**폰트, 원, 텍스트, 로고이(가) 표시된 사진

자동 생성된 설명Trip-Flutter 최종보고서**

**Triple(7조)**

|  |  |
| --- | --- |
| 과목명 | 소프트웨어개발프로젝트 |
| 담당교수 | 이현숙 |
| 학과 | 컴퓨터소프트웨어공학과(학) |
| 팀명 | Triple (7조) |
| 팀원 | 전채린, 김동빈, 정승민 |
| 제출일 | 2024-12-12 |

**목차**

1. **작품의 개요**
2. **시스템 구성도**
3. **사용자 매뉴얼과 요구사항 분석 결과 정리**
4. **구축 시스템 데모 효과 향상방안**
5. **기타**
6. **개인별 탐구 기술 정리**

**1.작품의 개요**

**1.1.프로젝트 개요**

‘트리플러터’[그림1]는 사용자가 국내 여행을 준비하는 모든 과정을 하나의 플랫폼에서 해결할 수 있도록 만든 AI 기반 국내 여행 계획 웹 서비스입니다. 트리플러터는 사용자를 개인 여행자와 팀 여행자로 구별하여 개인 여행자가 여행을 계획할 때 필요한 기능과 팀 여행자가 함께 여행을 준비할 때에 필요로 하는 기능을 각각 개발하였습니다.

일반적으로 국내 여행을 준비할 때 정보 수집, 일정 계획, 예약 등 많은 시간이 소요되지만 트리플러터를 활용하여 시간 단축 및 불필요한 준비 과정을 줄이는 것을 목표로 하였습니다.



그림 1

**1.2.개발 배경 및 개발 목표**

국내 여행을 계획할 때 많은 사람들은 어디로 가야하고 어떤 일정을 구성해야 할지 고민합니다. 가까운 곳이지만 정보 부족과 일정 조율 등의 어려움으로 준비에 많은 시간을 할애합니다.

여행지 정보 수집, 일정 계획, 숙박 및 교통 예약 등 복잡한 과정을 거쳐야 하고 특히 여러 명이 함께 여행하였을 때 각자의 일정과 취향을 조율하는 데에 추가적인 어려움이 있습니다. 일정 조율을 위해 여러 번 연락을 취해 조정해야 하며 이로 인한 효율성 문제가 발생합니다.

이런 문제를 해결하고자 저희 트리플은 ‘트리플러터’를 개발하였습니다. 트리플러터는 Open AI사의 LLM인 GPT API를 활용하여 사용자의 요구와 취향에 맞는 맞춤형 여행 일정을 자동으로 생성하는 기능을 도입했습니다.

또한 Tour API를 활용하여 국내 여행에 특화된 정보와 기능을 제공하며, 팀 여행 기능을 통해 친구나 가족들과 함께 여행할 때 효율적으로 여행 계획을 수립하고 공유할 수 있도록 하였습니다. 이를 통해 여행 준비 과정에서 불필요한 과정으로 인해 발생하는 시간 낭비를 최소화하고 보다 즐겁고 편리한 여행 경험을 제공하고자 합니다.

**1.3 개발**

**1.3.1 개발 환경**

**Front-End**

* Front-End 환경 구축을 위해 Node.js 기반 React.js 풀 스택 프레임워크인 Next.js [그림2]를 활용하였습니다.

**Back-End**

* RESTful API 제작을 위해 풀 스택 프레임워크인 Next.js 환경에서 Node.js 및 Express.js [그림3]을 활용하였습니다.
* 데이터베이스는 비교적 간단한 데이터 저장을 위해 NoSQL 기반의 DB, Mongo DB [그림 4]를 활용하였습니다.
* 배포 서버는 간편한 접근성을 이유로 AWS의 EC2 [그림 5]를 활용하였습니다.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| [그림2] Next.js | [그림3] Express.js | [그림 4] Mongo DB | [그림 5] AWS EC2 |

**1.3.2 적용 기술**

적용한 기술에 대한 설명은 아래와 같습니다.

ChatGPT API [그림 8]

* 여행 계획 수립을 자동화 하기 위해 사용.

한국 관광공사 TourAPI [그림 9]

* 국내 여행지 검색을 위해 사용.

JWT [그림 10]

* 사용자 로그인 및 인증, 인가를 위해 사용

GitHub [그림 11]

* 프로젝트 형상 관리 및 협업

Notion [그림 12]

* 프로젝트 산출 문서 정리 및 진행 일정 관리

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| [그림 8] ChatGPT API | [그림 9] Tour API | [그림 10] JWT | [그림 11] GitHub | [그림 12] Notion |

**1.4. 차별화전략**

**1.4.1. 유사시스템 | 트리플**

트리플은 사용자가 여행지나 목적지를 설정하면 해당 지역에 대한 관광지, 식당, 숙박 정보 등을 제공해주는 여행 정보 제공 서비스이다. 사용자는 관심 지역을 선택하고, 추천 여행 루트나 명소 정보를 확인할 수 있으며, 일정 계획 시 필요한 다양한 정보를 간편하게 얻을 수 있다. 이를 통해 사용자는 여행 준비 과정을 효율적으로 진행할 수 있다.

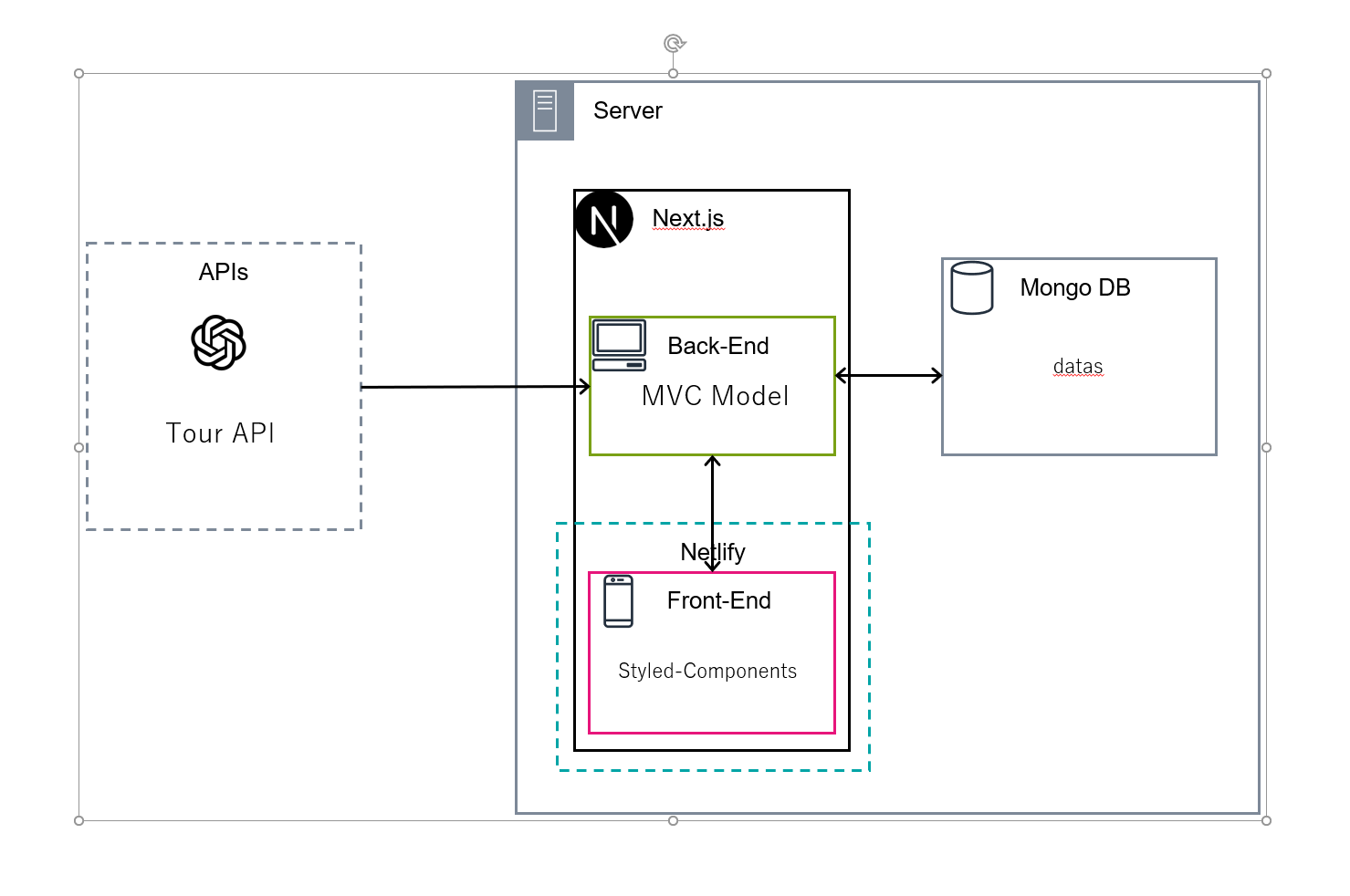
**1.4.2. 트리플러터만의 차별성**

트리플러터는 기존의 여행 정보 제공 서비스와 달리 ChatGPT 기반의 AI 기술을 활용하여 사용자가 원하는 여행지나 일정에 대한 추천을 손쉽게 받을 수 있습니다. 사용자가 희망하는 여행 스타일, 일정, 취향 등을 입력하면 AI가 적합한 여행지와 여행 일정 아이디어를 제안해주어, 맞춤형 여행 계획 수립이 가능해집니다.

또한 국내 여행지 정보 확인을 위해 TourAPI를 활용하여 다양한 관광 명소, 음식점, 숙박 시설에 대한 최신 정보를 제공합니다. 이를 통해 사용자는 단순한 정보 열람을 넘어, AI를 통한 맞춤형 여행 일정 추천 및 검증된 데이터를 활용한 여행 계획을 수립할 수 있으며, 이로써 기존 유사 서비스와의 명확한 차별화를 이룰 수 있습니다.

**2. 시스템 구성도**

**2.1.대략적인 전체 구성도**프로젝트의 구조는 [그림 13]과 같습니다.

  
[그림 13]

**2.2. 사용자 인터페이스 및 사용자 경험 설계 (UI / UX)**

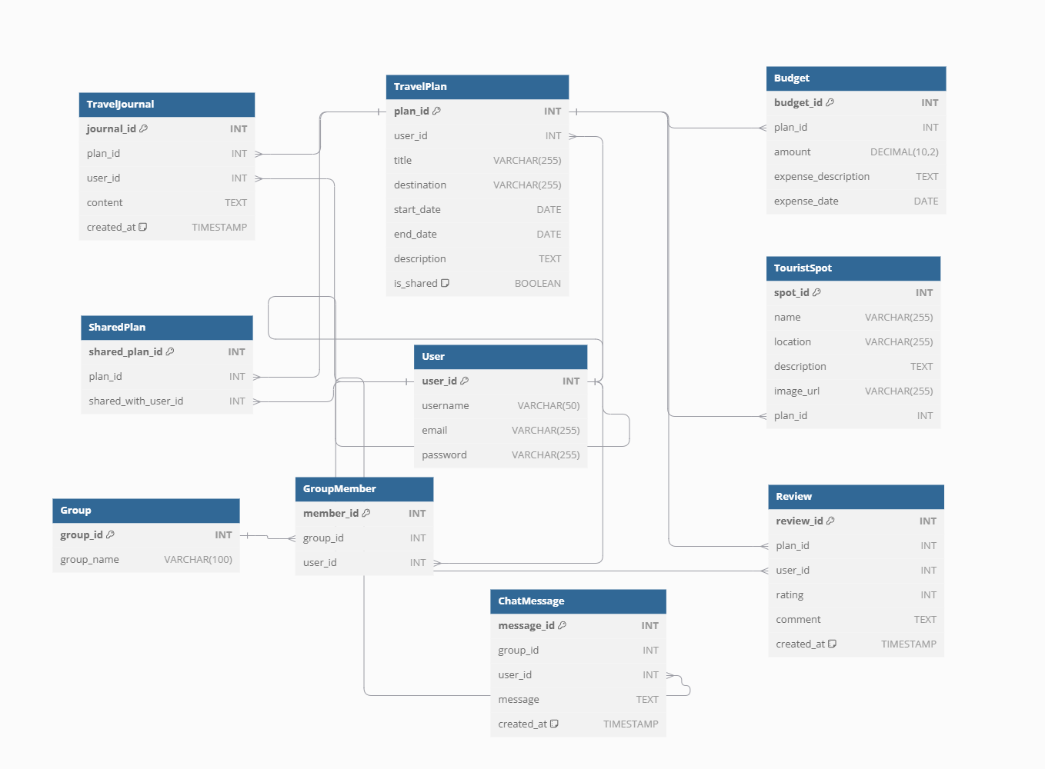
사용자 인터페이스 및 사용자 경험은 Figma를 통해 진행하였으며, 전반적인 사용자 인터페이스의 설계는 아래 [그림 14]와 같습니다.

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

[그림 14]

**2.3.데이터베이스 설계**스키마 구조는 [그림 14]와 같습니다.



[그림 14]

**User 테이블 :** 사용자 계정 정보를 저장하기 위한 테이블입니다. **TravelPlan 테이블 :** 사용자가 계획한 여행 정보(제목, 목적지, 여행 기간, 설명, 공개 여부 등)를 저장합니다.  
**Budget 테이블 :** 해당 여행 계획에 대한 비용 정보(지출 금액, 지출 내용, 지출 날짜)를 관리합니다.  
**TouristSpot 테이블 :** 여행 계획과 연계된 관광지 정보(이름, 위치, 설명, 이미지 URL)를 저장합니다.  
**TravelJournal 테이블 :** 여행 일정 중 사용자가 작성한 여행 기록(내용, 작성 시간) 정보를 저장합니다.  
**Review 테이블 :** 여행 계획 관련 피드백(평점, 코멘트, 작성 시간) 정보를 관리합니다.  
**SharedPlan 테이블 :** 특정 여행 계획을 다른 사용자와 공유한 이력(공유 대상 사용자 정보)을 저장합니다.  
**Group 테이블 :** 다수의 사용자들로 구성된 그룹 정보를 저장합니다. (그룹명 등).  
**GroupMember 테이블 :** 특정 그룹에 속한 사용자 정보를 관리합니다(어느 그룹에 어떤 사용자가 속하는지).  
**ChatMessage 테이블 :** 그룹 내에서 주고받은 채팅 메시지 정보(작성자, 내용, 작성 시간)를 저장합니다.

**3. 사용자 매뉴얼과 요구사항분석 결과 정리**

**3.1.사용자 시나리오 (간략한 채팅 예시)**

(1) 사용자: “가족 4명이 3박4일로 제주도 여행을 가고 싶어. 여유롭게 휴양 위주의 일정 추천 좀 해줘.”

(2) 시스템(ChatGPT): “좋습니다. 1일차는 숙소 근처 산책과 카페 휴식, 2일차는 해안도로 드라이브 후 섭지코지 방문, 3일차는 한라산 둘레길 짧은 산책, 4일차는 중문 해수욕장 근처 둘러보고 귀가 일정으로 제안할게요.”

(3) 사용자: “2일차에 성산일출봉을 넣지 않고 더 쉬운 코스로 바꿔줘.”

(4) 시스템(ChatGPT): “알겠습니다. 2일차는 성산일출봉 대신 애월 해안도로 드라이브 후 카페 휴식으로 변경할게요.”

(5) 사용자: 일정 선택 후 공유

(6) 여행 다녀온 후 후기 작성

**4. 구축 시스템 데모 효과 향상 방안**

**4. 1.구현 방법 및 적용 기술**

**<GPT의 여행지 추천 및 TourAPI 연동>**

사용자가 입력한 여행 관련 정보(지역명, 키워드 등)에 따라 OpenAI의 GPT 모델과 TourAPI를 활용하여 맞춤형 여행지 정보를 추천하고, 해당 정보를 DB에 저장 및 조회하는 기능입니다.

우선 사용자가 특정 지역 또는 제목으로 요청을 보내면, 서버는 TourAPI를 통해 해당 키워드에 맞는 여행지 데이터를 조회합니다. 이 과정에서 axios를 사용해 외부 API에 HTTP 요청을 보내고, 응답으로 받은 JSON 데이터 중 필요한 항목(예: title, firstimage, overview)을 추출합니다. 이후 MongoDB에 연결하여 views 컬렉션을 업데이트하거나 새로운 문서를 삽입합니다. 이를 통해 특정 여행지 정보가 조회될 때마다 조회 수(views)가 1씩 증가하고, 해당 여행지 제목, 이미지, 생성 시간 등이 DB에 기록됩니다. 이러한 구조를 통해 사용자는 본인이 찾고자 하는 여행지 정보를 실시간으로 확인할 수 있고, 시스템은 조회한 정보를 DB에 축적하고 관리하여 향후 참조나 통계 산출에 활용할 수 있게 됩니다.

**<여행지 목록 조회 기능>**

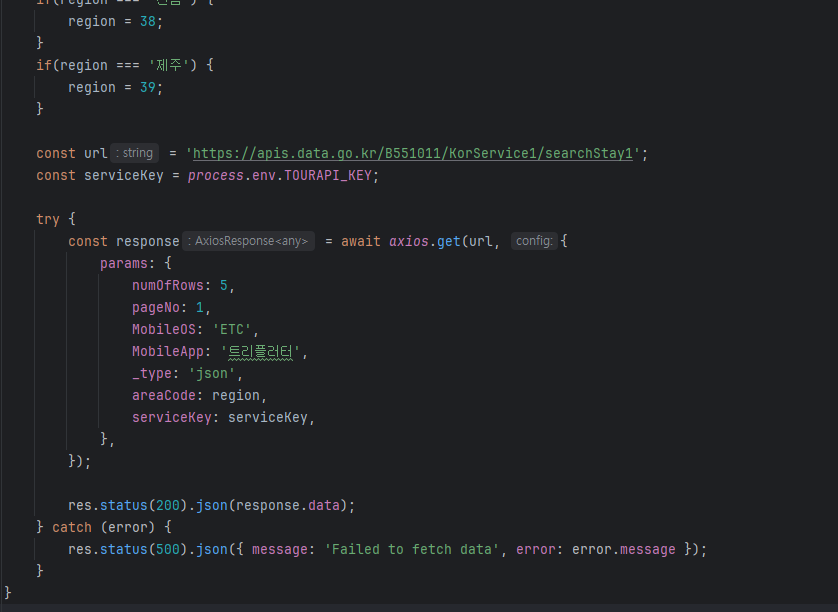
사용자가 특정 지역명을 GET 요청으로 전달하면, 해당 지역명(keyword)과 페이지 번호를 바탕으로 TourAPI에 HTTP 요청을 보내어 20개의 여행지 정보를 받아오는 기능입니다. axios를 사용하여 외부 API를 호출하고, 응답으로 받은 JSON 데이터 중 ‘item’ 리스트를 클라이언트에 반환합니다.

이를 통해 사용자는 지정한 지역에 해당하는 관광지 목록을 페이지 단위로 받아볼 수 있으며, 매 페이지마다 최대 20개의 관광지 정보를 확인할 수 있습니다. 코드 내에서 ‘region’과 ‘pageNo’를 파라미터로 받아 요청 URL 및 쿼리 파라미터를 동적으로 설정하고, API 응답이 성공적으로 수신되면 이를 JSON 형식으로 반환합니다. 에러 발생 시 상태 코드에 맞는 에러 메시지를 반환하여 클라이언트 측에서 적절히 대응할 수 있도록 합니다.

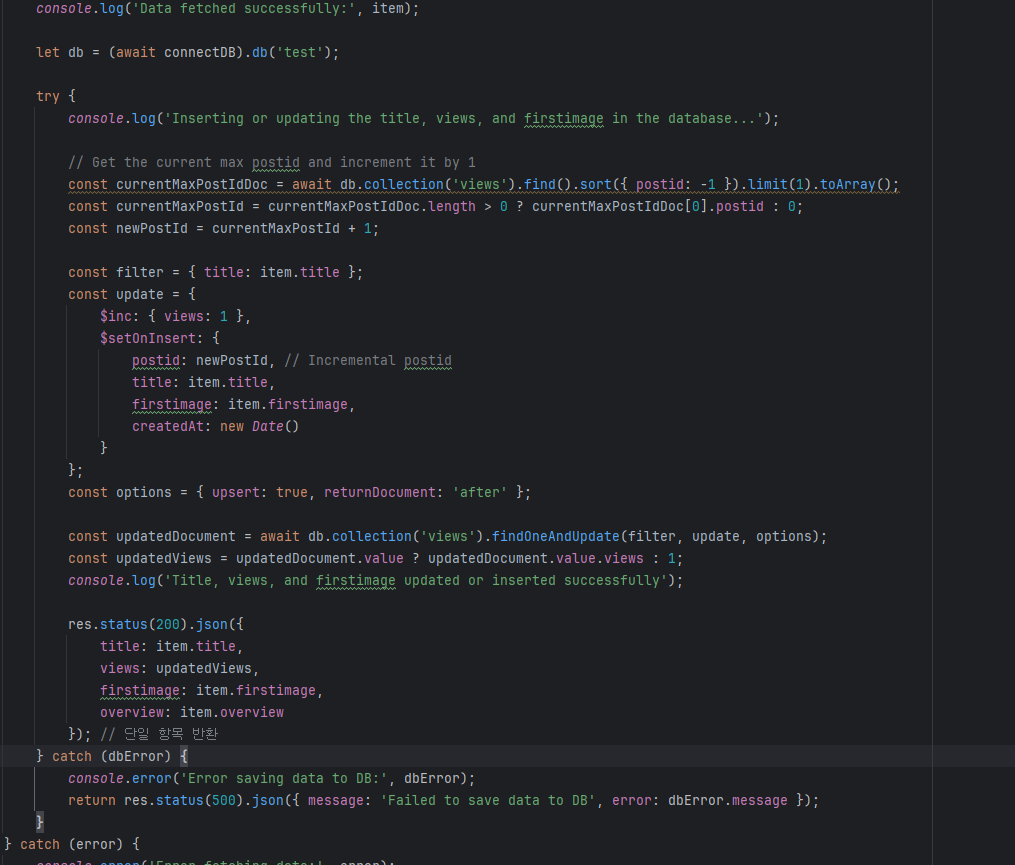
**<숙박 정보 조회 기능>**

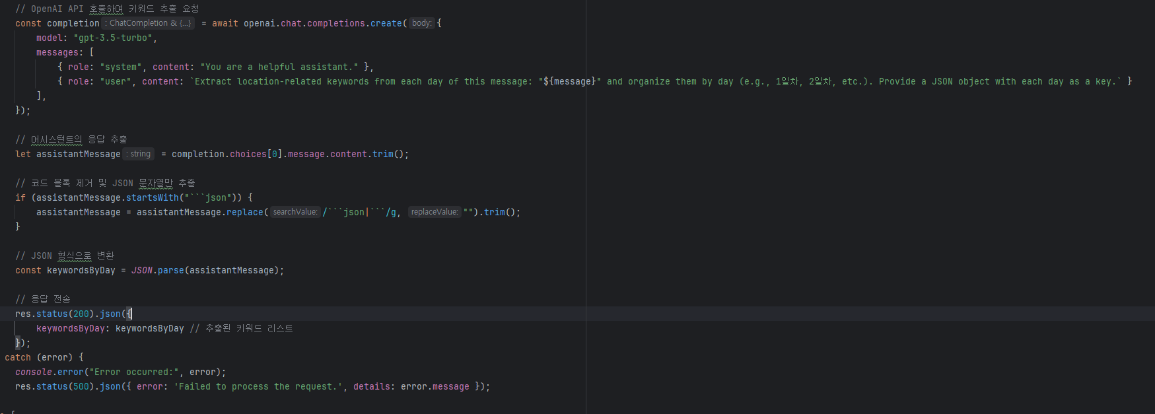
사용자가 특정 지역명을 GET 요청으로 전달하면, 해당 지역명을 행정 구역 코드를 통해 변환한 후 TourAPI의 searchStay1 엔드포인트를 호출하여 해당 지역의 숙박 정보를 조회하는 기능입니다. 코드는 지역명을 한국관광공사의 표준 지역 코드(서울=1, 부산=6 등)로 매핑한 뒤, axios를 사용해 외부 API 요청을 수행합니다. 응답으로 받은 JSON 데이터를 그대로 클라이언트에 반환하여, 사용자는 요청한 지역에 대한 5개 숙박 시설의 정보를 확인할 수 있습니다.

이를 통해 사용자는 지역명만 입력하여도 해당 지역의 숙박 정보를 빠르고 직관적으로 얻을 수 있으며, 향후 UI 레벨에서 이 정보를 가공하여 이용자가 쉽게 숙소를 선택할 수 있도록 할 수 있습니다.









**5. 기타 (이 프로젝트를 통해 알게 된 점)**

**5.1.외부 API 연동 및 응답 처리:**  
 다양한 외부 API(OpenAI, TourAPI)를 연동하고, axios를 통한 비동기 요청/응답 처리 과정을 경험하면서 API 호출 시 발생할 수 있는 예외 상황(응답 지연, 에러 응답, 데이터 파싱 문제)에 대비하는 로직 구현이 중요함을 깨달았습니다.

**5.2.데이터 관리 및 구조화:**  
 ChatGPT를 활용해 추출한 키워드나 일정 정보를 JSON 형태로 구조화하고, 이를 TourAPI와 결합하는 과정을 통해, 원시 텍스트 데이터에서 의미 있는 정보를 뽑아내고 이를 다른 서비스와 연계하는 데이터 파이프라인을 이해하게 되었습니다.

**5.3.DB 연동 및 상태 관리:**  
 MongoDB와 연동하여 조회한 관광지 정보를 저장하고, 조회 수를 갱신하는 기능을 구현하면서, 단순히 API 응답을 반환하는 것 이상으로 사용자 요청을 역사적으로 기록하고 분석할 수 있는 기반을 마련할 수 있음을 알게 되었습니다. 이를 통해 서비스 개선을 위한 데이터 수집과 활용 방안을 고려할 수 있게 되었습니다.

**5.4.유연한 아키텍처 설계 필요성:**  
 Next.js, Express.js, OpenAI, TourAPI, MongoDB 등 다양한 기술 스택을 조합하는 과정에서, 각 부분(인증, 데이터 처리, 외부 API 연동)을 모듈화하고 독립적인 역할을 갖도록 설계하는 것이 유지보수와 확장성 측면에서 중요하다는 점을 배웠습니다.

**5.5.사용자 경험(UX) 고려:**  
 사용자의 요구사항(여행 기간, 동반자, 예산, 선호 스타일 등)을 자연어로 입력받아 GPT로 처리한 뒤, 해당 결과를 실제 관광 정보와 매칭하는 과정을 거치면서, 단순한 데이터 제공이 아닌 "사용자가 이해하고 활용하기 쉬운 정보" 제공을 위해서는 응답 형식, 데이터 정렬, 필터링, UI/UX 기획까지 다각적인 요소를 고민해야 함을 깨달았습니다.

**6. 개인별 탐구 기술 정리**

**6.1.김동빈**

**6.1.1.JWT 기반 인증(Authorization) 기법**

**스터디 방향:** JWT의 구조, Access Token/Refresh Token 개념, 토큰 발급 및 검증 플로우 학습적용 내용: 로그인 시 서버에서 JWT를 발급하여 클라이언트에 전달, 추후 요청 시 헤더에 포함된 JWT를 검증하여 유효 사용자 식별

**6.1.2.ChatGPT API 연동을 통한 맞춤형 여행지 추천 기능**

**스터디 방향:** OpenAI ChatGPT API의 사용 방법, 프롬프트 엔지니어링, API 호출 시 보안/에러처리 방법 숙지

**적용 내용:** 사용자의 요구사항에 따라 ChatGPT API를 호출하여 추천 리스트 확보

**6.1.3.TourAPI 연동을 통한 관광 정보 조회**

**스터디 방향:** 외부 공공 API(TourAPI) 사용 방법, 인증키 관리, RESTful API 응답 파싱, 비동기 통신 패턴 학습

**적용 내용:** ChatGPT가 추천한 여행지 리스트를 토대로 TourAPI를 호출하여 여행지 상세 정보를 획득 및 클라이언트에 제공Next.js 기반 백엔드 구현

**6.1.4. 풀스택 프레임워크 Next.js 활용**

**스터디 방향:** Next.js의 서버 사이드 렌더링 원리, API Routes 기능을 통한 백엔드 로직 구현 방식, 페이지 라우팅 구조 이해

**적용 내용:** Next.js의 API Routes를 활용하여 인증, 추천, 정보 조회를 위한 백엔드 엔드포인트를 구성하고, 클라이언트 요청에 대응하는 서비스 로직 구현

**6.1.5. Express.js를 통한 서버 설정 및 미들웨어 구성**

**스터디 방향:** Express.js를 통한 라우팅, 미들웨어 구성, 에러 핸들링, CORS 설정, 로깅 전략 숙지

**적용 내용:** 인증, 로깅, 에러 처리 등의 미들웨어를 Express로 구성하고, API 서버 안정성 강화

**6.1.6. MongoDB 연동 및 Mongoose 활용**

**스터디 방향:** MongoDB의 특성, Mongoose를 통한 스키마 정의, CRUD 연산, 인덱싱 및 성능 이슈 학습

**적용 내용:** 사용자 계정, 추천 결과, 관광지 관련 데이터를 MongoDB 컬렉션에 저장하고 관리하며, 필요 시 Mongoose를 통한 ORM으로 데이터 핸들링 로직 단순화

**6.2.정승민**

Figma를 활용하여 트리플러터의 전반적인 사용자 인터페이스와 사용자 경험을 설계, 디자인하였고 그것을 Next.js 프레임워크를 통해 웹 브라우저상에 렌더링하기 위한 퍼블리싱 작업과 프론트엔드 개발을 담당하였습니다.

Next.js 프레임워크의 최신 버전인 Next.js 14버전의 app router 활용을 위해 기존 pages router와 다른 점에 대해 탐구하고, app router가 제공하는 디렉토리 기반 컴포넌트 배치를 통해 트리플러터 서비스의 전반적인 페이지 URL 구축에 힘썼습니다.

또한 Next.js 프레임워크가 제공하는 next/layout 내장 라이브러리를 통해 헤더와 푸터 등 페이지 전역에 걸쳐 공통적으로 적용되어야 하는 레이아웃을 구성하기 위한 스터디를 진행했고, react-redux 라이브러리를 통해 사용자의 로그인 정보 등의 상태를 페이지 전역에서 참조할 수 있도록 탐구하고 적용하였습니다.

마지막으로 대표적인 CSS in JS 라이브러리인 styles-components가 클라이언트 사이드 렌더링 환경에서만 정상적으로 동작하여 SSR 프레임워크인 Next.js에서 사용했을 때 CSS 로드가 한 박자 느리다는 이슈가 발생해 styled-component가 정상적으로 동작할 수 있도록 webpack을 설정하는 방법에 대한 탐구를 통해 적용하였습니다.

**6.3.전채린**

**6.3.1.Next.js의 주요 특징**

Next.js는 React 기반의 프레임워크로, 서버 사이드 렌더링(SSR)과 정적 사이트 생성(SSG)을 지원하며, 성능 최적화 및 SEO 개선에 유리한 환경을 제공합니다. 주요 특징은 다음과 같습니다:

1. **서버 사이드 렌더링(SSR)**: 클라이언트 렌더링에 비해 초기 로딩 속도가 빠르고, SEO 친화적입니다.
2. **정적 사이트 생성(SSG)**: 빌드 시점에 HTML을 생성하여 빠른 페이지 로드를 제공합니다.
3. **파일 기반 라우팅**: 디렉토리 구조와 파일 이름을 기반으로 URL이 자동 생성됩니다.
4. **API 라우트**: 서버리스 함수로 동작하는 API를 쉽게 작성할 수 있습니다.
5. **Image Optimization**: 최적화된 이미지 처리로 페이지 성능 향상.

**6.3.2.TypeScript의 주요 특징**

TypeScript는 JavaScript의 상위 집합으로, 정적 타입을 지원하여 코드의 안정성과 유지보수성을 향상시킵니다. 주요 특징은 다음과 같습니다:

1. **정적 타입 검사**: 컴파일 단계에서 오류를 감지할 수 있습니다.
2. **IDE 지원 강화**: 코드 작성 중 자동 완성, 리팩토링, 타입 추론 기능 강화.
3. **인터페이스와 타입**: 데이터 구조를 명확히 정의하고, 협업 과정에서 코드 이해를 용이하게 합니다.
4. **ES6+ 지원**: 최신 JavaScript 기능과 함께 사용 가능.

**6.3.3.프로젝트 적용 사례**

**1.환경 설정**

**1.1.프로젝트 초기화**

Next.js와 TypeScript를 기반으로 프로젝트를 설정하기 위해 다음 단계를 수행했습니다:

1. **Next.js 설치**:

npx create-next-app@latest --typescript

--typescript 옵션을 통해 기본 설정부터 TypeScript를 활성화했습니다.

1. **필요한 패키지 설치**:
   * ESLint와 Prettier로 코드 스타일과 품질 유지.
   * Axios를 사용하여 API와의 통신 구현.
2. **TypeScript 설정 파일(tsconfig.json) 구성**:

{

"compilerOptions": {

"target": "es5",

"lib": ["dom", "dom.iterable", "esnext"],

"strict": true,

"baseUrl": ".",

"paths": {

"@components/\*": ["components/\*"],

"@utils/\*": ["utils/\*"]

}

}

}

**1.2.디렉토리 구조**

* **pages/**: Next.js의 파일 기반 라우팅을 위한 페이지 컴포넌트.
* **components/**: 재사용 가능한 UI 컴포넌트.
* **styles/**: CSS 모듈 및 전역 스타일 파일.
* **utils/**: 공통 함수 및 헬퍼 파일.
* **types/**: TypeScript 타입 정의 파일.

**2.주요 구현 내용**

**2.1 파일 기반 라우팅**

Next.js는 파일 구조를 기반으로 라우트를 자동 생성합니다. 예를 들어:

* **경로:** pages/index.tsx
  + 메인 페이지 렌더링.
* **경로:** pages/trip/[id].tsx
  + 동적 라우팅을 활용하여 특정 여행 계획의 상세 페이지를 구현.

코드 예제:

import { useRouter } from 'next/router';

const TripDetail = () => {

const { query } = useRouter();

const tripId = query.id;

return (

<div>

<h1>Trip ID: {tripId}</h1>

</div>

);

};

export default TripDetail;

**2.2 API 통합**

Next.js의 API 라우트를 활용해 서버리스 방식으로 API를 작성하고, 이를 통해 ChatGPT API와 통신을 구현했습니다.

* **API 파일 경로:** pages/api/chat.ts
* **API 구현:**

import { NextApiRequest, NextApiResponse } from 'next';

export default async function handler(req: NextApiRequest, res: NextApiResponse) {

const { userInput } = req.body;

const response = await fetch('https://api.openai.com/v1/chat/completions', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

Authorization: `Bearer ${process.env.OPENAI\_API\_KEY}`,

},

body: JSON.stringify({

model: 'gpt-4',

messages: [{ role: 'user', content: userInput }],

}),

});

const data = await response.json();

res.status(200).json({ result: data.choices[0].message.content });

}

**2.3 컴포넌트 설계**

재사용성을 고려하여 컴포넌트를 설계했습니다. 주요 컴포넌트는 다음과 같습니다:

* **ChatInput**: 사용자 입력을 처리하는 입력 컴포넌트.
* **TripCard**: 여행 계획을 요약하여 카드 형태로 표시.
* **MapDisplay**: Google Maps API를 활용하여 여행 경로를 시각화.

const TripCard: React.FC<{ trip: Trip }> = ({ trip }) => (

<div className="trip-card">

<h2>{trip.destination}</h2>

<p>{trip.startDate} - {trip.endDate}</p>

</div>

);

**2.4 상태 관리**

상태 관리는 Next.js 내장 상태와 Context API를 조합하여 구현했습니다. 필요한 경우 React Query를 사용하여 서버 데이터와 동기화.

**6.3.4.프로젝트 경험과 학습**

**6.3.4.1.배운 점**

1. **TypeScript의 강점**:
   * 타입 정의를 통해 코드 안정성 확보.
   * 팀 협업 시 데이터 구조의 명확성 제공.
2. **Next.js의 효율성**:
   * 서버 사이드 렌더링과 정적 사이트 생성의 이점을 활용.
   * 파일 기반 라우팅으로 생산성 향상.
3. **API 통합 경험**:
   * 외부 API(ChatGPT, Google Maps 등)와의 통신 방법 학습.
   * 서버리스 함수 작성 및 환경 변수 관리.
4. **성능 최적화**:
   * 이미지 최적화, 코드 스플리팅, 정적 페이지 생성 등 다양한 기법 적용.

**6.3.4.2.개선 방향**

1. **더 나은 상태 관리 도구 탐색**: Recoil이나 Zustand와 같은 라이브러리의 활용 가능성 검토.
2. **테스트 자동화 도입**: Jest와 Cypress를 활용한 테스트 커버리지 확장.
3. **사용자 피드백 반영**: UI/UX 개선을 위해 사용성 테스트 지속.

**6.3.5.결론**

Next.js와 TypeScript를 활용한 이번 프로젝트는 웹 개발의 현대적 기법을 학습하고 실무에 적용하는 좋은 기회가 되었습니다. 이를 통해 안정적이고 확장 가능한 프론트엔드 시스템을 구축할 수 있었으며, 사용자 경험을 향상시키는 데 중점을 두었습니다. 향후 개선 작업과 기술 확장을 통해 더 나은 서비스를 제공할 계획입니다.