DECOBET v2
[Easy To Translate Coding] Web Dev

What is this Project?

- DECOBET 프로젝트에서 보완이 필요한 점을 개선한 리빌딩 프로젝트입니다.
- JWT 인증 시스템을 고도화로 보안을 강화, 안정적인 서비스 운영을 목표로 합니다.
- HttpOnly, Secure, Samsite 등의 보안 설정을 적용하여 각종 보안을 강화합니다.

What are the main features?

- JWT RTR (Refresh Token Rotation)전략을 활용한 인증 시스템 고도화
- IP 기반 삐회원 사용횟수 제한 기능으로 트래픽 관리
- 1:1문의 시스템의 계층형 댓글 구조 적용으로 서비스 관리 품질 향상
- Junit & Mockito를 활용한 TDD(테스트 주도 개발) 적용으로 코드 안정성 확보
- React-Bootstrap을 활용한 UI 개선 및 사용자 경험 향상

프로젝트 소개 (DECOBET v2)

[DECOBET v2]

보안 및 성능을 대폭 개선한 코드 번역 웹사이트 (백엔드 & 프론트엔드 개발)

개인 프로젝트

프로젝트 기간: 2025. 02.07 – 2025. 02. 28(진행 중)

GitHub - https://github.com/kwanghunk/TEC-WEB-Project

[Tech Stack]

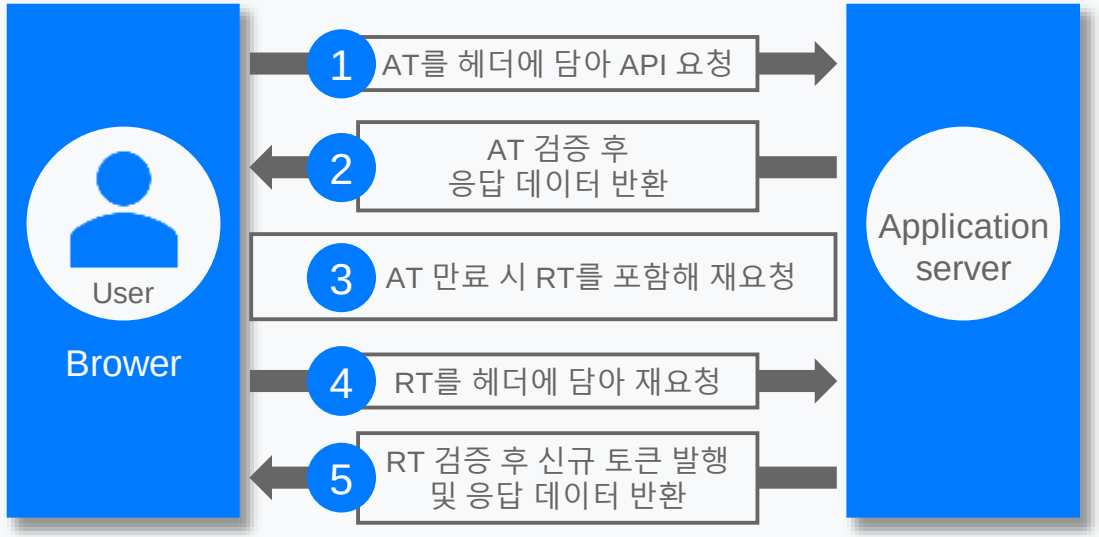
[핵심 구현기능]

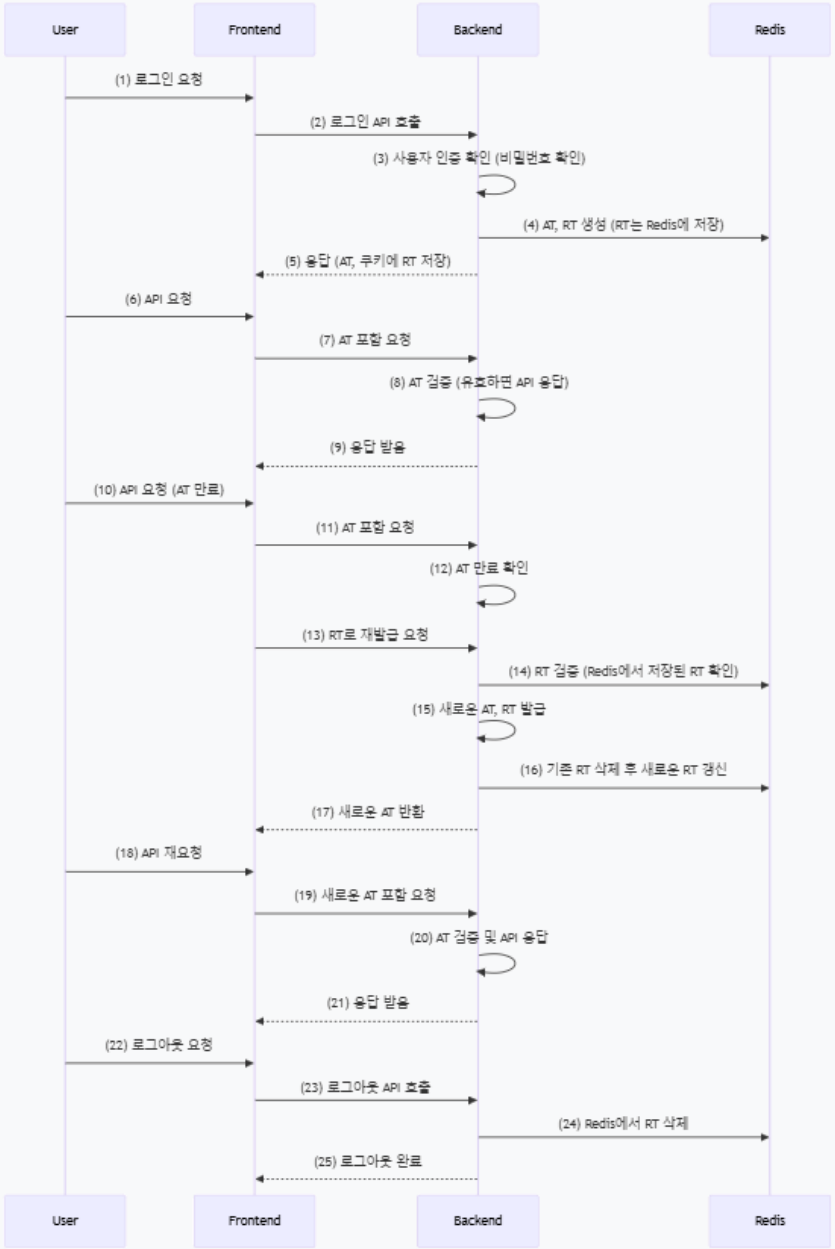
보안 강화(JWT RTR 적용)	Refresh Token Rotation을 적용하여 보안성 향상
비회원 사용 제한 기능	IP 기반 요청 제한을 추가하여 트래픽 관리
1:1 문의 시스템 개선	계층형 댓글 구조 도입으로 서비스 관리 품질 향상
TDD(테스트 주도 개발) 적용	Junit & Mockito를 활용하여 안정적인 API 제공

3-4. 프로젝트 – 시스템 흐름도

시스템 흐름도

- 사용자 로그인 및 API 요청 처리 과정을 시각화 한 것입니다.
- 로그인 시 AT와 RT를 함께 발급하며, RT는 Redis에 저장 보안을 강화합니다.
- API 요청 시 AT를 헤더에 포함하여 요청하며, AT가 만료되었을 경우 저장된 RT를 이용하여 신규 AT, RT를 자동 발급합니다.
- RTR적용으로 RT가 재사용될 경우 기존 RT는 폐기하고 사용자의 모든 세션을 강제 종료하여 탈취 방지를 강화합니다.





3-4. 프로젝트 – 담당역할

담당역할 – JWTFilter 동작

1. 주요 기능
- API 요청 시 JWTFilter를 통해 인증 및 권한 검사를 수행
- AccessToken(AT) 검증 및 갱신(RTR 전략 적용)

2. 문제 정의
- 기존 JWT 인증 방식에서는 AT 만료 시 로그인 필요
- 토큰이 만료될 경우 자동 갱신 기능 부재

3. 핵심 코드
- JWTFilter를 통해 AT 검증 및 만료된 경우 RT를 활용하여 자동 갱신 처리
- Redis를 활용하여 RT 저장 및 관리

4. 결과
- RT를 활용한 자동갱신으로 사용자 경험 개선
- 인증 강화 및 API 요청 시 불필요한 로그인 감소

```
String authorization= request.getHeader("Authorization");
if (authorization == null || !authorization.startsWith("Bearer ")) {
    response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
    response.getWriter().write("Unauthorized - No Access Token");
    return;
}

String token = authorization.replace("Bearer ", "");

if (jwtUtil.parseTokenExpirationCheck(token)) {
    String refreshToken = jwtUtil.getRefreshTokenFromCookie(request);
    if (refreshToken == null || !jwtUtil.isRefreshTokenValid(refreshToken)) {
        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
        response.getWriter().write("Unauthorized - Invalid Refresh Token");
        return;
    }
```

```
Claims refreshClaims = jwtUtil.parseToken(refreshToken);
String username = refreshClaims.get("username", String.class);
String userType = refreshClaims.get("userType", String.class);
String tokenFamily = refreshClaims.get("tokenFamily", String.class);

jwtUtil.revokeRefreshToken(refreshToken);

String newAccessToken = jwtUtil.createAccessToken(username, userType, 1000L * 60 * 15);
String newRefreshToken = jwtUtil.createRefreshToken(username, userType, tokenFamily);

Cookie newRefreshCookie = new Cookie("refreshToken", newRefreshToken);
newRefreshCookie.setHttpOnly(true);
newRefreshCookie.setSecure(true);
newRefreshCookie.setAttribute("SameSite", "Strict");
newRefreshCookie.setPath("/");
response.addCookie(newRefreshCookie);

response.setHeader("Authorization", "Bearer " + newAccessToken);

token = newAccessToken.replace("Bearer ", "");
```

담당역할 – 번역 기록 관리

```
SaveHistory entity = SaveHistory.builder()
        .username(username)
        .requestCode(saveDTO.getRequestCode())
        .responseCode(saveDTO.getResponseCode())
        .typeCode(saveDTO.getTypeCode())
        .historyTitle(saveDTO.getHistoryTitle())
        .saveTime(LocalDateTime.now())
        .build();
saveHistoryRepository.save(entity);
```

1. 주요 기능
- 사용자의 번역 기록을 관리하며 이전 번역 기록의 조회, 저장, 다운로드 기능 제공
- 사용자가 요청한 기록만 데이터베이스(DB)에 저장하여 불필요한 데이터 증가 방지

2. 문제 정의
- 사용자의 모든 번역 기록을 DB에 저장할 경우 저장공간 낭비 및 조회 성능 저하 발생 우려

3. 핵심 코드
- 기본적으로 클라이언트 측에서 세션을 활용하여 최근 10개 기록만 유지
- 사용자가 명시적으로 요청한 기록만 DB에 저장

4. 결과
- 세션 기록과 DB 기록을 분리하여 저장공간 최적화

```
public String generateFileContentFromDB(int saveHistoryNo, String username) {
    Optional<SaveHistory> optionalHistory =
            saveHistoryRepository.findBySaveHistoryNoAndUsername(saveHistoryNo, username);
    if (optionalHistory.isPresent()) {
        SaveHistory history = optionalHistory.get();

        StringBuilder fileContent = new StringBuilder();
        fileContent.append(LocalDateTime.now()).append("\n")
            .append("Saved On: ").append(history.getSaveTime()).append("\n\n")
            .append("Language: ").append(history.getTypeCode()).append("\n\n")
            .append("OriginCode:\n").append(history.getRequestCode()).append("\n\n")
            .append("TranslationCode:\n").append(history.getResponseCode()).append("\n\n");
        return fileContent.toString();
    } else {
        throw new IllegalArgumentException("데이터를 찾을 수 없습니다.");
    }
}
```

3-4. 프로젝트 – 담당역할

담당역할 – 비회원 기능 제한(IP)

1. 주요 기능
- 비회원 사용자의 API 요청 횟수 제한
- IP 기반 요청 카운트 관리 및 일정 주기로 초기화

2. 문제 정의
- 무분별한 API 요청으로 서버 부하 발생 가능
- 비회원과 회원 간의 서비스 사용 차별성 미비

3. 핵심 코드
- IpService를 활용하여 IP 기반 요청 횟수 관리
- IpResetService를 활용하여 매일 자정마다 요청 횟수 초기화

4. 결과
- 비회원의 과도한 API요청을 방지하여 서비스 안정성 향상

```java
// 비회원 요청 가능여부 확인 및 요청 카운트 증가
@Transactional
public boolean isRequestAllowed(HttpServletRequest request) {
    String ipAddress =  ipUtil.getClientIp(request);

    Optional<Ip> ipRecord = ipRepository.findByIpAddress(ipAddress);
    Ip ipData = ipRecord.orElseGet(() -> createNewIp(ipAddress));

    if (ipData.getRequestCount() >= GUEST_LIMIT) return false; // 요청 제한 초과

    // 요청이 가능하면 카운트 증가 후 DB 저장
    ipData.setRequestCount(ipData.getRequestCount() + 1);
    ipData.setLastRequest(LocalDateTime.now());
    ipRepository.save(ipData);

    return true;
}
```

```java
public String getClientIp(HttpServletRequest request) {
    String[] headers = {
            "X-Forwarded-For", // 프록시 서버 뒤의 실제 클라이언트 IP
            "Proxy-Client-IP", // Proxy-Client-IP 헤더 확인
            "WL-Proxy-Client-IP", // WebLogic Proxy-Client-IP
            "HTTP_X_FORWARDED_FOR", // 일부 프록시 환경
            "HTTP_CLIENT_IP" // 일부 프록시 호나경
    };
    String ipAddress = null;
    for (String header : headers) {
        String ipList = request.getHeader(header);
        if (ipList != null && !ipList.isEmpty() && !"unknown".equalsIgnoreCase(ipList)) {
            ipAddress = ipList.split(",")[0].trim();
            break;
        }
    }
    if (ipAddress == null || ipAddress.isEmpty() || "unknown".equalsIgnoreCase(ipAddress)) {
        ipAddress = request.getRemoteAddr();
    }
    return ipAddress;
}
```

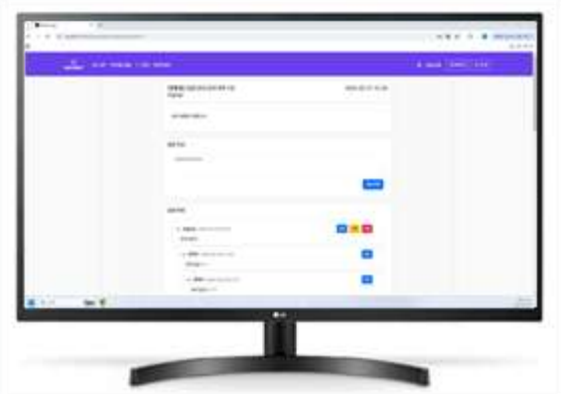
담당역할 – 1:1 문의

1. 주요 기능
- 사용자와 관리자가 1:1 문의로 직접 소통 가능
- 계층형 댓글 구조로 서비스 품질 개선 향상

2. 문제 정의
- 단순 1:1 문의 방식에서는 추가 답변이 어려움
- 만족할 만한 문의 답변을 받기 위해 반복적으로 새로운 문의를 작성해야 함

3. 핵심 코드
- SupportService에서 문의 데이터를 계층 구조로 저장하여 다중 답변 지원
- 부모-자식 관계를 이용하여 답변이 이어질 수 있도록 설계

4. 결과
- 다중 답변을 지원하여 효율적인 소통 가능



```
UserSupport parentInquiry = parentInquiryOpt.get();
UserSupport reply = UserSupport.builder()
        .username(username)
        .title("RE: " + parentInquiry.getTitle())
        .content(replyContent)
        .status(InquiryStatus.IN_PROGRESS)
        .category(parentInquiry.getCategory())
        .isDeleted("N")
        .createdDate(LocalDateTime.now())
        .parentInquiry(parentInquiry) // 부모 문의 연결
        .build();

supportRepository.save(reply);
```

```
// 재귀적으로 모든 답변 조회
private List<UserSupportDetailDTO> getRepliesRecursively(UserSupport parent) {
    List<UserSupport> replies = supportRepository.findAllRepliesByParentInquiryNo(parent.getInquiryNo());

    return replies.stream()
            .filter(reply -> "N".equals(reply.getIsDeleted())) // 삭제되지 않은 데이터만 필터링
            .map(reply -> {
                UserSupportDetailDTO dto = UserSupportDetailDTO.fromEntity(reply);
                dto.setReplies(getRepliesRecursively(reply)); // 재귀 호출하여 하위 답변까지 가져오기
                return dto;
            })
            .collect(Collectors.toList());
}
```