캡스톤 디자인 I 최종결과 보고서

프로젝트 제목(국문): Talk2Travel: 여행 일정 에이전트
프로젝트 제목(영문): Talk2Travel: Travel Agent

프로젝트 팀(원): 학번: 20201047 이름: 구남석

1. 중간보고서의 검토결과 심사위원의 '수정 및 개선 의견'과 그러한 검토의견을 반영하여 개선한 부분을 명시하시오.

없음

2. 기능, 성능 및 품질 요구사항을 충족하기 위해 본 개발 프로젝트에서 적용한 주요 알고리즘, 설계방법 등을 기술하시오.

- **항공·숙소 Top-3 추천 로직**

  Amadeus에서 가져온 원본 오퍼 배열을 가격 기준으로 정렬한 뒤 상위 3개만 잘라내는 휴리스틱을 적용했다. 구체적으로 rank_flights_by_price()와 rank_hotels_by_price() 함수가 각각 sorted(offers, key=…)[:3] 패턴으로 O(n log n) 정렬을 수행하며, 가격 파싱에 실패한 항목은 float("inf")를 반환해 자동으로 목록 뒤로 밀린다. 캡스톤디자인 2에서는 동일 인터페이스에 ML re-ranking을 넣을 여지도 확보된다.

- **외부 API Service Layer 분리**

  Amadeus·Stripe 호출은 Flask 라우트와 분리된 flight_service.py, hotel_service.py 등 Service Layer 모듈에 감쌌다. 이 코드는 API 키나 버전이 바뀌더라도 모듈 내부 수정만으로 종속성을 격리할 수 있어 유지보수성을 높인다. 예를 들어 호텔 검색은 hotel_offers_search.get()를 래핑한 search_hotels() 함수가 담당하며, 오류 시 ResponseError를 로깅한 뒤 빈 리스트를 반환해 라우트 코드가 복잡한 예외 처리를 다시 하지 않도록 했다.

- **캘린더 UX – DOM Patch 방식**

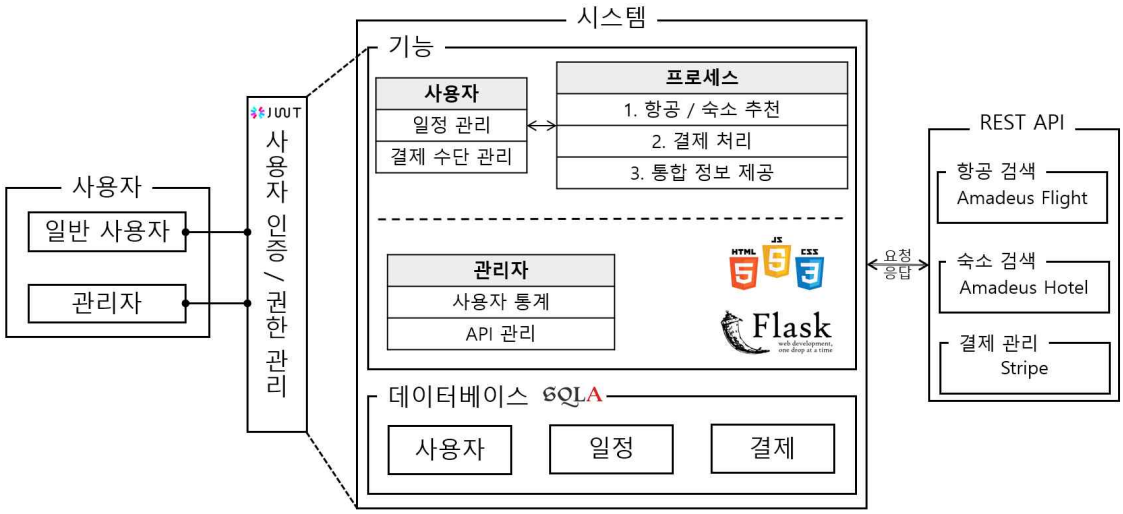
  일정 저장이 성공하면 서버가 돌려준 JSON을 그대로 FullCalendar의 addEvent()에 넘겨 새 이벤트 노드 하나만 DOM에 삽입한다. 덕분에 사용자는 페이지 새로고침 없이도 즉시 일정이 나타나는 경험을 얻는다. 이 방식을 DOM patch라고 부르는데, 전체 캘린더를 다시 렌더링하지 않고 변경된 부분(또는 새 이벤트 요소)만 DOM API로 추가해 레이아웃 재계산·리페인트를 최소화한다. 이러한 구현은 fullcalendar.js에서 calendar.addEvent({...}) 호출로 확인할 수 있다.

- **공항 자동완성 – IATA DB + ILIKE 와일드카드**

  사용자가 출발·도착 공항 입력란에 두 글자 이상을 타이핑하면 suggestAirports()가 /api/airports?city=키워드를 비동기로 호출한다. 백엔드 라우트는 Airport 테이블에서 city_en, name_en, name_ko 세 칼럼에 대해 ILIKE '%keyword%' 조건을 OR로 묶어 검색하고, 알파벳순으로 정렬해 최대 10개만 돌려준다. 클라이언트는 받은 리스트를 code – name 형태로 렌더링하고, 항목을 클릭하면 숨겨진 <input name="departure_airport">·arrival_airport 필드에 IATA code를 저장한다. 이 과정 전체가 airports_api() 라우트와 bindAuto() 함수에 구현돼 있다.

- 데이터 모델(미구현 항목)
- 결제(미구현 항목)
- 보안(미구현 항목)

3. 요구사항 정의서에 명세된 기능 및 품질 요구사항에 대하여 최종 완료된 결과를 기술하시오.



<그림 1. Talk2Travel 시스템 아키텍처>

<그림 1>은 Talk2Travel 전체 시스템을 기능별·구성 요소별로 나누어 보여주는 아키텍처 다이어그램이다. 왼쪽에는 사용자 인증 및 권한 관리 영역, 중앙에는 시스템 내부(비즈니스 로직)와 데이터베이스, 오른쪽에는 외부 연동 API가 배치되어 있다.

[사용자 인증 및 권한 관리]

사용자의 종류는 크게 일반 사용자(일정 관리, 결제 수단 관리 기능 사용)와 관리자(사용자 통계 조회, API 관리 기능 사용)로 구분된다.

이 두 사용자 그룹은 시스템 접근 시 JWT(JSON Web Token) 기반의 인증·인가 절차를 거친다. 즉, 사용자가 로그인하면 JWT를 발급받아 이후 모든 REST API 호출 시 헤더에 토큰을 담아 보냄으로써 자신의 신원을 증명하고, 서버는 해당 토큰을 검증해 권한을 확인한다.

[사용자 기능]

사용자(일반)는 주로 '일정 관리(Schedule Management)'와 '결제 수단 관리(Payment Method Management)' 기능을 이용한다. 프로세스(서비스 로직) 영역에는 다음 세 가지 주요 비즈니스 로직 모듈이 존재한다.

<첫 번째, 항공/숙소 추천 모듈>사용자가 일정 정보를 등록하면(Create Schedule → DB 저장), Flask 백엔드에서 amadeus SDK를 이용해 Amadeus Flight Offers Search API와 Hotel Offers Search API를 호출한다. 반환된 데이터는 recommendation.py 모듈에서 rank_flights_by_price, rank_hotels_by_price 함수를 통해 가격 기준 상위 N개(top-3)를 추출하고, simplify_flight, simplify_hotel 함수를 사용해 사용자에게 보여줄 최소한의 정보(한국어 키로 변환된 가격·항공사 코드·객실 설명 등)를 가공한다.

<두 번째, 결제 처리 모듈> 사용자가 특정 오퍼를 선택하면, 결제 모듈이 Stripe 결제 API를 호출하여 결제 트랜잭션을 생성하고, 성공 시 해당 거래 정보를 PaymentTransaction 테이블에 저장한다. Flask 백엔드(app.py) 내에서 Stripe SDK 혹은 REST API 호출 로직을 구현해, 토큰화된 결제 수단 ID, 금액을 전달하고, 결제 성공(또는 실패) 여부에 따라 사용자에게 결과를 푸시 알림 형태로 전달한다.

<통합 정보 제공> 일정 관리, 항공/숙소 추천, 결제 결과, 결제 수단 정보 등을 하나로 묶어 사

용자가 편리하게 조회할 수 있도록 REST API 레벨에서 여러 엔드포인트(/api/schedules, /api/airports, /api/payment-methods, /api/transactions 등)를 제공한다. 관리자 전용 API 또한 별도로 구현하여, 사용자 통계 분석(User Statistics), API 사용량 모니터링(API Management) 등을 지원한다.

[프론트엔드 구현]

사용자 인터페이스(UI)는 HTML5, CSS3, JavaScript, Bootstrap, FullCalendar 라이브러리를 활용하여 구현되어 있다. index.html에 FullCalendar 달력을 배치하고, 모달 폼은 Bootstrap 컴포넌트를 사용한다. 자동완성(Auto-complete)은 JavaScript(fetch API)를 통해 /api/airports?city=… 엔드포인트에서 반환된 데이터를 기반으로 <ul> 요소를 실시간으로 채워 넣는 방식으로 구현되어 있다.

[백엔드 구현]

Python의 Flask 프레임워크를 사용해 웹 서버를 구축하며, RESTful API(/api/schedules, /api/airports, /api/...)를 제공한다. SQLAlchemy를 이용해 데이터베이스 모델(User, Schedule, PaymentMethod, PaymentTransaction, Airport)과 테이블을 정의 및 매핑한다. Amadeus API(항공·숙소 검색) 호출, Stripe API(결제) 호출 등의 비즈니스 로직을 해당 서비스 모듈(flight_service.py, hotel_service.py, recommendation.py 등)에 분리하여 관리한다. 사용자 인증/인가(JWT) 로직도 백엔드에서 처리하며, 로그인 시 JWT를 발급하고 요청마다 토큰을 검증한다.

[데이터베이스 구현]

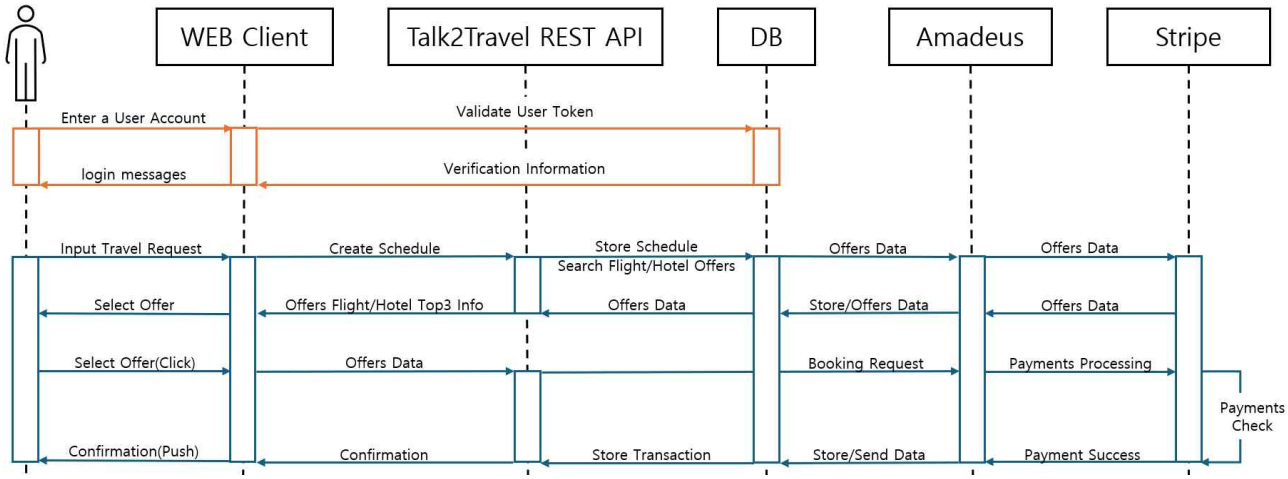
ERD에 정의된 대로 User, Schedule, PaymentMethod, PaymentTransaction 테이블을 갖는다.
Flask 애플리케이션이 구동될 때(db.create_all()), 로컬파일 기반 SQLite(sqlite:///schedules.db) 혹은 추후 MySQL/PostgreSQL로 변경이 가능한 설정을 통해 테이블을 생성하고 연결한다.
관리자는 관리자 콘솔을 통해 사용자 테이블, 일정 테이블, 결제 관련 테이블을 직접 조회 및 통계용 쿼리를 실행할 수 있다.

[외부 REST API 연동]

<Amadeus> 항공권/호텔 오퍼를 검색할 때 사용하며, config.py에 설정된 AMADEUS_CLIENT_ID, AMADEUS_CLIENT_SECRET을 통해 OAuth 인증을 받아 사용한다. 항공은 flight_service.py의 search_flights 함수, 숙소는 hotel_service.py의 get_hotel_ids 및 search_hotels 함수를 통해 호출된다.

<Stripe> 결제 수단 등록(tokenization), 결제 트랜잭션 생성, 결제 성공 여부 확인을 위해 사용한다. Flask 코드 내에서 Stripe SDK 혹은 HTTP 요청을 통해 토큰화된 카드 정보를 전달하고, 응답으로 반환된 PG사 거래 ID를 PaymentTransaction.transaction_id 필드에 저장한다.

이러한 구성도를 통해 Talk2Travel 시스템은 인증/인가, 일정 관리, 항공·숙소 추천, 결제 처리, 통합 정보 제공, 관리자 기능까지 end-to-end의 통합된 플로우로 구현되어 있으며, 프론트엔드와 백엔드, 데이터베이스, 외부 API가 명확히 분리되어 상호작용한다.



<그림 2. 사용자 시나리오 - 시퀀스 다이어그램>

<그림 2>는 Talk2Travel 시스템에서 사용자가 로그인부터 여행 일정 생성, 항공·숙소 검색, 결제까지 이루어지는 전 과정을 시점별로 표시한 그림이다.
왼쪽부터 순서대로, User(사용자), Web Client(클라이언트), Talk2Travel(Flask), DB(데이터베이스), Amadeus API, Stripe API가 배치되어 있으며, 각 컴포넌트 사이에 주고받는 메시지(화살표)와 순서를 통해 사용자 시나리오를 표현한다.

[로그인 절차]

1. 사용자가 웹 클라이언트에 계정 정보를 입력한다(Enter a User Account).
2. 웹 클라이언트는 입력된 토큰을 REST API에 전달하여 유효성을 검사해 달라고 요청한다(Validate User Token).
3. REST API는 DB에 접근하여 해당 토큰 또는 인증 정보를 확인한 뒤, 사용자 계정이 존재하고 올바른지 검사한다.
4. DB가 인증 결과(Verification Information)를 REST API로 반환하면, REST API는 로그인 성공/실패 여부와 관련된 메시지(login messages)를 웹 클라이언트에 돌려준다.
5. 웹 클라이언트는 이 응답에 따라 화면에 로그인 결과를 표시하거나 에러 메시지를 보여준다.

[여행 일정 생성 및 항공·숙소 정보 조회]

1. 로그인 후, 사용자는 웹 클라이언트 화면에서 여행 요청(출발·도착 공항, 날짜, 인원, 예산 등)을 입력하고 제출한다(Input Travel Request).
2. 웹 클라이언트는 이 정보를 REST API에 전송한다(Create Schedule). 내부적으로 JSON Body에 user_id, start_date, end_date, departure_airport, arrival_airport, passengers, budget 등의 필드를 포함한다.
3. REST API는 Schedule(Trip)을 DB에 메시지를 보내며 저장한다(Store Schedule). 이때 DB에는 고유 식별자(UUID), 사용자 ID, 출발일, 도착일, 출발 공항 코드, 도착 공항 코드, 인원 수, 예산, 생성 시각(created_at), 수정 시각(updated_at) 등이 삽입된다.
4. 저장이 완료되면 REST API는 Amadeus 항공·숙소 검색 API를 호출한다(Search Flight/Hotel Offers). 이때 Schedule 테이블에 들어간 출발지, 도착지, 날짜, 인원 수 등의 파라미터를 전달한다.

5. Amadeus 서버는 해당 요청에 대한 결과(항공 및 숙소 오퍼 데이터 리스트)를 반환한다(Offers Data).
6. REST API는 도착한 Offers 데이터를 받아서, 필요하다면 DB에 별도로 저장하거나(Store/Offers Data) 가공한 뒤, Recommendation 추천 로직 모듈을 통해 가격순 상위 3개(Top-3)를 추출하여 웹 클라이언트에 돌려준다(Offers Flight/Hotel Top3 Info).
7. 웹 클라이언트는 사용자가 보기 편하도록 항공·숙소 Top-3 정보를 화면에 표시하고, 사용자는 제시된 옵션 중 하나를 선택할 수 있다.

[옵션 선택 및 결제 처리]

1. 사용자가 웹 클라이언트에서 특정 항공/숙소 오퍼(=Top-3 중 하나)를 클릭하여 선택하면(Select Offer(Click)), 웹 클라이언트는 선택된 Offer 정보를 REST API에 다시 전송한다(Offers Data → REST API).
2. REST API는 이 오퍼 정보를 바탕으로 DB에 결제 및 예약 정보를 저장한다(Booking Request). 이때 Trip과 연결된 거래 내역을 생성하기 위한 초기 트랜잭션 레코드를 만들 수 있으며, payment_status를 ‘pending’으로 두고 결제 준비 상태로 표시한다.
3. DB가 저장을 마치면 REST API는 Stripe로 요청을 보낸다(Payments Processing). 이때 payment_method_id, 금액(amount), 통화(currency) 등의 정보를 넘겨서 실제 카드 결제(또는 기타 결제 수단) 트랜잭션을 시작한다.
4. Stripe는 결제 요청을 처리한 뒤 “Payment Success” 혹은 “Payment Failure” 상태를 REST API에 돌려준다.
5. REST API는 Stripe로부터 받은 결제 성공 정보를 바탕으로 DB의 PaymentTransaction 테이블에 메시지를 보내며(Store Transaction) 트랜잭션 레코드를 최종 업데이트(결제 상태(payment_status)를 ‘completed’로 변경, PG사 거래 ID(transaction_id) 저장 등)한다.
6. DB 저장이 끝나면 REST API는 웹 클라이언트에 메시지를 보내 사용자가 결제 완료 및 예약 확정 상태를 화면에서 확인하도록 한다.(Confirmation(Push))

<그림 3. Talk2Travel ERD>

<그림 3>은 Talk2Travel 시스템의 핵심 엔티티(테이블) 간 관계를 보여주는 ERD이다. 주요 테이블은 USER, PAYMENT_METHOD, TRIP, PAYMENT_TRANSACTION이며, 각 테이블의 컬럼과 제약조건(PK, FK, NOT NULL 등), 그리고 테이블 간 참조(1:N 관계)를 식별할 수 있다.

[관계]
- USER ← 1:N → PAYMENT_METHOD
- USER ← 1:N → TRIP
- TRIP ← 1:N → PAYMENT_TRANSACTION
- PAYMENT_METHOD ← 1:N → PAYMENT_TRANSACTION

[USER Table(회원 정보)]
- 사용자 식별번호(id, UUID, PK): 각 회원을 고유하게 구분하는 기본 키.
- 이메일(email, VARCHAR, NOT NULL, UNIQUE): 로그인 및 알림 수신용 이메일 주소.
- 암호화된 비밀번호(password_hash, VARCHAR, NOT NULL): bcrypt 등으로 해시된 비밀번호를 저장한다.
- 생성 시점(created_at, DATETIME, NOT NULL): 회원이 처음 생성된 시각.
- 수정 시점(updated_at, DATETIME, NOT NULL): 회원 정보가 마지막으로 업데이트된 시각.
- SQLAlchemy 모델(app.py) 상의 User 클래스가 바로 이 테이블과 매핑된다.

[TRIP Table(일정 정보)]
- 일정 식별번호(id, UUID, PK): 여행 일정 단위를 고유하게 식별.
- 사용자 식별번호(user_id, UUID, NOT NULL, FK → USER.id): 이 일정을 등록한 회원의 외래 키.
- 여행 시작일(start_date, DATE, NOT NULL)
- 여행 종료일(end_date, DATE, NOT NULL)
- 여행지(city, VARCHAR, NOT NULL): 사용자가 입력한 도착 도시(예: “KIX” 등 IATA 코드) 또는 자유 텍스트.
- 인원 수(passenger, INT, NULL): 여행에 동행할 인원 수(입력 선택).
- 예산(budget, DECIMAL, NULL): 사용자가 설정한 여행 예산(선택 입력).

- 생성 시점(created_at, DATETIME, NOT NULL)
- 수정 시점(updated_at, DATETIME, NOT NULL)
- SQLAlchemy 모델은 Schedule 클래스이며, 내부적으로 __tablename__ = "schedules"로 지정되어 있지만 ERD 상에서는 TRIP으로 명명되어 있다.

[PAYMENT_METHOD Table(결제 수단)]
- 결제 식별번호(id, UUID, PK): 각 결제 수단 레코드를 고유하게 나타내는 기본 키.
- 사용자 식별번호(user_id, UUID, NOT NULL, FK → USER.id): 이 결제 수단을 등록한 회원을 가리키는 외래 키.
- 결제 수단(type, VARCHAR, NOT NULL): 카드, 계좌이체 등 결제 유형을 나타내는 문자열.
- 결제 API 제공자(provider, VARCHAR, NOT NULL): Stripe 실제 토큰을 생성한 PG사를 나타냄.
- 토큰화된 결제 수단 식별자(token, VARCHAR, NOT NULL): 결제 API가 발급한 토큰(Stripe의 payment_method ID).
- 카드 뒷 4자리(card_last4, CHAR(4), NOT NULL): 사용자에게 표시해 줄 카드 번호 뒷자리.
- 결제 수단 유효기간(월)(exp_month, INT, NOT NULL)
- 결제 수단 유효기간(년)(exp_year, INT, NOT NULL)
- 생성 시점(created_at, DATETIME, NOT NULL)
- 수정 시점(updated_at, DATETIME, NOT NULL)
- 실제 SQLAlchemy 모델은 PaymentMethod 클래스이며, user = db.relationship("User", back_populates="payment_methods")로 USER 테이블과 연결되어 있다.

[PAYMENT_TRANSACTION Table(거래/결제 내역)]
- 결제 식별번호(id, UUID, PK): 각 결제 트랜잭션을 고유하게 식별하는 기본 키.
- 일정 식별번호(trip_id, UUID, NOT NULL, FK → TRIP.id): 이 결제가 연결된 여행 일정의 외래 키.
- 결제 수단 식별번호(payment_method_id, UUID, NOT NULL, FK → PAYMENT_METHOD.id): 이 거래에 사용된 결제 수단의 외래 키.
- 금액(amount, DECIMAL, NOT NULL): 결제 금액.
- 통화(currency, VARCHAR, NOT NULL): KRW, USD 등 통화 단위.
- 결제 상태(payment_status, VARCHAR, NOT NULL): pending, completed, failed 등 현재 결제 진행 상태.
- PG사 거래 ID(transaction_id, VARCHAR, NOT NULL): Stripe나 다른 결제 플랫폼에서 생성된 고유 거래 ID.
- 생성 시점(created_at, DATETIME, NOT NULL)
- 수정 시점(updated_at, DATETIME, NOT NULL)
- SQLAlchemy 모델은 PaymentTransaction 클래스이며, trip = db.relationship("Schedule", back_populates="transactions")와 payment_method = db.relationship("PaymentMethod", back_populates="transactions")를 통해 TRIP(=Schedule) 및 PAYMENT_METHOD 테이블과 연결된다.

[코드 상으로 존재하는 AIRPORT Table(공항 관련)]
- IATA 코드(iata_code, CHAR(3), PK)
- ICAO 코드(icao_code, CHAR(4), NULL)
- 공항명(영문)(name_en, VARCHAR, NOT NULL)
- 공항명(한글)(name_ko, VARCHAR, NULL)
- region, VARCHAR, NULL), 국가명(영문/한글), 도시명(city_en) 등
- 이 테이블은 자동완성 기능(/api/airports)을 위해 사용되며, FullCalendar 모달 폼에서 출발·도착 공항을 입력할 때 실시간으로 검색해주는 용도로 활용된다.

4. 구현하지 못한 기능 요구사항이 있다면 그 이유와 해결방안을 기술하시오,

최초 요구사항	구현 여부(미구현, 수정, 삭제 등)	이유(일정부족, 프로젝트 관리미비, 팀원변동, 기술적 문제 등)
캘린더 UI 화면 구현	구현	해당사항 없음
일정 관리: 사용자 입력 파싱	수정	토큰 호출 비용 - 지원금 없음(캡스톤 2에서 구현 예정)
일정 관리: 날짜 드래그 후, 여행 정보 입력	구현	해당사항 없음
지역 검색 시 공항 정보 자동완성 목록 조회	구현	해당사항 없음
일정 등록 시 데이터베이스에 저장	구현	해당사항 없음
Amadeus Flight API: 항공 정보 호출	구현	해당사항 없음
Amadeus Hotel API: 도시 호텔 정보 호출	구현	해당사항 없음
항공/호텔 정보 3건 추출	구현	해당사항 없음
상위 3건 기준: 최저가 (캡스톤2: 거리, 리뷰, 가격에 가중치 다르게 두어 추천)	구현	해당사항 없음
IATA CODE DB 저장	구현	해당사항 없음
일정-정보 추천 파이프라인 구축: 일정 등록 후 자동으로 항공/숙소 추천 3건 정보 제공	구현	해당사항 없음
추천 정보 요약 카드 구현	구현	해당사항 없음
항공/숙소 세부 내용 제공	미구현	일정부족(5월부터 주제 변경해서 시작) 구현 계획: Amadeus의 백엔드 엔드포인트를 추가(Flight/Hotel Booking API)하여 호출 결과에서 편명, 객실별 구성, 평점, 부대시설 정보등을 JSON으로 전달해 팝업 형태로 제공한다.
결제 정보 데이터베이스 암호화	미구현	일정부족(5월부터 주제 변경해서 시작) 구현 계획: 대칭키 방식을 사용 및 서버 환경 변수에 암호화 키를 보관하여 구현한다.
Amadeus Booking API, Stripe API 연결	미구현	일정부족(5월부터 주제 변경해서 시작) 구현 계획: Flight Order Create / HotelCreate Booking 엔드포인트를 추가하여 JSON 형태로 클라이언트에 반환한다.
일정-정보 추천 파이프라인과 연결	미구현	일정부족(5월부터 주제 변경해서 시작) 구현 계획: 모든 모듈이 구현된 경우, JSON 형태로 반환하여 다음 모듈을 수행하게 구현한다.

5. 요구사항을 충족시키지 못한 성능, 품질 요구사항이 있다면 그 이유와 해결방안을 기술하시오.

분류(성능, 속도 등) 및 최초 요구사항	충족 여부(현재 측정결과 제시)	이유(일정부족, 프로젝트 관리미비, 팀원변동, 기술적 문제 등)
API 호출 응답 속도 및 전체 응답 시간 2초 미만	측정하지 않음	일정 부족: 프로젝트 초기에 기능 구현에 집중하면서, 기능 구현 일정이 촉박해져 벤치마크 도구를 도입해 테스트를 돌릴 여유가 부족함.
사용자 계정 토큰화 및 결제 관련 메타데이터를 DB에 암호화하여 저장	미구현	일정 부족: 프로젝트 초기에 기능 구현에 집중하면서, 키 관리를 위한 암호화 라이브러리 설정 작업을 수행하는 데 여유가 부족함.
예외 처리 및 오류 검증	부분 구현	일정 부족: 프로젝트 초기에 기능 구현에 집중하면서 미뤄진 상태.

6. 최종 완성된 프로젝트 결과물(소프트웨어, 하드웨어 등)을 설치하여 사용하기 위한 사용자 매뉴얼을 작성하시오.

① 접속 전 준비 사항

- 인터넷 연결이 되어 있어야 한다.
- Stripe 결제를 위해 유효한 신용카드 또는 체크카드를 준비해야 한다.
- 이메일 알림을 수신할 수 있는 이메일 계정이 필요하다.

② 웹사이트 접속 및 회원 가입

- 웹 브라우저를 열고 주소창에 Talk2Travel 서버 도메인을 입력한다.
- 메인 화면 상단의 메뉴 바에서 “회원 가입” 버튼을 클릭한다.
- 회원 가입 폼이 나타나면 다음 정보를 입력한 뒤 “가입하기” 버튼을 클릭한다.
- 가입이 완료되면 자동으로 로그인 페이지로 이동하거나, 입력한 이메일로 인증 메일이 발송된다.
- 인증 메일이 도착하면 메일 내의 “이메일 인증하기” 링크를 클릭하여 가입을 확정한다.

③ 로그인

- 메인 화면 또는 회원 가입 후 자동으로 리디렉션된 로그인 페이지에서 가입된 이메일 주소와 비밀번호를 입력한다.
- “로그인” 버튼을 클릭한다.
- 로그인에 성공하면 상단 메뉴 바에 사용자 이름(또는 닉네임)이 표시되며 캘린더 페이지로 이동한다.

④ 여행 일정 등록

- 로그인 후 처음 보이는 페이지는 캘린더이다.
- 날짜를 드래그하면, 일정을 등록할 수 있다.
- (캡스톤디자인2 구현 예정)화면 하단에 “2025년 6월 15일부터 6월 24일까지 오사카 2명 100만 원”처럼 자유롭게 입력한다.
- (캡스톤디자인2 구현 예정)화면 ENTER 키를 누르거나 옆의 “일정 등록” 버튼을 클릭하면 시

스템이 자연어를 파싱하여 여행지(city), 시작일(start_date), 종료일(end_date), 인원(passenger), 예산(budget)을 추출한다.
- 파싱 결과가 올바르게 나타나면 자동으로 일정이 캘린더에 등록되고 하단에 추천 항공·숙소 목록이 표시된다.
- 검색창 아래에 "출발 공항"·"도착 공항" 입력란이 있으며 두 글자 이상 입력하면 자동으로 IATA 코드(예: "ICN", "KIX")와 공항명을 제안한다.
- 원하는 공항을 선택하면 해당 코드가 자동으로 입력된다. 이 기능을 사용하면 자연어가 아닌 구체적인 출발/도착 공항을 설정할 수 있다.

⑤ 추천 항공·숙소 확인
- 일정 등록 후 화면 하단에 추천 항공 오퍼와 추천 숙소 오퍼가 각각 탭 형태로 나타난다.
- 항공 오퍼 탭
    - Amadeus API를 통해 가져온 여러 오퍼 중 가격 기준 상위 3개가 표시된다.
    - 각 오퍼는 항공사 이름, 출발·도착 시간, 경유 횟수, 총 요금 등의 정보가 있다.

- 숙소 오퍼 탭
    - Amadeus API에서 제공된 숙소 정보를 가격 기준 상위 3개로 필터링하여 보여준다.
    - 각 오퍼는 호텔 이름, 별점(★ 표시), 위치, 총 숙박 비용(체크인 – 체크아웃 기준)을 포함한다.

⑥ 오퍼 선택 및 결제
- 추천된 Top 3 항공/숙소 카드의 "예약하기" 버튼을 클릭한다.
- 선택 후 결제 페이지로 이동하며 결제하려는 인원 수, 택스/수수료 포함 여부를 확인한 뒤 카드 정보를 입력한다.
- "결제하기" 버튼을 클릭하면 Stripe 결제 시스템을 통해 결제가 진행된다.
- 결제 과정 중 3D Secure 인증(휴대폰 문자로 전송된 인증 코드 입력)이 필요한 경우 별도 창이 열리며 안내에 따라 인증을 완료한다.
- 결제가 승인되면 "결제 성공" 메시지가 화면에 표시되며 사용자가 입력한 이메일 주소로 예약 확인 메일이 발송된다.
- 화면 우측 상단 메뉴 바에 "예약 내역" 링크가 활성화되며 클릭 시 예약 상세 정보를 확인할 수 있다.

⑦ 예약 확인 및 관리
- 상단 메뉴 바의 "예약 내역" 메뉴를 클릭하면 예약 상태(확정/취소 대기 등), 여행지, 일정, 인원, 결제 금액, 예약 번호(Booking ID), 이메일·SMS 알림 상태 등의 상세 정보가 표시된다.
- 예약 내역 페이지에서 각 예약 건 우측의 "수정" 또는 "취소" 버튼을 클릭할 수 있다.
- 예약 확정 이후 화면 하단의 "알림 설정" 메뉴에서 여행 D-7 알림(이메일/푸시), 여행 D-1 알림(문자/SMS) 등을 켜고 끌 수 있다. 기본으로 이메일 알림이 활성화되어 있으며 휴대폰 번호를 등록하면 SMS 알림도 받을 수 있다.

⑧ 로그아웃
화면 우측 상단에 사용자 이름 또는 프로필 아이콘이 표시된다.

해당 아이콘을 클릭하면 “로그아웃” 메뉴가 나타난다.
“로그아웃”을 클릭하면 정상적으로 로그아웃되며 메인 로그인 화면으로 이동한다.

7. 캡스톤디자인 결과의 활용방안

- **사회적 파급효과**

  기존 플랫폼들은 항공편 예약, 숙소 검색, 일정 관리 기능이 각기 분산되어 있어 사용자가 여러 웹사이트를 오가야 하는 불편함이 있었다. 반면 Talk2Travel는 단일 인터페이스 안에서 여행 계획 수립부터 예약 확정까지의 과정을 직관적으로 처리할 수 있도록 설계되었다. 특히 사용자의 예산과 선호도를 반영한 추천 시스템을 통해 개인별 최적화된 여행 옵션을 제시함으로써 이용자 만족도를 크게 높일 수 있다. 이를 통해 바쁜 현대인들이 짧은 시간에도 간편하게 여행 일정을 짤 수 있고, 관광에 대한 접근성이 보다 쉬워진다.

- **기술적 파급효과**

  Talk2Travel는 자연어 입력(추후 음성 인식 기능 포함)으로 일정 등록을 지원하며, Amadeus API를 활용해 실시간 항공·숙소 정보를 받아오고, Stripe 연동을 통해 플랫폼 내에서 바로 결제까지 진행할 수 있다. 이처럼 핵심 기능을 모듈화된 서비스 레이어(예: flight_service.py, hotel_service.py)에 위임하여 구현함으로써, 외부 API 버전 변경이나 인증키 교체 시 내부 수정 범위를 최소화했다. 또한 일정 관리, 예약, 결제 기능이 하나의 시스템으로 통합되어 있기 때문에, 서로 다른 서비스 간 연동 문제를 원천적으로 제거하고 사용자 경험을 대폭 향상할 수 있다. 결과적으로 개발·운영 효율이 높아지고, 향후 기능 확장(머신 러닝 기반 추천, 챗봇 형태)에도 유연하게 대응할 수 있다.

- **경제적 파급효과**

  기존의 대형 여행사 중심 예약 시스템에서는 중개 수수료가 높고, 개인 여행자는 여러 플랫폼을 비교·검색하는 데 드는 시간·비용 부담이 있었다. Talk2Travel은 중소 여행사나 개인 여행자도 직접 비용 효율적인 여행 계획을 세울 수 있도록 지원하여, 중간 수수료를 절감할 수 있다. 이 과정에서 플랫폼이 차지하는 수수료를 최소화하고, 이용자에게 합리적인 가격의 여행 옵션을 제공함으로써, 여행 시장의 경쟁력을 높이는 효과를 기대할 수 있다.
  이처럼 사회·기술·경제적 측면에서 기존 시장의 한계를 극복하고, 사용자에게 편의성과 만족도를 동시에 제공함으로써 새로운 패러다임을 제시할 수 있는 플랫폼으로 평가받을 수 있다.