[TP-7] 최종 보고서 및 소스코드

2025. 06. 12.

학번: 202000826　　이름: 김연범

7-1. 구축된 DB의 각 테이블 내용

1. Airplane

```
SELECT * FROM AIRPLANE;
```

스크립트 출력 x　질의 결과 x

SQL | 인출된 모든 행: 36(0.01초)

	AIRLINE	FLIGHTNO	DEPARTUREDATETIME	DEPARTUREAIRPORT	ARRIVALDATETIME	ARRIVALAIRPORT
1	KOREAN	K123	25/05/04	인천	25/05/04	싱가폴
2	KOREAN	K123	25/05/03	인천	25/05/03	싱가폴
3	KOREAN	K123	25/05/05	인천	25/05/05	싱가폴
4	TWAY	T123	25/05/06	인천	25/05/06	일본
5	TWAY	T123	25/05/07	인천	25/05/07	일본
6	TWAY	T123	25/05/08	인천	25/05/08	일본
7	ASIANA	A123	25/05/04	인천	25/05/04	제주
8	ASIANA	A123	25/05/03	인천	25/05/03	제주
9	ASIANA	A123	25/05/05	인천	25/05/05	제주
10	ASIANA	A123	25/05/06	인천	25/05/06	제주
11	ASIANA	A123	25/05/07	인천	25/05/07	제주
12	ASIANA	A123	25/05/08	인천	25/05/08	제주
13	ASIANA	A123	25/05/09	인천	25/05/09	제주
14	ASIANA	A123	25/05/10	인천	25/05/10	제주

2. Customer

```
SELECT * FROM CUSTOMER;
```

스크립트 출력 x　질의 결과 x

SQL | 인출된 모든 행: 6(0.009초)

	CNO	NAME	PASSWD	EMAIL	PASSPORTNUMBER
1	12345	YEONBUM	1234	KYB@NAVER.COM	M123S1234
2	12346	JEONGEN	5678	JGN@NAVER.COM	M124S1235
3	12347	SULHWA	9012	SHW@NAVER.COM	M126S1237
4	12348	JUNGKU	3456	JK@NAVER.COM	M128S1239
5	12349	BOHYEN	7890	BHN@NAVER.COM	M113S1134
6	12350	ENJUNG	0987	EJ@NAVER.COM	M133S1334

3. Cancel

```
SELECT * FROM CANCEL;
```

스크립트 출력 × 질의 결과 ×

SQL | 인출된 모든 행: 10(0.005초)

	FLIGHTNO	DEPARTUREDATETIME	SEATCLASS	REFUND	CANCELDATETIME	CNO
1	A123	25/05/20	ECONOMY	300	25/05/08	12346
2	A123	25/05/26	ECONOMY	300	25/05/09	12346
3	A123	25/05/29	ECONOMY	300	25/05/10	12346
4	A123	25/05/12	ECONOMY	300	25/05/11	12347
5	A123	25/05/15	ECONOMY	300	25/05/12	12347
6	A123	25/05/18	ECONOMY	300	25/05/13	12347
7	A123	25/05/21	ECONOMY	300	25/05/14	12347
8	A123	25/05/24	ECONOMY	300	25/05/15	12347
9	A123	25/05/10	ECONOMY	300	25/05/02	12345
10	A123	25/05/13	ECONOMY	300	25/05/10	12345

4. Reserve

```
SELECT * FROM RESERVE;
```

질의... ×

SQL | 인출된 모든 행: 22(0.042초)

	FLIGHTNO	DEPARTUREDATETIME	SEATCLASS	PAYMENT	RESERVEDATETIME	CNO
1	T123	25/05/08	ECONOMY	400	25/05/03	12350
2	K123	25/05/05	ECONOMY	1000	25/05/02	12349
3	A123	25/05/10	ECONOMY	300	25/05/01	12345
4	A123	25/05/13	ECONOMY	300	25/05/01	12345
5	A123	25/05/16	ECONOMY	300	25/05/01	12345
6	A123	25/05/19	ECONOMY	300	25/05/01	12345
7	A123	25/05/22	ECONOMY	300	25/05/01	12345
8	A123	25/05/25	ECONOMY	300	25/05/02	12345
9	A123	25/05/28	ECONOMY	300	25/05/02	12345
10	A123	25/05/11	ECONOMY	300	25/05/03	12346
11	A123	25/05/14	ECONOMY	300	25/05/04	12346
12	A123	25/05/17	ECONOMY	300	25/05/05	12346
13	A123	25/05/20	ECONOMY	300	25/05/06	12346
14	A123	25/05/23	ECONOMY	300	25/05/07	12346
15	A123	25/05/26	ECONOMY	300	25/05/08	12346
16	A123	25/05/29	ECONOMY	300	25/05/09	12346
17	A123	25/05/12	ECONOMY	300	25/05/10	12347
18	A123	25/05/15	ECONOMY	300	25/05/11	12347
19	A123	25/05/18	ECONOMY	300	25/05/12	12347
20	A123	25/05/21	ECONOMY	300	25/05/13	12347
21	A123	25/05/24	ECONOMY	300	25/05/14	12347
22	A123	25/05/27	ECONOMY	300	25/05/15	12347

5. Seats

```
SELECT * FROM SEATS;
```

스크립트 출력 | 질의 결과

SQL | 인출된 모든 행: 42(0.004초)

	FLIGHTNO	DEPARTUREDATETIME	SEATCLASS	PRICE	NO_OF_SEATS
1	K123	25/05/03	BUSINESS	2000	50
2	K123	25/05/04	BUSINESS	2000	50
3	K123	25/05/05	BUSINESS	2000	50
4	K123	25/05/05	FIRST	3000	10
5	K123	25/05/04	FIRST	3000	10
6	K123	25/05/03	FIRST	3000	10
7	K123	25/05/03	ECONOMY	1000	500
8	K123	25/05/04	ECONOMY	1000	500
9	K123	25/05/05	ECONOMY	1000	500
10	T123	25/05/06	ECONOMY	200	200
11	T123	25/05/07	ECONOMY	200	200
12	T123	25/05/08	ECONOMY	400	200
13	A123	25/05/03	ECONOMY	300	300
14	A123	25/05/04	ECONOMY	300	300

7-2. 구현된 시스템 상세 설명

1. 구현 환경

본 시스템은 웹 기반 항공권 예약 플랫폼으로, 사용자에게 항공편 조회, 예약, 예약 내역 확인, 공지사항 및 FAQ 열람 등 다양한 서비스를 제공한다. 전체 시스템은 Python 기반의 Flask 프레임워크를 중심으로 구성되었으며, 클라이언트와 서버 간의 상호작용을 효과적으로 처리할 수 있도록 설계되었다. 구현 환경은 다음과 같다.

가. 개발 언어 및 프레임워크

- 프로그래밍 언어: Python 3.10.8
- 웹 프레임워크: Flask 3.0.3
- 템플릿 엔진: Jinja2 (Flask 기본 내장)
- HTML/CSS/JavaScript: 사용자 인터페이스 구현을 위한 Front-end 구성

나. 데이터베이스

- DBMS: Oracle Database 21
- 연결 방식: cs_Oracle을 사용해 Flask 서버에서 Oracle DB로 직접 쿼리 수행

다. 서버 환경

- 운영체제: Window 11
- 웹 서버: Flask 내장 개발 서버(local 서버)

2. 각 모듈 별 주요 알고리즘

본 시스템은 항공권 예약 서비스를 중심으로 사용자의 예약, 조회, 취소, 회원 정보 관리, 공지사항 및 FAQ 검색 등 다양한 기능을 제공하며, 각각의 기능은 Flask 기반 라우팅과 Oracle 데이터베이스 연동을 통해 처리된다. 주요 모듈별 알고리즘 흐름은 다음과 같다.

가. 로그인 화면

1) 기능 설명: 사용자가 id, password를 입력하면 해당 조건에 부합하는 회원을 Customer 테이블에서 조회하고, 해당 회원 정보를 가져온다. 로그인에 실패했을 경우 error 메시지를 호출하고, 로그인에 성공했을 경우 main 화면으로 렌더링된다.

2) 알고리즘 흐름

Algorithm 1: User Login Process

Algorithm 1: User Login Procedure

Input: User ID `id`, Password `password`
Output: Success: Redirect to main page; Failure: Show error message

Step 1: Receive input credentials `id`, `password` from the login form
Step 2: Query the `Customer` table in the database where the values match:

```
SELECT * FROM Customer WHERE cno = :id AND passwd = :password
```

Step 3: **if** *a matching record is found* **then**
    Fetch user information into session variables
    Redirect the user to the `main.html` page
else
    Display a flash message indicating login failure
    Remain on or redirect to the login page

나. 회원가입 화면

1) 기능 설명: 사용자가 id, name, password, email, passport_number를 입력하면 해당 데이터를 만족하는 회원을 Customer 테이블에 추가한다. 회원가입에 성공했을 경우 회원가입에 성공했다는 메시지를 호출한다.

2) 알고리즘 흐름

Algorithm 2: User Registration Process

Algorithm 1: User Registration Procedure

Input: User ID `id`, Name `name`, Password `password`, Email `email`, Passport Number `passport_number`

Output: Success: Show registration success message; Failure: Show error message

Step 1: Receive input values from the registration form: `id`, `name`, `password`, `email`, `passport_number`
Step 2: Validate inputs for completeness and format
if *any required field is missing or invalid* **then**
    Display an error message and terminate the process
Step 3: Construct SQL insert query:

```
INSERT INTO Customer
VALUES (:id, :name, :password, :email, :passport_number)
```

Step 4: if *insertion is successful* **then**
    Display a message indicating successful registration
else
    Display an error message indicating registration failure

다. 메인 화면

1) 기능 설명: 상단에 CNU Airline, My Airline, 고객센터 메뉴가 있으며, 사용자가 마우스 커서를 올리면 각각 {항공권 예약, 예약 조회}, {나의 예약 조회, 나의 취소 내역, 회원 정보 관리}, {공지사항, FAQ} 소 메뉴가 list-out 된다. 각 소 메뉴를 클릭 시, 해당 페이지로 렌더링된다.

2) 알고리즘 흐름

Algorithm 3: Navigation Menu Interaction and Routing

Algorithm 1: Top Menu Interaction and Page Routing

Input: User interface events (mouse hover, menu click)
Output: Display submenu or redirect to corresponding page
Step 1: Display the top-level navigation menu with the following items:
    `CNU Airline`, `My Airline`, `Customer Service`
Step 2: foreach *top-level menu item* **do**
    if *mouse hovers over the item* **then**
        Show corresponding submenu as a dropdown list
Step 3: Submenu Mapping:
    `CNU Airline` → {`Book Flight`, `Check Reservations`}
    `My Airline` → {`My Reservations`, `My Cancellations`, `Profile Settings`}
    `Customer Service` → {`Notices`, `FAQ`}
Step 4: if *user clicks on a submenu item* **then**
    Redirect to the corresponding route and render the appropriate page
        Example: Click on `My Reservations` → route to `/reservation_lookup`

라. 항공권 예약 화면

1) 기능 설명: 사용자가 출발지, 도착지, 출발 날짜, 좌석 등급, 정렬 기준을 입력하면, 해당 조건에 부합하는 항공편 목록을 조회하고, 개별 항공편 옆의 "예약" 버튼을 통해 선택 항공편을 예약할 수 있다. 예약에 성공하면 사용자에게 항공편 티켓을 이메일로 전송한다. 티켓 내용 중 남은 좌석 수는 (SEATS 테이블의 각 항공편 별 좌석 수) – (RESERVE 테이블의 동일 항공편을 예약한 예약 건수)를 통해 계산된다.

2) 알고리즘 흐름

Algorithm 4: Flight Search and Reservation Process

Algorithm 1: Flight Search and Reservation Flow

Input: Departure Airport, Arrival Airport, Departure Date, Seat Class, Order Criterion

Output: List of matching flights and reservation status

Step 1: Query AIRPLANE table with user input

```
SELECT A.flightno, A.departureairport, A.arrivalairport,
TO_CHAR(A.departuredatetime, 'YYYY-MM-DD HH24:MI') as
 departure_time,
TO_CHAR(A.arrivaldatetime, 'YYYY-MM-DD HH24:MI') as
 arrival_time
S.seatclass, S.price, S.no_seats
FROM AIRPLANE A
JOIN SEATS S ON A.flightno = S.flightno
ND A.departuredatetime = S.departuredatetime
WHERE A.departureairport = :1
AND A.arrivalairport = :2
AND A.departuredatetime BETWEEN TO_DATE(:3, 'YYYY-MM-DD')
 AND TO_DATE(:4, 'YYYY-MM-DD') + 1
 AND S.seatclass = UPPER(:5)
ORDER BY S.price ASC or S.departuredatetime ASC
```

Step 2: Calculate the remain seats of each ticket using Count query

Step 3: Display the list of matching flights with flight information

Step 4: **if** *user clicks "Reserve"* **then**

- **(a)** Get user's customer number (CNO) from login session
- **(b)** Insert reservation into RESERVE table:

```
INSERT INTO RESERVE (FLIGHTNO, DEPARTUREDATETIME,
 SEATCLASS, PAYMENT, RESERVATEDATETIME, CNO)
VALUES (selected_flightno, selected_datetime,
 input_seatclass, price, CURRENT_TIMESTAMP,
 session_cno)
```

- **(c) sending email to customer with the ticket**

마. 예약 조회 화면

1) 기능 설명: 사용자가 예약한 모든 항공편 목록을 조회할 수 있는 화면이다.

2) 알고리즘 흐름

Algorithm 5: Retrieve All Reservations for Logged-in User

Algorithm 1: Reservation History Query for Logged-in User

Input: Logged-in user's customer number (`CNO`) from session
Output: List of reserved flight details

Step 1: Retrieve the customer's ID from session
```
cno = session[cno]
```
Step 2: Execute SQL query to join `RESERVE` and `AIRPLAIN` tables:
```
SELECT a.airline, r.flightno, a.departureairport,
 a.arrivalairport,
r.departuredatetime, a.arrivaldatetime, r.payment
FROM RESERVE r, AIRPLAIN a
WHERE r.CNO = cno
AND r.flightno = a.flightno
AND r.departuredatetime = a.departuredatetime;
```
Step 3: Fetch the result set containing all matching reservation records
Step 4: Display the list of reservations to the user in the front-end interface

바. 나의 예약/취소 내역 화면

1) 기능 설명: 사용자가 기간을 입력하고 조회 버튼을 누르면, 해당 조건에 부합하는 예약, 취소 내역을 조회할 수 있는 화면이다.

2) 알고리즘 흐름

Algorithm 6: Retrieve Reservations and Cancellations by Date Range

Algorithm 1: Date-Filtered Reservation and Cancellation History Retrieval

Input: Start Date (`start_date`), End Date (`end_date`)
Output: Filtered reservation and cancellation history for the user

Step 1: Obtain logged-in user's customer number (`CNO`) from session
```
cno = session[cno]
```
Step 2: Retrieve reservation history from `RESERVE` table
Step 3: Retrieve cancellation history from `CANCEL` table
Step 4: For each record in reservation and cancellation results:
- **4.1:** Join with `AIRPLANE` table using `FLIGHTNO` and `DEPARTUREDATETIME`
- **4.2:** Extract the following details:
  - Airline name, Departure and Arrival airports
  - Departure and Arrival datetimes
  - Seat class and Payment or Refund amount

Step 5: Render results into HTML template
- Separate sections for Reservations and Cancellations
- Each entry includes full flight and transaction information

사. 회원 정보 관리 화면

1) 기능 설명: 사용자가 id, name, password, email, passport_number를 입력하고 변경사항 저장하기 버튼을 누르면, Customer 테이블에서 로그인된 사용자의 cno를 만족하는 회원 정보를 업데이트한다.

2) 알고리즘 흐름

Algorithm 7: Update Customer Profile Information

Algorithm 1: Customer Profile Information Update

Input: Updated values: `id`, `name`, `password`, `email`, `passport_number`
Output: Status message indicating success or failure of the update

Step 1: Get the logged-in user's `cno` from session
```
cno = session[cno]
```
Step 2: Validate the input values (e.g., non-empty fields, valid formats)
```
if not valid(id, name, password, email, passport_number):
    flash("Invalid input.  Please check your data.")
    return
```
Step 3: Execute SQL update query on `Customer` table
```
UPDATE Customer SET
  cno = id,
  name = name,
  passwd = password,
  email = email,
  passportnumber = passport_number
WHERE cno = :cno;
```
Step 4: Commit the transaction
```
db.commit()
```

아. 공지사항 및 FAQ 화면

1) 기능 설명: 사용자가 키워드를 입력하면, 해당 키워드를 포함하는 제목의 공지 또는 FAQ 항목만 필터링하여 보여준다.

2) 알고리즘 흐름

Algorithm 8: Filter Notices or FAQs by Keyword

Algorithm 1: Filter Notices or FAQs by Keyword

Input: Search keyword `K`, Mode `M` $\in$ {`"notice"`, `"faq"`}
Output: Filtered list of entries whose title contains keyword `K`

Step 1: Read user input for search keyword
```
K = request.form[keyword]
```
Step 2: Determine target table from mode `M`
```
if M == "notice" then
    table = "NOTICE"
else
    table = "FAQ"
```
Step 3: Execute SQL query to filter titles

Step 4: Render filtered results

자. 관리자 화면

1) 기능 설명: 관리자가 로그인 화면에서 관리자 계정으로 접속하면, 항공권 취소 이력이 있는 고객의 모든 정보와 취소 횟수를 확인할 수 있다.

2) 알고리즘 흐름

Algorithm 9: Admin Query Process

Algorithm 1: Admin Query Process

Output: User ID `cno`, UserName `name`, Email `email`, Password `password`, PassPortNumber `passportnumber`, CancelCount `count`, CancelRank `rank`

1 **Step 1:** Push the `check` button
2 **Step 2:** Query the `Customer` table and `Cancel` table in the database where the values match:

```
3    SELECT a.cno, b.name, b.passwd, b.email,
      b.passportnumber, a.cancel_count, RANK() OVER (ORDER BY
      cancel_count DESC) AS cancel_rank
4    FROM ( SELECT cno, COUNT(*) AS cancel_count FROM CANCEL
      GROUP BY CNO ) a, CUSTOMER b
5    WHERE b.cno = a.cno
6    ORDER BY cancel_rank
```

7 **Step 3:** Render the admin.html with customer's information

7-3. User Manual

1. 로그인 화면

가. 초기 화면

나. 기능 설명: id(=cno)와 password를 입력 후 로그인 버튼을 누르면, 메인 화면으로 렌더링된다. 회원가입 버튼을 누르면 회원가입 화면으로 렌더링된다.

2. 회원가입 화면

가. 초기 화면

나. 기능 설명: id(=cno), 이름, 비밀번호, 이메일, 여권 번호를 입력 후 회원가입 버튼을 누르면, 회원가입 완료 알림과 함께 Customer 테이블에 회원 정보가 새로 삽입된다. 뒤로가기 버튼을 누르면 로그인 화면으로 렌더링된다.

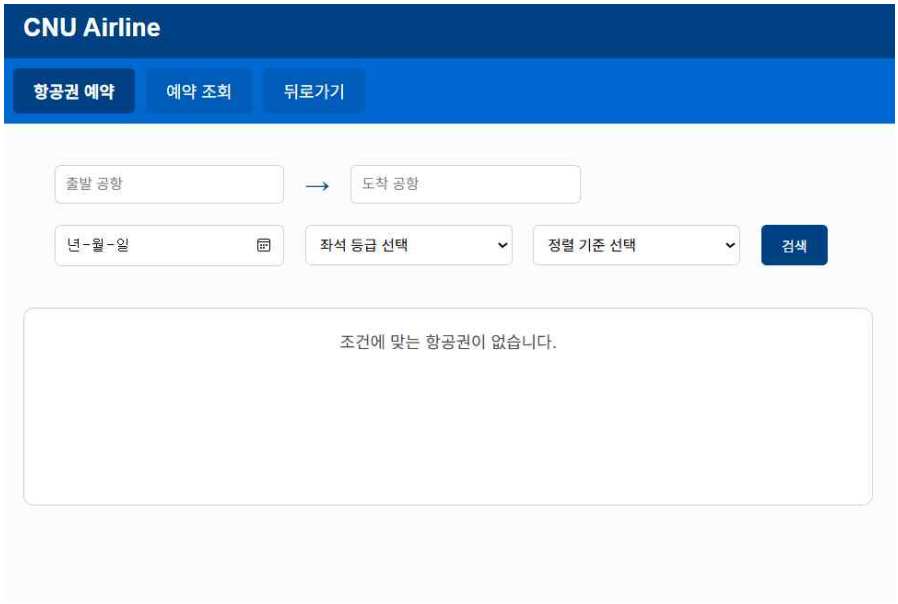
3. 메인 화면

가. 초기 화면

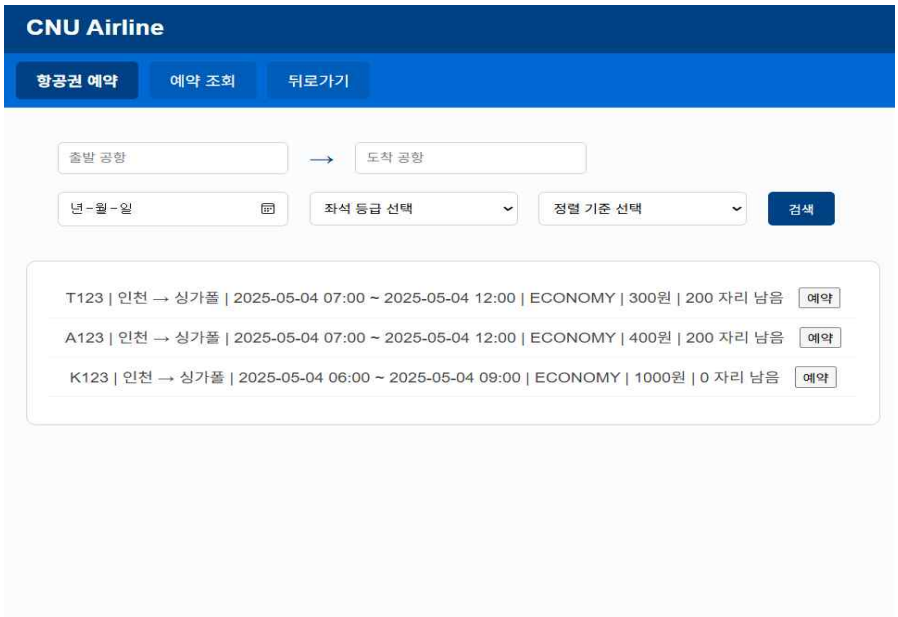
나. 기능 설명: 항공권 예약, 예약 조회, 나의 예약/취소 내역, 회원 정보 관리, 공지사항, FAQ 메뉴로 이동할 수 있는 메인 화면으로, CNU Airline에 hover 시 {항공권 예약, 예약 조회} 서브 메뉴가, My Airline에 hover 시 {나의 예약 내역, 나의 취소 내역, 회원 정보 관리} 서브 메뉴가, 고객센터에 hover 시 {공지사항, FAQ} 서브 메뉴가 list-out 된다.

4. 항공권 예약 화면

가. 초기 화면



나. 검색 버튼 클릭 화면



다. 기능 설명: 항공권을 예약할 수 있는 화면으로, {출발 공항, 도착 공항, 출발 날짜, 좌석 등급, 정렬 기준}을 입력한 후 검색 버튼을 클릭하면 해당하는 항공권을 보여준다. 항공권은 정렬 기준에 따라 정렬된 채 출력된다. 항공권 옆에 예약 버튼을 누르면, 해당 항공권이 예약된 후 예약 조회 화면으로 자동 렌더링된다. 이와 동시에, 사용자에게 티켓이 이메일로 전송된다(이메일 형식은 별첨1 참고). 예약 조회 버튼을 부르면 예약 조회 화면으로, 뒤로가기 버튼을 두르면 메인 화면으로 렌더링된다. 만약 자리가 남아있지 않은 항공편을 예약할 경우, '예약할 수 없는 항공편입니다. 다른 항공편을 선택하세요.' 라는 문구와 함께 구매 불가 처리가 된다.

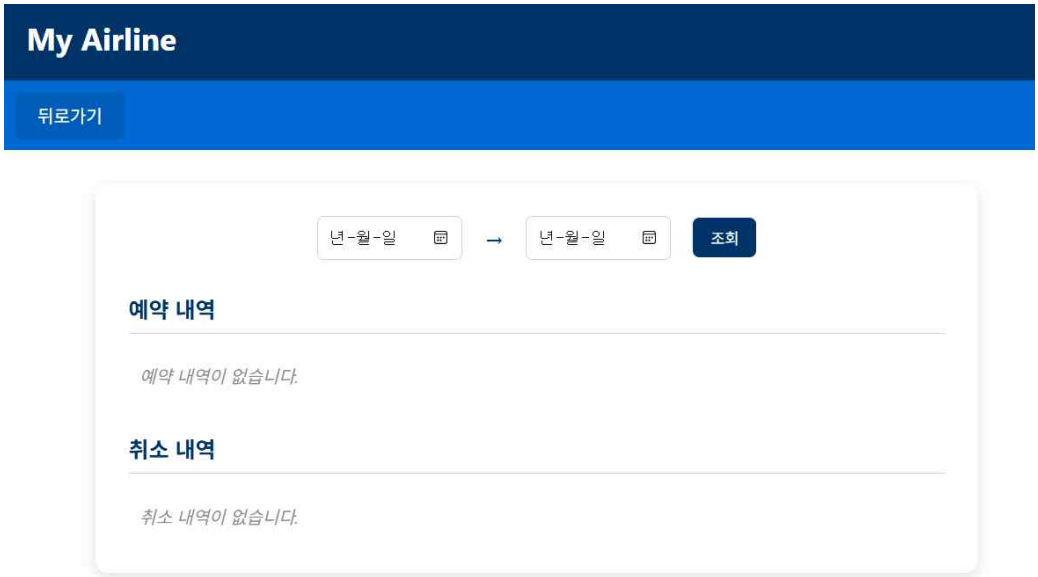
5. 예약 조회 화면

가. 초기 화면

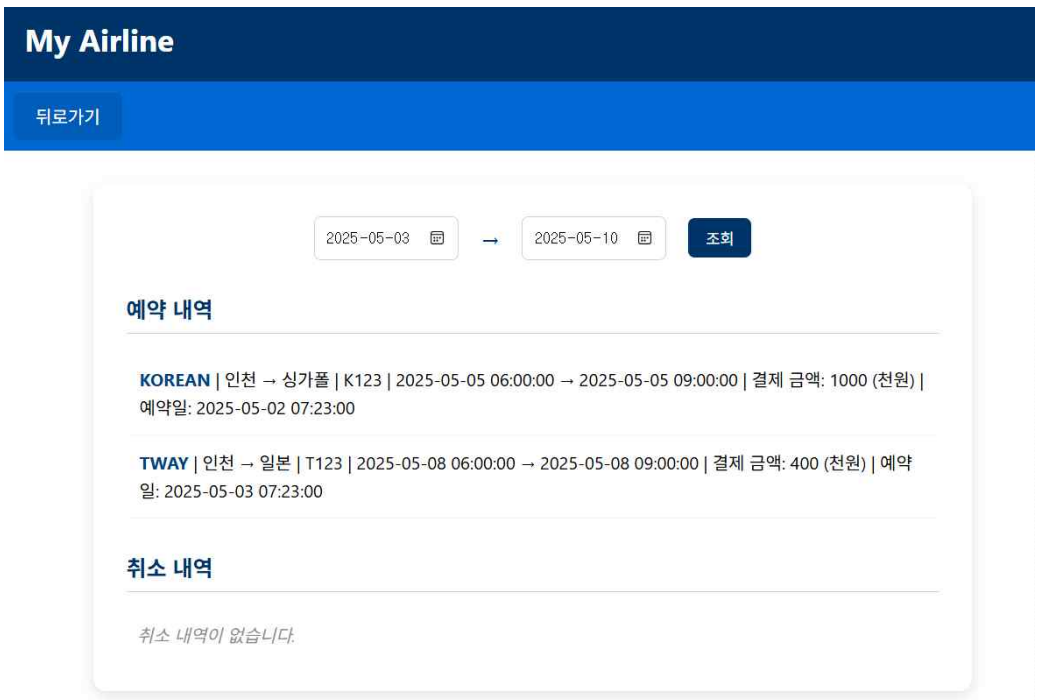
나. 기능 설명: 로그인된 세션의 회원을 기준으로, 예약된 항공권을 보여주는 화면으로, 각 항공권 별 {출발->도착 공항, 출발->도착 시간, 요금(천원)} 정보를 보여준다. 예약 취소 버튼을 클릭하면 해당하는 항공권의 예약 내용이 취소된다. 항공권 예약 취소 시 **전액 환불** 처리되며 Cancel 테이블에 이력이 저장된다. 항공권 예약 버튼을 누르면 항공권 예약 화면으로, 뒤로 가기 버튼을 누르면 메인 화면으로 렌더링된다.

6. 나의 예약/취소 내역 화면

가. 초기 화면



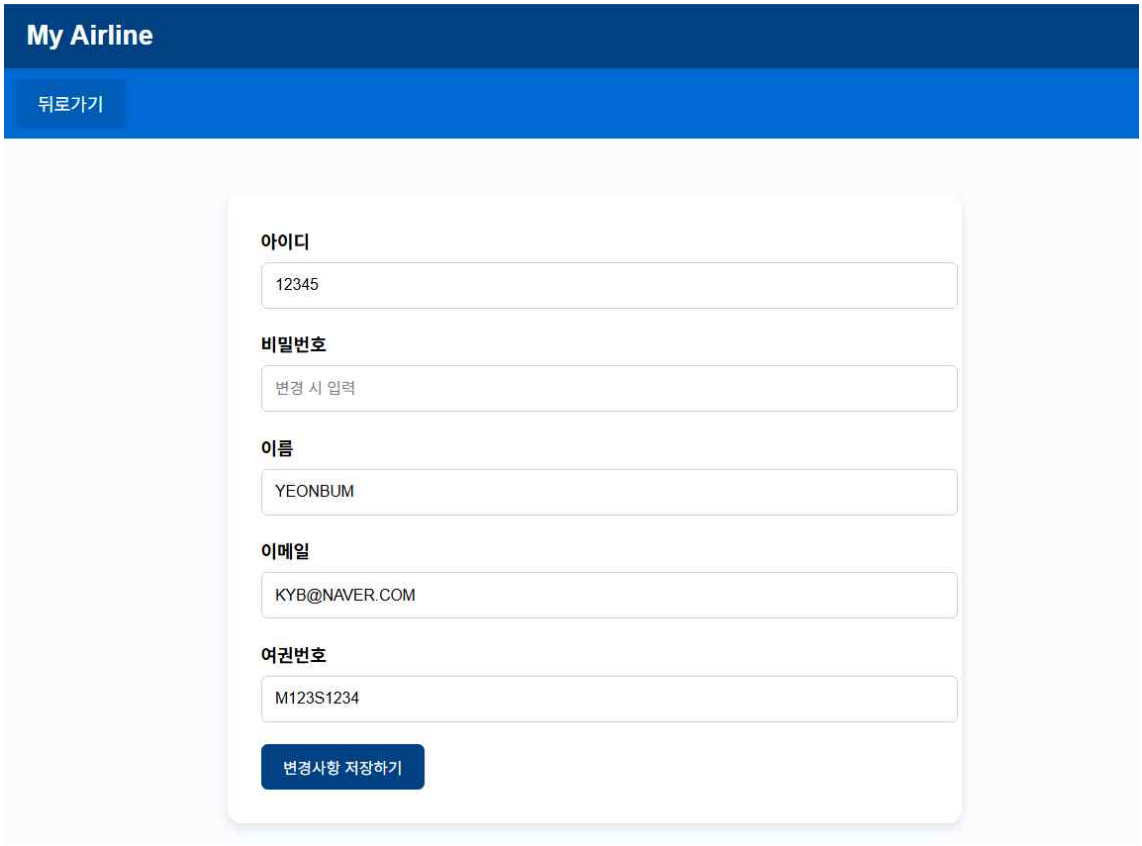
나. 조회 버튼 클릭 화면



다. 기능 설명: 기간 설정 후 조회 버튼을 누르면 해당하는 기간 내에 출발하는 항공권의 예약/취소 내역을 보여주는 화면이다. 뒤로가기 버튼을 누르면 메인 화면으로 렌더링 된다.

7. 회원 정보 관리 화면

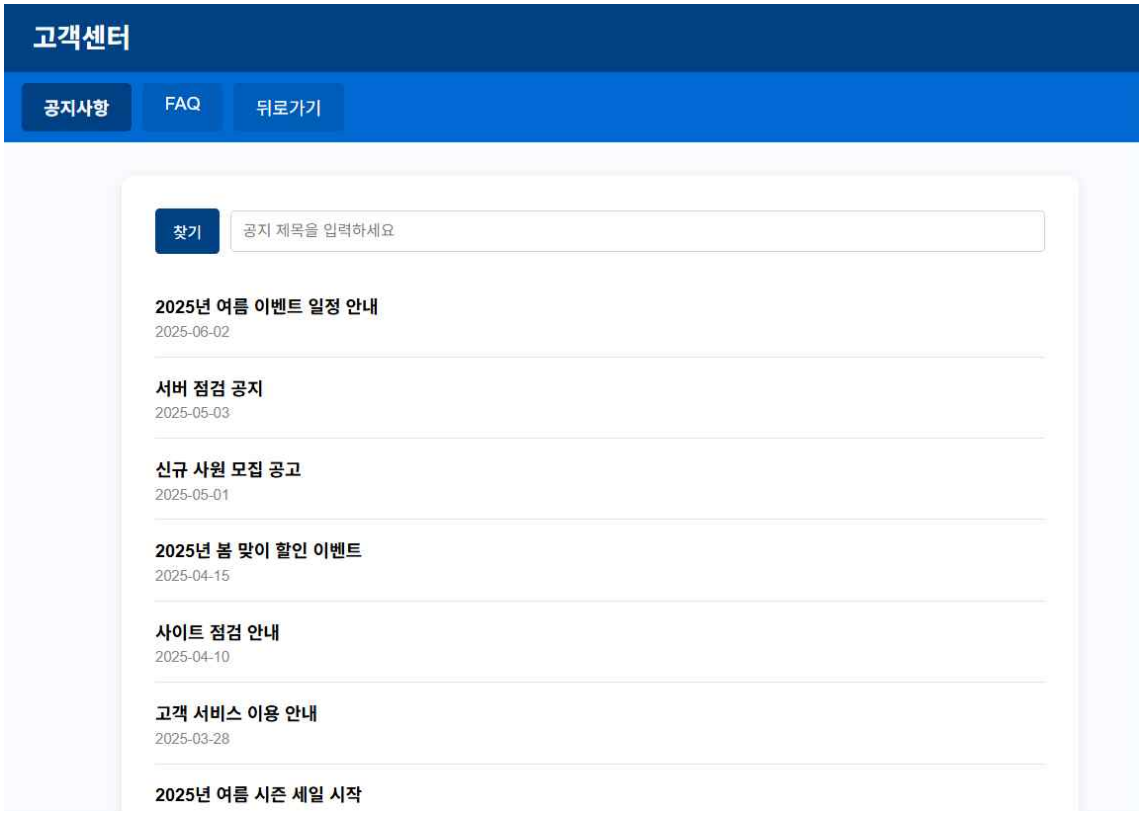
가. 초기 화면



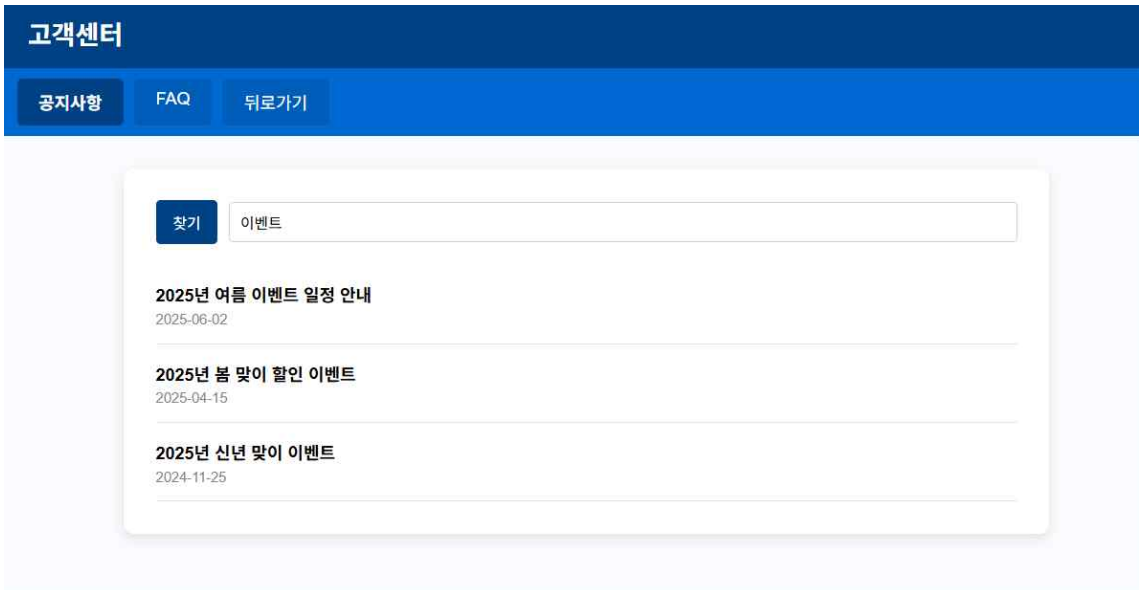
나. 기능 설명: 현재 로그인된 회원의 {아이디, 이름, 이메일, 여권번호}를 알려주는 화면으로, 새로 설정할 {아이디, 비밀번호, 이름, 이메일, 여권번호}를 입력하고 변경사항 저장하기 버튼을 누르면, 해당하는 회원(로그인된 세션의 CNO 기준)의 데이터를 업데이트한다. 뒤로가기 버튼을 누르면 메인 화면으로 렌더링된다.

8. 고객센터 화면

가. 초기 화면



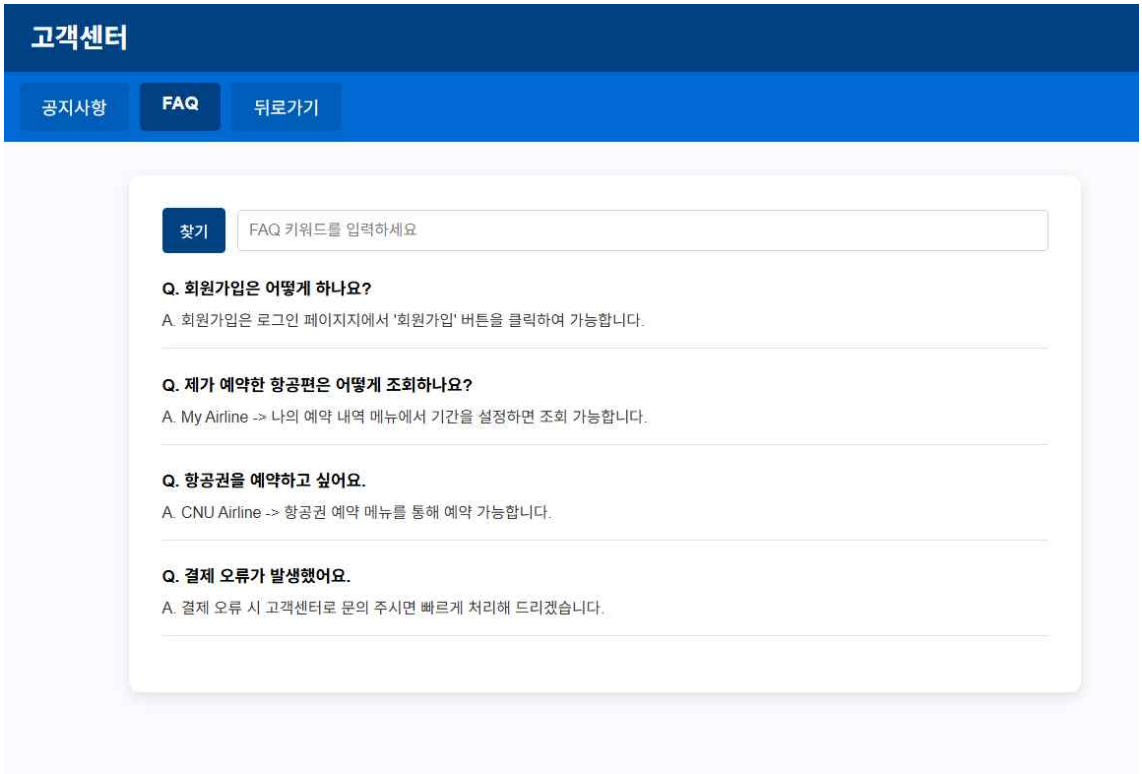
나. 찾기 버튼 클릭 화면



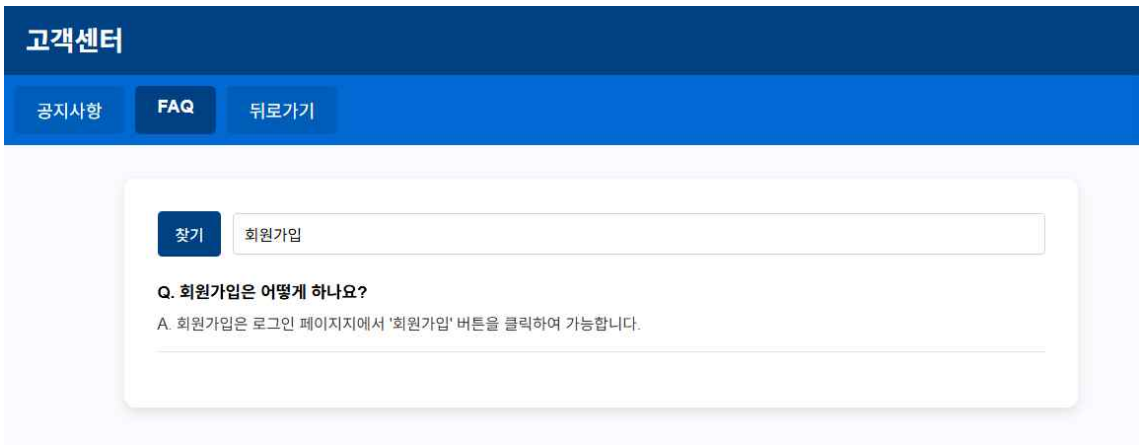
다. 기능 설명: 공지사항을 개제 순서대로 보여주는 화면으로, 키워드 입력 후 찾기 버튼을 클릭하면, 해당 키워드가 존재하는 제목의 공지사항만 보여준다. FAQ 버튼 클릭 시 FAQ 화면으로, 뒤로가기 버튼 클릭 시 메인 화면으로 렌더링된다.

9. FAQ 화면

가. 초기 화면



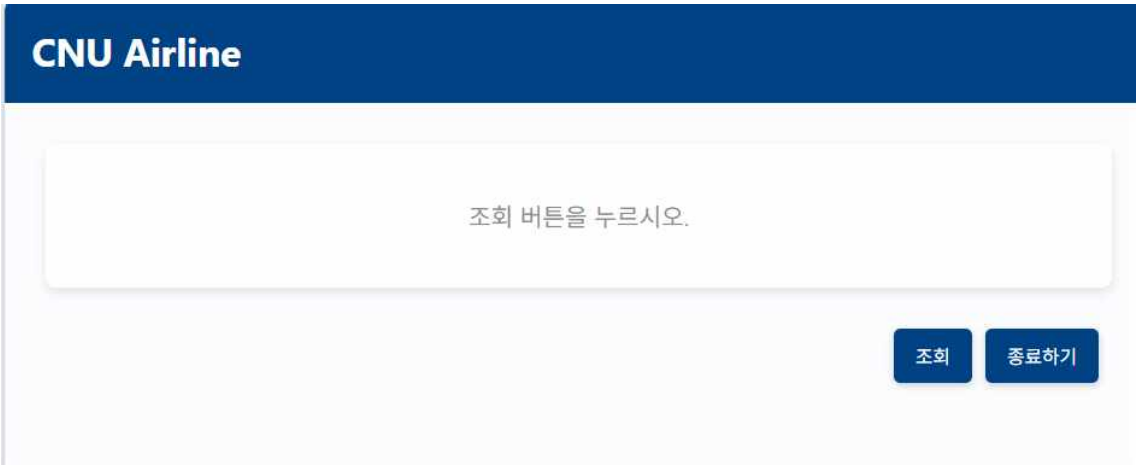
나. 찾기 버튼 클릭 화면



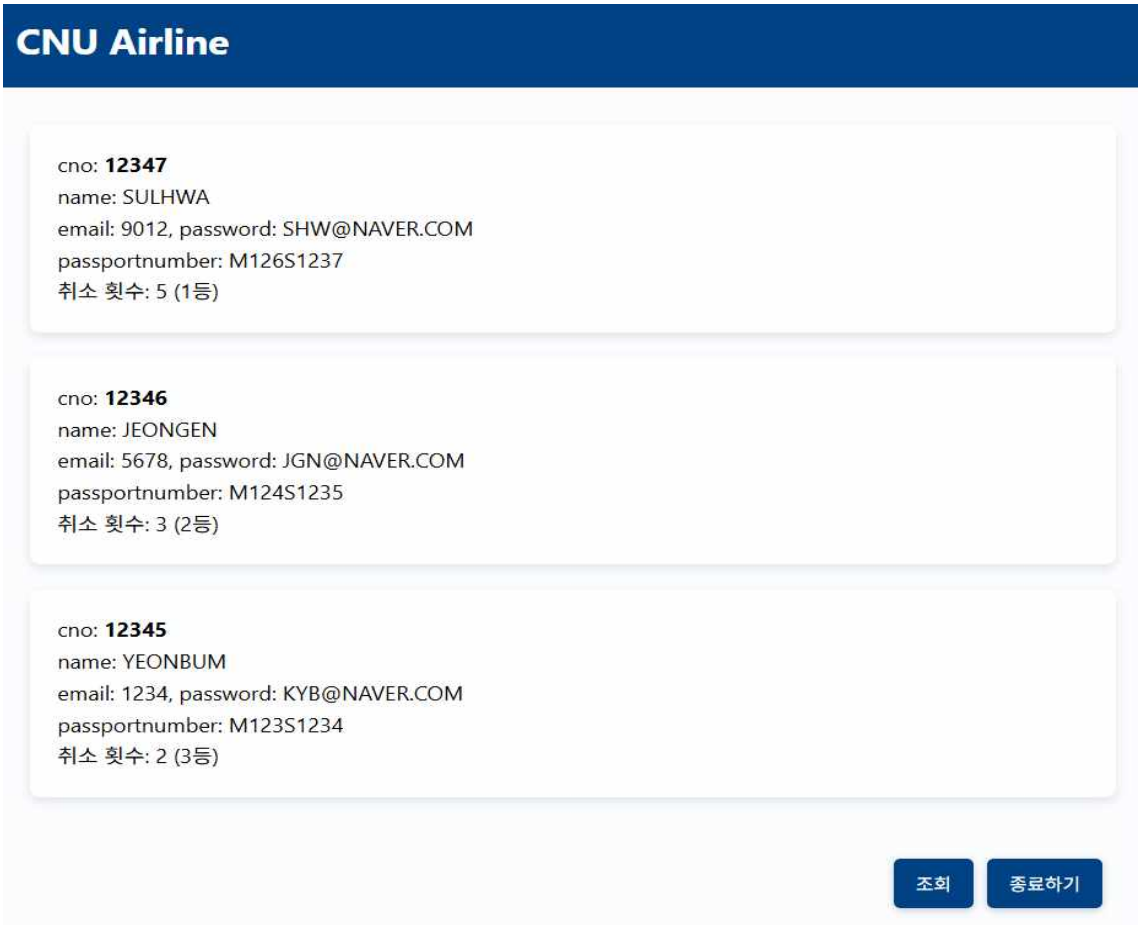
다. 기능 설명: FAQ를 보여주는 화면으로, 키워드 입력 후 찾기 버튼을 클릭하면, 해당 키워드가 속한 FAQ를 보여준다. 공지사항 버튼 클릭 시 공지사항 화면으로, 뒤로가기 버튼 클릭 시 메인 화면으로 렌더링된다.

10. 관리자 화면

가. 초기 화면



나. 조회 버튼 클릭 화면



다. 기능 설명: 관리자가 로그인했을 때 볼 수 있는 화면으로, 조회 버튼을 누르면 항공권 취소 이력이 있는 고객의 모든 정보와 취소 횟수, 이를 기반으로 하는 등수를 확인할 수 있다. 종료하기 버튼을 누르면 로그인 화면으로 렌더링된다.

7-4. 소스 코드

1. 소드 코드 전체 구성

본 소스코드는 Python의 Flask 프레임워크를 활용해 구현된 웹 기반 항공권 예약 시스템으로, 전체 구성은 사용자 인증, 항공권 예약 및 조회, 예약 취소, 마이페이지 기능 등을 포함한 전형적인 CRUD 기반 웹 애플리케이션 구조를 따르고 있다.

가. 기본 구성 요소

- Flask: 경량 웹 프레임워크로 HTTP 요청 처리 및 라우팅 담당
- oracledb: Oracle DB와의 연결 및 SQL 실행
- render_template: HTML 템플릿과 연동하여 동적 페이지 생성
- session: 사용자 로그인 세션 관리
- flash: 사용자에게 알림 메시지 전달

나. 사용자 인증 관련 기능

- / (GET): 로그인 페이지 렌더링
- /login (GET\POST): 사용자 로그인 처리. id/password를 Oracle DB의 CUSTOMER 테이블과 대조하여 검증함. 관리자 id/password일 경우 관리자 화면으로 렌더링함.
- /signup (GET\POST): 사용자 회원가입 처리. 회원 정보를 입력으로 받아 DB에 저장
- 로그인 성공 시 session을 통해 사용자 정보(cno, name)를 저장해 인증 상태 유지

다. 항공권 조회 및 예약 기능

- /ticket_reservation (GET): 항공권 검색 초기 화면 렌더링
- /ticket_find (GET): 사용자가 입력한 출발지, 도착지, 날짜, 좌석 등급, 정렬 기준으로 항공편 조회. 각 항공편 별 남은 좌석 수는 SEATS 테이블과 RESERVE 테이블을 근거로 계산함
- /reserve (POST): 항공권 예약 요청 처리. RESERVE 테이블에 예약 정보를 삽입하고 DB에 반영 및 사용자에게 티켓을 이메일로 전송. 남은 좌석 수를 근거로 예외 처리 적용

라. 예약 내역 확인 및 취소 기능

- /reservation_lookup (GET): 로그인한 사용자의 항공권 예약 내역 조회. RESERVE, AIRPLANE 테이블을 조인하여 출력함
- /cancel_reservation (POST): 사용자의 예약 취소 처리. 예약 조건을 만족하는 RESERVE 레코드 삭제 및 삭제 이력 레코드를 CANCEL 테이블에 저장

마. 사용자 마이페이지 기능

- /my_history (GET): 사용자 마이페이지 초기 화면 렌더링
- /get_my_history (GET): 기간을 기준으로 로그인 사용자의 예약 기록을 조회하여 JSON 반환

바. 관리자 통계 질의 기능

- /load_customer (POST): 취소 이력이 있는 고객의 모든 정보와 취소 횟수, 취소 횟수 기반 등수를 조회/계산하여 JSON 반환

사. 기타 데이터 구성

- notices: 공지사항 데이터는 리스트 형태로 코드 상단에 직접 정의
- faqs: FAQ(자주 묻는 질문)도 코드 상단에 하드코딩
- ADMIN_CNO, ADMIN_PASSWORD: 관리자 정보는 코드 상단에 하드코딩

2. 소스 코드

가. 라이브러리 및 초기 하드코딩된 데이터들

```python
from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify
import oracledb
from datetime import datetime
from email.mime.text import MIMEText
import smtplib

app = Flask(__name__)
app.secret_key = '1234kyb'

# Oracle DB 연결
connection = oracledb.connect(user="d202000826", password="21-76018500KYB", dsn="localhost:1521/xeXDB")

# 공지사항 데이터
notices = [
    {"title": "2025년 여름 이벤트 일정 안내", "date": "2025-06-02"},
    {"title": "서버 점검 공지", "date": "2025-05-03"},
    {"title": "신규 사원 모집 공고", "date": "2025-05-01"},
    {"title": "2025년 봄 맞이 할인 이벤트", "date": "2025-04-15"},
    {"title": "사이트 점검 안내", "date": "2025-04-10"},
    {"title": "고객 서비스 이용 안내", "date": "2025-03-28"},
    {"title": "2025년 여름 시즌 세일 시작", "date": "2025-03-22"},
    {"title": "신규 기능 업데이트", "date": "2025-03-10"},
    {"title": "고객센터 휴무 안내", "date": "2025-02-25"},
    {"title": "2025년 첫 결제 프로모션", "date": "2025-02-15"},
    {"title": "서버 확장 완료 안내", "date": "2025-02-01"},
    {"title": "2025년 고객 만족도 조사", "date": "2025-01-20"},
    {"title": "사내 워크샵 공지", "date": "2025-01-10"},
    {"title": "신규 상품 출시 안내", "date": "2025-01-01"},
    {"title": "2024년 결산 보고", "date": "2024-12-25"},
    {"title": "2024년 연말 행사 안내", "date": "2024-12-15"},
    {"title": "계정 보안 강화 안내", "date": "2024-12-01"},
    {"title": "2025년 신년 맞이 이벤트", "date": "2024-11-25"},
    {"title": "고객 개인정보 보호 공지", "date": "2024-11-10"},
    {"title": "서비스 이용약관 변경 안내", "date": "2024-10-30"}
]

# FAQ 데이터
faqs = [
    {"question": "회원가입은 어떻게 하나요?", "answer": "회원가입은 로그인 페이지지에서 '회원가입' 버튼을 클릭하여 가능합니다."},
    {"question": "제가 예약한 항공편은 어떻게 조회하나요?", "answer": "My Airline -> 나의 예약 내역 메뉴에서 기간을 설정하면 조회 가능합니다."},
    {"question": "항공권을 예약하고 싶어요.", "answer": "CNU Airline -> 항공권 예약 메뉴를 통해 예약 가능합니다."},
    {"question": "결제 오류가 발생했어요.", "answer": "결제 오류 시 고객센터로 문의 주시면 빠르게 처리해 드리겠습니다."}
]

# 관리자 계정
ADMIN_CNO = 'admin'
ADMIN_PASSWORD = '1234'
```

나. 로그인 페이지 렌더링 및 로그인 처리

```python
# 홈 페이지 렌더링 (기본적으로 로그인 페이지로 연결)
@app.route('/')
def home():
    return render_template('login.html')

# 로그인 처리 라우팅
@app.route('/login', methods=['POST', 'GET'])
def login():
    if request.method == 'GET':
        return render_template('login.html')

    # 로그인 요청 처리 (POST)
    cno = request.form['id']
    password = request.form['password']

    cursor = connection.cursor()
    cursor.execute("SELECT * FROM CUSTOMER WHERE cno=:1 AND passwd=:2", [cno, password])
    user = cursor.fetchone()

    if user:
        # 로그인 성공 -> 세션 저장 후 메인 화면(대시보드)으로 이동
        session['user'] = user[1]
        session['cno'] = user[0]
        return redirect(url_for('dashboard'))
    else:
        # 로그인 실패 시 오류 메시지 전달
        return render_template('login.html', error='로그인 정보가 올바르지 않습니다.')
```

다. 회원가입 페이지 및 로그인 화면 렌더링

```python
# 회원가입 처리 라우팅
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        # 입력 정보 수집
        username = request.form['username']
        name = request.form['name']
        password = request.form['password']
        email = request.form['email']
        passport = request.form['passport']

        # DB에 신규 고객 정보 삽입
        cursor = connection.cursor()
        cursor.execute(
            """
            INSERT INTO CUSTOMER
            VALUES (:1, :2, :3, :4, :5)
            """, [username, name, password, email, passport]
        )
        connection.commit()

        flash("회원가입이 완료되었습니다!", "success")

    return render_template('signup.html')

# 로그인 후 메인 페이지(대시보드)
@app.route('/main')
def dashboard():
    if 'user' not in session:
        return redirect(url_for('home'))
    return render_template('main.html', user=session['user'])
```

라. 항공권 예약 페이지 렌더링 및 항공권 조회 처리

```
# 항공권 예약 화면
@app.route('/ticket_reservation')
def ticket_reservation():
    return render_template('ticket_reservation.html', tickets=[])

# 항공권 조회 처리
@app.route('/ticket_find')
def ticket_find():
    # 사용자가 입력한 검색 조건 수집
    departure = request.args.get('departure')
    arrival = request.args.get('arrival')
    departure_date = request.args.get('departure_date')
    seat_class = request.args.get('seat_class')

    cursor = connection.cursor()

    # 정렬 기준 선택
    sort_by = request.args.get("sort_by")

    if sort_by == "time":
        # 항공편 및 좌석 정보 조회 쿼리
        query = """
        SELECT A.flightno, A.departureairport, A.arrivalairport,
               TO_CHAR(A.departuredatetime, 'YYYY-MM-DD HH24:MI') as departure_time,
               TO_CHAR(A.arrivaldatetime, 'YYYY-MM-DD HH24:MI') as arrival_time,
               S.seatclass, S.price, S.no_of_seats
            FROM AIRPLANE A
            JOIN SEATS S ON A.flightno = S.flightno
            AND A.departuredatetime = S.departuredatetime
            WHERE A.departureairport = :1
            AND A.arrivalairport = :2
            AND A.departuredatetime BETWEEN TO_DATE(:3, 'YYYY-MM-DD') AND TO_DATE(:4, 'YYYY-MM-DD') + 1
            AND S.seatclass = UPPER(:5)
            ORDER BY S.departuredatetime ASC
        """
```

```
    elif sort_by == "price":
        # 항공편 및 좌석 정보 조회 쿼리
        query = """
        SELECT A.flightno, A.departureairport, A.arrivalairport,
               TO_CHAR(A.departuredatetime, 'YYYY-MM-DD HH24:MI') as departure_time,
               TO_CHAR(A.arrivaldatetime, 'YYYY-MM-DD HH24:MI') as arrival_time,
               S.seatclass, S.price, S.no_of_seats
            FROM AIRPLANE A
            JOIN SEATS S ON A.flightno = S.flightno
            AND A.departuredatetime = S.departuredatetime
            WHERE A.departureairport = :1
            AND A.arrivalairport = :2
            AND A.departuredatetime BETWEEN TO_DATE(:3, 'YYYY-MM-DD') AND TO_DATE(:4, 'YYYY-MM-DD') + 1
            AND S.seatclass = UPPER(:5)
            ORDER BY S.price ASC
        """
    else: query = ""
    cursor.execute(query, [departure, arrival, departure_date, departure_date, seat_class])
    rows = cursor.fetchall()

    # 결과를 리스트로 정리하여 템플릿에 전달
    tickets = []
    for row in rows:
        tickets.append({
            'flight_no': row[0],
            'departure': row[1],
            'arrival': row[2],
            'departure_time': row[3],
            'arrival_time': row[4],
            'seat_class': row[5],
            'price': row[6],
            'no_seats': row[7]
        })

    for ticket in tickets:
        query = """
            SELECT COUNT(*)
            FROM RESERVE
            WHERE flightno = :1
            AND departuredatetime = :2
        """
        cursor.execute(query, [(ticket['flight_no']),
                               datetime.strptime(ticket['departure_time'], "%Y-%m-%d %H:%M")])
        number = cursor.fetchone()
        ticket['no_seats'] = int(ticket['no_seats']) - int(number[0])

    return render_template('ticket_reservation.html', tickets=tickets)
```

마. 항공권 예약 내역 조회 처리

```python
# 예약 내역 조회
@app.route('/reservation_lookup')
def reservation_lookup():
    if 'user' not in session:
        return redirect(url_for('home'))

    cursor = connection.cursor()
    cursor.execute("""
        SELECT a.airline, r.flightno, a.departureairport, a.arrivalairport,
               r.departuredatetime, a.arrivaldatetime, r.payment
        FROM RESERVE r, AIRPLANE a
        WHERE r.CNO = :1
        AND r.flightno = a.flightno
        AND r.departuredatetime = a.departuredatetime
    """, [session['cno']])
    rows = cursor.fetchall()

    reservations = [
        {
            'airline': row[0],
            'flight_number': row[1],
            'departure_airport': row[2],
            'arrival_airport': row[3],
            'departure_time': row[4],
            'arrival_time': row[5],
            'fare': row[6]
        }
        for row in rows
    ]

    return render_template('reservation_lookup.html', reservations=reservations)
```

바. 항공권 예약 처리

```python
# 항공권 예약 처리
@app.route('/reserve', methods=['POST'])
def reserve():
    if "user" not in session:
        return redirect(url_for('home'))

    cursor = connection.cursor()

    # 예약 정보 수집
    flightno = request.form['flightno']
    departuredatetime_str = request.form['departuredatetime']
    seatclass = request.form['seatclass']
    payment = request.form['payment']
    reservedatetime = datetime.now()

    # 현재 로그인된 사용자의 CNO, email 확인
    cursor.execute("SELECT cno, email FROM CUSTOMER WHERE name=:1", [session['user']])
    row = cursor.fetchone()
    if not row:
        flash("회원 정보를 찾을 수 없습니다.")
        return redirect("/ticket_reservation")
    cno = row[0]
    email = row[1]

    # 출발 일시 문자열을 datetime 객체로 변경
    departuredatetime = datetime.strptime(departuredatetime_str, "%Y-%m-%d %H:%M")

    cursor.execute("""
                   INSERT INTO RESERVE VALUES (:1, :2, :3, :4, :5, :6)
                   """, [flightno, departuredatetime, seatclass, payment, reservedatetime, cno])

    connection.commit()

    flash("예약이 완료되었습니다.")

    # 탑승권 내용 작성
    ticket_content = f"""
    항공편 예약이 완료되었습니다!

    ▶ 항공편 번호: {flightno}
    ▶ 출발 일시: {departuredatetime}
    ▶ 좌석 등급: {seatclass}
    ▶ 결제 금액: {payment}원

    감사합니다.
    """
    if email:
        send_email(email, "탑승권 예약 완료", ticket_content)

    return redirect("/reservation_lookup")
```

사. 이메일 전송 매커니즘 함수

```python
# 이메일 전송 메커니즘 함수
def send_email(to_email, subject, body):
    # 전송자 정보 하드코딩
    from_email = "kyb8732@gmail.com"
    from_password = [redacted]

    msg = MIMEText(body)
    msg['Subject'] = subject
    msg['From'] = from_email
    msg['To'] = to_email

    try:
        with smtplib.SMTP_SSL('smtp.gmail.com', 465) as smtp:
            smtp.login(from_email, from_password)
            smtp.send_message(msg)
        print("이메일 전송 완료")
    except Exception as e:
        print("이메일 전송 실패: ", e)
```

아. 예약 취소 처리 및 사용자 히스토리 페이지 렌더링

```python
# 예약 취소 처리
@app.route('/cancel_reservation', methods=['POST'])
def cancel_reservation():
    if 'user' not in session:
        return redirect(url_for('home'))

    flight_number = request.form['flight_number']
    departure_time = request.form['departure_time']
    try:
        cursor = connection.cursor()
        # Cancel 테이블 업데이트를 위한 취소 대상 항공권 정보 불러오기
        cursor.execute("""
            SELECT SEATCLASS, PAYMENT
            FROM RESERVE
            WHERE cno = (
                SELECT cno FROM CUSTOMER WHERE name = :1
            ) AND flightno = :2 AND departuredatetime = TO_DATE(:3, 'YYYY-MM-DD HH24:MI:SS')
        """, [session['user'], flight_number, departure_time])

        row = cursor.fetchone()
        seatclass = row[0]
        payment = row[1] # refund 금액은 무조건 전액 환불로 설정!
        canceldatetime = datetime.now()

        # 해당 예약 정보 삭제
        cursor.execute("""
            DELETE FROM RESERVE
            WHERE cno = (
                SELECT cno FROM CUSTOMER WHERE name = :1
            ) AND flightno = :2 AND departuredatetime = TO_DATE(:3, 'YYYY-MM-DD HH24:MI:SS')
        """, [session['user'], flight_number, departure_time])
        connection.commit()

        # 출발 일시 문자열을 datetime 객체로 변경
        departuredatetime = datetime.strptime(departure_time, "%Y-%m-%d %H:%M:%S")

        # Cancel 테이블에 해당 취소 내역 저장
        cursor.execute("""
            INSERT INTO CANCEL
            VALUES (:1, :2, :3, :4, :5, :6)
            """, [flight_number, departuredatetime, seatclass, payment, canceldatetime, session['cno']])
        connection.commit()
        flash("예약이 성공적으로 취소되었습니다.", "success")
    except Exception as e:
        print("예약 취소 오류:", e)
        flash("예약 취소 중 오류가 발생했습니다.", "error")

    return redirect(url_for('reservation_lookup'))

# 사용자 히스토리 조회 페이지
@app.route("/my_history")
def my_history():
    if 'user' not in session:
        return redirect(url_for('home'))
    return render_template('my_history.html', user=session['user'])
```

자. 이스토리 데이터 반환 처리

```python
# 히스토리 데이터 반환
@app.route("/get_my_history")
def get_my_history():
    if 'user' not in session:
        return redirect(url_for('home'))

    # 기간 필터 수집
    start_date = request.args.get('start_date')
    end_date = request.args.get('end_date')

    cursor = connection.cursor()
    # 항공권 예약 내역 조회
    cursor.execute("""
        SELECT a.airline,
        a.departureairport,
        a.arrivalairport,
        a.flightno,
        a.departuredatetime,
        a.arrivaldatetime,
        b.payment,
        b.reservedatetime
        FROM AIRPLANE a
        JOIN RESERVE b
        ON a.flightno = b.flightno
        AND a.departuredatetime = b.departuredatetime
        WHERE a.departuredatetime BETWEEN TO_DATE(:1, 'YYYY-MM-DD HH24:MI:SS')
                                      AND TO_DATE(:2, 'YYYY-MM-DD HH24:MI:SS') + 1
        ORDER BY  a.departuredatetime ASC
                """, [start_date, end_date])
    rows = cursor.fetchall()

    reservations = [
        {
            'airline': row[0],
            'departure': row[1],
            'arrival': row[2],
            'flight_no': row[3],
            'departure_time': row[4],
            'arrival_time': row[5],
            'payment': row[6],
            'reserved_date': row[7]
        }
        for row in rows
    ]
```

```python
cursor.execute("""
    SELECT a.airline,
    a.departureairport,
    a.arrivalairport,
    a.flightno,
    a.departuredatetime,
    a.arrivaldatetime,
    b.refund,
    b.canceldatetime
    FROM AIRPLANE a
    JOIN CANCEL b
    ON a.flightno = b.flightno
    AND a.departuredatetime = b.departuredatetime
    WHERE a.departuredatetime BETWEEN TO_DATE(:1, 'YYYY-MM-DD HH24:MI:SS')
                                  AND TO_DATE(:2, 'YYYY-MM-DD HH24:MI:SS')+1
    ORDER BY  a.departuredatetime ASC
            """, [start_date, end_date])

rows = cursor.fetchall()
cancellations = [
    {
        'airline': row[0],
        'departure': row[1],
        'arrival': row[2],
        'flight_no': row[3],
        'departure_time': row[4],
        'arrival_time': row[5],
        'refund': row[6],
        'cancelled_date': row[7]
    }
    for row in rows
]
# JSON 형태로 데이터 탬플릿에 전달
return render_template('my_history.html', reservations=reservations, cancellations=cancellations)
```

차. 고객 정보 렌더링 및 공지사항/FAQ 페이지 렌더링

```python
# 세션에 로그인된 고객 정보
@app.route("/customer")
def customer_info():
    if 'user' not in session:
        return redirect(url_for('home'))
    return render_template('customer.html', user=session['user'])

# 공지사항
@app.route("/notice")
def notice():
    if 'user' not in session:
        return redirect(url_for('home'))
    return render_template('notice.html', user=session['user'])

# FAQ
@app.route("/faq")
def faq():
    if 'user' not in session:
        return redirect(url_for('home'))
    return render_template('faq.html', user=session['user'])
```

카. 고객 초기 정보 렌더링

```python
# 세션에 로그인된 고객 정보 -> 초기 화면
@app.route("/update_profile")
def update_profile():
    if 'user' not in session:
        return redirect(url_for('home'))

    cursor = connection.cursor()
    cursor.execute(
        """
        SELECT cno, name, passwd, email, passportnumber
        FROM CUSTOMER
        WHERE cno =:1
        """, [session['cno']]
    )

    row = cursor.fetchone()

    user_info = {
        'userid': row[0],
        'name': row[1],
        'email': row[3],
        'passport': row[4]
    }
    return user_info
```

타. 고객 정보 업데이트 처리

```
# 세션에 로그인된 고객 정보 업데이트 -> 업데이트 버튼 클릭 시
@app.route("/update_customer", methods=['POST'])
def update_customer():
    if 'user' not in session:
        return redirect(url_for('home'))

    user_id = request.form['userid']
    password = request.form['password']
    name = request.form['name']
    email = request.form['email']
    passport = request.form['passport']
    print(user_id, password, name, email, passport)
    try:
        cursor = connection.cursor()
        cursor.execute(
            """
            UPDATE CUSTOMER
            SET CNO =:1, NAME =:2, PASSWD =:3, EMAIL =:4, PASSPORTNUMBER =:5
            WHERE CNO =:6
            """, [user_id, name, password, email, passport, session['cno']]
        )
        connection.commit()
        flash("고객 정보가 성공적으로 변경됐습니다.", "success")
    except Exception as e:
        print("고객 정보 변경 오류:", e)
        flash("고객 정보 변경경 중 오류가 발생했습니다.", "error")

    return render_template('customer.html', user=session['user'])
```

파. 키워드 기반 공지사항/FAQ 검색 처리 및 코드 실행 처리

```
# 키워드 기반 공지사항 검색
@app.route("/get_notice")
def get_notice():
    keyword = request.args.get('keyword', '').lower()  # 입력된 키워드
    if keyword:
        filtered_notices = [notice for notice in notices if keyword in notice['title'].lower()]
    else:
        filtered_notices = notices
    return jsonify(filtered_notices)

# 키워드 기반 FAQ 검색
@app.route('/get_faq', methods=['GET'])
def get_faq():
    keyword = request.args.get('keyword', '').lower()  # 검색어를 소문자로 변환
    filtered_faqs = [faq for faq in faqs if keyword in faq['question'].lower() or keyword in faq['answer'].lower()]
    return jsonify(filtered_faqs)

if __name__ == '__main__':
    app.run(debug=True)
```

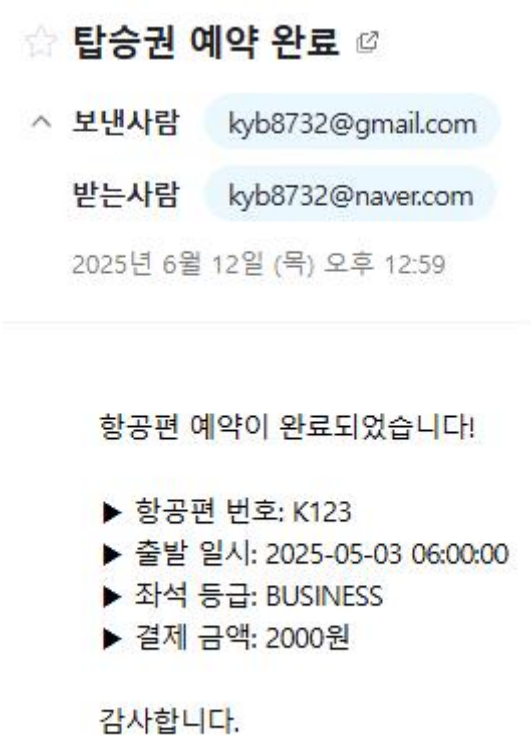
아. 관리자용 통계 질의 처리

```
# 관리자용 통계 질의
@app.route('/load_custoer', methods=['POST'])
def load_customer():
    cursor = connection.cursor()
    cursor.execute("""
    SELECT
        a.cno,
        b.name,
        b.passwd,
        b.email,
        b.passportnumber,
        a.cancel_count,
        RANK() OVER (ORDER BY cancel_count DESC) AS cancel_rank
    FROM (
        SELECT
            cno,
            COUNT(*) AS cancel_count
        FROM CANCEL
        GROUP BY CNO
    ) a, CUSTOMER b
    WHERE b.cno = a.cno
    ORDER BY cancel_rank
                    """,[])

    rows = cursor.fetchall()
    print(rows)
    customers = [
    {
        "cno": row[0],
        "name": row[1],
        "email": row[2],
        "password": row[3],
        "passportnumber": row[4],
        "count": row[5],
        "rank": row[6]
    }
    for row in rows
]


    return render_template('admin.html', customers=customers)
```

[별첨1] 이메일 형식


탑승권 예약 완료

보낸사람 kyb8732@gmail.com

받는사람 kyb8732@naver.com

2025년 6월 12일 (목) 오후 12:59

항공편 예약이 완료되었습니다!

▶ 항공편 번호: K123
▶ 출발 일시: 2025-05-03 06:00:00
▶ 좌석 등급: BUSINESS
▶ 결제 금액: 2000원

감사합니다.