Mini Payment Demo 개인 미니 프로젝트

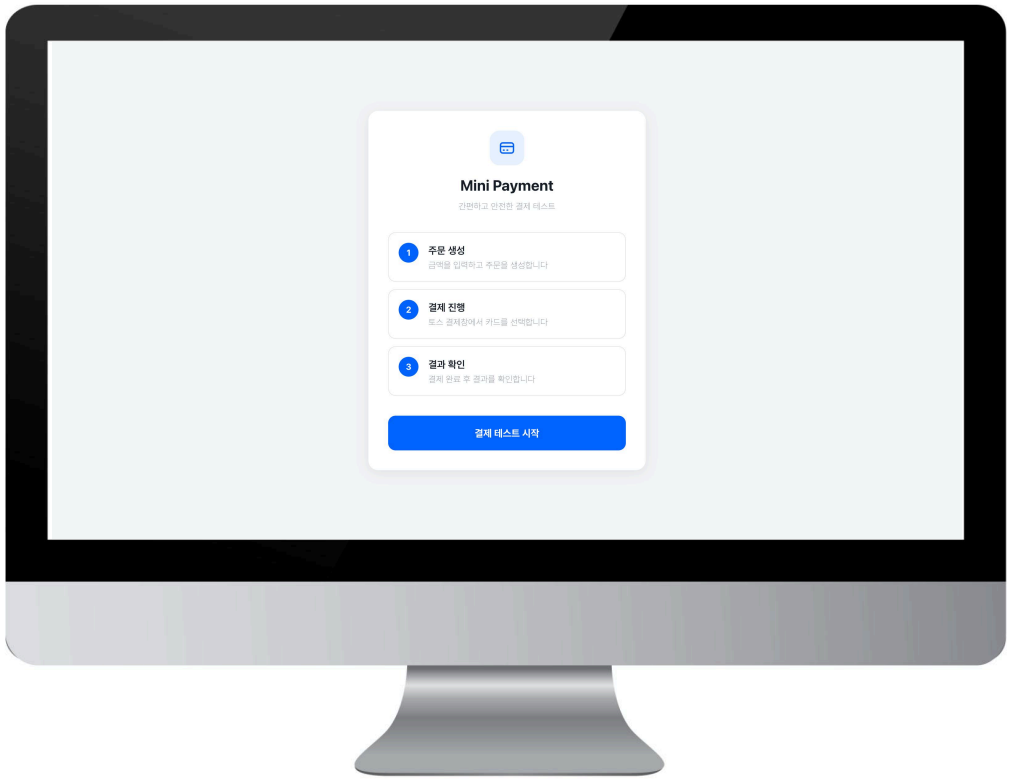
2026.02

프로젝트 개요

핀테크 서비스 환경을 가정하고, 주문 생성 → 결제 요청 → 승인 → 결과 확인 → 중복 방어 → Webhook 구조 설계까지
하나의 결제 플로우를 구현한 미니 프로젝트입니다. 단순 UI 구현이 아니라, 서버에서의 결제 상태 관리
결제 승인과 Webhook 분리 설계 테스트 환경에서의 실제 PG 연동 구조 이해에 초점을 맞추었습니다.

목표

결제의 전체 라이프사이클을 이해하고 직접 구현
프론트엔드와 서버 간 역할 분리 설계
중복 결제 방어 로직 구현
TossPayments 테스트 환경 연동



Mini Payment Demo 개인 미니 프로젝트

2026.02

기술 스택

Front-end

- Next.js (App Router), TypeScript, Tailwind CSS

Back-end

- Next.js Route Handler, Prisma, PostgreSQL (Neon)

PG

- TossPayments 테스트 환경 (requestPayment 방식)

역할

주문생성

- 서버에서 orderId + paymentToken 생성
- DB에 CREATED 상태로 저장

TossPayments requestPayment 연동

- 성공 시 서버 confirm API 호출

결제 승인 (Confirm)

- paymentKey + orderId + amount 검증
- Toss API 승인 요청
- 성공 시 DB 상태 PAID 업데이트

중복 결제 방어 및 실패 케이스 처리

- 이미 결제된 주문은 재결제 차단
- 서버 400/401/500 분기 처리

Mini Payment Demo 개인 미니 프로젝트

2026.02

결제 상태 신뢰성

문제

"결제 완료" 버튼을 눌렀다고 해서 실제 결제가 정상적으로 처리되었다고 보장할 수 없었다.

고민

결제의 최종 상태는 클라이언트가 아니라 서버가 가져야 하지 않을까?

UI 상태와 DB 상태가 어긋나면 어떻게 할 것인가?

해결

주문 상태를 DB에서 관리 (CREATED → PAID)

결제 완료 API에서 서버가 상태를 직접 검증

UI는 서버 응답만을 기준으로 상태 표시

Mini Payment Demo 개인 미니 프로젝트

2026.02

중복 결제 방어

문제

같은 orderId로 결제 요청이 두 번 들어오면 결제가 중복 처리될 위험이 있었다.

고민

결제 API는 네트워크 재시도, 더블클릭 등으로 여러 번 호출될 수 있음

이미 결제된 주문에 대해 어떻게 처리해야 할까?

에러를 줄 것인가? 성공을 반환할 것인가?

해결

DB에서 status가 PAID 인 경우 → 상태 변경 없이 기존 정보 반환

실제 update는 최초 1회만 수행

Mini Payment Demo 개인 미니 프로젝트

2026.02

결제 성공/실패/중복 구분

문제

DB에서는 결제 상태가 구분되지만 UI에서는 이를 명확히 표현하지 못했다.

고민

단순 성공/실패가 아니라 정상 결제 이미 결제됨 강제 실패를 어떻게 구분할 것인가?

해결

UI 상태를 세 가지로 분리:

- SUCCESS
- DUPLICATE DETECTED
- FAILURE

배지 + 메시지 + 처리 시간 표시로 명확한 피드백 제공

Mini Payment Demo 개인 미니 프로젝트

2026.02

회고

처음에는 "버튼 누르면 결제 완료"라는 단순 흐름으로 구현했지만, 실제 결제 시스템에서는 클라이언트가 신뢰 기준이 될 수 없다는 것을 깨달았다.
결제의 최종 상태는 서버가 결정해야 하며, UI는 서버 상태를 단순히 표현하는 계층에 불과하다는 점을 체감했다.
→ 상태의 주체를 명확히 분리하는 설계의 중요성을 배웠다.

중복 결제 테스트를 구현하면서 "같은 요청이 여러 번 들어올 수 있다"는 전제를 두고 API를 설계해야 한다는 걸 배웠다.
단순히 중복을 막는 게 아니라 이미 처리된 요청에 대해 "같은 결과를 반환하는 것"이 더 안전한 구조라는 점을 이해했다.
→ 결제 시스템은 실패보다 '중복'이 더 위험하다는 걸 인지하게 되었다.

orderId만으로 결제 완료 처리가 가능하다는 걸 깨닫고 paymentToken 검증 로직을 추가했다.
이 과정을 통해 보안은 나중에 붙이는 옵션이 아니라 처음 설계 단계에서 반드시 고려해야 할 기본 전제라는 걸 배웠다.