .email("loadtest" + index + "@example.com")
                    .frequency(SubscriptionFrequency.DAILY)
                    .build();

                return newsletterController.subscribe(request,
mockRequest);
            } finally {
                latch.countDown();
            }
        });
        futures.add(future);
    }

    latch.await(30, TimeUnit.SECONDS);
    long endTime = System.currentTimeMillis();

    // Then
    long totalTime = endTime - startTime;
    double avgResponseTime = (double) totalTime / concurrentUsers;

    assertThat(avgResponseTime).isLessThan(1000); // 평균 응답 시간 1초 이하

    // 모든 요청이 성공했는지 확인
    for (Future<ApiResponse<SubscriptionResponse>> future : futures) {
        ApiResponse<SubscriptionResponse> response = future.get();
        assertThat(response.isSuccess()).isTrue();
    }

    executor.shutdown();
}
```

TC-022: 대용량 데이터 조회 성능 테스트

```
@Test
@DisplayName("1000개 구독 목록 조회 성능 테스트")
```

```
void largeDataRetrievalPerformanceTest() {
    // Given
    int largeDataSetSize = 1000;
    List<SubscriptionResponse> largeDataSet = new ArrayList<>();

    for (int i = 0; i < largeDataSetSize; i++) {
        largeDataSet.add(SubscriptionResponse.builder()
            .subscriptionId((long) i)
            .userId(1L)
            .email("perftest" + i + "@example.com")
            .status("ACTIVE")
            .build());
    }

    when(newsletterService.getMySubscriptions(1L))
        .thenReturn(largeDataSet);

    // When
    long startTime = System.currentTimeMillis();
    ResponseEntity<ApiResponse<List<SubscriptionResponse>>> response =
        newsletterController.getMySubscriptions(mockRequest);
    long endTime = System.currentTimeMillis();

    // Then
    long responseTime = endTime - startTime;
    assertThat(responseTime).isLessThan(2000); // 2초 이하
    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
    assertThat(response.getBody().getData()).hasSize(largeDataSetSize);
}
```

5.2 메모리 사용량 테스트

TC-023: 메모리 누수 테스트

```
@Test
@DisplayName("반복적인 API 호출 시 메모리 누수 확인")
void memoryLeakTest() {
    // Given
    Runtime runtime = Runtime.getRuntime();
    long initialMemory = runtime.totalMemory() - runtime.freeMemory();

    // When
    for (int i = 0; i < 1000; i++) {
        SubscriptionRequest request = SubscriptionRequest.builder()
            .email("memorytest" + i + "@example.com")
            .frequency(SubscriptionFrequency.DAILY)
            .build();

        newsletterController.subscribe(request, mockRequest);

        // 가비지 컬렉션 강제 실행
```

```
            if (i % 100 == 0) {
                System.gc();
            }
        }

        // Then
        long finalMemory = runtime.totalMemory() - runtime.freeMemory();
        long memoryIncrease = finalMemory - initialMemory;

        // 메모리 증가량이 초기 메모리의 50% 이하여야 함
        assertThat(memoryIncrease).isLessThan(initialMemory * 0.5);
    }
```

6. 보안 테스트 (Security Test)

6.1 인젝션 공격 테스트

TC-024: SQL 인젝션 공격 방지 테스트

```
    @Test
    @DisplayName("SQL 인젝션 공격 시도 시 안전하게 처리되는지 확인")
    void sqlInjectionPreventionTest() {
        // Given
        String maliciousEmail = "test@example.com'; DROP TABLE subscriptions;
    --";
        SubscriptionRequest request = SubscriptionRequest.builder()
            .email(maliciousEmail)
            .frequency(SubscriptionFrequency.DAILY)
            .build();

        // When & Then
        // SQL 인젝션 시도가 안전하게 처리되어야 함
        assertThatThrownBy(() -> newsletterController.subscribe(request,
    mockRequest))
            .isInstanceOf(MethodArgumentNotValidException.class);
    }
```

TC-025: XSS 공격 방지 테스트

```
    @Test
    @DisplayName("XSS 공격 시도 시 안전하게 처리되는지 확인")
    void xssPreventionTest() {
        // Given
        String maliciousKeywords = Arrays.asList("<script>alert('XSS')
    </script>");
        SubscriptionRequest request = SubscriptionRequest.builder()
            .email("test@example.com")
            .frequency(SubscriptionFrequency.DAILY)
            .keywords(maliciousKeywords)
```

```
            .build();

        // When
        ApiResponse<SubscriptionResponse> response =
newsletterController.subscribe(request, mockRequest);

        // Then
        // XSS 공격이 방지되어야 함
        assertThat(response.isSuccess()).isTrue();
        // 저장된 키워드에 스크립트 태그가 포함되지 않아야 함
        assertThat(response.getData().getKeywords()).doesNotContain("
<script>");
    }
```

6.2 권한 테스트

TC-026: 다른 사용자 구독 정보 수정 시도 방지

```
    @Test
    @DisplayName("다른 사용자의 구독 정보 수정 시도 시 권한 거부")
    void unauthorizedModificationTest() {
        // Given
        Long otherUserSubscriptionId = 999L;
        Map<String, String> statusRequest = Map.of("status", "PAUSED");


    when(newsletterService.changeSubscriptionStatus(otherUserSubscriptionId,
    1L, "PAUSED"))
            .thenThrow(new NewsletterException("접근 권한이 없습니다.",
    "ACCESS_DENIED"));

        // When
        ResponseEntity<ApiResponse<SubscriptionResponse>> response =

    newsletterController.changeSubscriptionStatus(otherUserSubscriptionId,
    statusRequest, mockRequest);

        // Then

    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.BAD_REQUEST);

    assertThat(response.getBody().getErrorCode()).isEqualTo("ACCESS_DENIED");
    }
```

7. 테스트 데이터 관리

7.1 테스트 데이터 설정

```
@BeforeEach
void setUp() {
    // 테스트용 사용자 생성
    testUser = User.builder()
        .id(1L)
        .email("test@example.com")
        .nickname("테스트 사용자")
        .build();

    // 테스트용 구독 생성
    testSubscription = Subscription.builder()
        .id(1L)
        .userId(1L)
        .email("test@example.com")
        .frequency(SubscriptionFrequency.DAILY)
        .status(SubscriptionStatus.ACTIVE)
        .preferredCategories(Arrays.asList(NewsCategory.POLITICS))
        .build();

    // Mock 설정
    when(mockRequest.getHeader("Authorization"))
        .thenReturn("Bearer valid-jwt-token");
}
```

7.2 테스트 데이터 정리

```
@AfterEach
void tearDown() {
    // 테스트 데이터 정리
    subscriptionRepository.deleteAll();
    deliveryRepository.deleteAll();
}
```

8. 테스트 실행 가이드

8.1 테스트 실행 명령어

```
# 전체 테스트 실행
./gradlew test

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

# 특정 테스트 메서드 실행
./gradlew test --tests NewsletterControllerTest.subscribe_Success

# 통합 테스트만 실행
./gradlew integrationTest
```

```
# 성능 테스트만 실행
./gradlew performanceTest
```

8.2 테스트 커버리지 확인

```
# 테스트 커버리지 리포트 생성
./gradlew test jacocoTestReport

# 커버리지 리포트 확인
open build/reports/jacoco/test/html/index.html
```

8.3 테스트 환경 설정

```
# application-test.yml
spring:
  datasource:
    url: jdbc:h2:mem:testdb
    driver-class-name: org.h2.Driver
  jpa:
    hibernate:
      ddl-auto: create-drop
    show-sql: true
  test:
    database:
      replace: none

logging:
  level:
    com.newsletterservice: DEBUG
```

9. 테스트 결과 검증

9.1 성공 기준

- 모든 단위 테스트 통과 (커버리지 80% 이상)
- 모든 통합 테스트 통과
- API 응답 시간 2초 이하
- 메모리 누수 없음
- 보안 취약점 없음

9.2 테스트 리포트

- JUnit 테스트 리포트
- JaCoCo 커버리지 리포트
- 성능 테스트 리포트

- 보안 테스트 리포트

이 테스트 케이스 설계서를 통해 뉴스레터 서비스의 품질을 보장하고 안정적인 서비스를 제공할 수 있습니다.