tspoonProject

강경민

목차

01

프로젝트 개요

목표와 목적

1. **목표**와 목적
2. 주요 기능
3. 프로젝트의 차별점

1. 목표
   a. **주요 기능 개발:** 기본적인 HTML과 JS 파일을 바탕으로, 주로 폼 데이터 맵핑과 AJAX를 통해 사용자와의 상호작용을 원활하게 만드는 것.
   b. **JSP와 서블릿 활용:** JSP와 서블릿을 활용하여 서버 측에서 데이터 처리와 동적 페이지 생성을 효율적으로 수행하는 것.
   c. **백엔드 중심의 설계:** 데이터베이스 설계와 백엔드 로직 구현에 중점을 두어, 효율적이고 안정적인 데이터 처리 및 저장 방안을 마련하는 것.

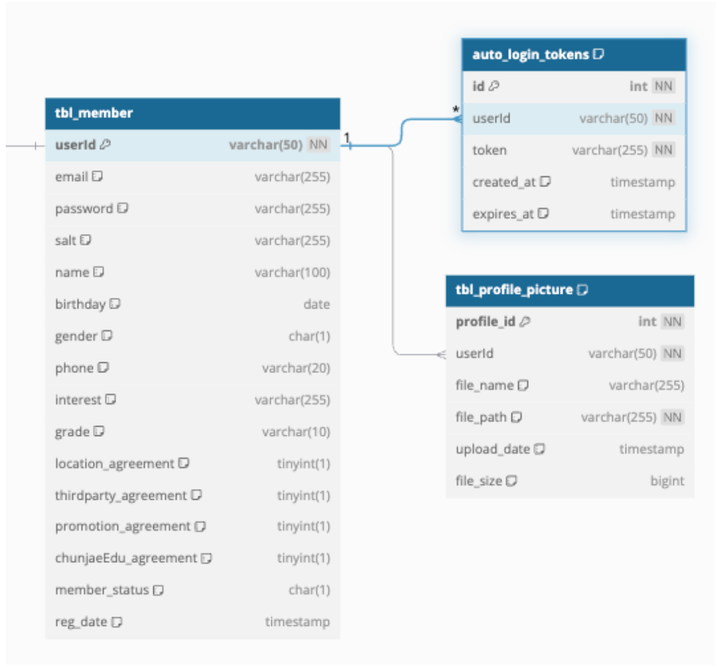
주요기능

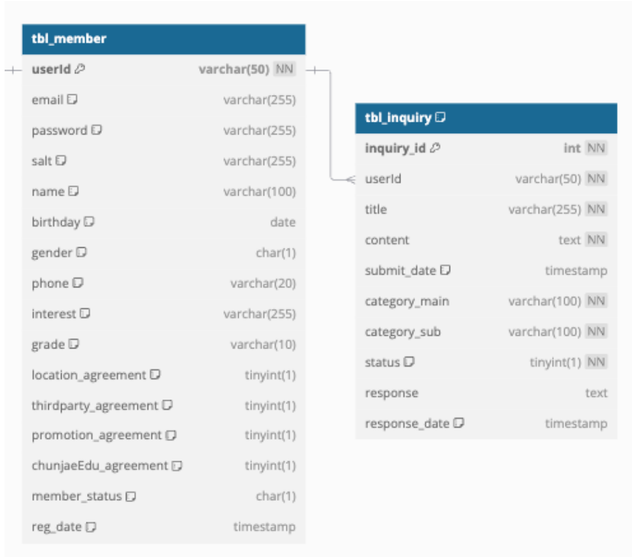
1. 목표와 목적
2. 주요 기능
3. 프로젝트의 차별점

1. 회원 관리 시스템
   a. 회원가입
   b. 사용자 정보 유효성 검증 (SHA-256 + Salt)
   c. 기본 프로필 사진 자동 생성
   d. 로그인/ 로그아웃
   e. 세션 기반 인증
   f. 자동 로그인 (쿠키 + 토큰)
   g. 비밀번호 검증
   h. 로그인 상태 유지 (30일)

ERD

Entity-Relationship Diagram





ERD

Entity-Relationship Diagram

주요기능

1. 목표와 목적
2. 주요 기능
3. 프로젝트의 차별점

2. 메시지 시스템
   a. 메시지 관리
   b. 메시지 작성 (제목, 내용, 첨부파일)
   c. 수신/발신함 분리
   d. 페이지네이션 구현 (AJAX)
   e. 읽음/안읽음 상태 관리
   f. 메시지 삭제 기능(논리삭제 , 물리삭제)
   g. 첨부파일 및 파일 업로드

주요기능

1. 목표와 목적
2. 주요 기능
3. 프로젝트의 차별점

3. 문의사항 시스템
    a. 문의 등록, 수정/삭제(답변 미등록시)
    b. 카테고리 분류
    c. 답변 상태 추적
    d. 페이지네이션 (AJAX)

주요기능

1. 목표와 목적
2. 주요 기능
3. 프로젝트의 차별점

4. 보안기능
    a. 비밀번호 암호화 (SHA-256)
    b. Salt 생성 및 관리
    c. 세션 관리
    d. XSS 방지 (입력 검증)
    e. 로그인 체크 필터

주요기능

1. 목표와 목적
2. 주요 기능
3. 프로젝트의 차별점

5. 데이터베이스 관리
   a. JNDI 기반 커넥션풀
   b. 트랜잭션 처리
   c. SQL injection 방지

프로젝트의 차별점

1. 목표와 목적
2. 주요 기능
3. 프로젝트의 차별점

1. 자동 로그인 기능: 사용자가 "로그인 상태 유지"를 선택하면 UUID로 생성된 암호화된 토큰을 사용하여 자동 로그인을 구현, 사용자 편의성을 극대화함.

2. 보안 강화: 비밀번호는 SHA-256 해싱과 솔트를 이용해 안전하게 저장하고, 자동 로그인 토큰도 암호화하여 보안을 강화함. 만료된 토큰은 주기적으로 삭제하여 데이터베이스의 안전성을 높임.

3. 유동적인 토큰 관리: 사용자의 자동 로그인 상태를 유연하게 관리하며, 만료 시간을 연장하여 로그인 세션을 지속적으로 유지할 수 있도록 지원함.

프로젝트의 차별점

1. 목표와 목적
2. 주요 기능
3. 프로젝트의 차별점

4. 실제 홈 서버 배포: 홈 서버를 사용하여 포트 포워딩을 통해 실제 서비스 환경에 배포함으로써, 실제 운영 환경에서의 성능과 안정성을 확인함.

5. HTTPS 구현: Let's Encrypt를 통해 HTTPS를 적용하여 데이터 전송 시 보안을 강화하고, 사용자 정보를 안전하게 보호함.

02

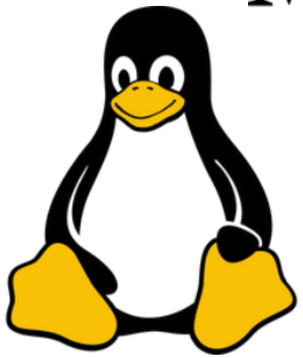
개발 환경 및 기술 스택

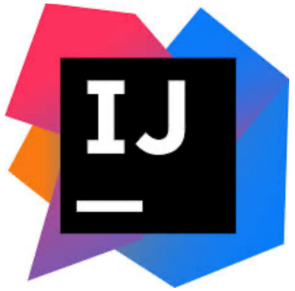
개발 환경 및 기술 스택

언어 및 프레임워크

개발 환경 및 기술 스택

툴과 플랫폼





03

기능 시연 (Demo)

04

주요 코드 리뷰

주요 코드 리뷰

db 쿼리 파라미터 바인딩 자동화

```
// 파라미터 있는 쿼리 Object 배열 파라미터 받음 (String || Integer)
//파일 업로드 시 파일 사이즈 받아오기 위해 Long 타입 추가
//null 처리 추가를 위해 다른파라미터들을 setObject로 변경후 setNull 추가 :(
public DbQueryUtil(Connection conn, String sql, Object[] parameters) throws SQLException {
    this.pstmt = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
    if (parameters != null) {
        for (int i = 0; i < parameters.length; i++) {
            if (parameters[i] != null) {
                pstmt.setObject(i + 1, parameters[i]);
            } else {
                pstmt.setNull(i + 1, java.sql.Types.NULL);
            }
        }
    }
}
```

PreparedStatement를 사용함으로써 파라미터 바인딩을 자동화하여, setObject(i + 1, parameters[i])와 같은 형식으로 간편하게 쿼리와 파라미터를 연결할 수 있습니다. 이는 SQL 인젝션 공격을 방지하고 코드 가독성을 높이는 데 도움을 줍니다.
null 처리: parameters가 null이 아닐 경우, 각 파라미터를 순회하면서 setObject를 통해 SQL 문에 설정하며, null 값인 경우에는 setNull을 사용하여 해당 위치에 NULL 값을 설정합니다. 이렇게 함으로써 SQL 쿼리에서의 NULL 처리를 간편하게 수행할 수 있습니다.

PasswordUtil 클래스 메서드

```java
// 솔트 생성
public static String generateSalt() {
    SecureRandom random = new SecureRandom();
    byte[] salt = new byte[SALT_LENGTH];
    random.nextBytes(salt);
    return Base64.getEncoder().encodeToString(salt);
}
```

비밀번호 해싱에 사용할 랜덤한 솔트를 생성합니다. SecureRandom을 사용하여 안전하게 랜덤 바이트를 생성하고, 이를 Base64 인코딩하여 문자열로 반환합니다. 이는 솔트의 안전성을 높입니다.

PasswordUtil 클래스 메서드

```java
// 비밀번호 해싱
public static String hashPassword(String password, String salt) {
    try {
        MessageDigest md = MessageDigest.getInstance(HASH_ALGORITHM);
        md.update(Base64.getDecoder().decode(salt));
        byte[] hashedPassword = md.digest(password.getBytes());
        return Base64.getEncoder().encodeToString(hashedPassword);
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException(message:"Failed to hash password", e);
    }
}
```

주어진 비밀번호와 솔트를 이용해 비밀번호를 해싱합니다. SHA-256 알고리즘을 사용하여 비밀번호를 해싱하고, 해시 결과를 Base64로 인코딩하여 반환합니다. 이 과정에서 NoSuchAlgorithmException 예외를 처리하여 해싱 실패 시 런타임 예외를 발생시킵니다.

```java
// 비밀번호 검증
public static boolean verifyPassword(String inputPassword, String storedPassword, String salt) {
    String hashedInputPassword = hashPassword(inputPassword, salt);
    return hashedInputPassword.equals(storedPassword);
}

// 새 비밀번호 생성 (해시된 비밀번호와 솔트 반환)
public static String[] createNewPassword(String password) {
    String salt = generateSalt();
    String hashedPassword = hashPassword(password, salt);
    return new String[]{hashedPassword, salt};
}
```

PasswordUtil 클래스 메서드

verifyPassword :
입력된 비밀번호와 저장된 비밀번호를 비교하여 일치 여부를 검증합니다. 입력된 비밀번호를 해싱한 후, 저장된 해시와 비교하여 동일하면 true를 반환하고, 그렇지 않으면 false를 반환합니다.

createNewPassword:
새 비밀번호를 생성하는 메서드로, 새로운 솔트를 생성하고, 이를 통해 해시된 비밀번호와 솔트를 반환하는 배열을 생성합니다. 이 메서드는 사용자가 새 비밀번호를 설정할 때 유용하게 사용됩니다.

05

결과 및 개선점

결과 및 개선점

결과	주요 구현 기능

- 회원 관리 시스템
- 메시지 시스템
- 문의사항 시스템

성공적인 구현 사항

- 보안성 (세션기반 인증, sql 인젝션 방지, xss 방지)
- 확장성 (모듈화된 구조, MVC 패턴 적용, 재사용 가능한 컴포넌트)
- 데이터 무결성 (트랜잭션 처리, 사용자 권한 검증)

개선점 및 한계점

- 복잡한 데이터베이스 구조
- 코드 복잡도
- 성능이슈 예상

한계점 상세 및 결론

결론
메시지 시스템의 기본적인 기능은 성공적으로 구현되었으나, 데이터베이스 설계의 복잡성으로 인해 성능과 유지보수성에 제약이 있습니다. 향후 개선을 통해 더 효율적이고 확장 가능한 시스템으로 발전시킬 필요가 있습니다.

- 한계점
  - a. 복잡한 데이터 베이스 구조
    - i. 메시지 관련 테이블 3개 분리
    - ii. 조회시 복잡한 join 쿼리 필요
    - iii. 상태 업데이트 시 다중 테이블 수정 필요
  - b. 코드 복잡도
    - i. 비즈니스 로직과 데이터 접근 로직 혼재
    - ii. 반복적인 예외 처리 코드
    - iii. 복잡한 상태 관리 로직
  - c. 성능 이슈 예상
    - i. 복잡한 쿼리로 인한 처리시간 증가
    - ii. 불필요한 중복 조회 발생
    - iii. 대량의 데이터 처리시 비효율 예상