

Secure Coding

| [19반] 정민석_7000

물품 거래 어플리케이션을 구현하고, 소프트웨어 생명주기 6단계를 고려하여, 서비스를 설계하고 구현하였습니다. 아래 git 레포지터리에 소스코드를 업로드하였습니다.

<https://github.com/j93es/whs-assignment/tree/main/secure-coding/secure-coding>

1. 요구사항 도출

물품 거래 어플리케이션의 요구사항을 도출하였습니다.

- 사람들이 플랫폼에 가입할 수 있어야 함
- 상품들을 올리고 볼 수 있어야 함.
- 플랫폼 사용자들끼리 소통이 가능해야함.
- 악성 유저나 상품을 차단 해야 함.
- 유저들 간의 송금이 가능해야함
- 상품의 검색할 수 있어야 함.
- 관리자가 플랫폼의 모든 요소를 관리할 수 있어야 함.

2. 시스템 명세

시스템이 어떤 기능을 해야하는지, 보안과 관련하여 어떤 기능이 필요한지를 명세하였습니다.

기능 명세

| 기능 분류 | 기능명 | 상세 설명 |
|-------|----------|-------------------------|
| 회원 | 회원가입 | 사용자 정보 입력 및 등록 |
| | 로그인 | 등록된 계정으로 로그인 |
| | 프로필 관리 | 소개글, 비밀번호 업데이트 |
| | 회원 조회 | 다른 사용자 조회 |
| | 회원 상세 | 다른 사용자의 조회에서 상세페이지를 열람 |
| | 회원 신고 | 불량 사용자 신고 |
| 물품 | 물품 등록 | 가격, 제목, 소개를 포함 |
| | 물품 조회 | 물품 정보, 판매자 정보 열람 |
| | 물품 검색 | 물품을 이름으로 검색 |
| | 물품 수정 | 내가 등록한 물품 수정 |
| | 물품 삭제 | 내가 등록한 물품 삭제 |
| | 물품 신고 | 불량 물품 신고 |
| 채팅 | 전체 채팅 | 모든 사용자 간의 채팅 |
| | 1:1 채팅 | 회원 간의 채팅 |
| 송금 | 지갑 조회 | 잔액, 내역 확인 |
| | 송금 | 회원간의 송금 |
| 관리자 | 회원관리 | 사용자 목록 조회 및 휴먼등록/복구 처리 |
| | 물품 관리 | 물품 등록 내역 조회 및 삭제 |
| | 물품 신고 관리 | 물품 신고 내역 확인 및 삭제 |
| | 회원 신고 관리 | 회원신고 내역 확인 및 휴먼등록/복구 처리 |
| | 채팅 | 전체 채팅 중 특정 채팅 삭제 |

보안 기능 명세

| 보안 기능 분류 | 보안 기능명 | 상세 설명 |
|----------|-----------|---|
| 회원 | 서버측 입력 검증 | 사용자명(username)와 비밀번호(password)에 대해 길이 허용 문자 집합, 형식 등 서버측 검증 수행. |
| | 비밀번호 보안 | 비밀번호를 평문으로 저장하지 않고 bcrypt, Argon2 등 강력한 해시 알고리즘과 고유 salt를 적용하여 암호화 저장하는지 확인 |

| | | |
|--------|----------------|---|
| | 세션 쿠키 설정 | 세션 쿠키에 HttpOnly 및 HTTPS 환경에서 Secure 플래그가 적용되어 있는지 확인 |
| | 세션 만료 및 재인증 | 일정 시간 이후 세션 만료 및 민감 작업 시 재인증 로직이 구현되어 있는지 확인 |
| | 실패 로그인 방어 | 로그인 실패 횟수에 따른 계정 잠금 혹은 지연(time-out) 메커니즘 적용 여부 확인 |
| 물품 | 폼 입력 검증 | 상품 제목, 설명, 가격 등의 입력 필드에 대해 서버측 검증 및 필수 항목 체크 여부 확인. 가격은 숫자 형식 및 범위 검증 적용 |
| | 소유자 확인 | 상품 수정 및 삭제 시, 요청한 사용자가 해당 상품의 소유자인지 검증하는 로직이 구현되어 있는지 확인 |
| 채팅 | 메시지 내용 검증 | 채팅 메시지에 대해 길이 제한 |
| | 연결 암호화 | 운영 환경에서 WSS(SSL/TLS 암호화된 웹소켓)를 사용하여 데이터 전송의 기밀성이 보장되는지 확인 |
| 송금 | 폼 입력 검증 | 적절한 숫자인지 검증 |
| 신고 | 폼 입력 검증 | 신고 대상(target_id) 및 신고 사유(reason)에 대해 서버측 입력 검증, 길이 제한 |
| | 신고 관리 | 신고 활동이 감사 로그로 기록되는지 확인 |
| | 신고 남용 방지 | 동일 사용자의 반복 신고 제한, 신고 건수 제한 및 관리자 검토 프로세스 등 신고 기능 남용 방지 로직이 구현되어 있는지 확인 |
| 전체 시스템 | ORM 및 파라미터 바인딩 | SQLAlchemy ORM 및 파라미터 바인딩을 통해 SQL 인젝션 공격에 대한 방어가 제대로 이루어지고 있는지 확인 |
| | 인증된 사용자만 접근 | 로그인 외의 서비스에 사용자가 접근 시, 인증이 되어있는지 확인 |
| | CSRF 보호 | 모든 폼에 대해 CSRF 토큰 사용 여부를 확인하여 요청 위조 공격 방지 |
| | 데이터베이스 권한 | 데이터베이스 사용자 권한이 최소 권한 원칙에 따라 설정되어 민감 데이터 접근이 제한되어 있는지 확인 |
| | 데이터 무결성 | 데이터베이스에 저장되기 전 모든 필수 항목 및 형식이 올바른지 검증하는 로직이 있는지 확인 |
| | 에러 및 예외 처리 | 예외 처리 시 스택 트레이스, DB 정보 등 민감 정보가 사용자에게 노출되지 않고, 에러 로그에 민감 정보 기록 방지 로직이 구현되어 있는지 확인 |
| | XSS 방어 | XSS 공격 방지를 위해 HTML 태그 및 스크립트 코드 이스케이프 또는 입력값 필터링 및 인코딩 적용 여부 확인 |

| | | |
|--|----------------|--|
| | Rate Limiting | 동일 사용자가 단기간에 과도한 요청을 보내지 않도록 제한하는 기능(스팸 방지)이 구현되어 있는지 확인 |
| | 보안 헤더 설정 | Content-Security-Policy, X-Frame-Options, X-Content-Type-Options 등의 보안 헤더가 적용되어 있는지 확인 |
| | HTTPS 적용 | 운영 환경에서 HTTPS를 사용하여 데이터 전송 시 기밀성 및 무결성이 보장되는지 확인 |
| | 라이브러리 및 의존성 관리 | 사용 중인 Flask, SQLAlchemy, Flask-SocketIO 등 라이브러리의 최신 보안 패치 및 업데이트가 적용되어 있는지 정기적으로 점검 |

3. 아키텍처 설계

시스템을 구현할 때에 어떤 기능을 어떤 계층에서 수행해야 하는지, DB에는 어떤 정보가 저장되어야 하는지를 설계하였습니다.

- Model
 - 사용자: 고유번호, 아이디, 비밀번호, 소개, 휴먼 여부, 지갑 잔액, 로그인 실패 횟수, 로그인 금지 시간
 - 상품: 고유번호, 이름, 설명, 가격, 판매자 고유번호
 - 신고: 고유번호, 신고자 고유번호, 대상 고유번호, 이유, 신고 시간
 - 채팅 메시지: 고유번호, 보낸이 고유번호, 받는이 고유번호, 메시지, 보낸 시간
 - 송금 내역: 고유번호, 보낸이 고유번호, 받는이 고유번호, 송금액, 송금/입금, 송금 시간
- Repository
 - CRUD 추상화
- Service
 - 비즈니스 로직 수행
 - 입력값 검증
 - User, Admin service를 분리하여, 권한 관리
- Router

- 사용자 페이지 렌더링 담당(html 서빙)
- API 엔드포인트 정의
- 인증
- 인증되지 않은 사용자의 라우터 접근 차단

4. 주요 구현 사항

- 기본 기능 구현 완료
- 입력 검증 및 XSS 방어
 - Service 계층에서 아래 코드를 활용하여 입력을 필터링하였습니다.

```
def sanitize_input(text):
    """
    XSS 방지를 위해 HTML 태그 및 스크립트 코드를 이스케이프합니다.
    """
    text = safe_str(text) # 입력값을 문자열로 변환

    if not text:
        return ""

    # 1. 모든 태그 제거 (기본 필터링이 필요할 경우)
    tag_stripped = re.sub(r'<.*?>', '', text)

    # 2. HTML 엔티티 이스케이프 (가장 안전한 처리)
    escaped = html.escape(tag_stripped)

    return escaped
```

- 이외의 비밀번호 길이, 채팅 메시지 길이, 신고 제한 등 추가적으로 필터링이나 비즈니스 로직에 따라 구현이 필요한 부분을 고려하였습니다.
- 비밀번호 보안
 - bcrypt의 단방향 암호화를 통하여 유저의 비밀번호를 암호화하여 저장하였습니다.

```
# ...
# bcrypt를 사용하여 고유 salt와 함께 비밀번호 해시 생성
hashed_password = bcrypt.hashpw(password.encode('utf-8'), b
crypt.gensalt())
# ...
```

- 인증되지 않은 사용자 제한
 - HTTP(S) 라우터와 WS(S) 라우터에 flask의 데코레이터를 통하여, 인증되지 않은 사용자(휴먼 계정 포함)의 접근을 제한하였습니다.
- ORM 및 파라미터 바인딩
 - SQLAlchemy ORM 및 파라미터 바인딩을 통해 SQL 인젝션 공격에 대한 방어를 구현하였습니다.

```
def update_failed_attempts(user_id, count):
    session = SessionLocal()
    try:
        session.query(User).filter(User.id == user_id).update({"failed_a
ttempts": count})
        session.commit()
    finally:
        session.close()
```

- CSRF 보호

```
@app.context_processor
def inject_csrf_token():
    return dict(csrf_token=generate_csrf())
```

위의 코드로 CSRF 토큰을 생성하고, 아래 코드처럼 form 태그의 input 태그에 csrf 토큰을 삽입하여 위변조를 방지하였습니다.

```
<form>
  <input type="hidden" name="csrf_token" value="{{ csrf_token }}" />
  <button type="submit">삭제</button>
</form>
```

- 에러 및 예외 처리
 - 커스텀 에러 페이지를 구성하고, python app.py의 개발자용 실행이 아닌 배포 명령어로 실행하여, 에러페이지에 콜스택 등 민감정보가 표시되지 않도록 구성하였습니다.
- Rate Limiting
 - flask_limiter로 HTTP(S) 라우터에 Rate Limiting을 구현하였고, 아래 코드를 통하여 socket 통신에서의 Rate Limiting을 구현하였습니다.

```

socketio_rate_limits = {}

def socketio_rate_limit(key_func, limit=20, window=60):
    def decorator(f):
        def wrapper(*args, **kwargs):
            key = key_func()
            now = time.time()
            log = socketio_rate_limits.get(key, [])
            log = [ts for ts in log if ts > now - window] # 윈도우 내 요청만
            유지
            if len(log) >= limit:
                emit("error", {"message": "Too many messages, please slow down."})
                return
            log.append(now)
            socketio_rate_limits[key] = log
            return f(*args, **kwargs)
        return wrapper
    return decorator

```

- 보안 헤더 설정
 - 아래 코드로 보안 관련 헤더를 설정하였습니다.

```

from flask import current_app as app

def register_headers(app):
    @app.after_request
    def set_headers(response):
        if app.config["ENV"] == "production":

```

```

        pass
    else:
        return response

# 👉 캐시 관련
response.headers["Cache-Control"] = "no-cache"

# 👉 Content Security Policy
csp = [
    "default-src 'none'",
    "base-uri 'self'",
    "connect-src 'self'",
    "font-src 'self'",
    "form-action 'self'",
    "frame-ancestors 'none'",
    "img-src 'self'",
    "script-src 'self' cdnjs.cloudflare.com",
    "style-src 'self'",
    "manifest-src 'self'",
    "object-src 'self'",
    "upgrade-insecure-requests"
]
response.headers["Content-Security-Policy"] = "; ".join(csp)

# 👉 보안 관련 헤더들
response.headers["X-Content-Type-Options"] = "nosniff"
response.headers["X-Frame-Options"] = "DENY"
response.headers["Referrer-Policy"] = "strict-origin-when-cross-origin"
response.headers["Permissions-Policy"] = (
    "accelerometer=(),autoplay=(),camera=(),fullscreen=(self),"
    "geolocation=(),gyroscope=(),midi=(),microphone=(),magnetometer=(),"
    "payment=(),xr-spatial-tracking=()"
)
response.headers["X-XSS-Protection"] = "1; mode=block"
response.headers["Cross-Origin-Resource-Policy"] = "same-origin"

```



```

        response.headers["Cross-Origin-Opener-Policy"] = "same-origin"
        response.headers["Cross-Origin-Embedder-Policy"] = "require-corp"
        response.headers["X-Permitted-Cross-Domain-Policies"] = "none"
        response.headers["Strict-Transport-Security"] = "max-age=31536000; includeSubDomains"

    return response

```

- template에서 inline script, inline style로 구현한 부분이 **CSP에 위배**되었습니다. 따라서 static 폴더에 해당 스크립트를 따로 분리하고, **template에서 link, script 태그로 해당 파일을 로드**하도록 구현하였습니다.
- 세션 쿠키 설정
 - 아래 코드를 통하여 운영환경에서 secure_flag를 설정하였습니다.

```

secure_flag = request.is_secure or current_app.config.get('ENV') = 'production'
response.set_cookie('jwt', token, httponly=True, secure=secure_flag)

```

- HTTPS/WSS 적용
 - certbot 혹은 인증서 기관의 인증서를 발급받아, Nginx 등에서 해당 인증서를 반영하여, HTTPS로 동작하게 할 수 있습니다.
 - WSS의 경우 HTTPS로 동작하면 WSS로 연결되도록 구현하였습니다.

5. 테스트

기능 테스트와 보안 고려사항을 체크하는 방향으로 테스트를 진행하였습니다.

기능 테스트

| 기능 분류 | 기능명 | 상세 설명 | 체크 |
|-------|------|----------------|----|
| 회원 | 회원가입 | 사용자 정보 입력 및 등록 | O |
| | 로그인 | 등록된 계정으로 로그인 | O |

| | | | |
|-----|----------|-------------------------|---|
| | 프로필 관리 | 소개글, 비밀번호 업데이트 | O |
| | 회원 조회 | 다른 사용자 조회 | O |
| | 회원 상세 | 다른 사용자의 조회에서 상세페이지를 열람 | O |
| | 회원 신고 | 불량 사용자 신고 | O |
| 물품 | 물품 등록 | 가격, 제목, 소개를 포함 | O |
| | 물품 조회 | 물품 정보, 판매자 정보 열람 | O |
| | 물품 검색 | 물품을 이름으로 검색 | O |
| | 물품 수정 | 내가 등록한 물품 수정 | O |
| | 물품 삭제 | 내가 등록한 물품 삭제 | O |
| | 물품 신고 | 불량 물품 신고 | O |
| 채팅 | 전체 채팅 | 모든 사용자 간의 채팅 | O |
| | 1:1 채팅 | 회원 간의 채팅 | O |
| 송금 | 지갑 조회 | 잔액, 내역 확인 | O |
| | 송금 | 회원간의 송금 | O |
| 관리자 | 회원관리 | 사용자 목록 조회 및 휴먼등록/복구 처리 | O |
| | 물품 관리 | 물품 등록 내역 조회 및 삭제 | O |
| | 물품 신고 관리 | 물품 신고 내역 확인 및 삭제 | O |
| | 회원 신고 관리 | 회원신고 내역 확인 및 휴먼등록/복구 처리 | O |
| | 채팅 | 전체 채팅 중 특정 채팅 삭제 | O |

보안 기능 테스트

| 보안 기능 분류 | 보안 기능명 | 상세 설명 | 체크 |
|----------|-----------|---|----|
| 회원 | 서버측 입력 검증 | 사용자명(username)와 비밀번호(password)에 대해 길이 허용 문자 집합, 형식 등 서버측 검증 수행. | O |
| | 비밀번호 보안 | 비밀번호를 평문으로 저장하지 않고 bcrypt, Argon2 등 강력한 해시 알고리즘과 고유 salt를 적용하여 암호화 저장하는지 확인 | O |

| | | | |
|--------|----------------|--|-----------------------------------|
| | 세션 쿠키 설정 | 세션 쿠키에 HttpOnly 및 HTTPS 환경에서 Secure 플래그가 적용되어 있는지 확인 | O |
| | 세션 만료 및 재인증 | 일정 시간 이후 세션 만료 및 민감 작업 시 재인증 로직이 구현되어 있는지 확인 | O |
| | 실패 로그인 방어 | 로그인 실패 횟수에 따른 계정 잠금 혹은 지연(time-out) 메커니즘 적용 여부 확인 | O |
| 물품 | 폼 입력 검증 | 상품 제목, 설명, 가격 등의 입력 필드에 대해 서버측 검증 및 필수 항목 체크 여부 확인. 가격은 숫자 형식 및 범위 검증 적용 | O |
| | 소유자 확인 | 상품 수정 및 삭제 시, 요청한 사용자가 해당 상품의 소유자인지 검증하는 로직이 구현되어 있는지 확인 | O |
| 채팅 | 메시지 내용 검증 | 채팅 메시지에 대해 길이 제한 | O |
| | 연결 암호화 | 운영 환경에서 WSS(SSL/TLS 암호화된 웹소켓)를 사용하여 데이터 전송의 기밀성이 보장되는지 확인 | 운영 환경에서 HTTPS를 사용하면 WSS를 사용하도록 구현 |
| 송금 | 폼 입력 검증 | 적절한 숫자인지 검증 | O |
| 신고 | 폼 입력 검증 | 신고 대상(target_id) 및 신고 사유(reason)에 대해 서버측 입력 검증, 길이 제한 | O |
| | 신고 관리 | 신고 활동이 감사 로그로 기록되는지 확인 | O |
| | 신고 남용 방지 | 동일 사용자의 반복 신고 제한, 신고 건수 제한 및 관리자 검토 프로세스 등 신고 기능 남용 방지 로직이 구현되어 있는지 확인 | O |
| 전체 시스템 | ORM 및 파라미터 바인딩 | SQLAlchemy ORM 및 파라미터 바인딩을 통해 SQL 인젝션 공격에 대한 방어가 제대로 이루어지고 있는지 확인 | O |
| | 인증된 사용자만 접근 | 로그인 외의 서비스에 사용자가 접근 시, 인증이 되어있는지 확인 | O |

| | | | |
|--|----------------|--|---|
| | | 인 | |
| | CSRF 보호 | 모든 폼에 대해 CSRF 토큰 사용 여부를 확인하여 요청 위조 공격 방지 | O |
| | 데이터베이스 권한 | 데이터베이스 사용자 권한이 최소 권한 원칙에 따라 설정되어 민감 데이터 접근이 제한되어 있는지 확인 | O |
| | 데이터 무결성 | 데이터베이스에 저장되기 전 모든 필수 항목 및 형식이 올바른지 검증하는 로직이 있는지 확인 | O |
| | 에러 및 예외 처리 | 예외 처리 시 스택 트레이스, DB 정보 등 민감 정보가 사용자에게 노출되지 않고, 에러 로그에 민감 정보 기록 방지 로직이 구현되어 있는지 확인 | O |
| | XSS 방어 | XSS 공격 방지를 위해 HTML 태그 및 스크립트 코드 이스케이프 또는 입력값 필터링 및 인코딩 적용 여부 확인 | O |
| | Rate Limiting | 동일 사용자가 단기간에 과도한 요청을 보내지 않도록 제한하는 기능(스팸 방지)이 구현되어 있는지 확인 | O |
| | 보안 헤더 설정 | Content-Security-Policy, X-Frame-Options, X-Content-Type-Options 등의 보안 헤더가 적용되어 있는지 확인 | O |
| | HTTPS 적용 | 운영 환경에서 HTTPS를 사용하여 데이터 전송 시 기밀성 및 무결성이 보장되는지 확인 | 운영 환경에서 certbot 혹은 다른 인증된 기관에서 key를 발급받아 HTTPS 적용 |
| | 라이브러리 및 의존성 관리 | 사용 중인 Flask, SQLAlchemy, Flask-SocketIO 등 라이브러리의 최신 보안 패치 및 업데이트가 적용되어 있는지 정기적으로 점검 | O |

6. 유지보수(TODO)

- pip-audit 을 통한 보안 업데이트 확인
- SSL 인증서를 주기적으로 업데이트
- 사용자 UI 개선
- 디렉토리 구조 개선
 - template 코드 admin, client 분리
 - Repository, Service, Router에서 유저, 물품, 송금, 신고 파일 분리
 - Model 파일 분리
- 의존성 역전/주입을 통하여 repository, service layer 추상화
- repository, service layer에서의 return 값 통일화(router에서 해당 리턴값으로 수행할 작업을 유지보수 용이하게 하기 위함)
- 빌드 과정의 단순화(Docker 등을 활용)