

1 한 일 + 추가된 기능

1. fast api로 연결 완료
2. CA로 authentication 가능하도록 구현
3. 주문에 대해 가게 승인 자동으로 처리되도록 구현
4. RBAC로 권한 관리
 - 각 api를 호출할 수 있는 권한인 scope를 정의하고, 그 scope의 모음인 role을 정의.
 - 사용자는 group context 안에서 role을 부여받음으로써 권한 획득.
5. 사용자가 multi wallet을 사용할 수 있도록 구현
 - 하나의 wallet은 하나의 계정에만 연결 가능
 - 하나의 wallet에 대해 다른 chain id로 연결 가능
6. USDT, USDC 결제 지원
7. 가게 생성 및 관리 기능 구현

1.1 사용자

- 비 로그인 유저: 가게, 상품 목록과 상세 정보 조회, 로그인 요청 가능
- 로그인한 고객: 자신의 프로필과 세션 조회, 주문 생성과 추적, 결제 트랜잭션 해시 제출 가능.
- 가게 직원/멤버: 소속 가게의 정보 조회, 역할 템플릿에 따른 상품과 재고 관리 가능
- 가게 매니저: 가게 직원 권한에 더해, 가게 프로필 업데이트와 가게 정책 내에서의 직원 초대 가능
- 가게 오너: 가게 매니저 권한에 더해, RBAC 정책 내에서의 오너 전용 가게 관리 요청 가능. 오너 전환은 가게 자체 서비스로 제공되지 않음
- 플랫폼 admin: 모든 권한을 가진 주체. 가게를 생성해서 사용자에게 가게에 대한 권한을 부여할 수 있음.

1.2 API

- 자세한 사항은 [API 명세](#)에서 확인 가능합니다.
 - 추후 기능이 완전히 구현이 완료되면 gitbook으로 API 문서화 작업을 진행할 예정입니다.
- Operator 관련 api는 구현만 되어 있는 상태. 현재로서는 중요해 보이지 않지만 추후를 생각해서 남겨 놓은 상태.

1.3 시연

가게 생성 먼저 시연해보겠습니다.

Filter database objects

Search Select column Select filter Filter value

1 rows

public	user_id	wallet_address	role	active	last_login_at	created_at	updated_at
auth_group_invitations	2fb7b956-c624-4870-9362-043df5f7d75f	0x32b31c74fe628e9164996f727f0d11a3c49ec27f	ADMIN	true	NULL	2026-05-22T19:31:33.570312+09:00	2026-05-22T19:31:33.570312+09:00

Tables 34

- auth_group_invitations
- auth_group_memberships
- auth_groups
- auth_login_challenges
- auth_permissions
- auth_role_permissions
- auth_roles
- auth_sessions
- auth_user_profiles
- auth_user_wallets
- auth_users

- 기본적으로는 환경변수에서 설정해준 admin 계정만 존재합니다.
- 해당 admin의 권한으로 가게를 생성할 수 있습니다.
- 일반 user의 가게 생성 요청에 관한 기능은 아직 없습니다.

HTTP Token Payments Local Cookie Auth > Cookie Auth Happy Path > 1. Request login challenge

POST

Docs Params Authorization Headers 11 Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 {
2   "walletAddress": "{{walletAddress}}",
3   "domain": "{{loginDomain}}",
4   "uri": "{{loginUri}}",
5   "chainId": {{chainId}}
6 }

```

Body Cookies 1 Headers 6 Test Results 3/3 201 Created 201 ms 1.52 KB Save Response

{ } JSON Preview Visualize

domain	token-payments.local
expirationTime	2026-05-22T10:45:39.266679+00:00
expiresAt	2026-05-22T10:45:39.266679+00:00
issuedAt	2026-05-22T10:40:39.266679+00:00
nonce	e34074c812c241a0b731ac0eb20f8124
signatureVerification {7}	
signingMessage	token-payments.local wants you to sign in with your Ethereum account: 0x32b31c74fe628e9164996f727f0d11a3c49ec27f URI: https://token-payments.local Version: 1 Chain ID: 1337 Nonce: e34074c812c241a0b731ac0eb20f8124 Issued At: 2026-05-22T10:40:39.266679+00:00 Expiration Time: 2026-05-22T10:45:39.266679+00:00
uri	https://token-payments.local
version	1
walletAddress	0x32b31c74fe628e9164996f727f0d11a3c49ec27f

Figure 1: admin으로 로그인 challenge

- login을 시도하면 siwe 규약에 맞는 message가 발급됩니다.
- 해당 signingMessage를 복사해서 브라우저에서 meta mask로 sign을 해보겠습니다.

```

(async () => {
  if (!window.ethereum) {
    console.error("메타마스크가 설치되어 있지 않습니다.");
    return;
  }

  // 1. 메타마스크 지갑 연결 및 주소 가져오기
  const accounts = await window.ethereum.request({ method: 'eth_requestAccounts' });
  const fromAddress = accounts[1];

  // 2. SIWE 메시지 생성 (Address 부분을 실제 지갑 주소로 변경)
  const message = `token-payments.local wants you to sign in with your Ethereum account:
  ${fromAddress}

  URI: https://token-payments.local
  Version: 1
  Chain ID: 1337
  Nonce: e34074c812c241a0b731ac0eb20f8124
  Issued At: 2026-05-22T10:40:39.266679+00:00
  Expiration Time: 2026-05-22T10:45:39.266679+00:00`;

  try {
    // 3. 메타마스크 personal_sign 호출
    const signature = await window.ethereum.request({
      method: 'personal_sign',
      params: [message, fromAddress],
    });

    console.log("=== 테스트 결과 ===");
    console.log("사용된 메시지:\n", message);
    console.log("생성된 서명 값 (Signature):\n", signature);
  } catch (error) {
    console.error("서명 실패:", error);
  }
})();

```

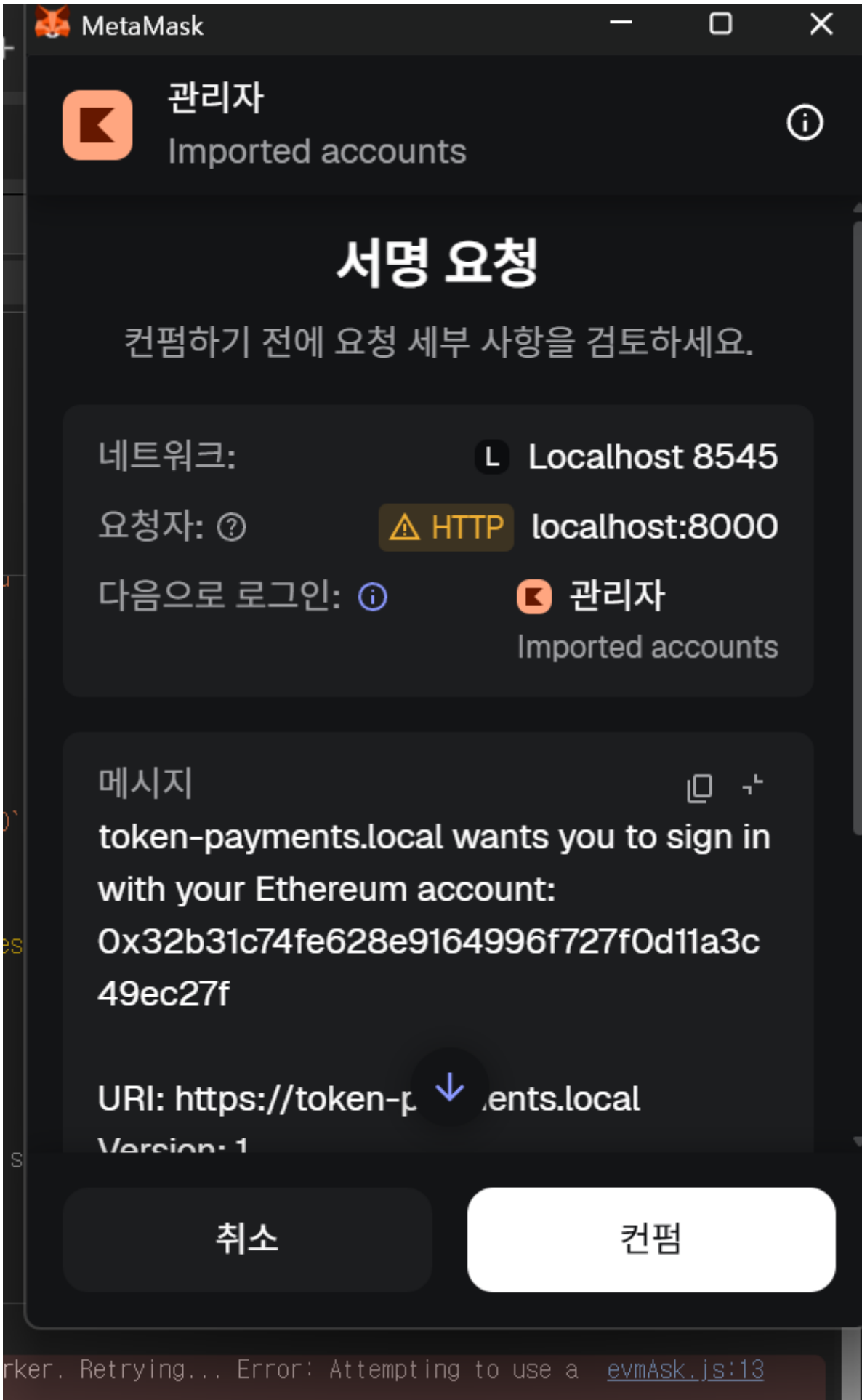
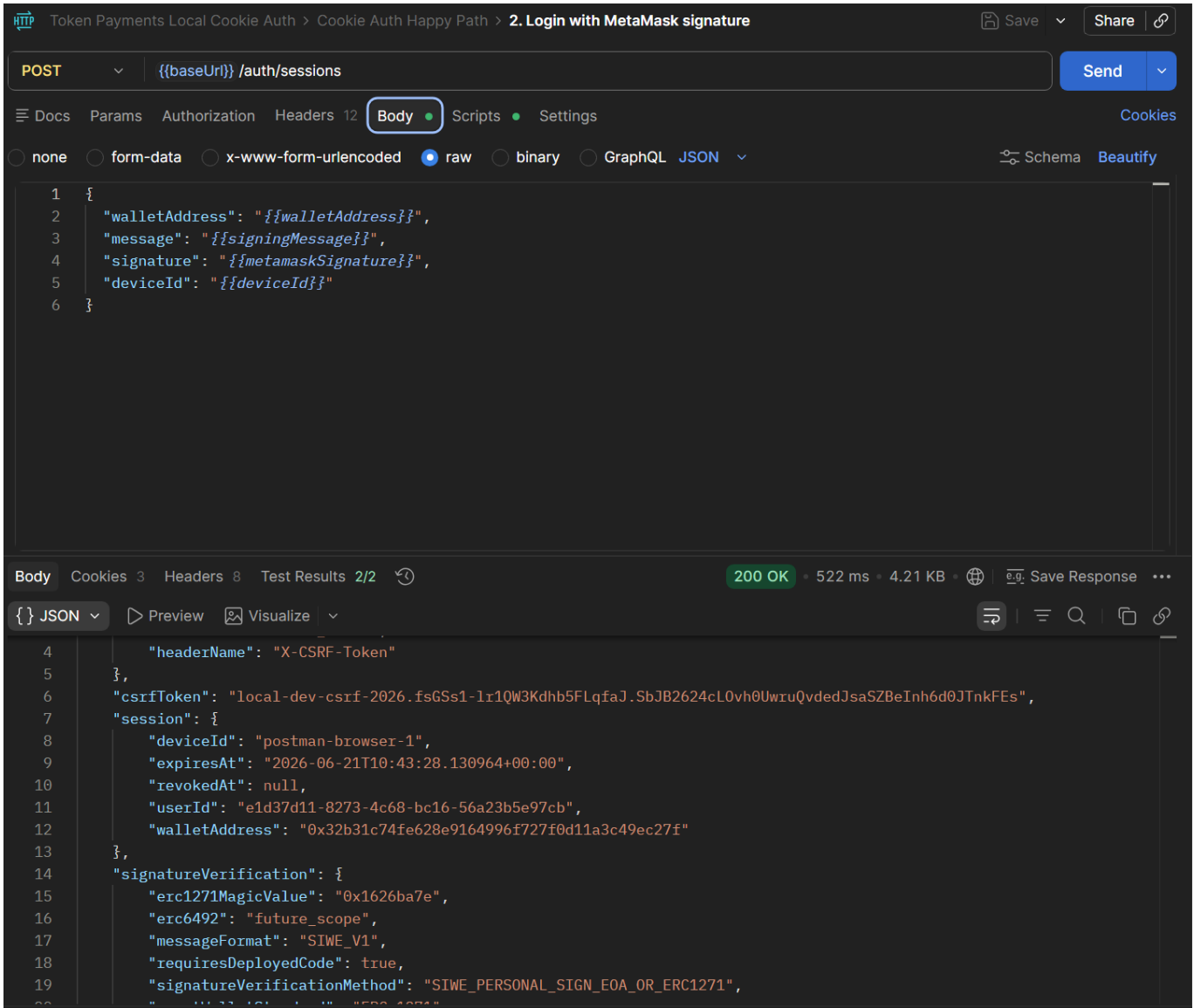


Figure 2: metamask 창

```
생성된 서명 값 (Signature): YM42:31  
  
0x8e86d6bd082088dfa56dcc6cf2bddb44126215a9631cfe7d948c389f0fc329ac77387c3d29cab57813d0788d6cfdbed57d0bd73c7  
efe00a923552f26142b020c1c
```

Figure 3: 서명 결과

- 서명을 가지고 다시 서버에 로그인 요청을 보내면 로그인에 성공하는 것을 확인할 수 있습니다.



- 이제 관리자의 권한으로 가게를 생성해보겠습니다.
- 먼저 가게 생성 요청을 보낸 사람이 현재 우리 서비스에 가입이 안된 유저라고 가정하겠습니다.
- 먼저 해당 유저의 wallet address로 계정을 생성해줍니다.

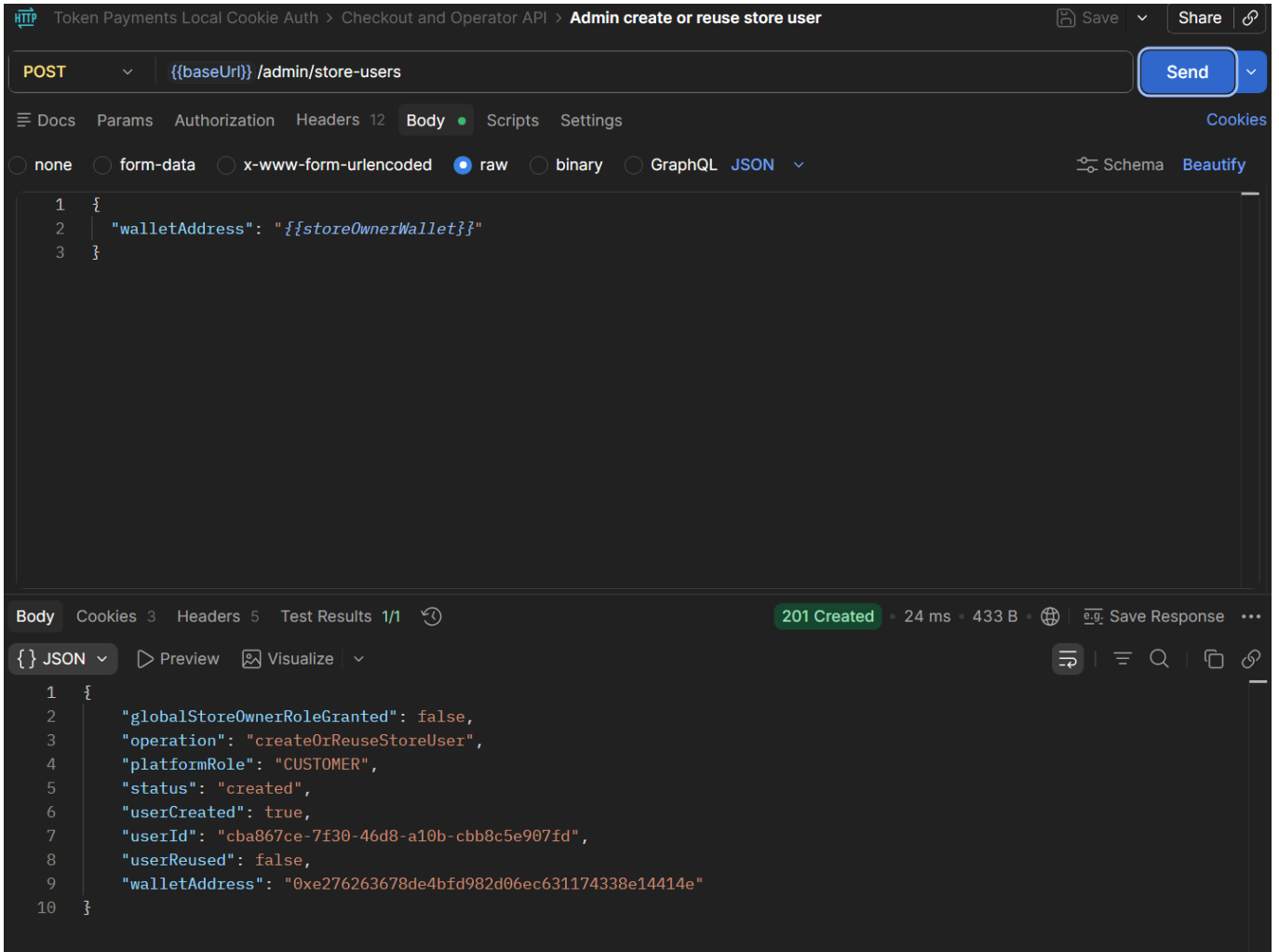


Figure 4: 계정 생성 api

The screenshot shows a database interface with the 'auth_users' table selected. The table has columns for `user_id`, `wallet_address`, `role`, `active`, `last_login_at`, `created_at`, and `updated_at`. There are two rows of data displayed.

user_id	wallet_address	role	active	last_login_at	created_at	updated_at
e1d37d11-8273-4c68-bc16-56a23b5e97cb	0x32b31c74fe628e9164996f727f0d11a3c49ec27f	ADMIN	true	2026-05-22T19:43:28.130964+09:00	2026-05-22T19:40:14.332678+09:00	2026-05-22T19:43:28.130402+09:00
cba867ce-7f30-46d8-a10b-cbb8c5e907fd	0xe276263678de4bfd982d06ec631174338e14414e	CUSTOMER	true	NULL	2026-05-22T19:45:50.56013+09:00	2026-05-22T19:45:50.56013+09:00

- 계정이 이제는 2개가 생성되어 있는 것을 확인할 수 있습니다.
- 이어서 가게를 생성해보겠습니다.
- 참고로 store wallet과 store owner wallet은 다른 지갑으로 설정할 수 있습니다.
- 이번 시연에서는 동일하게 설정하겠습니다.

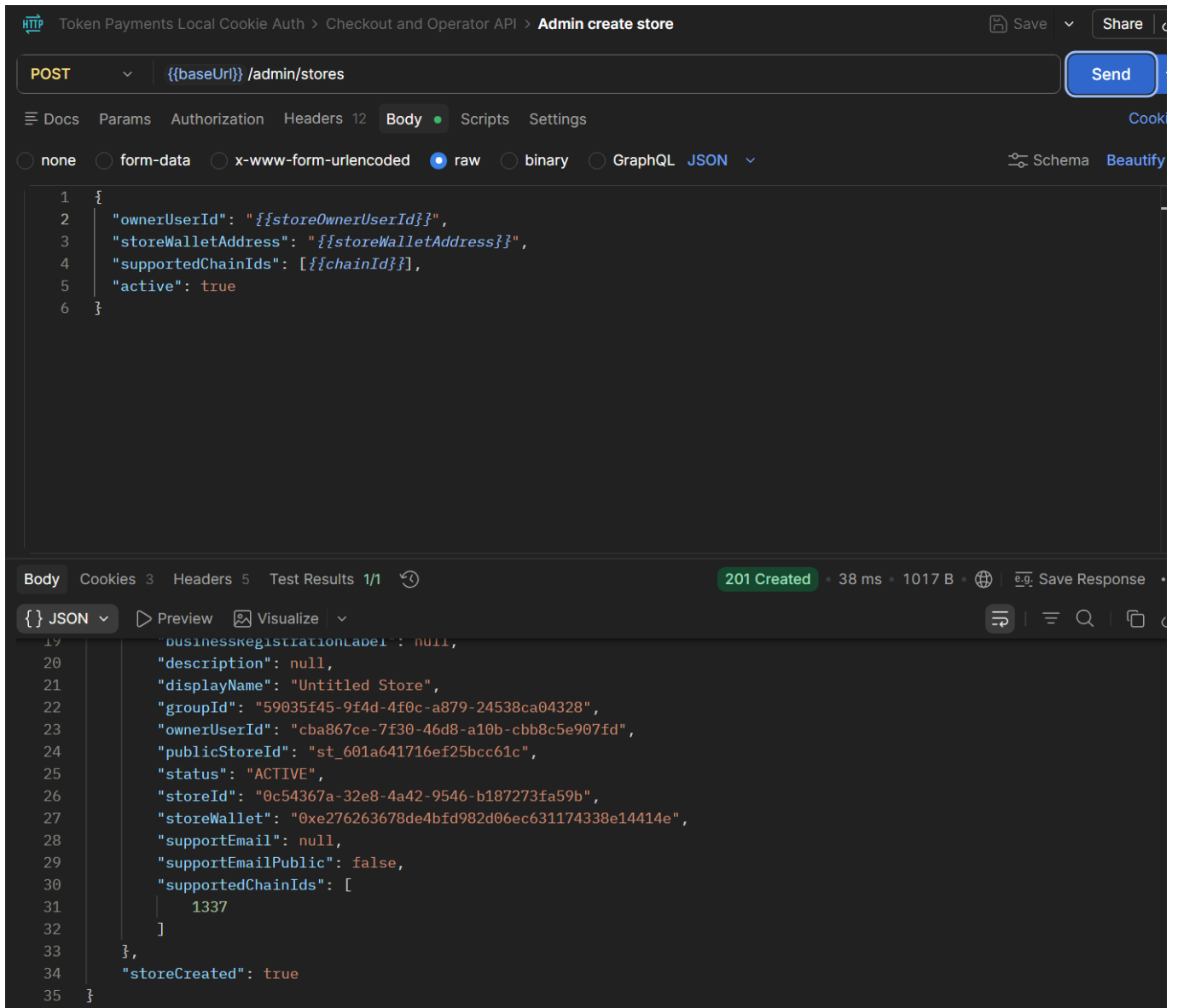


Figure 5: 가게 생성

- 가게 생성이 완료되었습니다.
- 그런데 시연하면서 깜빡하고 display name같은 가게 정보를 입력 안해줬네요.
- 하지만 admin 권한으로 가게 정보를 수정하는 것도 가능합니다.
- 먼저 가게 list를 조회해보겠습니다.

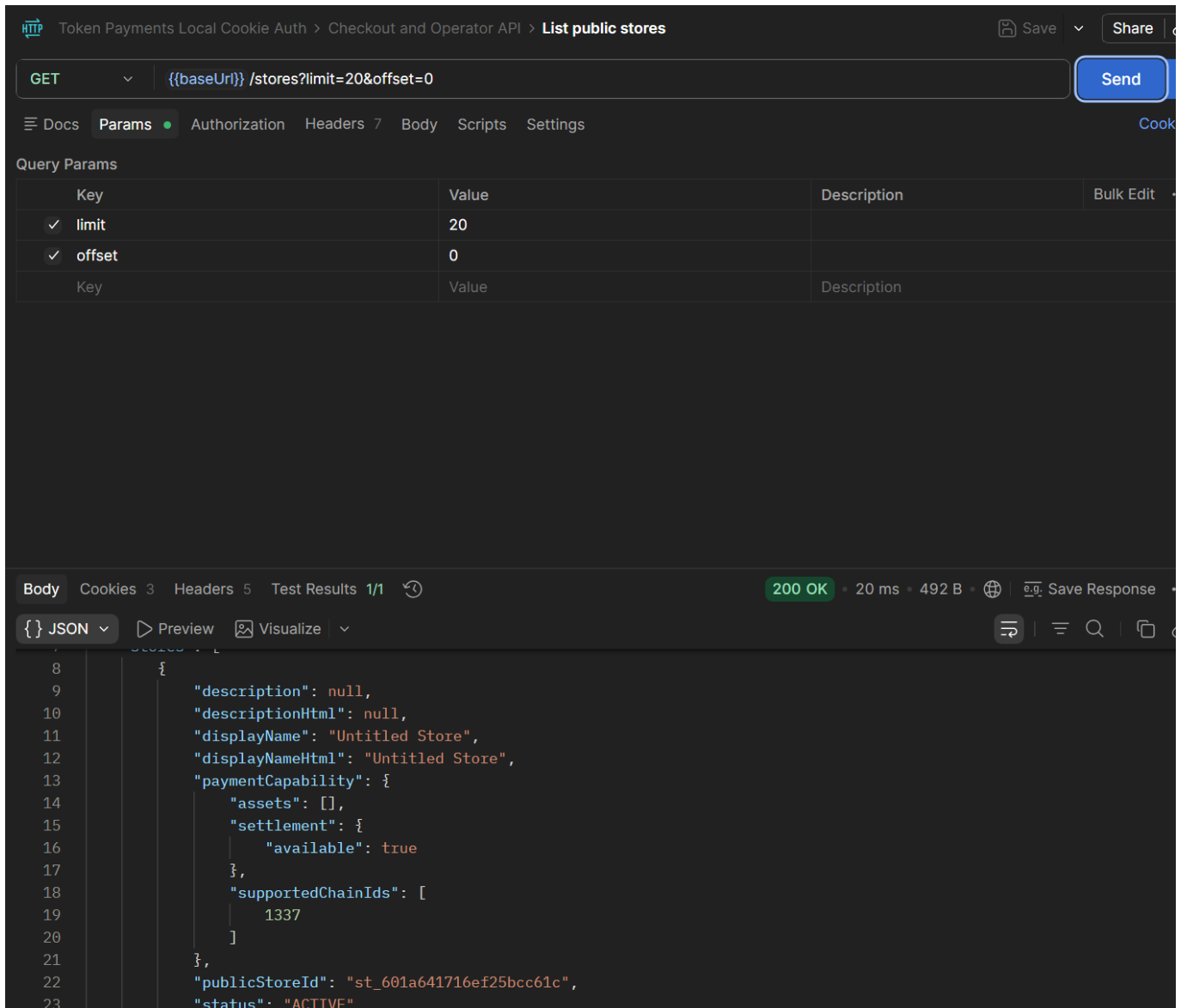


Figure 6: 가게 조회

- 우리가 만든 가게가 조회되고 있습니다.
- 이름이랑 설명은 아직 제대로 입력이 안되어 있습니다.
- 여기서 나온 public store id로 가게 프로필을 업데이트 해보겠습니다.

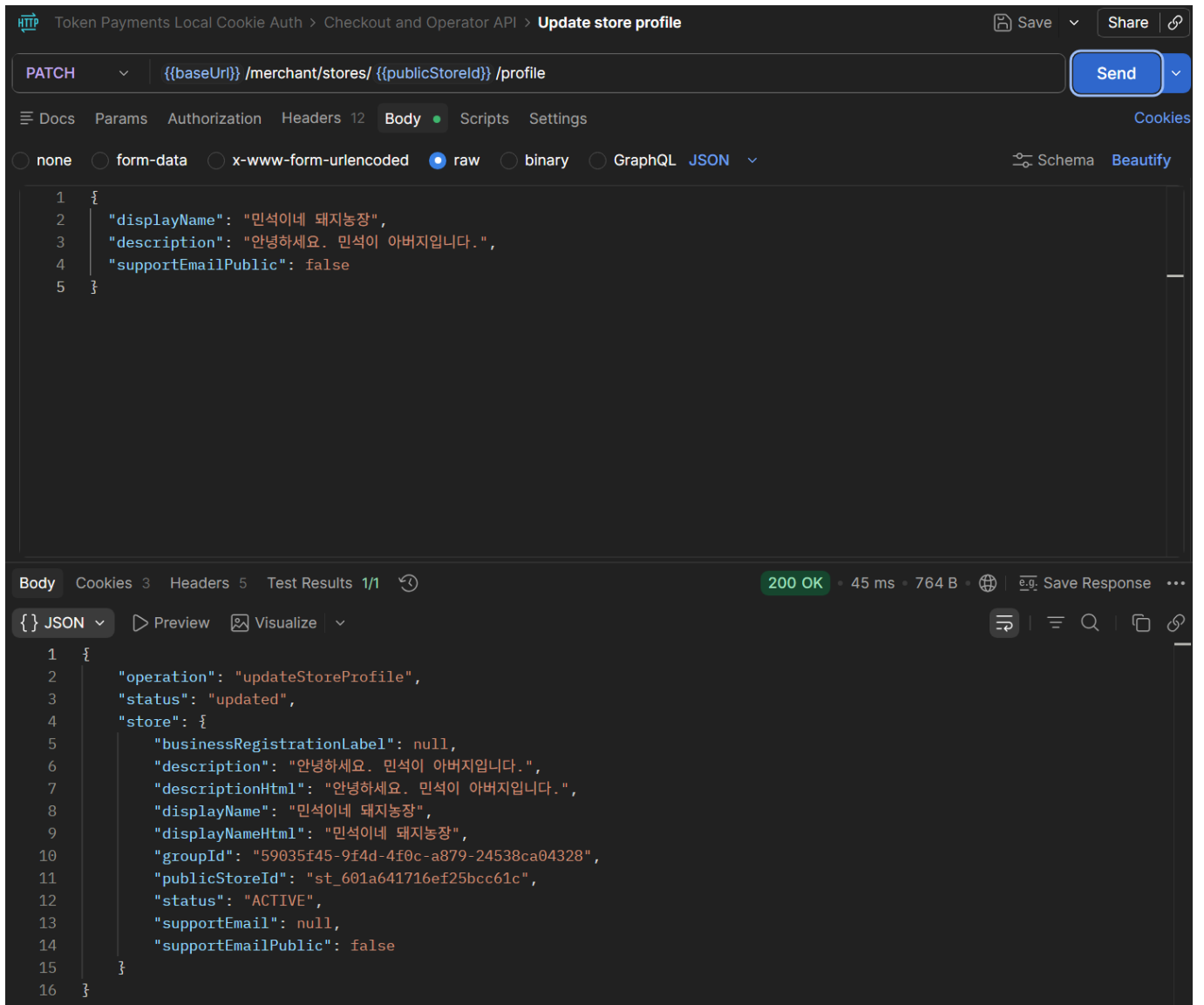


Figure 7: 가게 프로필 업데이트

- 네 가게 프로필을 업데이트 했습니다.

Token Payments Local Cookie Auth > Checkout and Operator API > List public stores

GET `{{baseUrl}} /stores?limit=20&offset=0` Send

Params • Authorization Headers 7 Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit	...
✓ limit	20			
✓ offset	0			
Key	Value	Description		

Body Cookies 3 Headers 5 Test Results 1/1 200 OK • 21 ms • 734 B • Save Response

JSON Preview Visualize

```

2   "pagination": {
3     "limit": 20,
4     "nextOffset": null,
5     "offset": 0
6   },
7   "stores": [
8     {
9       "description": "안녕하세요. 민석이 아버지입니다.",
10      "descriptionHtml": "안녕하세요. 민석이 아버지입니다.",
11      "displayName": "민석이네 돼지농장",
12      "displayNameHtml": "민석이네 돼지농장",
13      "paymentCapability": {
14        "assets": [],
15      }

```

- 다시 조회했을 때도 업데이트된 가게 프로필이 조회되는 것을 확인할 수 있습니다.
- 이제 정상적으로 가게를 생성했으니 가게 주인 계정으로 로그인해서 품목들을 추가해보겠습니다.

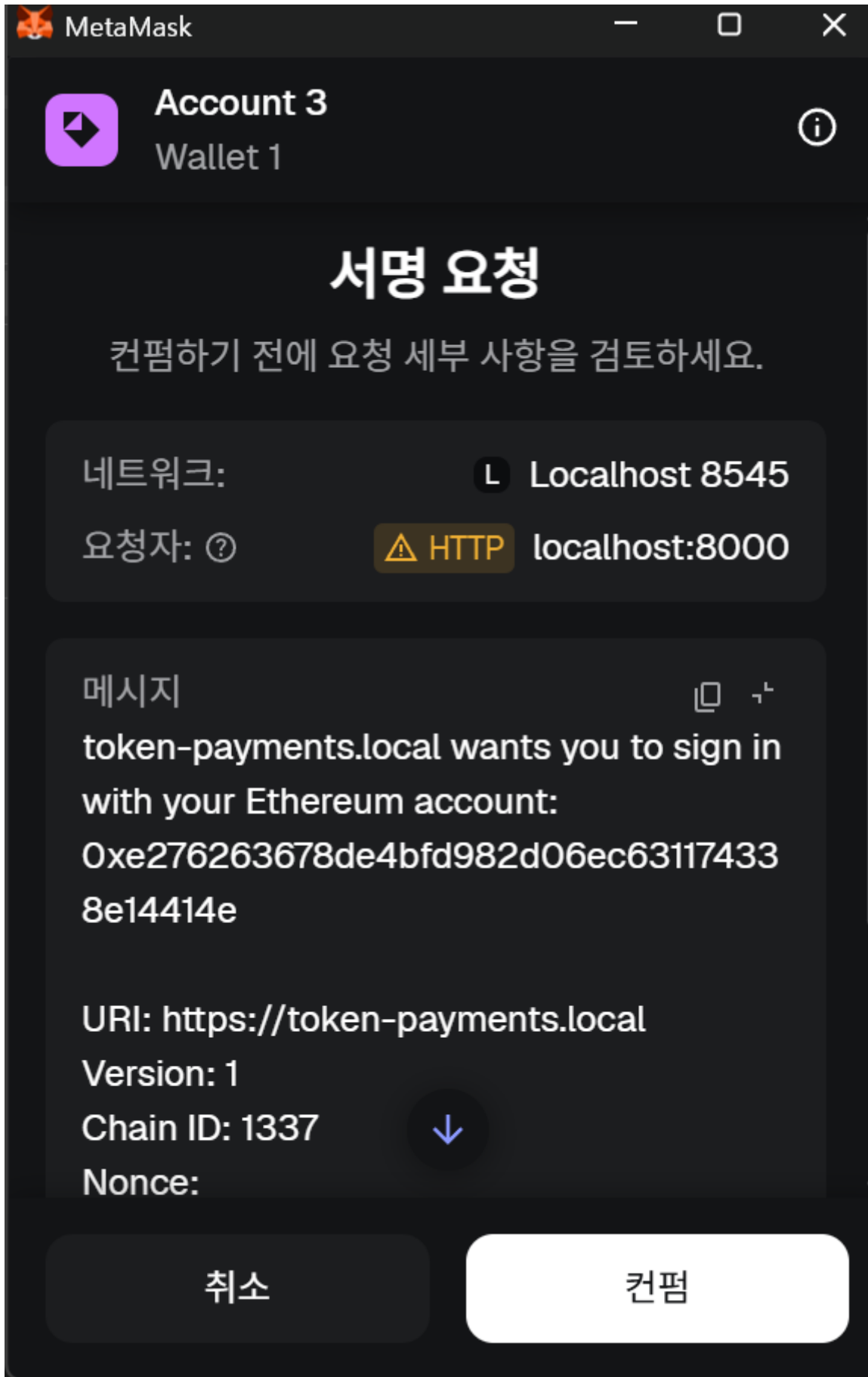


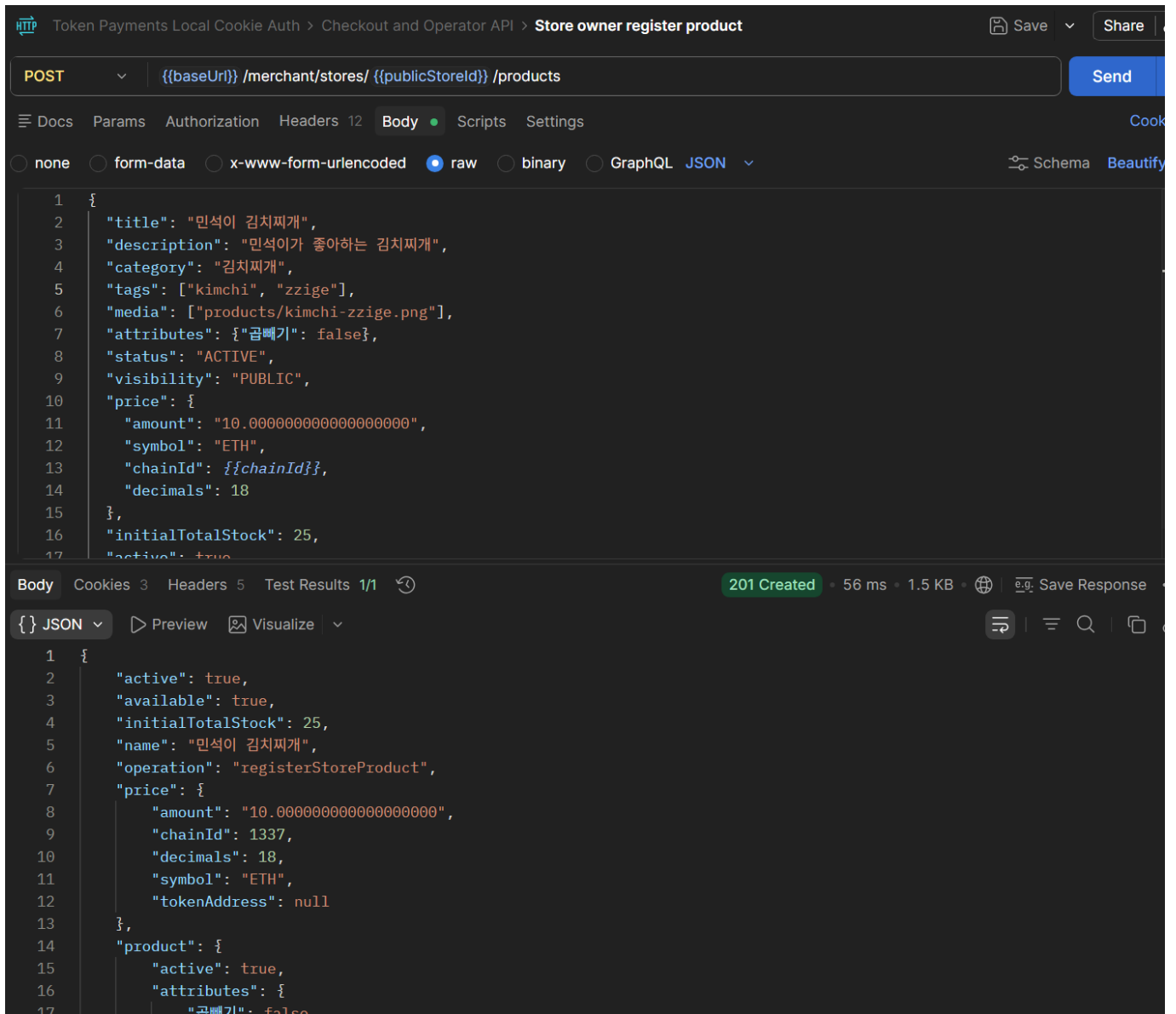
Figure 8: 가게 주인 로그인

- 이제 가게 주인의 권한으로 가게 리스트를 조회해보겠습니다.

The screenshot shows an API client interface for a GET request to `{{baseUrl}}/merchant/stores`. The response status is `200 OK` with a response time of `29 ms` and a body size of `586 B`. The response body is displayed in JSON format:

```
1 {
2   "stores": [
3     {
4       "description": "안녕하세요. 민석이 아버지입니다.",
5       "descriptionHtml": "안녕하세요. 민석이 아버지입니다.",
6       "displayName": "민석이네 돼지농장",
7       "displayNameHtml": "민석이네 돼지농장",
8       "publicStoreId": "st_601a641716ef25bcc61c",
9       "status": "ACTIVE"
10    }
11  ]
12 }
```

- 정상적으로 가게 리스트가 조회되는 것을 확인할 수 있습니다.



- 품목도 추가해주었습니다. 음식점이라기 보다는 일종의 밀키트로 봐주시면 될 것 같습니다.
- 현재 상품 이미지를 업로드 하는 기능은 없습니다.
- 이제 일반 소비자 계정으로 로그인해서 해당 상품을 주문해보도록 하겠습니다.



Account 4

Wallet 1



서명 요청

컨펌하기 전에 요청 세부 사항을 검토하세요.

네트워크: **L** Localhost 8545

요청자: ⓘ **⚠ HTTP** localhost:8000

메시지



token-payments.local wants you to sign in with your Ethereum account:

Oxdd7f01c6b548cbaf406579ebd17f00972e65663b

URI: https://token-payments.local

Version: 1

Chain ID: 1337

Nonce:

7513751d02a4442a83f1754bcf991838

Issued At: 2026-05-

22T11:31:09.202704+00:00

Expiration Time: 2026-05-

22T11:36:09.202704+00:00

취소

컨펌

Figure 9: 소비자 로그인

Token Payments Local Cookie Auth > Checkout and Operator API > List public store products

GET {{baseUrl}} /stores/ {{publicStoreId}} /products?limit=20&offset=0&q=Local

Params Authorization Headers 7 Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
limit	20		
offset	0		
q	Local		

Body Cookies 3 Headers 5 Test Results 1/1 200 OK 20 ms 745 B Save Response

{ } JSON Preview Visualize

```
1 {
2   "pagination": {
3     "limit": 20,
4     "nextOffset": null,
5     "offset": 0
6   },
7   "products": [],
8   "store": {
9     "description": "안녕하세요. 민석이 아버지입니다.",
10    "descriptionHtml": "안녕하세요. 민석이 아버지입니다.",
11    "displayName": "민석이네 돼지농장",
12    "displayNameHtml": "민석이네 돼지농장",
13    "paymentCapability": {
14      "assets": [],
15      "settlement": {
16        "available": true
17      }
18    }
19  }
20 }
```

- 먼저 상품 목록을 조회합니다.
- 가게는 뜨는데 상품이 안뜨네요. 아직 버그가 있는 것 같습니다.



- 주문 생성이 완료된 모습입니다.

2 추가 작업 필요

- 통합 테스트
- 리팩토링
 - api에 아직 테스트용 데이터 반환값이 남아 있는 상태. 보안적으로 문제가 되지 않도록 수정이 필요하다.
 - 기능이 많이 추가되면서 legacy code랑 충돌이 나는 부분이 생겼다.
 - DB에 정규화가 덜 된 부분이 있다.

3 구현 예정 기능

1. media upload (상품 이미지에 한해서만)
2. 개인 정보(전화번호, 주소) DID로 관리
 - 서버에 개인정보를 아무것도 저장하지 않는 방향을 원하는 상태.
3. AI Agent Payment

4 먼 훗날 추가하면 좋지 않을까 하고 상상만 하고 있는 기능

1. 상품 리뷰 기능: 급해 보이는 기능은 아니다.
2. 검색 지원(elasticsearch): elastic search를 사용할 줄 모르기 때문에 일단은 우선순위가 낮다.
3. 이메일 인증으로 계정 복구 기능: 있으면 좋은 기능이라고 생각하지만 아직 어떻게 구현할지 감이 잡히지 않는 상태.
4. SCM: 가게 주인 입장에서 공급망 관리 기능을 제공하면 좋을 것 같다는 생각. 해당 기능이 구현된다면 AI Agent 협상 기능에서 공급망 vendor들과의 수수료 협상이 가능하도록 구현할 수 있을 것 같다.
 - 가게
 - 재고 보유 주체 선택, 수송 주체 선택, 소비자 방문 수령 / 방문 구매 가능 여부
 - 창고
 - 다크 스토어 / 풀필먼트 등, 다른 유저에게 공간 대여 가능
 - 수송
 - 항공, 소화물, 화물 트럭, 철도 화물, 해상 화물, 파이프라인, 드론, 오토바이, 자전거 등
 - 단, 우리 서비스가 중고거래 플랫폼처럼 되는 것을 원하지는 않는 상태.