# 로또 웹사이트 구축 개발 및 테스트 보고서

## 사이버보안학과 32213184 이성엽

## 1. 프로젝트 개요 및 목표

### 1.1 프로젝트 목표

본 프로젝트는 Python의 웹 프레임워크인 Django를 기반으로 로또 구매 및 관리 기능을 제공하는 웹사이트를 구축하고, 이를 Docker 환경에 배포하는 것을 목표로 한다. 개발 과정의 난이도를 높이기 위해, Docker Compose를 활용하여 Django 애플리케이션 컨테이너와 PostgreSQL 데이터베이스 컨테이너를 통합하는 다중 컨테이너 환경을 구성하였다.

### 1.2 주요 요구사항 분석

| **구분** | **기능 명세** | **구현 상태** |
| --- | --- | --- |
| **사용자 (User)** | 로또 번호 생성 (수동/자동) | 완료 |
|  | 로또 구매 및 구매 기록 조회 | DB 연결 통합 완료 (SQLite 기반) |
|  | 당첨 번호 및 당첨 내역 확인 | 로직 구현 완료 |
| **관리자 (Admin)** | 로또 추첨 (당첨 번호 생성) | 로직 구현 완료 |
|  | 로또 판매 실적 확인 | DB 연결 통합 완료 |
|  | 당첨자 확인 및 등수 처리 | 핵심 알고리즘 구현 완료 |
| **배포 환경** | Docker 기반 다중 컨테이너 환경 구축 | Docker 파일 및 설정 완료 (DB 연결 문제로 임시 SQLite 전환) |

## 2. 시스템 설계 및 데이터 모델링

### 2.1 데이터 모델 (Django Model) 설계

웹사이트의 핵심 비즈니스 로직을 처리하기 위해 총 두 개의 주요 모델을 설계하였다.

#### 1) LottoRound (회차 정보 모델)

로또의 회차별 추첨 정보(6개 당첨 번호, 1개 보너스 번호, 추첨일)를 저장한다.

#### 2) LottoTicket (구매 티켓 모델)

사용자가 구매한 로또 1장의 정보를 저장한다. 구매자(User 외래키), 구매 번호 6개, 구매 유형, 당첨 여부, 최종 당첨 등수를 포함한다. 이 모델 내부에 핵심 비즈니스 로직인 check\_rank() 메서드를 포함하여 데이터와 로직의 응집도를 높였다.

### 2.2 당첨 등수 판정 알고리즘 (check\_rank)

로또 당첨 판정은 서비스의 핵심 로직이며, LottoTicket 모델에 메서드로 구현하였다.

| **등수** | **조건** | **구현 로직** |
| --- | --- | --- |
| **1등** | 메인 번호 6개 일치 | match\_count == 6 |
| **2등** | 메인 번호 5개 + 보너스 번호 일치 | match\_count == 5 AND is\_bonus\_match |
| **3등** | 메인 번호 5개 일치 | match\_count == 5 |
| **4등** | 메인 번호 4개 일치 | match\_count == 4 |
| **5등** | 메인 번호 3개 일치 | match\_count == 3 |
| **낙첨** | 그 외 | return 0 |

## 3. 개발 환경 구축 및 테스트 과정

### 3.1 Docker 환경 구축 및 난관 분석

본 프로젝트는 Docker를 통한 PostgreSQL DB와 Django 앱의 통합을 시도하였다.

1. 파일 구성: Dockerfile, docker-compose.yml, .env, requirements.txt를 프로젝트 루트 디렉토리에 구성하였다.
2. 초기 문제 (Connection Refused): docker-compose up 후 로컬 Django 환경에서 migrate 시 connection to server at "localhost" (127.0.0.1), port 5432 failed: Connection refused 오류가 반복 발생했다. 이는 DB 컨테이너의 비정상 종료 또는 로컬 포트 노출 문제로 진단되었고, docker-compose up -d db 명령으로 DB 실행을 우선하며 대응했다.
3. 2차 난관 (Connection Abort): DB 서버가 켜진 후에도 connection to server at "localhost" (::1), port 5432 failed: Software caused connection abort 오류가 발생했다. 이는 환경 변수 불일치로 인한 인증 실패(password authentication failed) 또는 DB 볼륨 충돌 문제로 분석되었다.
4. 전략적 전환: 반복적인 DB 연결 문제로 인해 PostgreSQL 대신 SQLite3를 임시 DB로 설정

### 3.2 핵심 기능 구현 상세

| **기능** | **상세 로직 및 구현 위치** | **테스트 결과** |
| --- | --- | --- |
| **자동 구매** | lotto/views.py 내 purchase\_lotto 함수에서 random.sample(range(1, 46), 6)을 사용하여 1~45 중 중복 없는 6개 번호 생성 및 LottoTicket 모델에 저장. | **성공:** 매번 다른 6개 번호가 생성되며 DB에 저장됨. |
| **수동 구매** | lotto/views.py에서 폼 데이터 6개 번호를 수신하고, **views와 프론트엔드(JS)** 양쪽에서 **1~45 범위 검사 및 중복 검사**를 수행하여 유효성 확보. | **성공:** 유효하지 않은 입력 시 에러 메시지 반환 및 DB 저장 거부. |
| **로또 추첨** | lotto/views.py 내 perform\_draw 함수에서 당첨 번호를 생성하고 LottoRound 모델에 저장한 후, 해당 회차의 모든 LottoTicket을 순회하며 check\_rank 메서드를 호출하여 등수를 일괄 업데이트. | **성공:** 추첨 번호 생성 후, DB 데이터가 등수에 따라 정상적으로 업데이트됨. |
| **사용자 인증** | Django 기본 User 모델을 사용하며, login\_required 데코레이터를 사용하여 구매 기능 접근 제어. | **성공:** 로그인한 사용자만이 구매 기록을 남길 수 있도록 구현. |

### 3.3 테스트 시나리오 및 결과

| **시나리오** | **기대 결과** | **실제 결과** |
| --- | --- | --- |
| **시나리오 A: 1등 당첨** | 구매 번호 6개가 추첨 번호 6개와 정확히 일치 시 1등으로 판정됨. | **성공** |
| **시나리오 B: 2등 당첨** | 구매 번호 5개가 메인 번호와 일치하고, 나머지 1개가 보너스 번호와 일치 시 2등으로 판정됨. | **성공** |
| **시나리오 C: 중복 번호 구매 시도** | 수동 구매 시 폼 유효성 검사(프론트엔드/백엔드)에 의해 DB 저장 거부 및 에러 메시지 표시. | **성공** |

## 4. 소스 코드 및 향후 계획

### 4.1 소스 코드 제출

본 프로젝트의 모든 소스 코드 및 설정 파일은 다음 GitHub 저장소에 제출되었습니다.

GitHub 저장소 링크:

https://github.com/lsystar0625-sudo/dkuassignmentlotto

## 5. 참고 문헌 및 용어 설명

### 5.1 참고 문헌

* Django 공식 문서 (Documentation)
* PostgreSQL 공식 문서
* Docker 및 Docker Compose 공식 가이드

### 5.2 용어 설명

| **용어** | **설명** |
| --- | --- |
| **Django** | Python 기반의 하이 레벨 웹 프레임워크. 신속한 개발(Rapid Development)을 목표로 하며, MTV(Model-Template-View) 아키텍처를 따름. |
| **Docker** | 애플리케이션과 그 실행 환경(라이브러리, 설정 등)을 컨테이너라는 형태로 패키징하고 격리하여, 어디서든 동일하게 실행되도록 보장하는 플랫폼. |
| **Docker Compose** | 다수의 컨테이너로 구성된 애플리케이션을 정의하고 실행하는 도구. 본 프로젝트에서는 web(Django)과 db(PostgreSQL) 컨테이너를 통합 구성하는 데 사용됨. |
| **psycopg2** | Python에서 PostgreSQL 데이터베이스에 접속하기 위한 DB 어댑터(Driver). |
| **Migration** | Django 모델의 변경 사항을 데이터베이스 스키마에 적용하는 과정. makemigrations로 파일 생성 후 migrate로 DB에 반영. |
| **SQLite3** | 파일 기반의 경량 데이터베이스. 별도의 서버 구성 없이 단일 파일로 동작하여 개발 초기에 사용하기 용이함. |