

POPBLE

KD ACADEMY Project

우민경

GitHub : https://github.com/minkii-w/popble_wmk

사용기술 스택

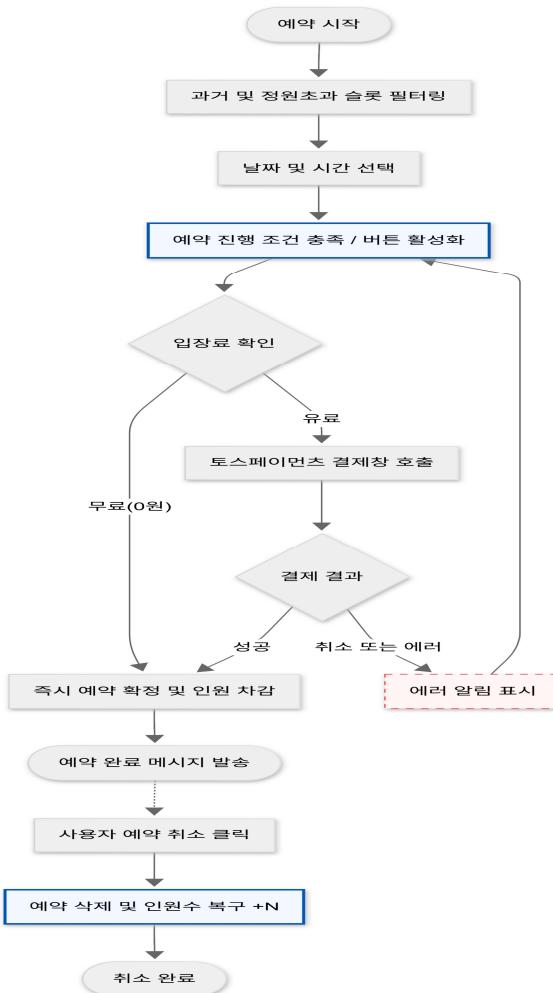
프론트엔드

백엔드

협업툴



FlowChart



설계 목표

사용자가 날짜를 선택하고 결제한 뒤, 예약이 최종 확정되기까지의 흐름을 시각화했습니다.
단순히 기능 구현에 그치지 않고, 결제 도중 발생할 수 있는 오류 상황까지 고려하여 설계했습니다.

핵심 구현 포인트

- 예약 가능 여부 확인

- 현재 시간을 기준으로 지난 일정은 비활성화로 사용자에게 직관적으로 표현하여 선택불가능의 상황을 제공
- 해당 시간의 정원수를 초과한 경우에도 비활성화를 통해 사용자에게 편의를 제공

- 비용 조건에 따른 결제 경로 설정

- 무료 입장 : 즉시 예약이 확정되도록 설계
- 유료 입장 : 결제창이 뜨도록 나누어 상황에 맞는 진행 경로를 설정

- 예약 재시도 상황 흐름 고려

- 결제진행도중 사용자의 변심, 결제오류상황일 시에는 현재 선택한 날짜/시간을 재선택하는 번거로움이 없이 바로 결제를 시도할 수 있는 화면으로 유지함

- 데이터 정합성을 고려한 인원수 복구

- 결제오류, 결제취소일 시에는 예약 진행상황이 불가한 점을 판단하여 예약인원 수 차감을 진행하지 않도록 구현
- 사용자가 예약을 취소할 경우에는 줄어들었던 인원수가 다시 즉시 복구되도록 구현

주요 기능 설명

예약

```
// 예약 등록  
@Override  
public Long register(ReservationDTO dto) {  
  
    Reservation reservation = dtoToEntity(dto);  
    ReservationTime rTime = reservation.getReservationTime();  
  
    // 예약 후 해당 예약시간의 잔여인원수 업데이트용  
    if(rTime != null) {  
        if(rTime.getCurrentCount() + reservation.getReservationCount() > rTime.getMaxCount()) {  
            throw new IllegalArgumentException("예약 가능 인원 초과 예제지");  
        }  
        rTime.setCurrentCount(rTime.getCurrentCount() + reservation.getReservationCount());  
        reservationTimeRepository.save(rTime);  
    }  
    reservationRepository.save(reservation);  
    return reservation.getId();  
}
```

```
// 예약 취소  
@Override  
public void cancel(Long id) {  
    Reservation reservation = reservationRepository.findById(id)  
        .orElseThrow();  
    // 예약취소 시 예약가능한 인원수 돌아옴  
    ReservationTime rTime = reservation.getReservationTime();  
    if(rTime != null) {  
        rTime.setCurrentCount(Math.max(0, rTime.getCurrentCount() - reservation.getReservationCount()));  
        reservationTimeRepository.save(rTime);  
    }  
  
    reservationRepository.deleteById(id);  
}
```

```
// 예약 가능 잔여 인원수 체크  
public List<ReservationDTO> getByPopupStoreAndDateTime(Long popupstoreId, LocalDate date, LocalTime startTime){  
    return reservationRepository.findByPopupStore_IdAndReservationTime_DateAndReservationTime_StartTime  
        (popupstoreId, date, startTime)  
        .stream()  
        .map(this::entityToDto)  
        .collect(Collectors.toList());  
}
```

예약 등록

- 사용자가 입력한 예약데이터(DTO)를 **Reservation Entity**로 변환하여 저장
- 현재 예약 인원(getCurrentCount)과 신규 예약 인원을 합산하여 최대 수용 인원 초과 여부를 사전에 검증
- 초과 시 예외 발생시키고, 프론트엔드(React)에서 예약불가안내(alert) 처리
- 예약 가능 자리가 있을 시, 현재 예약 인원수에 새로 예약 인원수를 합산하여 잔여 인원 갱신 후 DB 저장

예약 취소

- 취소 희망 사용자의 정보로 예약내역을 확인(reservationRepository.findById)
- Math.max함수를 사용해 음수 값 발생 가능성을 방지하여 데이터 정합성 유지
- 취소 완료 시 예약 데이터(DB) 및 잔여 인원 복구 처리

예약 가능 인원수 체크

- 사용자에게 보여지는 잔여 인원수 업데이트 코드
- 최대수용인원수(getMaxCount) - 현재예약인원수(getCurrentCount) 계산

주요 기능 설명

예약 시간 배열 I

```
//여러 개의 시간 슬롯을 한 번에 등록  
@Override  
public void createReservationTimes(List<ReservationTimeDTO> dtoList) {  
    dtoList.forEach(this::createReservationTime);  
}
```

```
//팝업스토어의 첫타임, 마지막타임 정보 조회(상세보기용)  
@Override  
public List<ReservationTimeDTO> getAllTimes(Long popupstoreId){  
    List<ReservationTime> result = reservationTimeRepository.findAllByPopupStoreId(popupstoreId);  
    return result.stream()  
        .map(this::entityToDTO)  
        .collect(Collectors.toList());  
}
```

```
//시간대 목록 가져오기  
public List<ReservationDTO> getByPopupStoreAndDateTime(Long popupstoreId, LocalDate date, LocalTime startTime){  
    return reservationRepository.findByPopupStore_IdAndReservationTime_DateAndReservationTime_StartTime  
        (popupstoreId, date, startTime)  
        .stream()  
        .map(this::entityToDto)  
        .collect(Collectors.toList());  
}
```

행사기업의 예약 시간대 등록(정보등록)

- 여러 운영 시간대, 최대 입장 인원을 컬렉션 기반으로 한 번에 등록하도록 구현

행사기업 운영시간 안내

- DB에 저장된 행사기업의 전체 운영 시간대를 조회
- stream을 사용해 entity를 사용하기 쉬운 형태의 DTO List로 전환

시간대 목록

- 행사기업의 운영시간대(최대입장인원수 포함) 목록 생성
- 예약 화면에 시간 정보와 인원 정보를 함께 응답

주요 기능 설명

예약 시간 배열 //

```
// 특정 날짜의 예약 가능한 시간 슬롯 목록을 조회
@Override
public List<ReservationTimeDTO> getAvailableTimesByDate(Long popupstoreId, LocalDate date) {
    List<ReservationTime> times = reservationTimeRepository.findByPopupstoreIdAndDateOrderByStartTimeAsc(
        popupstoreId, date);

    return times.stream()
        .map(time -> ReservationTimeDTO.builder()
            .id(time.getId())
            .popupstoreId(popupstoreId)
            .date(time.getDate())
            .startTime(time.getStartTime())
            .endTime(time.getEndTime())
            .maxCount(time.getMaxCount())
            .remainingSeats(time.getMaxCount() - time.getCurrentCount())
            .build())
        .collect(Collectors.toList());
}
```

예약 시간

오전

10:00 - 11:00 (10석) 11:00 - 12:00 (10석)

오후

12:00 - 13:00 (10석) 13:00 - 14:00 (10석)
14:00 - 15:00 (10석) 15:00 - 16:00 (10석)
16:00 - 17:00 (10석) 17:00 - 18:00 (10석)
18:00 - 19:00 (10석)

```
{amTimes.length > 0 ? (
    amTimes.map(t => {
        const today = new Date();
        const isToday =
            selected.date.getFullYear() === today.getFullYear() &&
            selected.date.getMonth() === today.getMonth() &&
            selected.date.getDate() === today.getDate();

        const isPastSlot = isToday && isPastTime(t.startTime, selected.date);
        const isDisabled = t.remainingSeats <= 0 || isPastSlot;
    })
)}
```

특정 날짜 선택 시 예약 가능 시간 슬롯 나타내기

- 사용자가 선택한 팝업스토어와 날짜를 기준으로 예약 가능한 시간 슬롯 목록을 조회
- 조회된 시간대 정보를 builder 패턴을 사용해 예약 가능 여부, 잔여 인원, 시간대 상태 정보를 포함한 응답 DTO 구성

예약 버튼 비활성화

- 현재날짜,시간을 기준으로 이미 지난 시간대에 대해 프론트엔드(React)에서 비활성화 버튼상태 부여

주요 기능 설명

결제 API

```
import { useEffect } from "react";

const TossPayment = ({ price, ordername, onSuccess, onFailure }) => {
  const clientKey = "test_ck_D5GePWyJnrK0W0k6q8gLzN97Eoq";

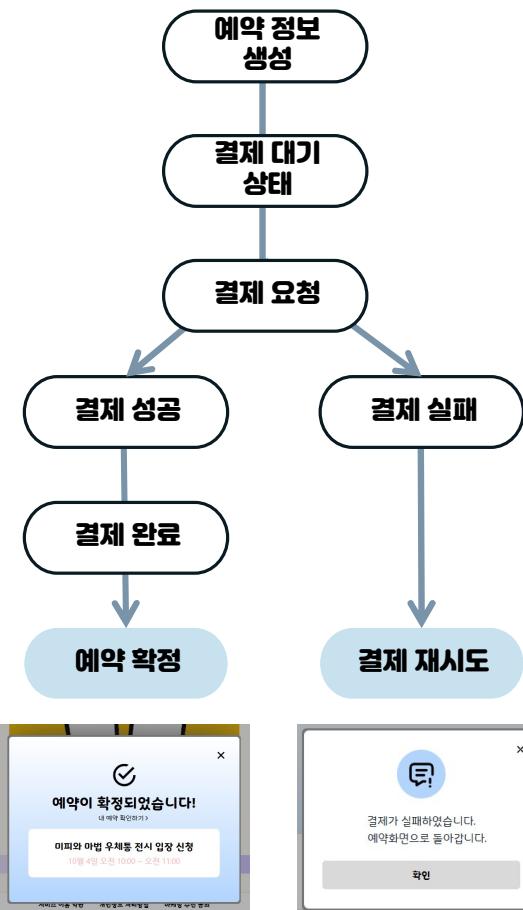
  useEffect(() => [
    if (!window.TossPayments) {
      return;
    }

    const tossPayments = window.TossPayments(clientKey);

    tossPayments
      .requestPayment("카드", {
        amount: price,
        orderId: `order_${new Date().getTime()}`,
        orderName: ordername,
        customerName: "예약자",
      })
      .then((result) => {
        console.log("결제 성공:", result);
        if (onSuccess) onSuccess(result);
      })
      .catch((error) => {
        console.error("결제 실패:", error);
        if (onFailure) onFailure(error);
      });
  ], [price, ordername, onSuccess, onFailure]);

  return null;
};

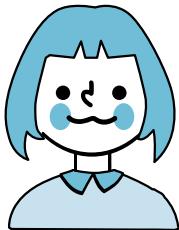
export default TossPayment;
```



결제 API

- Toss Payments API와 연동하여 예약에 대한 결제 요청 및 결과를 처리
- 결제 결과에 따라 백엔드 기준의 예약 상태 값을 변경하고, 해당 상태를 기준으로 예약 확정 여부를 판단하도록 설계
- 결제 실패시 예약을 확정하지 않고 재시도 가능한 상태로 유지하여 데이터 정합성 확보
- 프론트엔드는 결제 요청 트리거 역할만 수행

* 담당 및 프로젝트 회고 *



우민경

담당 : 예약, 결제

- 예약 진행 기능 구현
- 과거시간/최대 정원
도달 시 예약 비활성화
기능 구현
- 토큰 결제 API 연동

개선할 점 *

- **기초 개념 학습의 부족**
HTTP 통신, 트랜잭션, 동시성 처리 등 백엔드 핵심 개념에 대한 이해가 부족하다고 판단
기능은 동작하지만 '왜 이렇게 동작하는지?'에 대한 설명이 어렵다는 한계점을 느낌
➡ 대안 : 작은 프로젝트 및 기능 단위 구현을 통해 반복 학습 필요
- **API 개념 이해 부족**
요청-응답 흐름 및 내부 처리 로직에 대한 이해가 충분치 않아, 동작 원리 중심의 학습의 필요성
➡ 대안 : 예제 API를 직접 설계/구현하며 내부 순환 로직에 대한 이해 강화
- **실시간 예약 기능 미구현 및 동시성 처리 한계**
실시간 통신 방식에 대한 이해 부족으로 다른 사용자의 예약 상태를 즉시 반영하는 기능을 구현하지 못함
또한 현재는 단일 요청 기준으로 예약 로직이 설계되어 있어, 동시에 다수의 예약 요청이 발생하는 경우를 고려한
트랜잭션 및 Lock 기반의 동시성 제어를 적용하지 못함
➡ 대안 : WebSocket 기반 실시간 상태 반영 / @Transactional, DB Lock 등을 활용한 동시성 제어 적용하여
예약 데이터 정합성을 보장하도록 리팩토링 예정
- **실패 상황에 대한 예외 처리 부족**
정상 흐름 위주로 구현되어, 예외 상황에 대한 처리와 에러 응답 설계가 충분하지 못함
➡ 대안 : 실패 상황별 에러 응답 기준 정리 및 일관된 응답 구조 설계 강화
- **서비스 배포 환경 구성의 한계**
로컬 환경 중심으로 개발하려 실제 배포가 이루어지지 않음
➡ 대안 : 리눅스 서버 및 AWS를 활용한 실제 서버 설정 시도
- **테스트 코드 작성 경험 제한적**
로직 검증을 테스트 코드로 충분히 검증하지 못함
➡ 대안 : 리팩토링 시 안정성을 확보하기 위한 코드 작성 시도