1. 핵심 컨셉
현금 거래 없음 (사업자등록 불필요)
포인트는 활동으로만 획득 (충전X, 환전X)
1:1 쿠폰 교환 + 포인트 차액 정산
2. 포인트 시스템 설계

sql
```
-- 포인트 획득 규칙 테이블
CREATE TABLE point_rules (
    rule_id INT PRIMARY KEY AUTO_INCREMENT,
    activity_type ENUM(
        'DAILY_LOGIN',          -- 일일 출석: 10P
        'COUPON_REGISTER',      -- 쿠폰 등록: 50P
        'TRADE_COMPLETE',       -- 거래 완료: 30P
        'REVIEW_WRITE',         -- 거래 후기: 20P
        'COUPON_SHARE',         -- 쿠폰 정보 공유: 15P
        'REFERRAL',             -- 친구 초대: 100P
        'FIRST_TRADE'           -- 첫 거래: 200P
    ) UNIQUE,
    points INT NOT NULL,
    daily_limit INT DEFAULT NULL,
    description VARCHAR(255)
);

-- 사용자 포인트 이력 (현금 관련 없음)
CREATE TABLE user_points (
    point_id BIGINT PRIMARY KEY AUTO_INCREMENT,
    user_id BIGINT NOT NULL,
    points INT NOT NULL,
    activity_type VARCHAR(50),
    description VARCHAR(255),
    balance_after INT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    INDEX idx_user_date (user_id, created_at)
);
```
3. P2P 에스크로 시스템 설계
3.1 거래 유형

sql

```
-- 쿠폰 교환 메인 테이블
CREATE TABLE coupon_exchanges (
    exchange_id BIGINT PRIMARY KEY AUTO_INCREMENT,

    -- 거래 당사자 A
    user_a_id BIGINT NOT NULL,
    coupon_a_id BIGINT NOT NULL,

    -- 거래 당사자 B
    user_b_id BIGINT NOT NULL,
    coupon_b_id BIGINT,  -- NULL이면 포인트만 제공

    -- 차액 정산 (쿠폰 가치 차이)
    point_difference INT DEFAULT 0,  -- A가 B에게 줄 포인트 (음수면 반대)

    -- 거래 상태
    status ENUM(
        'PROPOSED',       -- A가 교환 제안
        'ACCEPTED',       -- B가 수락
        'LOCKED',         -- 양쪽 자산 잠금
        'CONFIRMED_A',    -- A 확인 완료
        'CONFIRMED_B',    -- B 확인 완료
        'COMPLETED',      -- 거래 완료
        'CANCELLED',      -- 거래 취소
        'EXPIRED'         -- 기한 만료
    ) DEFAULT 'PROPOSED',

    proposed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    expires_at TIMESTAMP DEFAULT (CURRENT_TIMESTAMP + INTERVAL 24
HOUR),
    completed_at TIMESTAMP NULL,

    FOREIGN KEY (user_a_id) REFERENCES users(user_id),
    FOREIGN KEY (user_b_id) REFERENCES users(user_id),
    INDEX idx_status (status),
    INDEX idx_expires (expires_at)
);
```

3.2 에스크로 잠금 관리

sql

```
-- 에스크로 잠금 상태 관리
```

```
CREATE TABLE escrow_locks (
    lock_id BIGINT PRIMARY KEY AUTO_INCREMENT,
    exchange_id BIGINT NOT NULL,

    -- 잠긴 자산 정보
    locked_type ENUM('COUPON', 'POINT') NOT NULL,
    user_id BIGINT NOT NULL,

    -- 쿠폰인 경우
    user_coupon_id BIGINT,

    -- 포인트인 경우
    locked_points INT,

    -- 잠금 상태
    is_locked BOOLEAN DEFAULT TRUE,
    locked_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    released_at TIMESTAMP NULL,

    FOREIGN KEY (exchange_id) REFERENCES coupon_exchanges(exchange_id),
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    INDEX idx_exchange (exchange_id),
    INDEX idx_user_locked (user_id, is_locked)
);
```

4. 거래 프로세스 상세 구현

4.1 교환 제안 플로우

python

```
class CouponExchangeService:

    def propose_exchange(self, proposer_id, my_coupon_id,
                         target_coupon_id, point_adjustment=0):
        """
        1:1 쿠폰 교환 제안
        point_adjustment: 내가 추가로 줄(+) 또는 받을(-) 포인트
        """

        # 1. 내 쿠폰 검증
        if not self.validate_coupon_ownership(proposer_id, my_coupon_id):
            raise Exception("내 쿠폰이 아니거나 이미 거래중")
```

```
        # 2. 상대 쿠폰 확인
        target_owner = self.get_coupon_owner(target_coupon_id)

        # 3. 포인트 잔액 확인 (내가 포인트를 줘야 하는 경우)
        if point_adjustment > 0:
            if not self.check_point_balance(proposer_id, point_adjustment):
                raise Exception("포인트 부족")

        # 4. 교환 제안 생성
        exchange = self.create_exchange(
            user_a_id=proposer_id,
            coupon_a_id=my_coupon_id,
            user_b_id=target_owner,
            coupon_b_id=target_coupon_id,
            point_difference=point_adjustment
        )

        # 5. 24시간 자동 만료 설정
        self.schedule_expiration(exchange.id, hours=24)

        # 6. 상대방에게 알림
        self.notify_user(target_owner, f"쿠폰 교환 제안이 도착했습니다!")

        return exchange
```

4.2 교환 수락 및 에스크로 잠금

```
python
def accept_exchange(self, exchange_id, acceptor_id):
    """
    교환 제안 수락 -> 에스크로 자동 잠금
    """

    exchange = self.get_exchange(exchange_id)

    # 1. 권한 확인
    if exchange.user_b_id != acceptor_id:
        raise Exception("권한 없음")

    # 2. 양쪽 자산 동시 잠금 (트랜잭션)
    with database.transaction():
        # A의 쿠폰 잠금
```

```
        self.lock_asset(
            exchange_id=exchange_id,
            user_id=exchange.user_a_id,
            asset_type='COUPON',
            coupon_id=exchange.coupon_a_id
        )

        # B의 쿠폰 잠금
        self.lock_asset(
            exchange_id=exchange_id,
            user_id=exchange.user_b_id,
            asset_type='COUPON',
            coupon_id=exchange.coupon_b_id
        )

        # 포인트 차액이 있는 경우
        if exchange.point_difference > 0:
            # A가 B에게 포인트를 줘야 함
            self.lock_asset(
                exchange_id=exchange_id,
                user_id=exchange.user_a_id,
                asset_type='POINT',
                points=exchange.point_difference
            )
        elif exchange.point_difference < 0:
            # B가 A에게 포인트를 줘야 함
            self.lock_asset(
                exchange_id=exchange_id,
                user_id=exchange.user_b_id,
                asset_type='POINT',
                points=abs(exchange.point_difference)
            )

        # 상태 변경
        exchange.status = 'LOCKED'
        exchange.save()

    # 3. 양쪽에 쿠폰 임시 전달 (사용은 불가)
    self.transfer_coupons_to_escrow(exchange_id)

    # 4. 72시간 내 양쪽 확인 필요
```

```
        self.notify_both_users(exchange_id,
            "쿠폰이 전달되었습니다. 72시간 내 확인해주세요.")
```

4.3 쿠폰 확인 및 최종 완료

```
python
def confirm_receipt(self, exchange_id, user_id):
    """
    받은 쿠폰 확인 (양쪽 모두 확인해야 완료)
    """

    exchange = self.get_exchange(exchange_id)

    # 1. 확인 상태 업데이트
    if user_id == exchange.user_a_id:
        exchange.status = 'CONFIRMED_A'
    elif user_id == exchange.user_b_id:
        exchange.status = 'CONFIRMED_B'

    # 2. 양쪽 모두 확인했는지 체크
    confirmations = self.get_confirmations(exchange_id)

    if confirmations == 2:  # 양쪽 모두 확인
        with database.transaction():
            # 3. 쿠폰 소유권 최종 이전
            self.finalize_coupon_transfer(exchange_id)

            # 4. 포인트 정산
            self.settle_points(exchange_id)

            # 5. 에스크로 잠금 해제
            self.release_all_locks(exchange_id)

            # 6. 거래 완료
            exchange.status = 'COMPLETED'
            exchange.completed_at = datetime.now()
            exchange.save()

            # 7. 거래 완료 보상 포인트 지급
            self.give_activity_points(exchange.user_a_id, 'TRADE_COMPLETE', 30)
            self.give_activity_points(exchange.user_b_id, 'TRADE_COMPLETE', 30)
```

4.4 자동 처리 시스템

python
```
class AutoEscrowProcessor:
    """
    자동 처리 스케줄러 (1시간마다 실행)
    """

    def process_expired_proposals(self):
        """24시간 지난 제안 자동 취소"""
        expired = self.get_expired_proposals()
        for exchange in expired:
            self.cancel_exchange(exchange.id)

    def process_unconfirmed_trades(self):
        """72시간 동안 확인 안된 거래 처리"""
        unconfirmed = self.get_unconfirmed_trades(hours=72)

        for exchange in unconfirmed:
            if exchange.confirmed_a and not exchange.confirmed_b:
                # A만 확인 -> B 페널티
                self.penalize_user(exchange.user_b_id, points=-50)
                self.complete_trade_forced(exchange.id)

            elif exchange.confirmed_b and not exchange.confirmed_a:
                # B만 확인 -> A 페널티
                self.penalize_user(exchange.user_a_id, points=-50)
                self.complete_trade_forced(exchange.id)

            else:
                # 둘 다 확인 안함 -> 거래 취소
                self.cancel_and_refund(exchange.id)
```

5. 거래 신뢰도 시스템

sql
```
-- 사용자 신뢰도 관리
CREATE TABLE user_trust_scores (
    user_id BIGINT PRIMARY KEY,
    total_trades INT DEFAULT 0,
    successful_trades INT DEFAULT 0,
    failed_trades INT DEFAULT 0,
    average_confirm_hours DECIMAL(5,2),
```

```
    trust_level ENUM('BRONZE', 'SILVER', 'GOLD', 'PLATINUM') DEFAULT 'BRONZE',
    last_calculated TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);

-- 신뢰도 레벨별 혜택
-- BRONZE: 기본 (동시 거래 3개)
-- SILVER: 동시 거래 5개, 수수료 10% 할인
-- GOLD: 동시 거래 10개, 수수료 20% 할인, 우선 매칭
-- PLATINUM: 무제한 거래, 수수료 면제, VIP 뱃지
```

6. 포인트만으로 쿠폰 구매

```
python
def request_coupon_for_points(self, buyer_id, coupon_id, offered_points):
    """
    쿠폰을 포인트로만 구매 요청
    """

    # 1. 포인트 잔액 확인
    if not self.check_point_balance(buyer_id, offered_points):
        raise Exception("포인트 부족")

    # 2. 쿠폰 소유자에게 제안
    owner_id = self.get_coupon_owner(coupon_id)

    # 3. 1:0 교환 생성 (내 쿠폰 없이 포인트만)
    exchange = self.create_exchange(
        user_a_id=buyer_id,
        coupon_a_id=None,  # 쿠폰 없음
        user_b_id=owner_id,
        coupon_b_id=coupon_id,
        point_difference=offered_points  # 포인트만 지불
    )

    return exchange
```

이렇게 설계하면 사업자 등록 없이 P2P 쿠폰 교환 플랫폼을 운영할 수 있습니다. 포인트는 순수하게 활동 보상으로만 얻을 수 있어 법적 이슈를 피할 수 있습니다.