[MailGuard 시스템]

[1]차 빌드 – 개발 명세서

2025 년 11 월 10 일

문서번호 : Genesis_SP1-TS_001

소 속 : 소프트웨어학부

팀 명 : Genesis

팀 원 : 고재현, 심수민, 전영우

교 수 : 조희승 교수님

<작성 가이드라인>

1. 주어진 양식에서 목차를 수정하지 않아야 합니다. 만약 임의 목차에 대하여 해당 사항이 존재하지 않는 경우는 해당 절의 목차를 삭제하지 않고, “해당 없음” 표시를 합니다.
2. 문서에는 반드시 페이지 번호를 기입합니다.
3. 주요 표와 그림은 해당 번호를 가져야 하며, 본문의 관련 내용에서 인용되어야 합니다.
4. 문서에 포함되는 모든 내용의 문구는 모호함이 없이 명확하게 표현되어야 합니다. ‘~ 할 수도 있다.’ 등의 표현은 사용하지 않도록 합니다.
5. 폰트는 맑은고딕을 사용하고, 제목 1은 20pt, 제목 2는 14pt, 제목 3은 12pt, 제목 4는 11pt, 표준은 10pt, 줄간격은 130%로 작성합니다. 목차 페이지는 11pt에 줄간격 160%로 작성합니다.
6. 문서 작성을 위한 편집기는 공동 활용을 위하여 ㅎㆍ글을 사용합니다.
7. 문서에 포함되는 그림은 고해상도의 이미지 형태로 삽입되어야 합니다.
8. 본 페이지는 삭제하고 작성합니다.

* 이 문서는 작성된 시점에서 고유한 문서 번호를 부여하고 관리 대상 문서에 포함시키나, 작성된 내용에 대한 변경 관리를 수행하지은 않습니다. 만약 해당 빌드의 요구사항이 변경되어 빌드 개발 계획서의 2장 요구사항 요약표가 변경되는 경우, 2장 분석 명세와 3장 설계명세에 변경된 사항을 반영하여 명세되어야 합니다.

목 차

1. 개발 개요

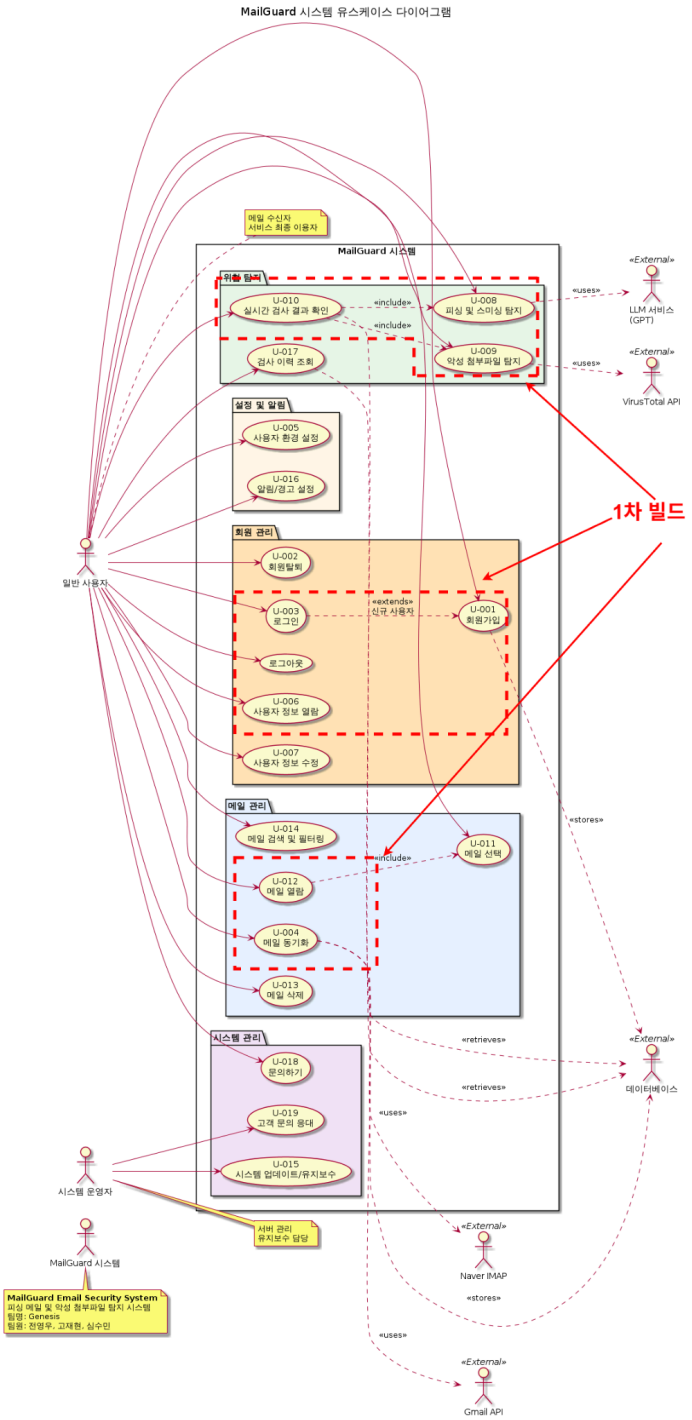
빌드 계획서에서 도출한 태스크 목록 테이블을 인용하여 정리합니다. 각 태스크별로 시작일과 종료일을 기술하여 해당 테스트의 완료 여부를 알 수 있도록 합니다. 종료되지 않은 태스크의 종료일은 “-”로 표시합니다.

태스크 번호	태스크 명	담당자	시작일	종료일	비고
T1-1	메일 DB에 저장된 본문에서 URL 분류	고재현	9/27		
T1-2	Mail API 연동 (Gmail, Naver)		9/27		
T1-3	UI 디자인		9/27		
T1-4	탐지 결과 인터페이스 표기 기능구현		9/27		
T1-5	관리/피드백 루프 구축		9/27		
T2-1	URL 차단 로직 구현	전영우	9/27	10/25	
T2-2	경고 메세지 팝업 알림		9/27	10/25	
T2-3	URL 의심 여부 판별		9/27	10/25	
T2-4	URL 접속 허용		9/27	10/25	
T2-5	URL 접속 차단		9/27	10/25	
T3-1	메일 본문/제목 분석용 LLM 모델 설정	고재현 전영우	9/27	-	
T3-2	메일 한국어 데이터셋 확보		9/27	-	
T3-3	LLM 모델 학습		9/27	-	
T3-4	어플리케이션과 상호작용 위한 API 구현		9/27	-	
T4-1	수신된 메일 목록 화면 구현(UI)	심수민	9/27	-	
T4-2	메일 목록에서 클릭 이벤트 구현		9/27	-	
T4-3	선택한 메일의 정보(첨부파일 유무, 본문 글)분석 로직 구현		9/27	10/19	
T5-1	첨부파일 정보 판별(압축파일/실행파일/문서파일 등 구분)		10/14	10/19	
T5-2	첨부파일 종류에 따른 처리 로직 구현(압축파일과 그외 파일일 경우로 분기)		10/14	10/19	
T5-3	파일 Hash 연산 수행 및 저장		10/14	10/19	
T5-4	압축파일 내부 정보 파싱(파일명 등)		10/14	10/19	
T6-1	파일 악성 판단 기준 설정		10/19	10/26	
T6-2	Virus Total API 연동		9/27	9/29	

T6-3	악성 파일 판단에 필요한 데이터 파싱		9/27	9/29	
T7-1	메일 정보 db연동		10/19	10/26	
T7-2	판단 기준에 따른 결과 제공 로직 구현		10/19	10/26	
T8-1	첨부파일 검사중 UI 구현		10/19	-	
T8-2	첨부파일 검사 완료/결과별 UI 구현		10/19	10/26	
T8-3	첨부파일 다운로드 경고 구현		10/19	-	
T8-4	검사 결과와 UI 연동		10/19	-	
T9-1	회원가입 페이지 UI 디자인	전영우	10/25	11/9	
T9-2	ID, 비밀번호 및 이메일 등록 기능 구현		10/25	11/9	
T9-3	db 스키마 구축 및 연동		10/25	11/9	
T9-4	개인정보 보호 위한 정보 보안 처리 (암호화 알고리즘)		10/25	-	
T10-1	본인인증 메일 생성 및 발송 기능 구현		10/25	-	
T10-2	발송 내역 및 상태 기록 기능 구현		10/25	-	
T11-1	발급된 코드와 일치 여부 확인 (유효기간 포함)		10/25	-	
T11-2	인증 실패 시 오류 안내		10/25	-	
T12-1	성공/실패 결과 출력 기능 구현		10/25	11/9	
T12-2	사용자 계정 활성화 처리 기능 구현		10/25	-	
T13-1	계정 삭제창 UI 디자인		10/25	-	
T13-2	계정 삭제 처리 구현		10/25	-	
T13-3	사용자 본인 확인(비밀번호 재입력, 2차 인증 등)		10/25	-	
T14-1	회원정보 UI 디자인		10/25	11/9	
T14-2	계정정보 수정 처리 구현		10/25	-	
T14-3	개인정보 보호 위한 정보 보안 처리 (암호화 알고리즘)		10/25	-	

2. 분석 명세

2.1 기능 명세 - Use Case Diagram (전체 시스템)



2.2 기능 명세 - (2) Use Case 설명서 (구현 시스템)

Use Case Name: 파일 관리	ID: 1	Importance Level: High
Primary Actor: 앱 사용자		Use Case Type: Essential, Overview

Stakeholders and interests:사용자: 수신한 이메일의 안전성을 확인하고 피싱 메일로부터 보호 받기를 원함

Brief Description:
사용자가 Gmail 또는 Naver 계정을 연동하면, 시스템이 이메일을 가져와서 URL 추출 및 규칙 기반 분석을 통해 피싱 위험도(0-100점)를 산출하고,
SAFE/SUSPICIOUS/DANGEROUS로 분류한 결과를 JSON 형식으로 반환합니다.

Trigger : user-initiated (사용자 시작)
Type : External

Relationships :
- Associations : Gmail/Naver 계정 연동
- Include : Oauth 2.0 인증 프로세스, 이메일 파싱 및 데이터 추출, 규칙
- Extend :
- Generalization :

Normal Flow of Event :

1. Gmail 계정 연동

    1.1 시스템이 Google OAuth 2.0 인증 플로우를 시작합니다
    - Google Cloud Console에 등록된 Client ID/Secret 사용
    - Redirect URI: http://localhost:8080/api/gmail/callback
    1.2 시스템이 사용자를 Google 인증 페이지로 리다이렉트합니다
    1.3 사용자가 Google 계정으로 로그인하고 Gmail 접근 권한을 승인합니다
    1.4 Google이 Authorization Code를 Redirect URI로 전달합니다
    1.5 시스템이 Authorization Code를 Access Token으로 교환합니다
    1.6 시스템이 Access Token을 저장하고 "연동 완료" 메시지를 표시합니다다
2. Naver 계정 연동

    2.1 사용자가 Naver 아이디와 비밀번호를 입력합니다
    2.2 시스템이 IMAP 프로토콜로 Naver 메일 서버에 연결합니다

3. 이메일 상세 내용 로드

    3.1 사용자가 분석할 이메일을 선택합니다
    3.2 시스템이 선택된 이메일의 상세 내용을 요청합니다
    3.3 시스템이 이메일 본문을 파싱합니다

4. 규칙 기반 피싱 탐지

    4.1 시스템이 발신자 도메인을 분석합니다

    4.2 시스템이 이메일 본문에서 의심 키워드를 탐지합니다

    4.3 시스템이 총 위험도 점수를 계산합니다 (각 항목 점수 합산)

5. 위험도 분류

    5.1 시스템이 계산된 점수를 기반으로 위험도를 분류합니다

Subflows:

Alternate/Exceptional Flows:

Gmail OAuth 인증 실패
Gmail Access Token 만료
Naver IMAP 인증 실패

Use Case Name: URL 차단	ID: 2	Importance Level: Low
Primary Actor: 앱 사용자		Use Case Type: Essential, Overview
Stakeholders and interests: 웹 사용자 - url 안전성 검증을 원합니다. 시스템 - 들어온 url 이 안전한 url인지 검증합니다.		
Brief Description: 해당 문서는 입력된 url을 차단하는 기능에 대한 use case description입니다.		
Trigger : URL 클릭 Type : External		
Relationships : - Associations : 웹 사이트 사용자 - Include : - Extend : - Generalization :		
Normal Flow of Event : 1. 시스템은 악성URL 리스트 db를 로딩합니다. 2. 사용자가 원하는 url로 리다이렉션을 시도합니다. 3. 시스템은 URL 이 악성 url 리스트 db에 있는지 확인합니다. 3.1. if URL이 안전하다면 리다이렉션을 진행합니다. 3.2. else if URL이 위험하다면 리다이렉션을 차단하고, 경고 메세지를 출력합니다.		
Subflows:		
Alternate/Exceptional Flows:		

Use Case Name: 메일 악성 검증 AI	ID: 3	Importance Level: High
Primary Actor: 시스템 설계자		Use Case Type: Essential, Overview
Stakeholders and interests: 웹 사용자 - 사용자는 받은 메일의 안전성 검증을 원합니다 시스템 - 제공된 메일의 제목, 본문을 분석하여 메일의 안전성을 검증합니다.		
Brief Description: 해당 문서는 메일의 안전성을 검증하는 AI에 대한 use case description입니다.		
Trigger : 메일 업로드 Type : External		

Relationships :
- Associations : 웹 사용자
- Include :
- Extend :
- Generalization :

Normal Flow of Event :
1. 시스템 설계자는 labeled된 데이터셋을 업로드합니다.
2. 시스템은 학습 / 검증용 데이터셋을 분할합니다.
3. 시스템은 학습에 사용할 수 있도록 데이터 토크나이징 / 데이터셋 변환을 진행합니다.
4. 시스템은 normal flow 3에서 반환된 데이터셋을 바탕으로, 모델 학습을 진행합니다.
5. 새로운 입력값이 들어오면 학습된 결과를 바탕으로 추론을 진행합니다.

Subflows:

Alternate/Exceptional Flows:

Use Case Name: 메일 정보 출력	ID: 4	Importance Level: High
Primary Actor: 웹 사이트 사용자		Use Case Type: Essential, Overview

Stakeholders and interests:
웹 사용자 - 메일 목록에서 메일을 선택합니다.
시스템 - 첨부파일을 받아와 파일 정보를 hash 값으로 변환합니다.

Brief Description:
해당 문서는 사용자가 등록한 외부 이메일 서비스 정보를 통해 메일 정보를 가져오는 기능에 대한 use case description입니다.

Trigger : 메일 목록에서 메일 클릭
Type :

Relationships :
- Associations : 첨부파일 처리, 파일 악성여부 출력
- Include : 회원가입, 로그인
- Extend :
- Generalization :

Normal Flow of Event :
1. 사용자는 메일 목록에 접속합니다.
2. 시스템은 등록된 사용자의 정보를 가지고 메일 목록을 출력합니다.
3. 3.1. if 사용자가 Google 계정 정보를 등록했을 경우→시스템은 Google Gmail API 이용.
   3.2. else 사용자가 Naver IMAP 정보를 등록했을 경우→시스템은 Naver IMAP 이용.
4. 사용자는 메일 목록에서 메일 하나를 선택합니다.
5. 시스템은 선택된 메일을 볼 수 있는 페이지로 리다이렉트 합니다.
6. 시스템은 선택된 메일의 정보(제목, 본문, 보낸 이, 첨부파일)를 출력합니다.

Subflows:
Alternate/Exceptional Flows: 1. Google 계정에 접속이 불가능할 경우. 2. Naver IMAP에 접속이 불가능할 경우.

Use Case Name: 첨부파일 처리	ID: 5	Importance Level: High
Primary Actor: 시스템		Use Case Type: Essential, Overview
Stakeholders and interests: 웹 사용자 - 선택한 메일의 첨부파일 다운로드 버튼을 누릅니다. 시스템 - 첨부파일을 받아와 파일 정보를 hash(SHA256) 값으로 변환합니다.		
Brief Description: 해당 문서는 이메일로 받은 첨부파일을 가져와 hash(SHA256) 값으로 연산하는 기능에 대한 use case description입니다.		
Trigger : 파일 다운로드 버튼, 파일 검사 버튼 Type :		
Relationships : - Associations : 웹 사용자, 시스템 - Include : - Extend : - Generalization :		
Normal Flow of Event : 1. 사용자는 첨부파일 다운로드 버튼을 클릭합니다. 2. 시스템은 첨부파일 데이터를 Google API/Naver IMAP을 통해 가져옵니다. 3. 시스템은 첨부파일이 압축파일(.zip)인지 확인합니다. 3.1. if 압축 파일일 경우 → 압축 해제 로직을 수행해 압축 전 파일로 변환해 사용합니다. 3.2. else 압축 파일이 아닐 경우 → 첨부파일을 그대로 사용합니다. 4. 시스템은 첨부파일 데이터를 이용해 hash(SHA256) 값을 구합니다.		
Subflows: 3.2.1 시스템은 압축 파일에 비밀번호가 걸려 있을 경우 사용자에게 비밀번호를 입력받습니다.		
Alternate/Exceptional Flows:		

Use Case Name: 파일 악성 여부 검사	ID: 6	Importance Level: High

Primary Actor: 시스템	Use Case Type: Essential, Overview
Stakeholders and interests: 웹 사용자 - 선택한 메일의 첨부파일을 다운로드 받습니다. 시스템 - 첨부파일 hash값을 가지고 악성파일 여부를 검사합니다.	
Brief Description: 해당 문서는 이메일로 받은 첨부파일이 악성인지 검사하는 기능에 대한 use case description입니다.	
Trigger : 파일 다운로드 버튼, 파일 검사 버튼 Type :	
Relationships : - Associations : 웹 사용자, 시스템, Virus Total API(외부 시스템), database - Include : 첨부파일 처리 - Extend : - Generalization :	
Normal Flow of Event : 1. 시스템은 VirusTotal API를 이용해 VirusTotal Service에 연결합니다. 2. 시스템은database에 저장된 정보와 현재 파일의 hash(SHA256)값을 비교합니다. 2.1 if database에 hash값이 존재하는 경우 → 3~6을 건너뛰고 7을 수행합니다. 3.2 else database에 hash값이 존재하지 않는 경우 → 3부터 수행합니다. 3. 시스템은 첨부파일의 Hash값을 VirusTotal에 전송합니다. 4. 시스템은 VirusTotal에서 Hash값에 해당하는 파일에 대한 JSON 정보를 받습니다. 3.1. if VirusTotal에 등록된 파일이 아닌 경우 → 확인할 수 없는 파일이라고 출력합니다. 3.2. else VirusTotal에 등록된 파일인 경우→ JSON 정보 파싱으로 넘어갑니다. 5. 시스템은 JSON 정보를 파싱해 악성, 의심, 안전 수치와 파일 명을 찾습니다. 6. 시스템은 검사완료한 첨부파일의 정보와 hash값을 database에 저장합니다. 7. 시스템은 첨부파일 검사 정보를 출력합니다. 6.1 if 악성, 의심 수치가 0을 초과할 경우 → 악성파일 판단 결과 출력합니다. 6.2 else 악성, 의심 수치가 0일 경우→정상 파일 판단 결과 출력합니다. 8. 사용자는 메일 첨부파일을 다운로드 받습니다.	
Subflows:	
Alternate/Exceptional Flows:	

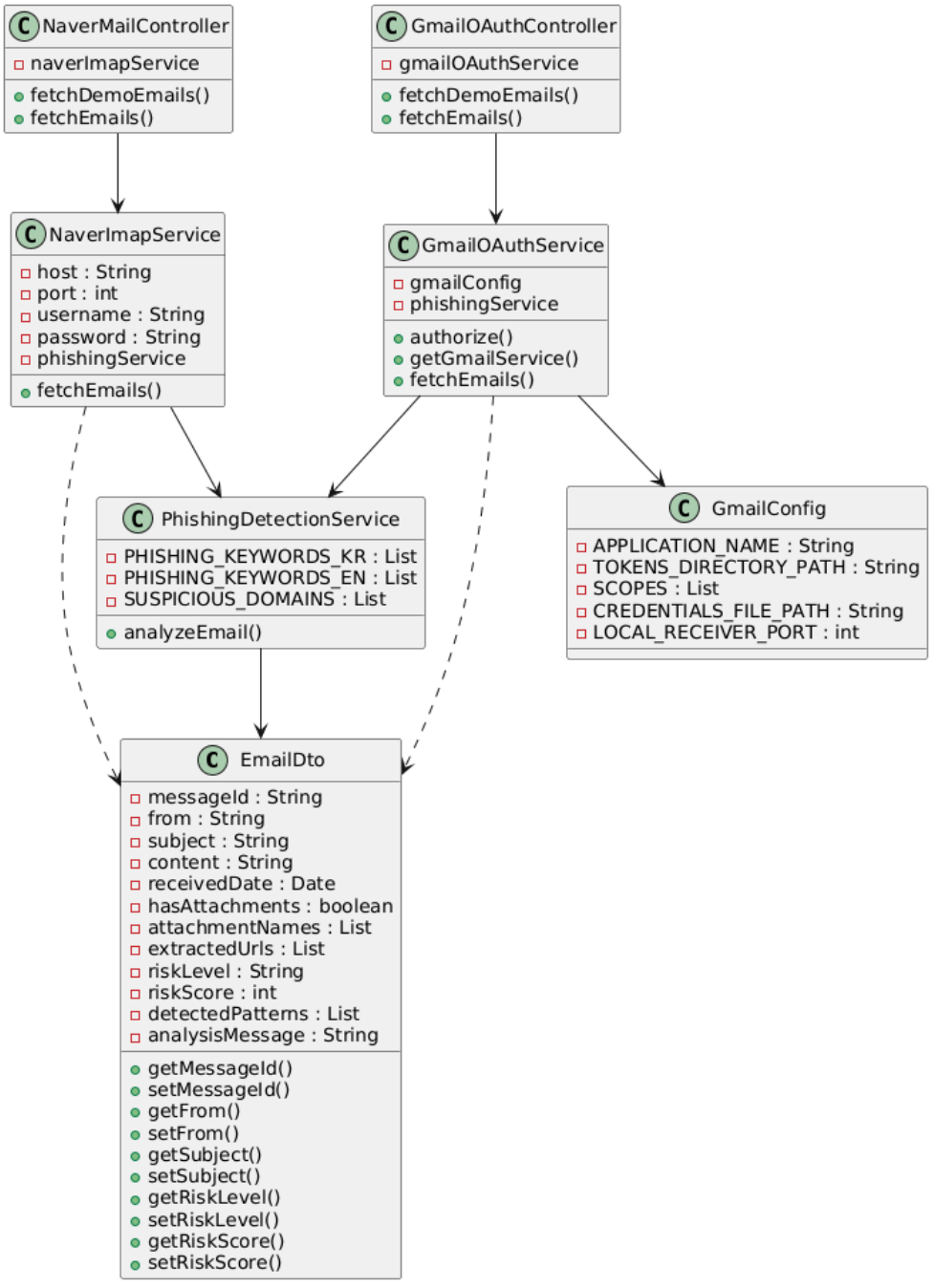
Use Case Name: 웹 사이트	ID: 7	Importance Level: Middle
Primary Actor: 웹 사용자		Use Case Type: Essential, Overview
Stakeholders and interests: 웹 사용자 - 웹사이트 이용을 원합니다..		

앱 시스템 - 요청된 사용자의 정보를 바탕으로 회원가입을 진행합니다.
Brief Description: 해당 문서는 웹 사이트에 대한 use case description입니다.
Trigger : 웹 사이트 접속 Type : External
Relationships : - Associations : 웹 사용자 - Include : - Extend : 회원가입, 로그인, gmail 불러오기 - Generalization :
Normal Flow of Event : 1. 사용자가 웹 사이트에 접속하면 welcome page를 보여줍니다. 1.1. if 홈페이지 버튼을 클릭 : 홈페이지로 이동합니다. 1.2. else if gmail 버튼 클릭 : gmail 메일 불러오기 기능을 수행합니다. 1.3. else if 로그인 버튼 클릭 : 로그인 기능을 수행합니다. 1.4. else if 프로필 버튼 클릭 : 사용자 프로필 화면을 보여줍니다. 2. 사용자가 웹사이트 사용을 종료합니다.
Subflows:
Alternate/Exceptional Flows:

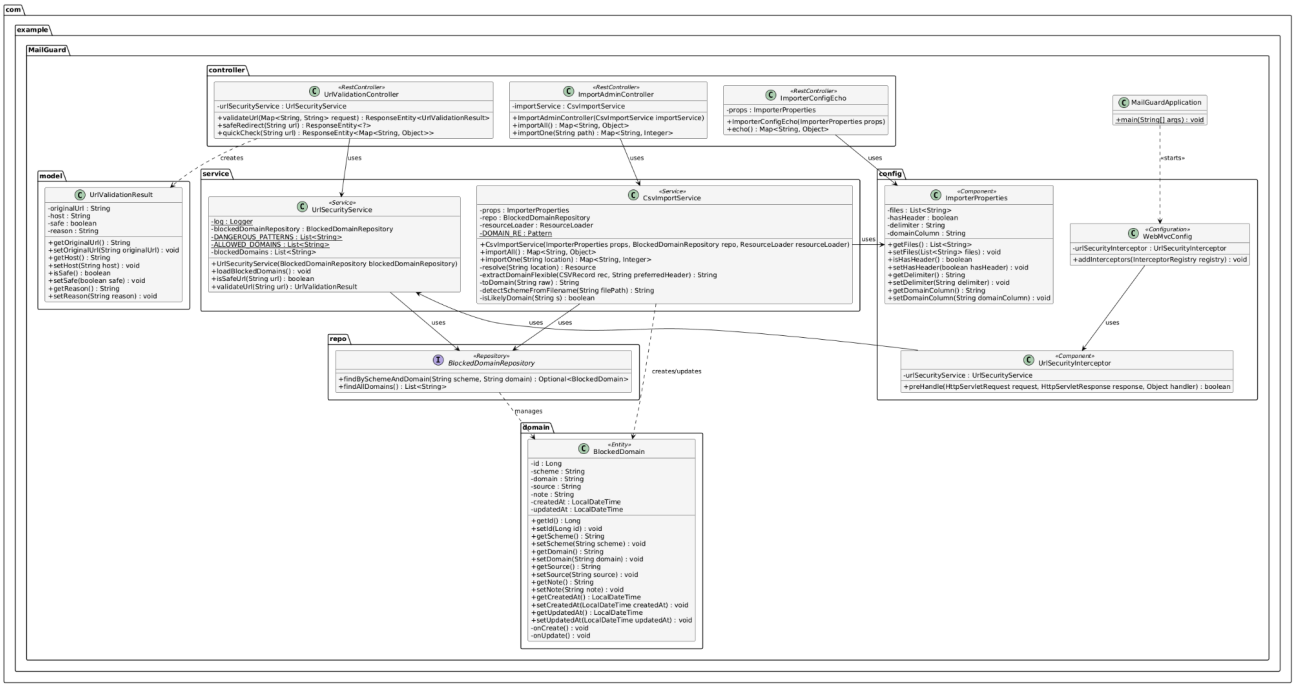
Use Case Name: 회원가입	ID: 8	Importance Level: Middle
Primary Actor: 웹 사용자		Use Case Type: Essential, Overview
Stakeholders and interests: 앱 사용자 - 애플리케이션 사용을 위해 회원가입을 원합니다. 앱 시스템 - 요청된 사용자의 정보를 바탕으로 회원가입을 진행합니다.		
Brief Description: 해당 문서는 웹 사이트 회원가입 로직에 대한 use case description입니다.		
Trigger : 회원가입 버튼 클릭 Type : External		
Relationships : - Associations : 웹 사용자 - Include : - Extend : - Generalization :		
Normal Flow of Event : 1. 사용자는 회원가입을 위해 이메일, 아이디, 비밀번호, confirm 비밀번호를 기입합니다. 2. 시스템은 사용자의 email이 중복되지 않는지 검사합니다. 2.1. if 사용자의 email이 이미 가입된 경우: 이메일 재기입을 요청합니다. 3. 시스템은 비밀번호와 confirm 비밀번호가 일치하는지 확인합니다. 3.1 if 비밀번호와 confirm 비밀번호가 매칭되지 않으면 : 비밀번호 확인을		

요청합니다. 4. database에 유저 정보를 업로드합니다.
Subflows:
Alternate/Exceptional Flows:

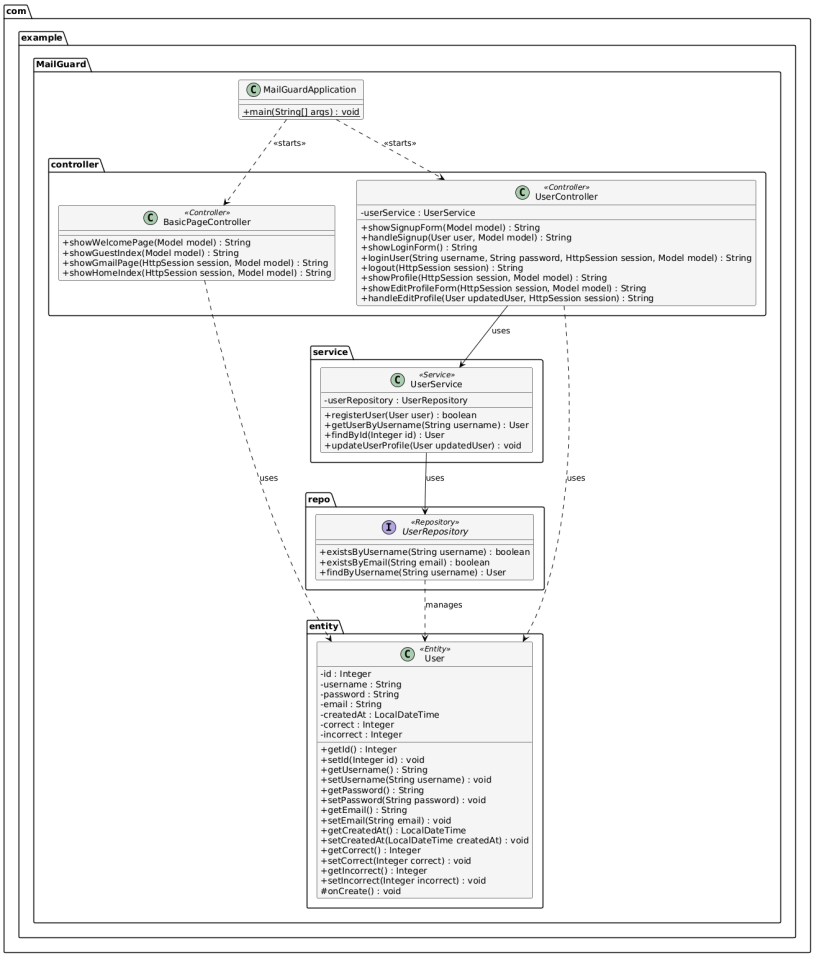
2.2 구조 명세 (구현 시스템)



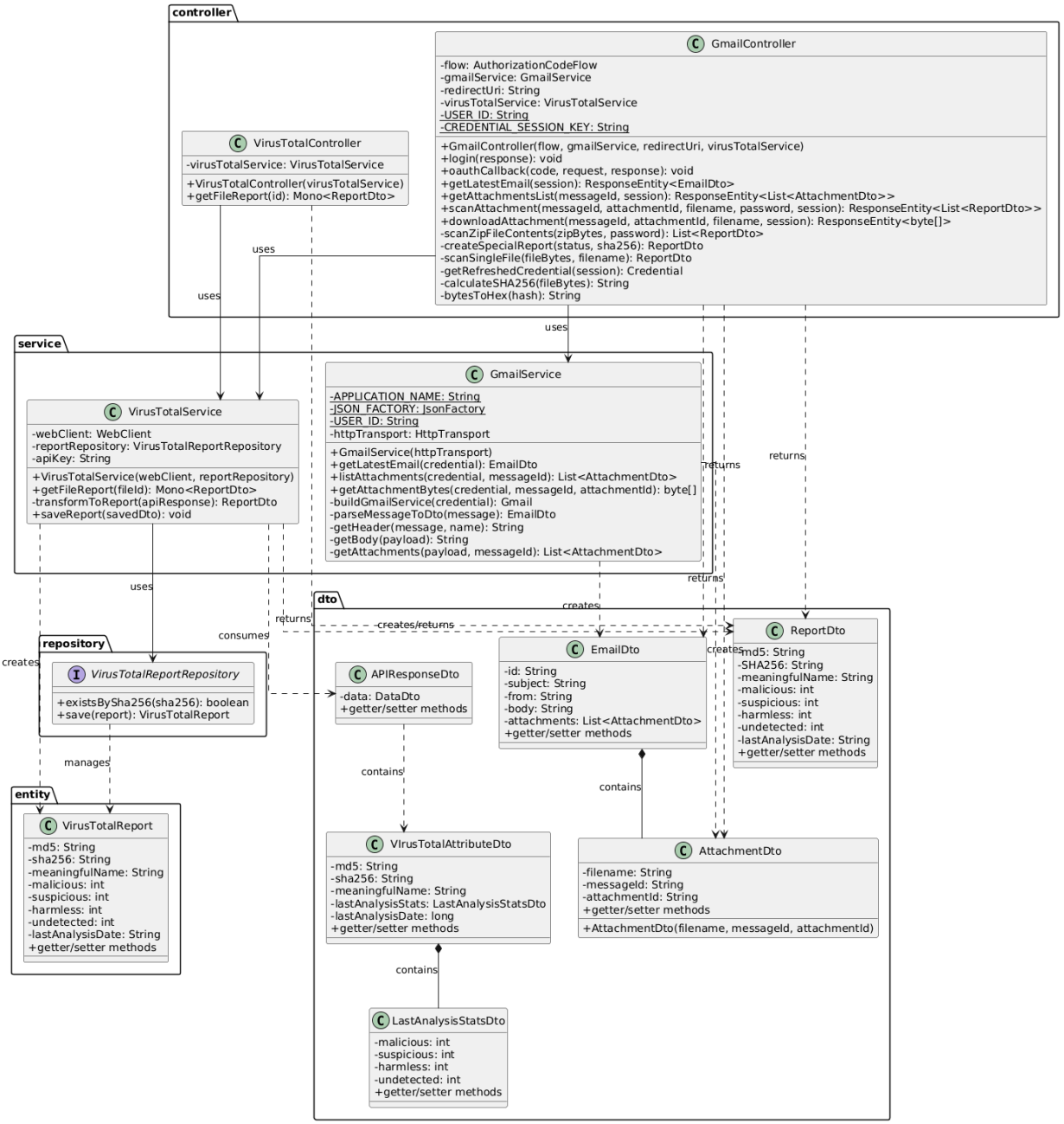
(Phishing detection 관련 Class Diagram)



(url 차단 로직 관련 Class Diagram)



(홈페이지 Class Diagram)



(메일 첨부파일 검사 관련 Class Diagram)

3. 설계 명세

3.1 핵심 알고리즘 설계 (구현 시스템)

1. ALGORITHM Gmail_OAuth_Authorization

INPUT: None

OUTPUT: Credential (Access Token)

```
BEGIN
    // 1. 토큰 저장소 확인
    IF StoredCredential EXISTS in tokens/ THEN
        LOAD credential FROM tokens/StoredCredential
        RETURN credential
    END IF

    // 2. 새로운 인증 필요
    LOAD client_secrets FROM credentials.json

    // 3. Authorization URL 생성
    auth_url = BUILD authorization_url WITH
        - client_id
        - redirect_uri = "http://localhost:8888/Callback"
        - scope = "GMAIL_READONLY"

    // 4. 사용자 브라우저 오픈
    OPEN default_browser WITH auth_url

    // 5. 로컬 서버에서 콜백 대기
    START local_receiver ON port 8888
    WAIT FOR authorization_code FROM Google
   // 6. Authorization Code를 Access Token으로 교환
    credential = EXCHANGE authorization_code FOR access_token

    // 7. 토큰 저장
    SAVE credential TO tokens/StoredCredential
```

```
    RETURN credential
END
```

- 토큰 확인 → 브라우저 오픈 → 코드 받기 → 토큰 교환 → 저장

2.ALGORITHM Naver_IMAP_Fetch_Emails

```
INPUT: maxEmails (integer)
OUTPUT: List<EmailDto> emails

BEGIN
    emails = []

    // 1. IMAP 연결 설정
    properties = CREATE mail_properties WITH
        - protocol = "imaps"
        - host = "imap.naver.com"
        - port = 993
        - ssl.enable = true

    session = CREATE mail_session WITH properties

    // 2. IMAP 서버 연결
    store = CONNECT TO session USING
        - username = NAVER_EMAIL
        - password = NAVER_PASSWORD

    // 3. INBOX 폴더 열기
    inbox = OPEN folder "INBOX" WITH READ_ONLY mode

    // 4. 전체 메시지 개수 확인
    total_messages = GET message_count FROM inbox
    start_index = MAX(1, total_messages - maxEmails + 1)

    // 5. 최신 메일부터 가져오기
```

```
messages = GET messages FROM inbox
    RANGE [start_index TO total_messages]

// 6. 각 메시지 처리
FOR EACH message IN REVERSE_ORDER(messages) DO
    email = NEW EmailDto()

    // 6.1 기본 정보 추출
    email.messageId = GET message_id FROM message
    email.from = GET sender FROM message
    email.subject = DECODE subject FROM message
    email.receivedDate = GET received_date FROM message

    // 6.2 본문 추출
    IF message IS multipart THEN
        email.content = EXTRACT text_content FROM multipart
    ELSE
        email.content = GET text_content FROM message
    END IF

    // 6.3 첨부파일 확인
    attachment_names = []
    IF message HAS attachments THEN
        FOR EACH part IN message.parts DO
            IF part IS attachment THEN
                ADD part.filename TO attachment_names
            END IF
        END FOR
    END IF

    email.hasAttachments = (LENGTH(attachment_names) > 0)
    email.attachmentNames = attachment_names

    // 6.4 피싱 분석 호출
    CALL phishingDetectionService.analyzeEmail(email)
```

```
        // 6.5 리스트에 추가
        ADD email TO emails
    END FOR

    // 7. 연결 종료
    CLOSE inbox
    CLOSE store

    RETURN emails

EXCEPTION HANDLING:
    IF MessagingException OCCURS THEN
        LOG error
        THROW custom exception WITH error message
    END IF
END
```

- IMAP 연결 → INBOX 열기 → 메시지 가져오기 → 파싱 → 분석

3. ALGORITHM: Main_Attachment_Security_Scan

```
MAIN PROCESS: Scan Attachment

INPUT: messageId, attachmentId, filename, password (optional), session
OUTPUT: List of security scan reports

STEP 1: Authenticate and Retrieve File
    - Validate user credential from session
    - Fetch attachment as byte array via Gmail API
    - Store file content in memory

STEP 2: Determine File Type and Route
    IF filename ends with ".zip" THEN
        CALL ZIP_EXTRACTION_PROCESS(fileBytes, password)
```

```
    ELSE
        CALL SINGLE_FILE_PROCESS(fileBytes, filename)
    END IF
```

4. ALGORITHM: ZIP_EXTRACTION_AND_SCAN

```
INPUT: zipBytes (byte array), password (optional string)
OUTPUT: reports (list of scan reports)

STEP 1: Prepare Temporary File
    CREATE temporary file on disk
    WRITE zipBytes to temporary file
    STORE file reference as tempFile

STEP 2: Initialize ZIP Handler
    CREATE ZipFile object from tempFile

STEP 3: Handle Encryption
    IF zipFile.isEncrypted() THEN
        IF password is NULL or EMPTY THEN
            CREATE report with status "Password Required"
            ADD report to reports list
            RETURN reports
        ELSE
            SET zipFile password to password.toCharArray()
        END IF
    END IF

STEP 4: Extract and Scan Internal Files
    GET list of all file headers from zipFile

    FOR EACH fileHeader IN fileHeaders DO
        IF fileHeader is NOT directory THEN
            OPEN input stream for fileHeader
            READ all bytes from stream into entryBytes
```

CLOSE input stream

CALCULATE SHA-256 hash from entryBytes
QUERY VirusTotal with hash
CREATE scan report with results
ADD report to reports list
END IF
END FOR

STEP 5: Handle Errors
IF ZipException occurs (wrong password or corrupted file) THEN
CLEAR reports list
CREATE report with status "Wrong Password / Corrupted Zip"
ADD error report to reports list
END IF

STEP 6: Cleanup
DELETE tempFile from disk
RETURN reports list

5. ALGORITHM: SINGLE_FILE_SCAN

INPUT: fileBytes (byte array), filename (string)
OUTPUT: scanReport (security analysis report)

STEP 1: Calculate File Hash
APPLY SHA-256 digest algorithm to fileBytes
CONVERT hash bytes to hexadecimal string
STORE as sha256Hash

STEP 2: Query Security Database
SEND sha256Hash to VirusTotal API
RECEIVE analysis report

IF report found THEN

```
        EXTRACT malware statistics (malicious, suspicious, harmless)
        FORMAT report with filename
    ELSE
        CREATE "Report Not Found" status
        INCLUDE calculated hash
    END IF

    RETURN formatted report
```

6. ALGORITHM: CALCULATE_SHA256

```
INPUT: fileBytes (byte array)
OUTPUT: hexString (lowercase hexadecimal string)

ALGORITHM:
    INITIALIZE SHA-256 MessageDigest
    PROCESS fileBytes through digest
    GET 32-byte hash result

    FOR EACH byte in hash DO
        CONVERT byte to hex (00-FF format)
        APPEND to result string
    END FOR

    RETURN hexadecimal string
```

6. ALGORITHM: URL BLACKLIST

```
BEGIN
  // 1. 악성 URL 리스트 로딩(필요 시 캐시)
  IF MaliciousUrlCache IS EMPTY OR Stale THEN
    malicious_set ← LOAD from DB (blocked_domain, patterns, wildcards)
    MaliciousUrlCache ← malicious_set
  END IF
```

```
// 2. 요청 URL 정규화
 norm_url ← NORMALIZE(request_url)
 host ← EXTRACT_HOST(norm_url)
 path_q ← EXTRACT_PATH_AND_QUERY(norm_url)

// 3. 화이트리스트(있으면) 우선 통과
 IF host ∈ WhiteList THEN
   RETURN (ALLOW, "whitelist pass", norm_url)
 END IF

// 4. 악성 URL 매칭 검사
 is_bad ← FALSE
 IF host ∈ MaliciousUrlCache.domains THEN
   is_bad ← TRUE
 ELSE IF MATCH_WILDCARD(host, MaliciousUrlCache.wildcards) THEN
   is_bad ← TRUE
 ELSE IF MATCH_RULES(norm_url, MaliciousUrlCache.url_rules) THEN
   is_bad ← TRUE
 END IF

// 5. 결정 및 처리
 IF is_bad = TRUE THEN
   LOG_SECURITY_EVENT(user_id, norm_url, "BLOCKED")
   SHOW_WARNING_PAGE("위험한 URL로 의심되어 차단되었습니다.", details=host)
   RETURN (BLOCK, "blocked by policy", null)
 ELSE
   LOG_SECURITY_EVENT(user_id, norm_url, "ALLOWED")
   REDIRECT_TO(norm_url)
   RETURN (ALLOW, "redirected", norm_url)
 END IF
END
```

3.2 데이터 설계 (구현 시스템)

사용 Database : MySQL

3.2.1. 악성 도메인 여부 검증 위한 table

```
blocked_domain (
    id BIGINT, scheme VARCHAR(10), domain VARCHAR(255),
    source VARCHAR(100), note VARCHAR(500),created_at DATETIME,
    updated_at DATETIME
)
```

3.2.2. 사용자 정보 저장용 table

```
user (
    id INT, username VARCHAR(50), password VARCHAR(255),
    email VARCHAR(100), created_at TIMESTAMP, correct INT, incorrect INT
)
```

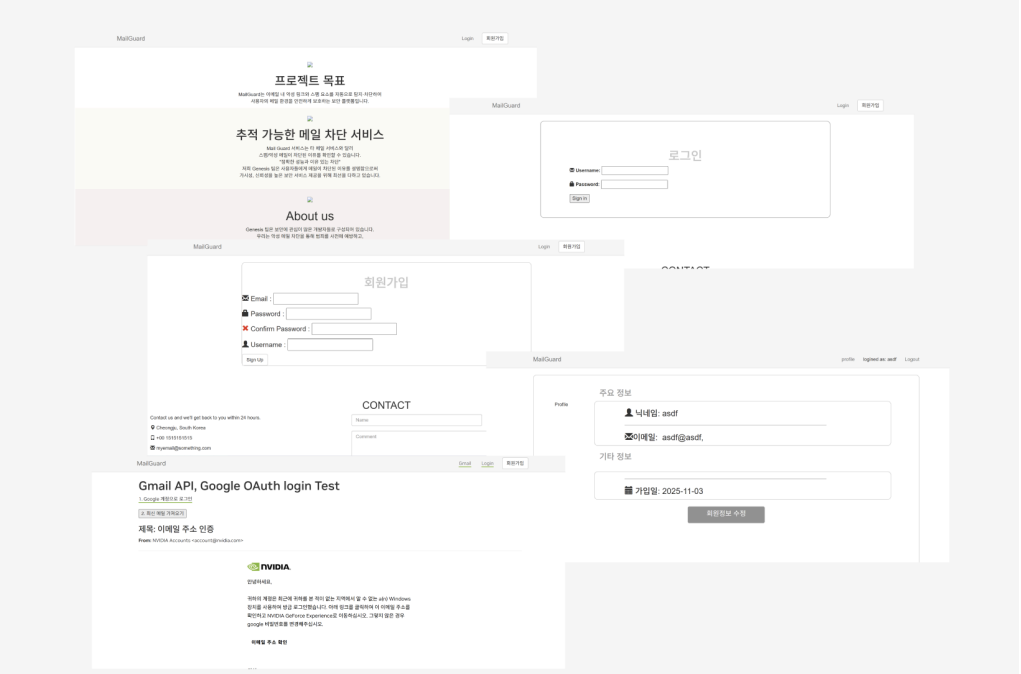
3.2.3 Gmail authorization key

```
oauth_tokens (
    user_id(fk) , access_token VARCHAR(2048),
    refresh_token VARCHAR(1024), id_token VARCHAR(2048),
    token_type VARCHAR(50),
    expires_in INT, created_at DATETIME
)
```

3.2.4 첨부파일 검사 결과 저장용 table

```
virus_total_data(
    save_id INT, md5 VARCHAR(32), sha256 VARCHAR(64) UNIQUE,
    meaningful_name VARCHAR(255), malicous_count INT,
    suspicious_count INT, harmless_count INT,
    undetected_count INT ,last_analysis_date VARCHAR(255),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
)
```

3.3 사용자 인터페이스 설계 (구현 시스템)



3.4 외부 인터페이스 설계

I/F name	Class Name	Signature	Description	R/R
gmail oauth2 api	GmailOAuthService	Credential authorize() throws IOException	Google OAuth 2.0 인증을 통해 Gmail API 접근 권한을 획득하고 Access Token을 반환	Request: GmailOAuthService / Response: Credential
VirusTotal API	VirusTotalService	Mono<ReportDto>getFileReport (String fileId)	파일 hash를 사용해 VirusTotal 외부 API를 호출하고 결과를 파싱해 ReportDto 형식으로 변환후 database에 저장한다.	Request: VirusTotalController/ Response: VirusTotalService
Naver IMAP3	NaverImapService	List<EmailDto> fetchEmails(int maxEmails) throws MessagingException	IMAP 프로토콜을 통해 Naver 메일 서버에 연결하여 INBOX 폴더에서 메일을 가져옴	Request: NaverImapService / Response: List<EmailDto>

4. 기타 구현 참고 사항 - 차기 구현시 반드시 고려할 사항

1. LLM 기반 정밀 분석 고도화

현재 GPT-4o-mini를 활용한 기본적인 피싱 탐지가 구현되어 있으나, 프롬프트 엔지니어링 최적화와 한국어 특화 학습 데이터 추가를 통해 탐지 정확도를 향상시켜야 한다. 또한 LLM 응답 시간이 길어질 경우 비동기 처리 및 캐싱 메커니즘 도입이 필요하다.