

Nest.js로 다시 시작한 디미페이

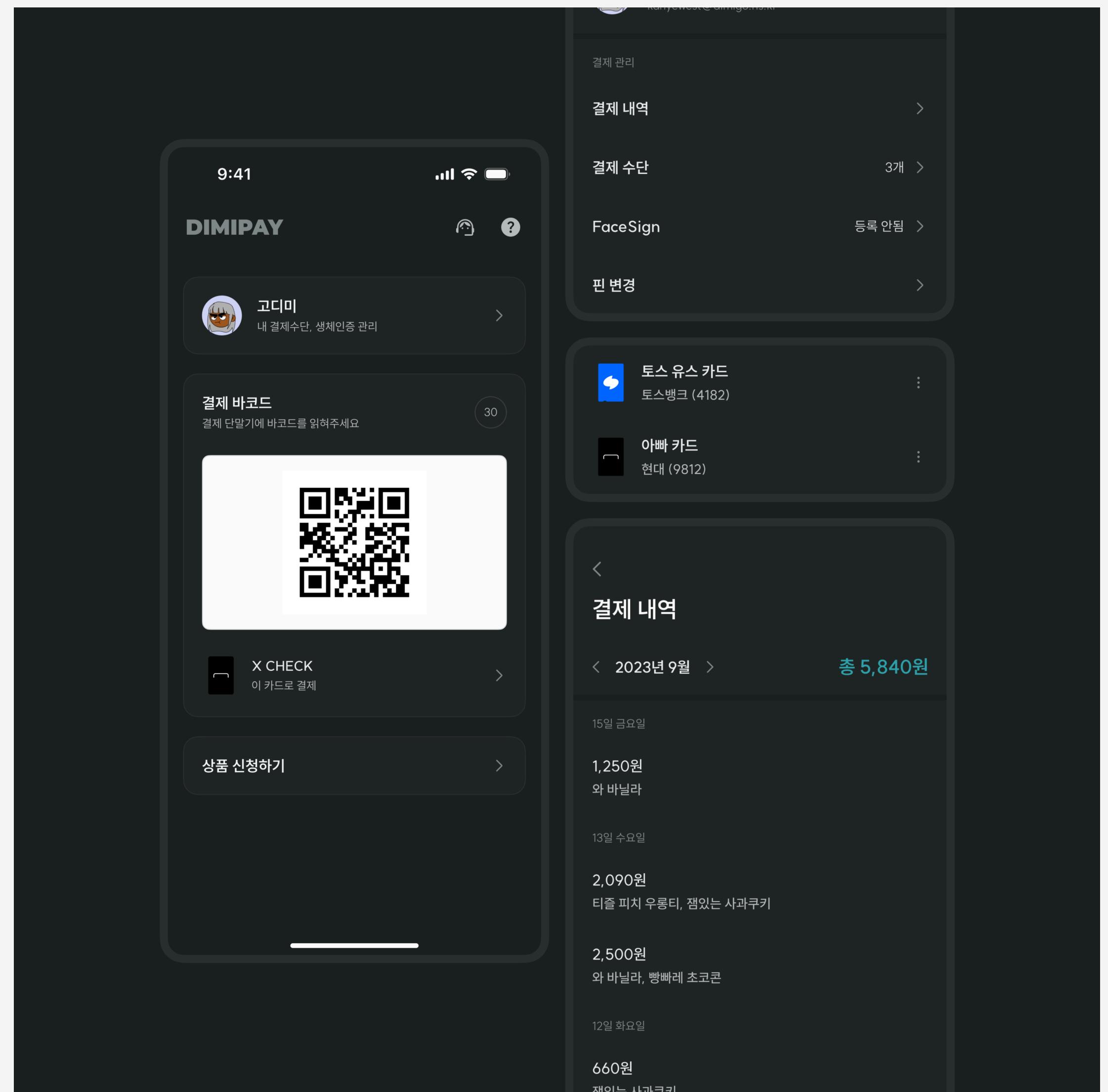
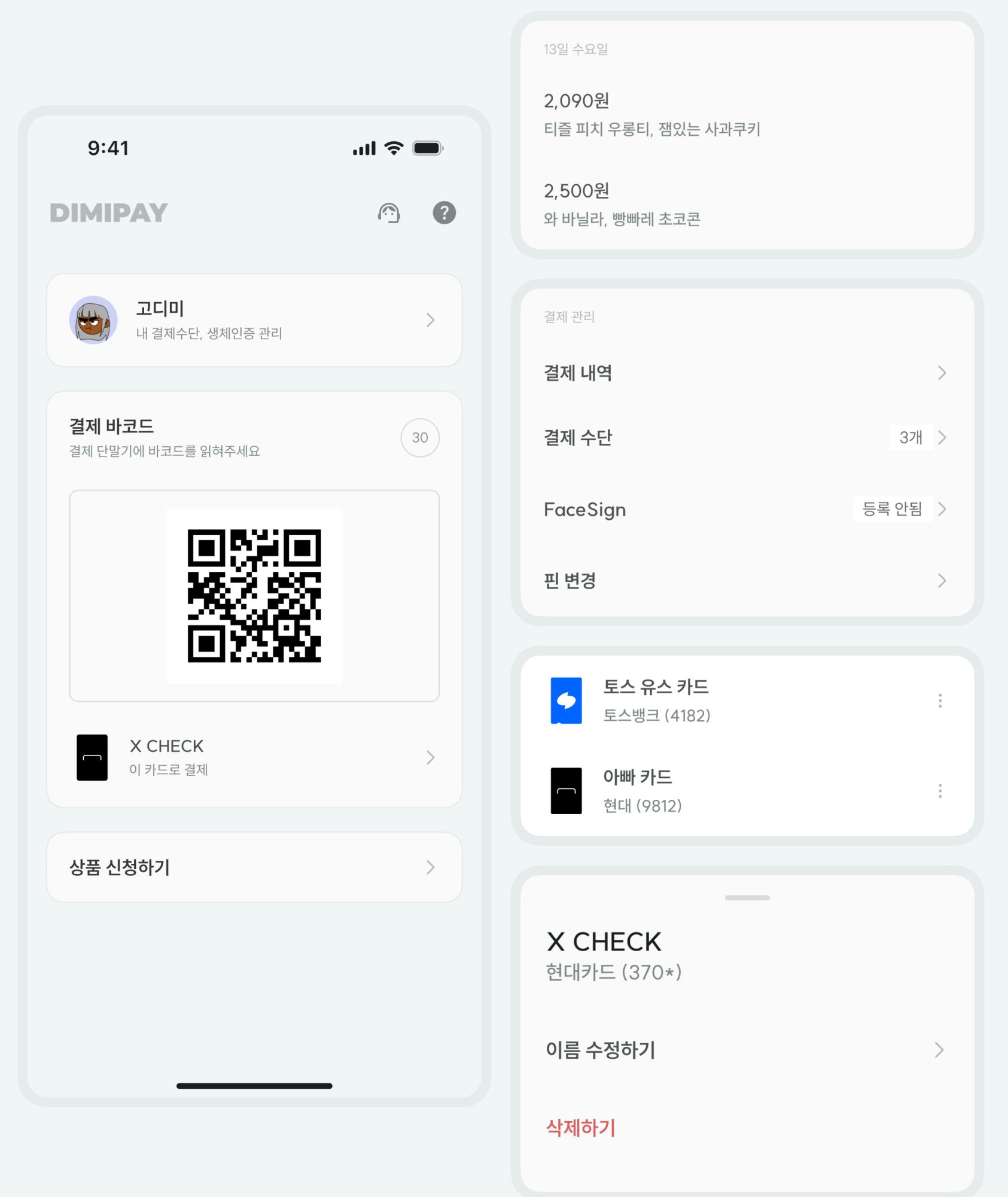
- 2025/04 ~ 당근마켓 당근알바 백엔드 개발자
- 2022년~ 디미페이 백엔드 개발자
- javien@daangn.com
- in/javien



디미페이

한국디지털미디어고등학교 교내 무인 매점 결제 솔루션

- 학생들로 이루어진 팀에서 모바일 앱, 결제 서버, 키오스크 앱, 어드민 개발
- 인터넷 없이 QR코드로 결제 또는 얼굴인식으로 결제



고*미님

얼굴 다시 인식하기

상품을 터치해서 삭제할 수 있어요

상품 전체 삭제

롯데) 옥동자 밀크
620원

동서) 제티(캔)
560원

붕어싸만코
1,250원

웅진) 티즐피치(우롱티)
1,450원

4개 상품
3,860원

X CHECK
이 카드로 결제

변경

결제하기

디미페이 앱의 결제 QR를 스캔해주세요

상품 스캔 창에서 결제 QR를 바로 스캔하면
빠르게 결제를 완료할 수 있습니다

20초

< 상품 스캔 화면으로 돌아가기

결제 비밀번호 입력

안전한 결제를 위해 결제 비밀번호를 입력해주세요.



4 1 5

2 8 6

7 9 3

- <https://www.youtube.com/watch?v=Mi5IdUZPD1c>
- <https://www.youtube.com/watch?v=zG9PC3to8N8>

- 2022년, 백엔드 개발자
- 2023년, PM/팀 리딩
- 현재, 백엔드 유지/팀 관리

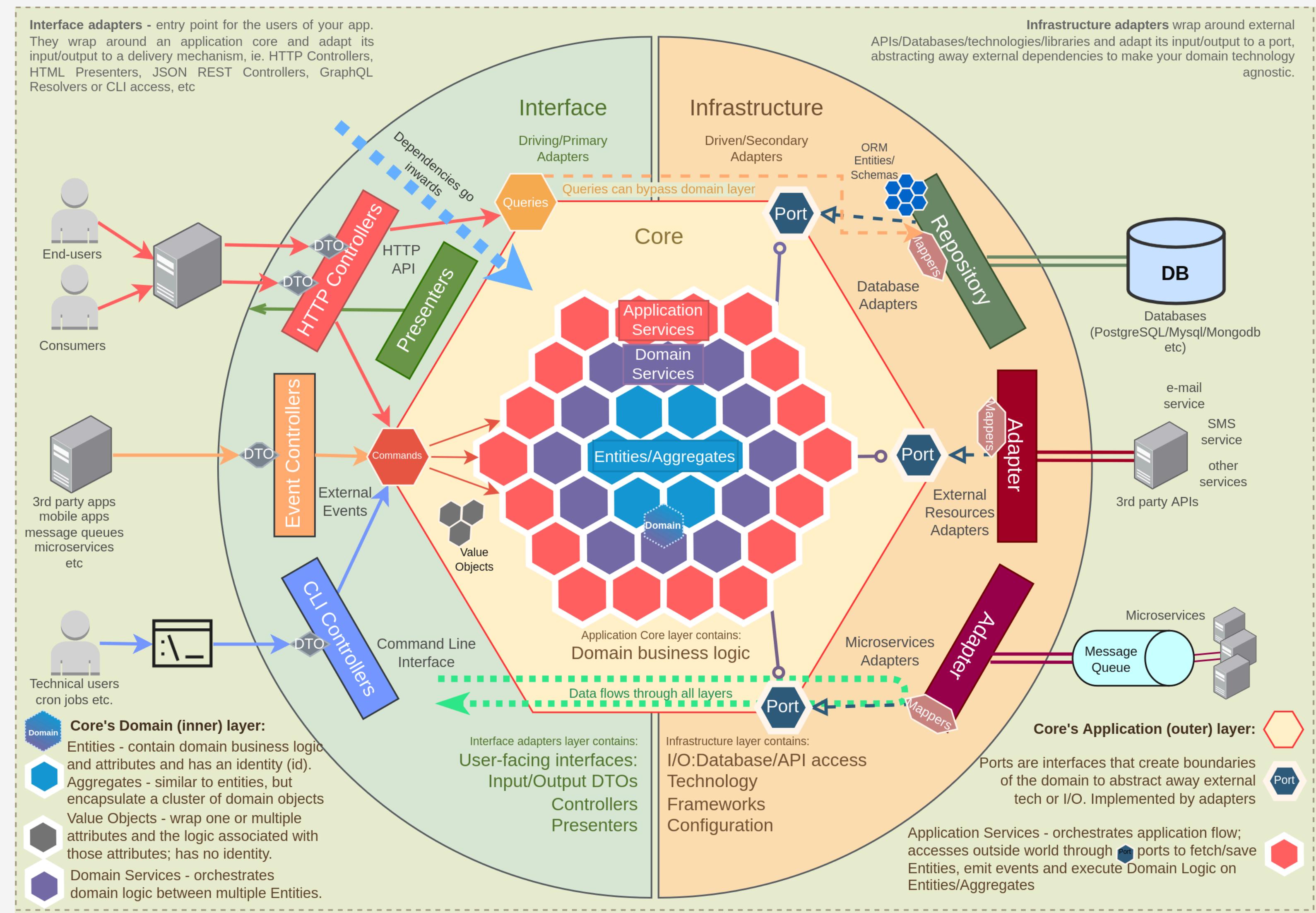
계층이란 개념도 모르던 학생이
Nestjs로 디미페이 백엔드를 처음부터 만든 과정

디미페이 v1

- express, typescript, prisma 기반 백엔드
- 서비스가 다뤄야할 문제는 점점 많아지고 있었음
- 사이드 이펙트 예측 불가
- 패턴 없이 즉흥적으로 만든 코드

디미페이 v1 – why?

- 복잡한 소프트웨어를 다루는 방법을 몰랐음
- 코드를 논리적으로 구분하는 방법을 몰랐음



Domain-driven hexagon

dimipay-back-v2 tech stack

- Nestjs와 typescript로 기술 스택 구성
- Bun을 runtime으로 사용
- Mikro ORM
 - 작업 단위(Unit of Work), 식별자 맵(Identity Mapper) 패턴을 구현
 - 공식 Nestjs 지원, 암시적 트랜잭션, result cache 등 지원

Bun v1.0.3

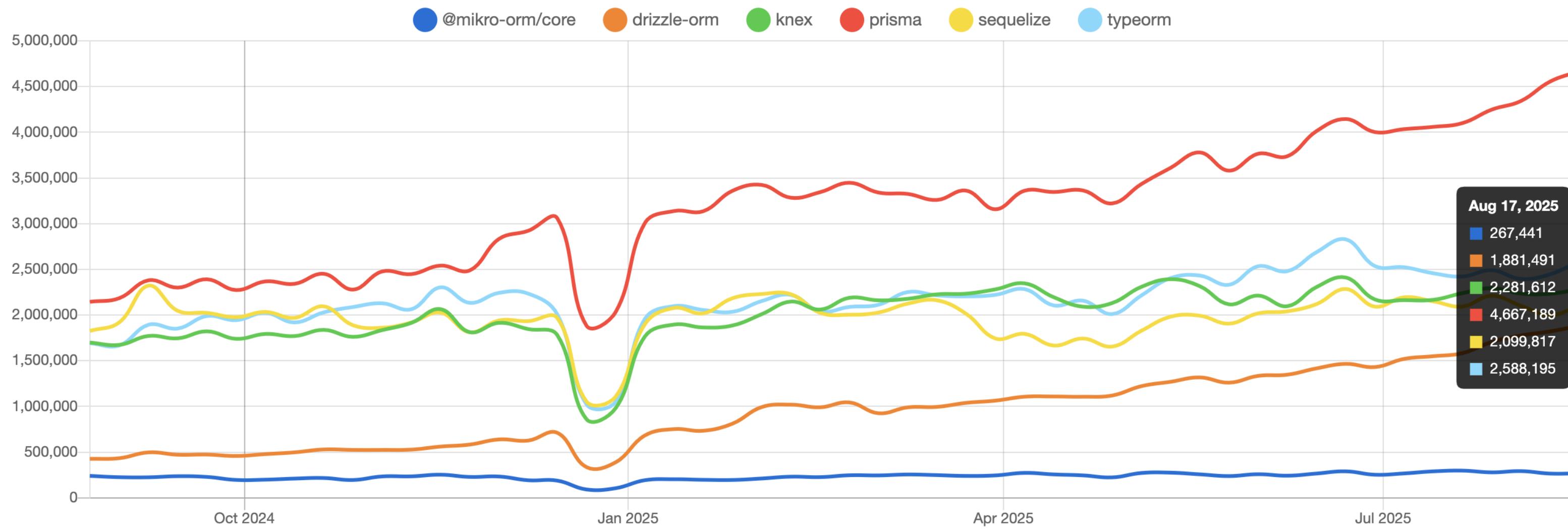
Colin McDonnell · September 22, 2023



Bun v1.0.3 fixes numerous bugs, adds support for metadata reflection via TypeScript's `emitDecoratorMetadata`, improves `fetch` compatibility, unblocks private registries like Azure Artifacts and Artifactory, bunx bugfix, `console.log()` depth 8 -> 2, Node.js compatibility improvements, and more.

@mikro-orm/core drizzle-orm knex prisma sequelize typeorm

Downloads in past 1 Year



Stats

		NPM	Github	Stars	Issues	Version	Updated	Created	Size
Blue square	@mikro-orm/core	NPM	GitHub	-	-	6.4.16	3 months ago	5 years ago	
Orange square	drizzle-orm	NPM	GitHub	29,954	1,725	0.44.5	2 days ago	4 years ago	
Green square	knex	NPM	GitHub	19,979	1,020	3.1.0	2 years ago	12 years ago	
Red square	prisma	NPM	GitHub	43,599	2,350	6.14.0	14 days ago	9 years ago	
Yellow square	sequelize	NPM	GitHub	30,136	984	6.37.7	5 months ago	14 years ago	
Light Blue square	typeorm	NPM	GitHub	35,719	2,484	0.3.26	8 days ago	9 years ago	

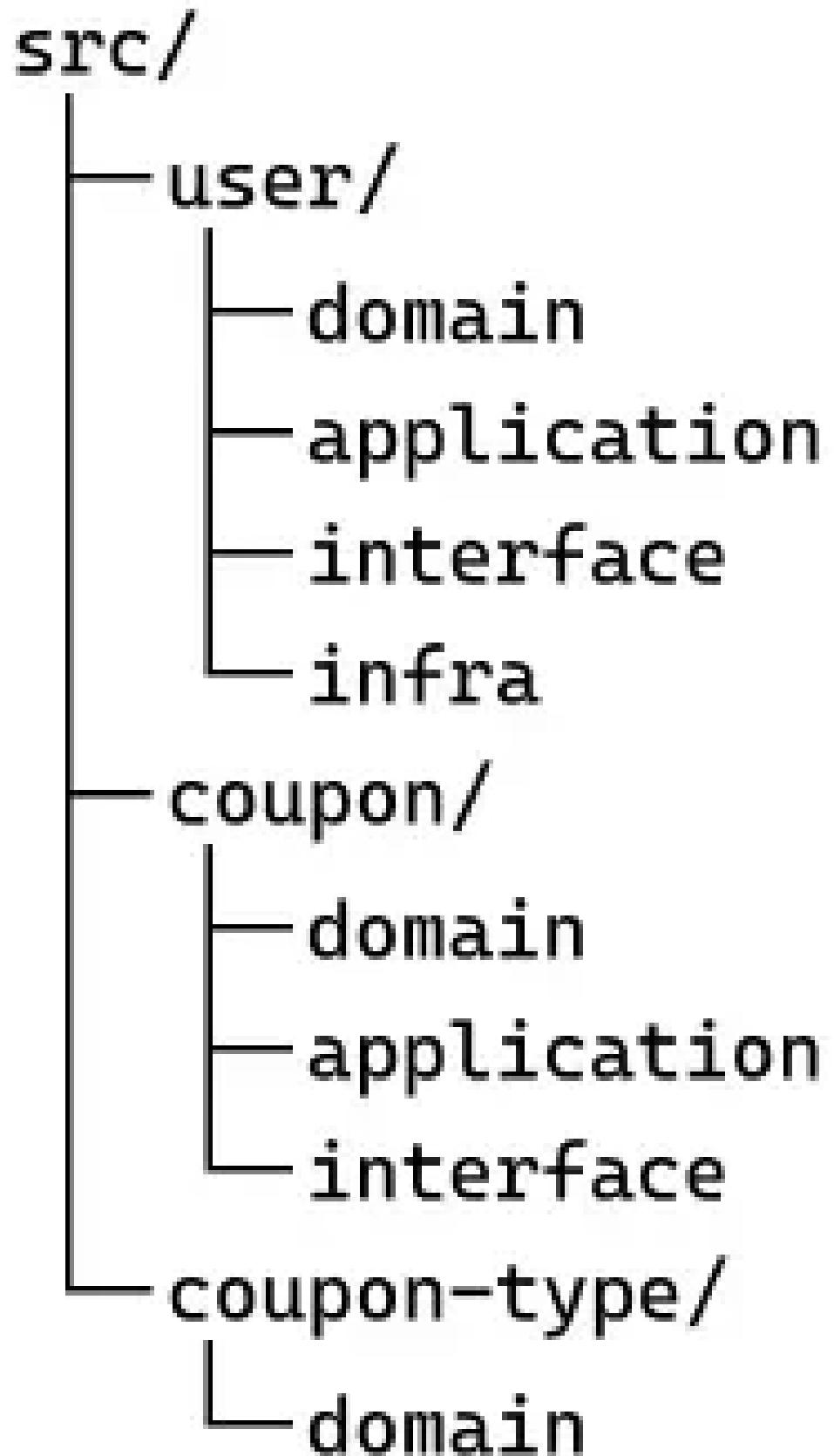
Folder structure

폴더구조 - 계층 기반

```
src/
  └── domain/
      ├── entities
      └── value-objects
  └── application/
      ├── services
      └── event-handlers
  └── infra/
      └── interface/
```

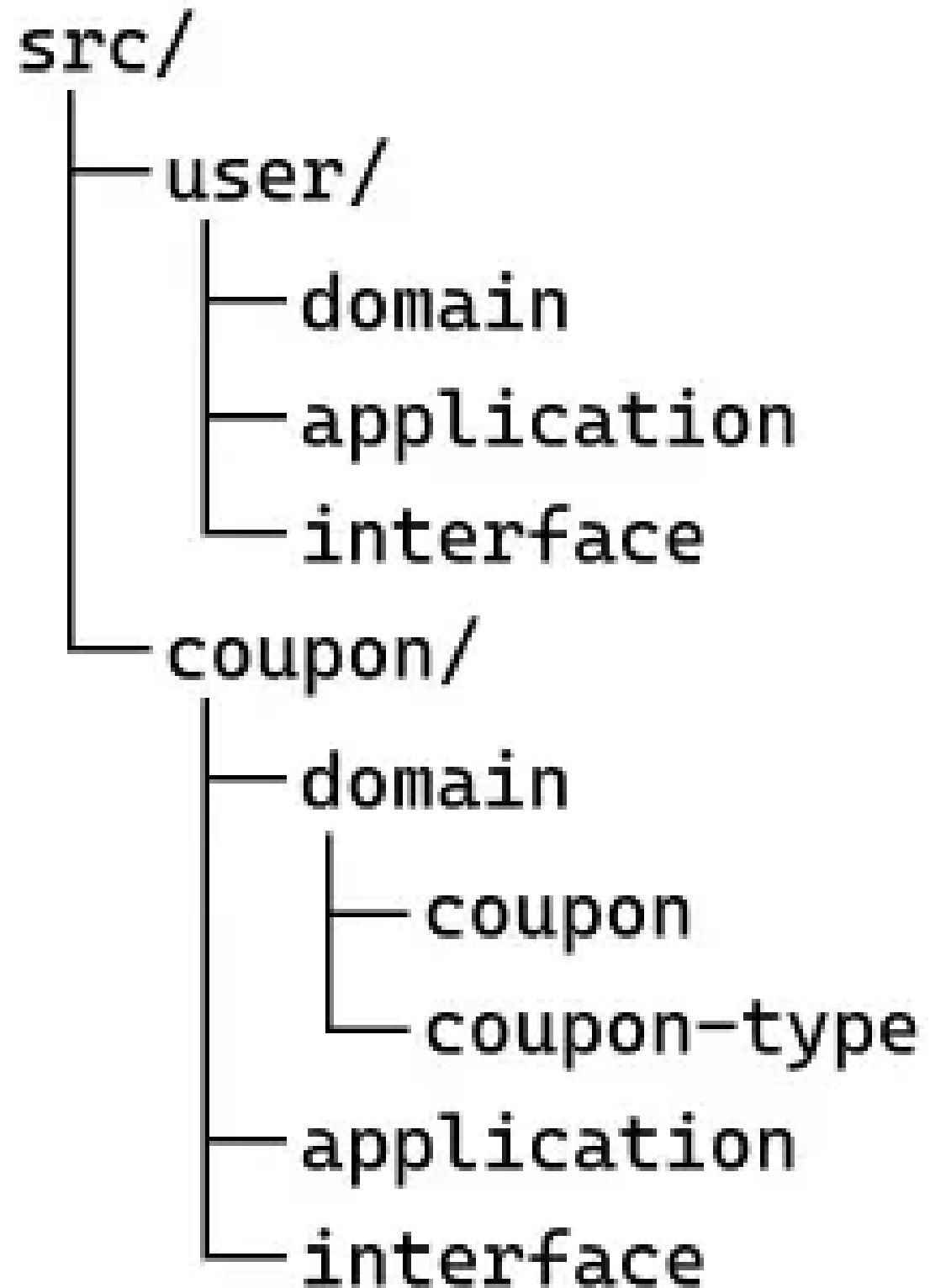
- 계층이 하나의 루트 폴더
- 계층간 결합도 ↑
- 도메인 응집도 ↓

폴더구조 - 모델 기반

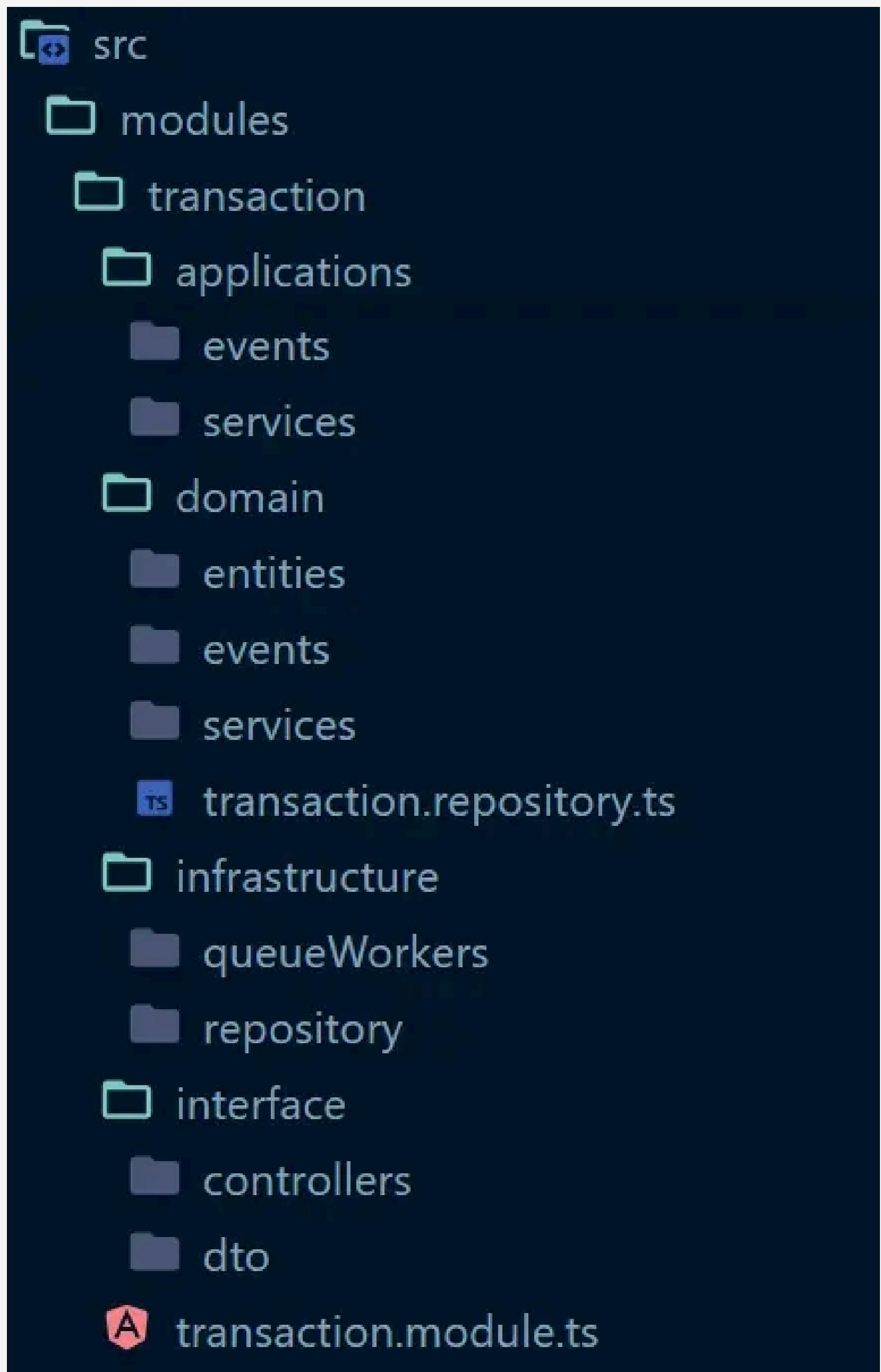


- 모델 폴더 안에서 계층을 나눔
- 느슨한 도메인간 관계

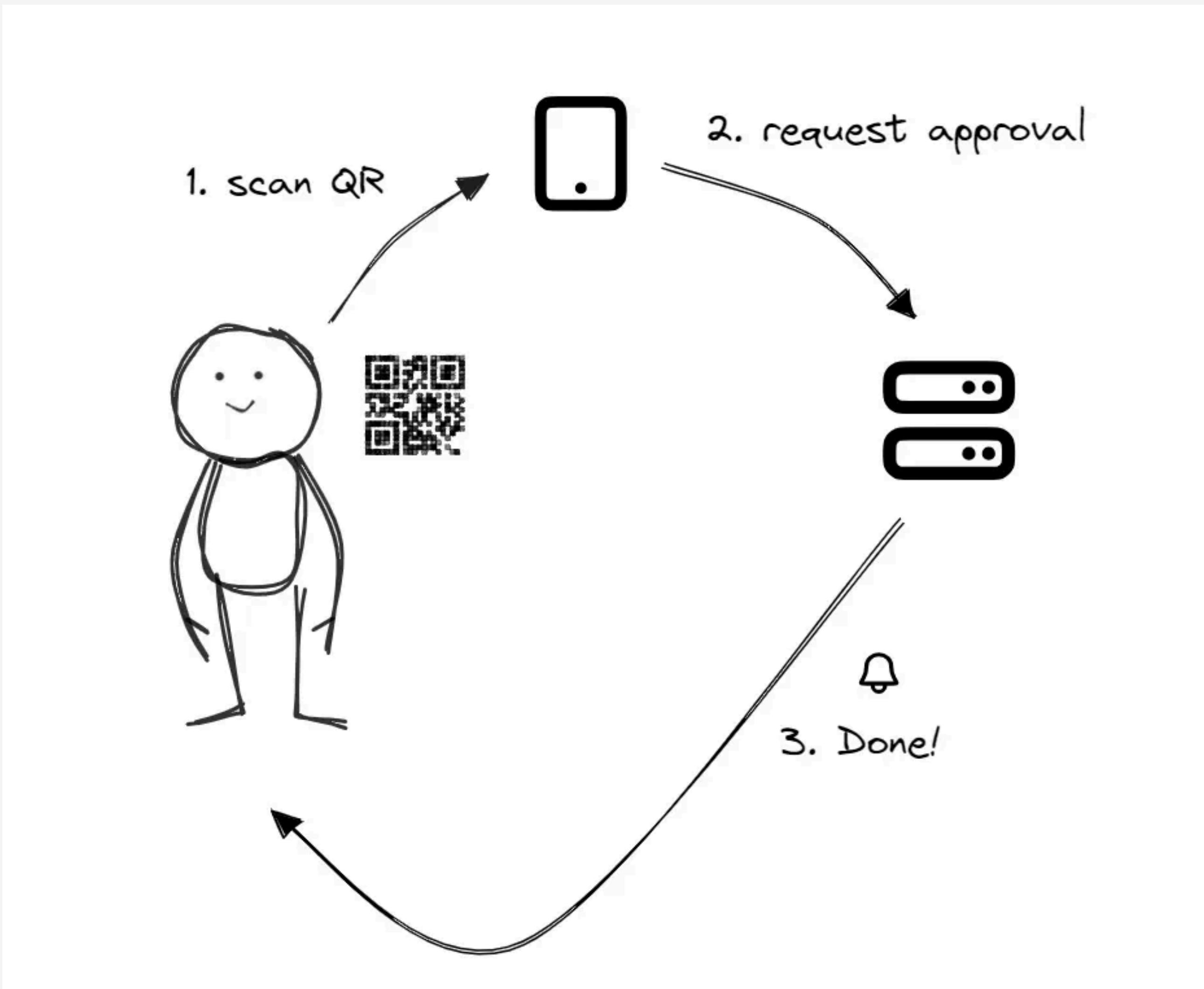
폴더구조 - 모듈 기반



- 같은 컨텍스트에 있는 개념끼리 묶고, 다시 계층으로 나눔
- Nestjs의 모듈 개념과 잘 어울림



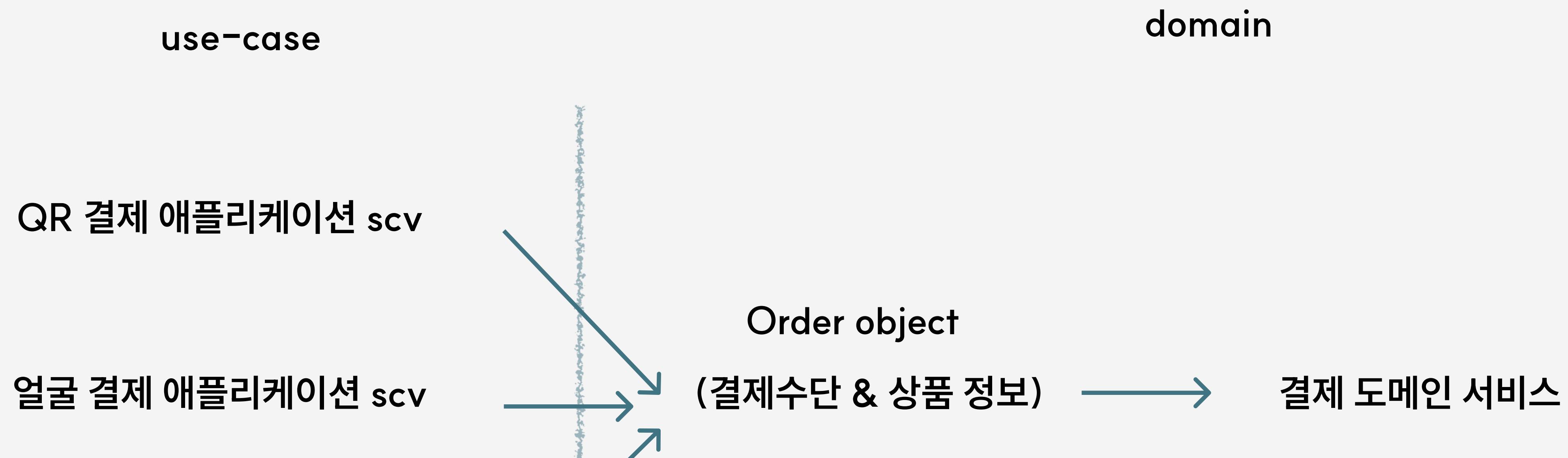
결제 과정



결제 과정 추상화

- QR: 결제 토큰에 저장된 결제 수단을 불러오고 입력된 상품의 구매 가능 여부를 확인한 뒤 결제 승인을 내리고 DB에 결제 내역을 저장
- 페이스 사인: 페이스 사인 인증 토큰으로 사용자가 선택한 결제 수단을 불러오고 상품 검증, 승인, 내역 저장
- 쿠폰 결제: 쿠폰을 불러오고 사용 가능한 쿠폰인지 확인한 뒤 상품 검증, 승인, 내역 저장

“결제 애플리케이션 서비스는 지급 수단을 불러오고 결제 로직을 호출한다”



```
export abstract class Order {
    readonly totalPrice: number

    constructor(
        readonly products: OrderProduct[],
        readonly transactionType: TransactionType,
        readonly purchaseType: PurchaseType
    ) {
        this.validateProducts()
        this.totalPrice = this.calculateTotalPrice()
    }

    abstract getUser(): Promise<User | null>

    public getProductsNames(): string[] {
        return // ...
    }

    private validateProducts() {
        if (this.products.length === 0) {
            throw new NoProductsInOrder()
        }

        for (const { product, amount } of this.products) {
            if (amount < 1) {
                throw new WrongAmount()
            }

            if (product.disabled) {
                throw new DisabledProductExists()
            }
        }
    }

    private calculateTotalPrice() {
        if(this.totalPrice !== undefined) return this.totalPrice
        return // ...
    }
}
```

- **GeneralCardOrder**

- **CouponOrder**

```
async approve(  
    command: ApproveRequestDto,  
    token: string,  
    dch: QRTransactionApproveDCH  
): Promise<TransactionApproveResponseDto> {  
    if (Coupon.isCoupon(token)) {  
        return this.couponTransactionService.approve({  
            coupon: token,  
            ...command,  

```

```
async approve(  
    command: ApproveRequestDto,  
    token: string,  
    dch: QRTransactionApproveDCH  
): Promise<TransactionApproveResponseDto> {  
    if (Coupon.isCoupon(token)) {  
        return this.couponTransactionService.approve({  
            coupon: token,  
            ...command,  
        })  
    }  
  
    const kiosk = this.contextService.getContextKiosk({ transactionId: true })  
    const paymentMethod = await this.getPaymentMethod(token, dch)  
  
    if (!paymentMethod) {  
        throw new WrongPayToken()  
    }  
  
    const orderProducts = await this.transactionDomainService.mapProducts(command.products)  
    const order = Order.createFromPaymentMethod(  
        TransactionType.APP_QR,  
        kiosk,  
        orderProducts,  
        paymentMethod  
    )  
  
    const transaction = await this.transactionDomainService.approve(order)  
  
    return {  
        status: transaction.status,  
        message: transaction.statusMessage,  
        totalPrice: transaction.totalPrice,  
    }  
}
```

```
@Transactional()
async approve(
    command: FaceSignTransactionApproveRequestDto,
    headers: FaceSignTransactionApproveHeaders,
    dch: FaceSignTransactionApproveDCH
): Promise<TransactionApproveResponseDto> {
    const kiosk = this.contextService.getContextKiosk({ transactionId: true })

    const userId =
        dch?.userId ??
        (await this.faceSignPaymentPinService.getUserIdFromPaymentPinOTP(
            kiosk.transactionId,
            headers.paymentPinOtp
        ))

    if (!userId) {
        throw new InvalidOTP()
    }

    const user = await this.userRepo.findByIdOrFail(userId)

    const paymentMethod = await this.paymentMethodRepo.findByIdOrFail(
        command.paymentMethodId,
        user.id
    )

    const orderProducts = await this.transactionDomainService.mapProducts(
        command.products
    )
    const order = Order.createFromPaymentMethod(
        TransactionType.FACESHOW,
        kiosk,
        orderProducts,
        paymentMethod
    )

    const transaction = await this.transactionDomainService.approve(order)
```

```
@Transactional()
async approve(
    command: FaceSignTransactionApproveRequestDto,
    headers: FaceSignTransactionApproveHeaders,
    dch: FaceSignTransactionApproveDCH
): Promise<TransactionApproveResponseDto> {
    const kiosk = this.contextService.getContextKiosk({ transactionId: true })

    const userId =
        dch?.userId ??
        (await this.faceSignPaymentPinService.getUserIdFromPaymentPinOTP(
            kiosk.transactionId,
            headers.paymentPinOtp
        ))

    if (!userId) {
        throw new InvalidOTP()
    }

    const user = await this.userRepo.findByIdOrFail(userId)

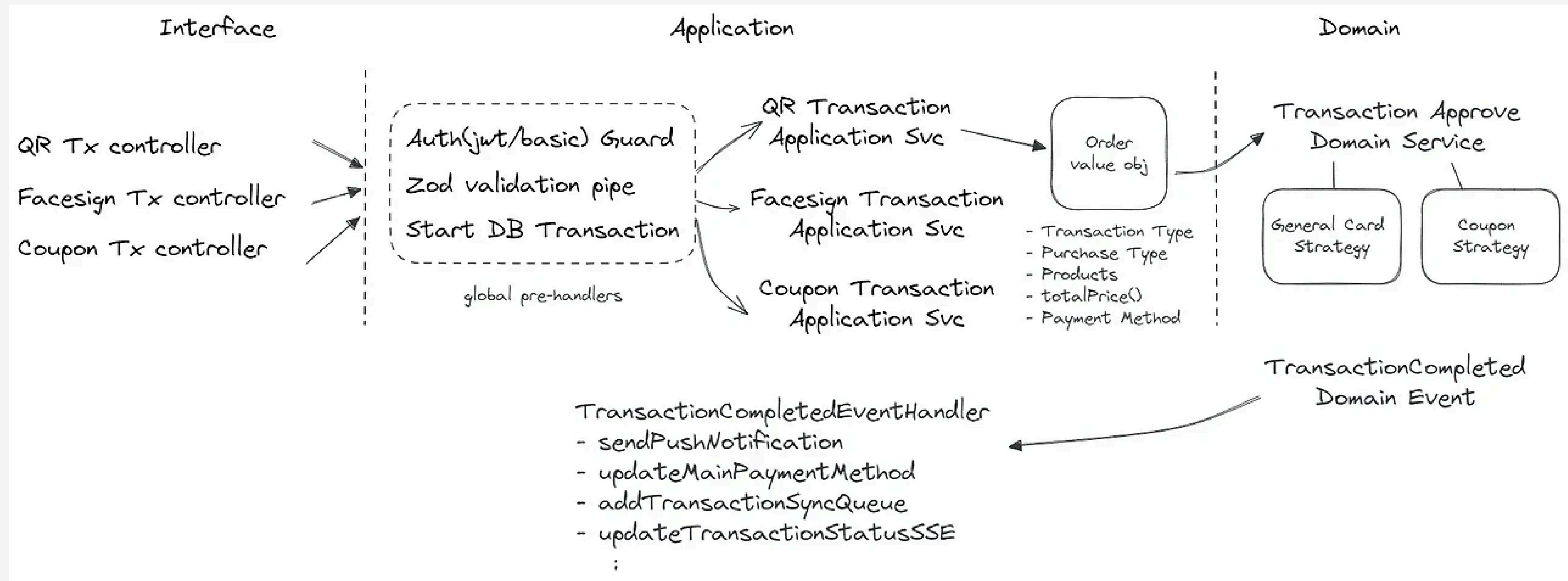
    const paymentMethod = await this.paymentMethodRepo.findByIdOrFail(
        command.paymentMethodId,
        user.id
    )

    const orderProducts = await this.transactionDomainService.mapProducts(
        command.products
    )
    const order = Order.createFromPaymentMethod(
        TransactionType.FACESHOW,
        kiosk,
        orderProducts,
        paymentMethod
    )

    const transaction = await this.transactionDomainService.approve(order)
```

결제 완료 이벤트

- 결제 성공 및 실패시 모두 발생되는 이벤트
- 알림 전송, 주 결제수단 변경, 재고 동기화, 결제 상태 SSE 업데이트
- 결과적 정합성을 요구되는 로직
- 재시도 정책을 타이트하게 가져갈 수 있는 장기실행 로직





쉽게 테스트 코드를 만들 수 있도록 설계



사이드 이펙트 없는 피처 개발



인지부하가 적인 소스코드



DI, 3rd party integration 등 기술적 구현 관심을 줄여준 Nestjs

- unit tests

```
187 pass
23 todo
0 fail
331 expect() calls
Ran 210 tests across 24 files. [25.07s]
```

- integration tests

```
64 pass
0 fail
168 expect() calls
Ran 64 tests across 17 files. [8.54s]
```

- e2e tests

```
42 pass
6 todo
0 fail
85 expect() calls
Ran 48 tests across 7 files. [6.01s]
```

감사합니다

- javien@daangn.com
- in/javien