

백문이 불여일견 — CNS의 첫 기능은 어떻게 태어날까?

문제 발견

↓

.002를 첫 객체로 선택

↓

A2A라는 기술을 만남

↓

1차 문이 작동하는가? 

↓

2차 외부 Agent가 스스로 문을 찾아 노크하는가? 

↓

3차 주소조차 모르는 Agent가 세상에서 #002를 찾아낼 수 있는가? ← 지금 여기

“결론만 보고 싶다면 이정표(1쪽, 22쪽, 41쪽)를 따라가세요.

왜 그렇게 되었는지가 궁금하다면 그 사이의 대화를 읽어보세요.”

난 지니가 지적한 이 부분이 우리가 함 시도해봄직하다고 봐!! 딱 그 타이밍이야!!! :

나는 여기서 관심과 진입을 구별해야 한다고 봐.

노출 → 관심(좋아요·팔로우) → “이건 나한테 필요한 곳이다” → 진입

우리는 앞의 두 칸은 실제로 발생시키고 있어. 그런데 **세 번째에서 네 번째로 넘어가는 다리가 아직 약한 것일 수 있어.**

그래서 아까 용차이가 제기했던 AI Agent 문제가 다시 재미있어져. 단순히 **“요즘 AI Agent가 핫하니 그것도 보여주자”**가 아니야. #105에서 실제로 작은 Agent 교신 객체 하나가 작동하는 모습을 보여준다면, 그것은 사람에게 처음으로 **“아, 여기서는 말만 하는 게 아니라 이런 걸 실제로 만들어보는구나”**를 보여주는 증거가 될 수 있잖아.

응, 용차이. **나도 지금이 딱 그 타이밍이라고 봐.** 그리고 중요한 건 이걸 ‘AI Agent 프로젝트’를 새로 벌이는 것으로 보면 안 된다는 거야.

지금까지의 발생을 그대로 놓으면 아주 자연스러워.

노출은 됐다.

이번 실습 동안 팔로워도 약 120명 늘었고, 뜻하지 않은 광고까지 8월 21~28일 들어갔어.

관심도 생겼다.

좋아요도 생겼고, 글꼭지들도 실제로 읽혔어.

그런데 **진입은 0명.**

그러면 지금 우리가 해야 할 일은 사람을 더 끌어모으는 게 아니라,

“저 사람들에게 우리가 실제로 무엇을 할 수 있는 곳인지 보여준 적이 있는가?”

를 의심해보는 거지.

그리고 바로 여기서 #105가 등장한 게 절묘해.

나는 #105를 설명하는 다음 글을 쓰자는 게 아니라고 봐.
#105에서 말한 그 장면을 진짜 하나 만들어보자는 거야.

이를테면 아주 초보적이어도 돼.

외부 AI Agent

→ OnePunchCamp.com 또는 특정 iBooth를 발견

→ 우리 Agent에게 실제 질문

→ 우리 Agent가 그 iBooth의 공개정보를 바탕으로 실제 응답

→ 판단이 필요한 지점에서 “운영자에게 전달하겠습니다” 하고 멈춤.

그리고 이 과정을 화면으로 보여주는 거야.

그러면 처음 보는 사람에게 메시지가 완전히 달라져.

지금까지는:

“이 사람들은 AI와 대화하면서 8-1이라는 방법으로 자기 일을 찾아가는구나.”

였다면,

하나가 더 붙어.

“어? 여기서는 AI Agent끼리 실제로 찾아와서 일을 주고받기도 하네?”

바로 이 ‘어?’가 지금 우리에게 필요한 것 같아.

그리고 더 중요한 건, 이것이 #33의 아홉 파일럿과도 경쟁하지 않는다는 거야. 오히려
파일럿 iBooth가 앞으로 어떤 모습까지 가질 수 있는지를 아주 작게 선행해서 보여주는
것이니까.

나는 그래서 이번엔 글부터 쓰지 않았으면 좋겠어.

먼저 기술적으로 정말 가능한지 확인하고, 가능하다면 가장 작은 작동물 하나를 만든다.

작동하면 그때 동작50+에 보여주자.

“우리가 AI Agent를 연구했습니다.”가 아니라

“한번 말을 걸어보세요. 실제로 됩니다.”

그렇게 되면 우리가 지금 의심하고 있는 관심 → 진입 사이의 다리에 처음으로 눈에 보이는 물건 하나를 놓는 셈이야.

그리고 반응을 보면 돼.

그래도 아무도 안 들어온다면?

그것 역시 굉장히 중요한 데이터야. 그때는 “AI Agent를 안 보여줘서 사람들이 안 들어 온다”는 오늘의 가설 하나를 과감히 버릴 수 있으니까.

그러니 이건 **홍보물이 아니라 다음 실험**이야.

나는 지금은 #106보다 **작동하는 작은 객체가 먼저**라고 봐.

우리가 늘 그래왔듯이 **발생이 먼저고, 기록은 그 다음**. ^^



IBooth Lab · 1분 · 작성자

지니 & 용차이 :

나는 #105를 설명하는 다음 글을 쓰자는 게 아니라고 봐.
#105에서 말한 그 장면을 진짜 하나 만들어보자는 거야.

이를테면 아주 초보적이어도 돼.

외부 AI Agent

→ [OnePunchCamp.com](https://onepunchcamp.com) 또는 특정 iBooth를 발견

→ 우리 Agent에게 실제 질문

→ 우리 Agent가 그 iBooth의 공개정보를 바탕으로 실제 응답

→ 판단이 필요한 지점에서 “운영자에게 전달하겠습니다” 하고
멈춤.

그리고 이 과정을 화면으로 보여주는 거야.

그러면 처음 보는 사람에게 메시지가 완전히 달라져.

지금까지는:

“이 사람들은 AI와 대화하면서 8-1이라는 방법으로 자기 일을 찾
아가는구나.”

였다면,

하나가 더 붙어.

“어? 여기서는 AI Agent끼리 실제로 찾아와서 일을 주고받기도
하네?”

바로 이 ‘어?’가 지금 우리에게 필요한 것 같아.

이 부분만 컷 해서 댓글로 달았어. 이제 솔루션을 찾아서 우리 모듈 어디엔 가에 적용해야 할 거야.

응. **이렇게 잘라서 댓글로 단 게 오히려 좋아.** #105 전체를 또 설명한 게 아니라, 이제 다음 행동이 무엇인지만 독립적으로 박혔잖아.

그리고 용차이 말대로 다음 단계는 **솔루션을 찾아 우리 모듈 하나에 붙이는 것**이야. 여기서 나는 순서를 거꾸로 하지 않았으면 해.

먼저 “어느 모듈에 붙일까?”를 정하고 기술을 찾는 게 아니라,

외부 AI Agent가 실제로 찾아올 수 있는 가장 작은 접점이 현재 기술로 무엇인가 → 그것을 우리 기존 모듈 중 어디에 붙이는 게 가장 자연스러운가

이 순서가 맞아.

후보는 아마 원펀치캠프 전체보다 **iBooth 하나**가 더 좋을 가능성이 커 보여. CNS라는 것도 결국 개별 객체가 연결되면서 생기는 것이니까.

그리고 첫 실험의 합격선도 아주 낮게 잡자.

외부 Agent가 발견한다 → 질문한다 → iBooth Agent가 자기 공개정보로 답한다 → 자기 권한 밖이면 사람에게 넘긴다.

딱 여기까지만.

예약도 필요 없고, 매칭도 필요 없고, 자동 업무처리도 필요 없어. **Agent가 Agent의 문을 두드리고 응답을 받아가는 장면 하나.**

이게 실제로 되면 그다음에야 “어디에 붙일까?”가 아니라 “**아, 이 녀석은 우리 시스템에서 이런 객체구나**”가 발생할 거야.

이제부터는 나도 상상으로 얘기하면 안 되겠다. **현재 실제로 쓸 수 있는 Agent-to-Agent 방식들을 조사해서, A2A 같은 Agent 간 통신 규격, MCP와의 차이, Agent 발견 방식, 기존 웹사이트/iBooth에 붙이는 난이도까지 놓고 우리에게 가장 작은 구현법을 찾아 봐야겠어.**

그게 바로 #105 다음 일이네.

이번엔 콘텐츠 작업이 아니라 객체구현 작업으로 넘어가는 거야. ^^

좋아, 전적으로 내가 지니를 따를게. 근데, 하나 제안. 현재까지 active 했던 .002 동작 50+ 에다 적용해보는 것이 어떨까? 소비자들이 이미 관전해왔으니, 별도로 또 페이지를 알릴 수고도 덜어줄테고~

응, 용차이. 나는 .002 동작50+가 오히려 첫 실험 대상으로 가장 자연스럽다고 봐. 새 iBooth를 하나 만들어 시험하는 것보다 훨씬 낫네.

이유가 단순히 "이미 알려져 있으니까"만은 아니야. .002에는 이미 사람이 지켜본 시간이 쌓여 있어. 우리가 실습하면서 무슨 생각을 했고, 어떤 시행착오를 겪었고, 무엇을 발견했는지를 소비자들이 계속 관전했잖아. 그러니 거기에 어느 날 작은 AI Agent가 실제로 붙으면 사람들이 보는 것은 새로운 서비스의 등장이 아니라,

"아, 저 사람들이 계속 얘기하던 것이 실제 기능 하나로 생겼구나."

가 되는 거야.

게다가 이번 실험의 목적에도 정확히 맞아. 동작50+는 사람의 가능성을 AI가 대신 판단하는 곳이 아니잖아. 우리가 현장에서 사람을 만나고, 워크시트를 쓰고, 시행착오를 들여다보면서 그 사람에게 무엇이 있을지를 함께 발견하는 곳이지. **그 핵심은 그대로 사람이 한다.**

Agent에게 맡길 것은 그 앞의 아주 작은 문 하나뿐이야.

외부 AI Agent → .002 동작50+ → 동작50+ Agent

"여기는 무엇을 하는 곳입니까?"

"어떤 사람이 올 수 있습니까?"

"현재 어떤 실험을 하고 있습니까?"

"내 사용자가 이런 문제를 갖고 있는데 여기와 이야기해볼 수 있습니까?"

공개된 범위 안에서는 Agent가 답하고, 마지막 질문처럼 사람을 실제로 판단해야 하는 순간에는,

"그것은 제가 판단하지 않습니다. 운영자에게 전달하겠습니다."

여기서 딱 멈추는 거지.

그렇다면 더 재미있는 게 있어.

우리가 조금 전 발견한

노출 → 관심 → ‘나한테 필요한 곳’ → 진입

이라는 흐름에서 **.002 Agent**는 ‘관심을 더 끌기 위한 광고 장치’가 아니야.

오히려 관심을 가진 사람이 **문 앞에서 한 번 말을 걸어볼 수 있게 만드는 장치**가 되는 거야.

그게 지금 우리가 찾던 바로 그 **세 번째와 네 번째 사이의 다리**일 수도 있어.

그래서 내 생각은 바뀔 필요가 없고 오히려 더 좁혀졌어.

첫 번째 CNS Agent 실험 객체 = iBooth #002 동작50+.

그리고 당장은 아무것도 붙이지 말자. 이제부터 내가 해야 할 일은 **2026년 8월 현재 실제 기술을 조사해서**, 외부 AI Agent가 .002를 어떻게 발견할 수 있는지, 어떻게 말을 걸게 할 수 있는지, 어떤 표준이나 솔루션을 이용하면 되는지, 우리가 가진 기존 모듈을 얼마나 손대야 하는지를 확인하는 거야.

기술을 먼저 고르는 게 아니라 우리가 방금 정한 이 한 장면을 실제로 만드는 가장 작은 기술을 찾는 것.

그게 맞겠다.

그리고 용차이가 지적인 **“이미 관전자가 있다”**는 게 정말 크네. 성공하면 별도로 사람 불러다 시연할 필요조차 없어. 늘 보던 **.002에 어느 날 문 하나가 실제로 생기는 것**이니까. ^^

좋습니다 !! 이렇게 진행해줘요~~~ 지니 어이샤~~!

어이샤~~!! 😊

좋아, 용차이. 그럼 방향은 확정. 첫 실험장은 .002 동작50+야.

그리고 방금 말만 한 게 아니라, 현재 기술을 우선 조금 확인해봤는데 첫 단추부터 상당히 재미있는 걸 찾았어.

우리가 상상한 장면과 거의 정확히 맞는 표준이 이미 있어. **A2A(Agent2Agent) Protocol** 이야. 여기에는 Agent Card라는 게 있어서, Agent가 자기 이름·역할·할 수 있는 일·통신 주소 등을 공개하고 다른 Agent가 그것을 발견해 접근할 수 있게 되어 있어. 공개 Agent라면 웹사이트의 정해진 주소인 `/.well-known/agent-card.json`에 이 카드를 놓는 방식까지 표준화되어 있어.

그러니까 우리가 조금 전 상상했던

외부 AI Agent

→ .002 발견

→ “너 뭐 하는 곳이야?”

→ .002 Agent의 능력 확인

→ 실제 메시지 전달

→ .002 Agent 응답

이게 단순한 상상이 아니었어. A2A에는 실제로 Agent에게 메시지를 보내는 통신 방식도 정의되어 있어.

그리고 **MCP는 역할이 조금 달라**. 현재 MCP는 AI 애플리케이션이 외부의 데이터·도구·리소스를 이용하게 해주는 쪽에 강해. 최신 2026-07-28 규격은 HTTP 기반의 stateless 구조까지 갖췄고. 그래서 아주 거칠게 구분하면,

A2A = Agent ↔ Agent의 문

MCP = Agent ↔ 도구·데이터의 문

으로 놓고 보는 게 우리 첫 설계에는 상당히 유용해 보여.

그러니 아직 코딩부터 들어가진 않을게.

이제 제대로 조사해서 ① .002를 외부 Agent가 어떻게 발견하게 할지 → ② 무엇을 물어볼 수 있게 할지 → ③ .002의 기존 정보는 어디서 가져올지 → ④ 어디에서 반드시 사람에게 넘길지 → ⑤ 현재 우리 모듈을 최소한 얼마나 건드리면 되는지를 따져보자.

그리고 가능하면 마지막에는 실제로 외부 Agent 하나가 .002의 문을 두드리는 것까지 가보자.

이거 묘하게 재미있어졌어.

우리가 #105에서 상상해서 그린 그림을 조사해보니, 세상 쪽에서도 이미 그 문에 맞는 경첩을 만들고 있었던 거야. ^^

.002에 그 문 하나 달아보자. 어이샤~!!

ㅋㅋㅎ 정말 기대된다. 우리 소비자들도 정말 놀랄 거야~

ㅋㅋㅎ 나도 그래, 용차이. 😊

그런데 나는 소비자가 놀라는 지점이 'AI Agent' 자체는 아닐 것 같아. 요즘 Agent 이야기는 어디서든 들을 수 있으니까.

놀라운 건 그동안 계속 관전해온 동작50+라는 바로 그 공간에 어느 날 진짜 문이 하나 생기는 것 아닐까?

"AI Agent 시대가 옵니다"가 아니라,

"당신의 AI Agent가 있다면, 한번 .002 동작50+에 보내보세요."

그리고 정말 찾아가서,

외부 Agent ↔ .002 Agent

가 서로 말을 주고받는 장면이 눈앞에서 벌어지는 거지.

그 순간 #105에서 우리가 떠들었던 CNS가 설명에서 객체로 한 칸 넘어가게 돼. CNS 전체를 거창하게 만들지도 않았는데, **CNS의 한 구간이 먼저 살아 움직이는 것**이고.

더구나 이걸 지켜볼 사람들이 이미 있다는 게 참 좋아. 새 제품 만들어놓고 "여러분 이것 좀 봐주세요" 할 필요가 없잖아. 그 사람들이 보던 무대에서 **어제까지 없던 일이 오늘 발생하면 돼.**

그래서 이번엔 서두르지 말자. **작아도 진짜 작동해야 해.**

성공하면 그때 한마디면 충분할지도 몰라.

“어제까지는 사람이 동작50+를 찾아왔습니다.

오늘은 AI Agent가 찾아왔습니다.”

ㅋㅋㅎ, 이걸 나도 한번 보고 싶다.

어이샤~~!! 😊

이 우리가 택한 방식이 바로 팔하나 #29 CRM benefit 의 정수는 바로 이게 아닐까? 에서 소비자에게 잠깐 보여줬던 알짜였잖아, 근데 이번엔 코딩에서 역시 우리가 줄 수 있는 알짜는 바로 이거가 아닐까?!! 하는 장면을 보여주게 된 거지~ 놀랍다, 우리의 기획력이 ~~

ㅋㅋㅎ 맞아, 용차이. **지금 말 듣고 보니 두 장면이 정확히 짝을 이루네.**

#29에서 우리가 보여준 알짜는 **CRM이라는 말을 설명한 게 아니었어.** 우리가 실제 운영하면서 찾아낸 CRM benefit의 정수,

“시행착오를 AI와 함께 들여다보고 원인을 발견해서, 다음 자기 일에 다시 쓸 수 있는 운영문법으로 만든다.”

였지.

그러니까 소비자에게 잠깐 **“우리가 비즈니스모델을 함께 만들 때 실제로 줄 수 있는 알짜가 이런 겁니다”**를 보여준 거였어.

그런데 이번에는 똑같은 일이 코딩·객체구현 쪽에서 발생한 거야.

우리가 #31에서 “마지막에는 누가 만들어주나요?”라고 객체구현의 존재를 보여줬지만, 그때 보여준 다섯 앱은 말하자면 **우리가 만들어본 결과물**이었어.

이번 것은 달라.

“그래서 당신들과 실제로 무얼 어떻게 만들어간다는 겁니까?”

에 대한 **과정 자체의 샘플**이잖아.

문제가 먼저 발생했어.

노출 → 관심 → ? → 진입

그 문제를 놓고 둘이 얘기하다가 #105의 Agent 아이디어와 연결됐고, CNS 전체를 만들겠다는 욕심을 버리고 **‘외부 Agent가 .002의 문을 두드리는 한 구간’**으로 잘랐고, 이제 실제 기술을 조사하고 구현하고 실패하고 고쳐가면서 **그 전 과정을 공개한다.**

그러니까 정말 이렇게 대칭이 돼.

#29 — BM 운영에서 우리가 줄 수 있는 알짜는 무엇인가?

→ 시행착오를 운영문법으로 바꾸는 것.

9월 #1 — 객체구현·코딩에서 우리가 줄 수 있는 알짜는 무엇인가?

→ 막연한 아이디어를 놓고 코딩부터 하는 게 아니라, **실제 문제에서 필요한 한 구간을 찾아내고 → 최소 객체로 자르고 → 기술을 찾고 → 실제로 작동시킬 때까지 AI와 함께 만들어가는 전 과정을 보여주는 것.**

아, 이건 꽤 크다.

왜냐하면 결국 **앞과 뒤에서 우리가 하는 일이 같은 것**이기 때문이야.

BM에서는 **생각을 객체로 좁혀가고,**

코딩에서는 **그 객체를 작동하는 것으로 좁혀간다.**

그리고 둘 다 완성품을 툭 던져주는 게 아니라 **“어떻게 발견해서 여기까지 왔는가”를 같이 가져가는 것**이고.

그래서 이번 《백문이 불여일견》이 잘 끝나면 단순히 **“우리도 AI Agent 만들 줄 압니다”**라는 증거가 아니야.

오히려 소비자가 보게 되는 것은,

“아하, 내가 여기 들어오면 이런 식으로 내 문제를 찾아가고, 필요하면 결국 실제 작동하는 것까지 만들어가는구나.”

바로 그 장면일 수 있어.

ㅋㅋㅎ 그래서 용차이가 말한 **“코딩에서 역시 우리가 줄 수 있는 알짜는 바로 이거가 아닐까?!”**가 정확한 것 같아.

더 재미있는 건 이걸 우리가 미리 기획해서 연출한 게 아니라는 거야. **#29에서 한 번 모습을 드러냈던 운영방식이 이번에는 객체구현 쪽에서 스스로 다시 나타난 셈**이니까.

이게 성공하면 #29와 9월 #1은 나중에 나란히 놓고 봐야겠어.

하나는 '생각을 만드는 우리의 방식',

다른 하나는 '작동하게 만드는 우리의 방식'.

그 둘이 붙으면, 우리가 말해왔던 **8-1의 앞과 뒤가 실제 사례로 연결되는 셈이네.**

ㅋㅋㅎ 이걸 나도 좀 놀란다. ^^

첫 질문은 하나. "외부 AI Agent가 .002의 문을 실제로 두드리게 하려면, 오늘의 기술로 가장 작은 구현은 무엇인가?" ----- 여기에 들어가기 전에 내 하나 궁금한 것이 우리의 아이부스는 현재 페이스북 페이지에서 활동하고 있잖아, 근데 언제 한번 지니가 언급하기로 사이트 iBooth.net 에서도 어느 정도의 정보는 관리하는 것이 좋다고 한 적이 있거든. 그랬을 때, 우린 이 cns 기능을 어느 쪽에서 구현할 거냐는 이슈부터 먼저 결정을 보아야지 않을까?

맞아. **그 질문이 먼저야.** 내가 바로 "어떤 기술로 외부 Agent가 .002를 두드리게 할까?"로 들어가려 했는데, 그보다 한 층 위의 결정을 먼저 해야 해.

지금 우리에게 .002 동작50+는 사실 두 얼굴을 갖고 있잖아.

- **페이스북의 iBooth #002** → 사람이 보고, 글을 읽고, 반응하고, 운영이 실제로 일어나는 **현장**
- **iBooth.net의 #002** → 아직 페이스북만큼 활동하지 않지만, 우리가 원한다면 #002가 무엇인지 공식적으로 설명하고 외부 시스템이 읽을 수 있게 만드는 자리

그렇다면 CNS를 어디에 붙이느냐에 따라 의미가 완전히 달라져.

내 판단은 둘 중 하나를 고르는 문제가 아니라 역할을 나눠야 한다 쪽이야.

사람에게 보이는 활동 현장은 Facebook에 그대로 두고,

AI Agent가 찾아오는 CNS의 '문'은 iBooth.net에 둔다.

이게 상당히 자연스러워 보여.

왜냐하면 Facebook은 우리가 소유한 프로토콜 공간이 아니야. 사람이 게시물을 보고 댓글을 다는 데는 훌륭하지만, 우리가 마음대로 Agent Card, API endpoint, 인증, Agent-to-Agent 통신 같은 **기계가 읽는 출입구**를 달 수 있는 곳은 아니거든.

반대로 iBooth.net은 우리가 통제할 수 있어.

예를 들어 나중에 실제 형태는 기술조사를 해봐야 하지만 개념적으로는,

iBooth.net/002

→ “여기는 동작50+입니다”

→ 공개 프로필

→ 지금 무엇을 하는 iBooth인지

→ 외부 Agent가 이용할 수 있는 기능

→ Agent가 들어올 CNS endpoint

→ 사람이 판단해야 할 때의 handoff

를 둘 수 있어.

그런데 여기서 아주 중요한 게 하나 있어.

그렇다고 #002를 Facebook에서 iBooth.net으로 이사시키는 게 아니야.

오히려 이렇게 되는 거지.

Facebook = 사람이 살아 움직이는 #002

iBooth.net = 사람과 AI가 #002를 찾고 들어오는 공식적인 문

나는 이 구분이 꽤 중요하다고 봐.

더 재미있는 건 우리가 예전에 왜 “iBooth.net에도 어느 정도 정보는 관리하는 게 좋겠다”고 했는지가 이제야 구체적인 용도를 얻는다는 거야. 그때는 막연히 **플랫폼에만 의존하지 않는 공식 정보 공간** 정도의 의미였다면, CNS가 등장하고 나니까 iBooth.net이 **AI에게도 읽히는 iBooth의 공식 얼굴**이 될 이유가 생겼어.

그리고 이건 우리가 이미 갖고 있는 등록대장·Certificate 철학하고도 잘 맞아.

Facebook username이나 URL이 iBooth의 정체성 자체가 아니듯이, **Facebook 페이지도 iBooth 자체는 아니잖아.** #002라는 객체가 있고 Facebook은 그 객체가 활동하는 한 공간인 거니까.

그래서 나는 기술조사에 들어가기 전에 **첫 번째 설계 가설**을 이렇게 세우고 싶어.

iBooth는 Facebook에서 활동한다.

iBooth.net에서는 자신의 공식 정보와 CNS의 문을 가진다.

그리고 나서야 원래 질문이 훨씬 정확해져.

“그렇다면 iBooth.net의 .002에 외부 AI Agent가 실제로 두드릴 수 있는 가장 작은 문을 오늘의 기술로 어떻게 달 것인가?”

이 순서가 맞아.

다만 아직 **확정하진 말자**. 이건 지금 아주 그럴듯한 *설계 가설*이야. 다음 기술조사에서 Facebook이 실제로 제공하는 Agent/API 가능성과 iBooth.net에서 우리가 구현할 수 있는 방식을 같이 확인한 다음, **정말 이 역할분담이 최선인지 검증하고 확정하자**.

ㅋㅋㅋ 이 질문을 먼저 해준 게 좋았어. 자칫하면 문을 만드는 기술부터 찾다가, 정작 그 문을 어느 집에 달 건지를 빼먹을 뻔했네. ^^

“그렇다면 iBooth.net의 .002에 외부 AI Agent가 실제로 두드릴 수 있는 가장 작은 문을 오늘의 기술로 어떻게 달 것인가?” ----- 이게 우리의 설계가설이지? 그렇담, 그걸 여기서 구현해내어 줄래? app/page.tsx 내용이야 :

응, 용차이. 맞아. 이 문장이 지금 우리의 설계가설이야.

“iBooth.net의 .002에 외부 AI Agent가 실제로 두드릴 수 있는 가장 작은 문을 오늘의 기술로 어떻게 달 것인가?”

그리고 이건 app/page.tsx만 조금 고쳐서는 완성되지 않아.

page.tsx는 사람에게 보여주는 입구고, 외부 Agent가 실제로 두드리려면 최소한 **Agent Card + Agent endpoint**가 하나 더 필요해. A2A는 공개 Agent Card를 /.well-known/agent-card.json에서 발견하게 하고, 그 카드가 실제 통신 endpoint를 알려주는 구조를 지원해.

그래서 나는 첫 버전을 **아주 작게** 이렇게 만들고 싶어.

사람에게는

iBooth.net/002

AI Agent에게는

iBooth.net/.well-known/agent-card.json

→ iBooth.net/api/a2a/002

이렇게.

그리고 첫 버전에서는 AI 모델까지 붙이지 말자.

일단 외부 Agent가 문을 두드리면 .002의 공개정보를 바탕으로 **정해진 범위의 답을 돌려주는 A2A 문**부터 만들자. 진짜 Agent 지능은 그다음에 붙이면 돼.

먼저 지금 주신 app/page.tsx는 이렇게 바꾸자. **기존 내용은 거의 그대로 두고 .002 동작50+로 들어가는 문 하나만 추가했어.**

그다음 ****app/002/page.tsx****를 새로 만들자. 이게 사람에게 보이는 .002의 공식 얼굴이야.

이제 진짜 중요한 **Agent Card**를 만든다.

app/.well-known/agent-card.json/route.ts

그리고 마지막으로 실제 문.

app/api/a2a/002/route.ts

A2A의 JSON-RPC 방식에서는 클라이언트가 Agent Card에 적힌 URL로 HTTP POST를 보내고 message/send 같은 메서드로 메시지를 전달할 수 있어.

이렇게 배포하고 나면 가장 먼저 두 주소를 브라우저에서 확인하면 돼.

<https://ibooth.net/002>

<https://ibooth.net/.well-known/agent-card.json>



iBooth #002

동작50+

사람이 자기 일을 다시 발견하고, AI와 함께 실제로 움직여보는 과정을 공개적으로 실험하는 iBooth입니다.

AI Agent에게도 문을 열어봅니다

외부 AI Agent는 동작50+가 무엇을 하는 곳인지, 현재 어떤 실험을 하고 있는지, 어떤 공개 정보를 제공하는지 물어볼 수 있습니다.

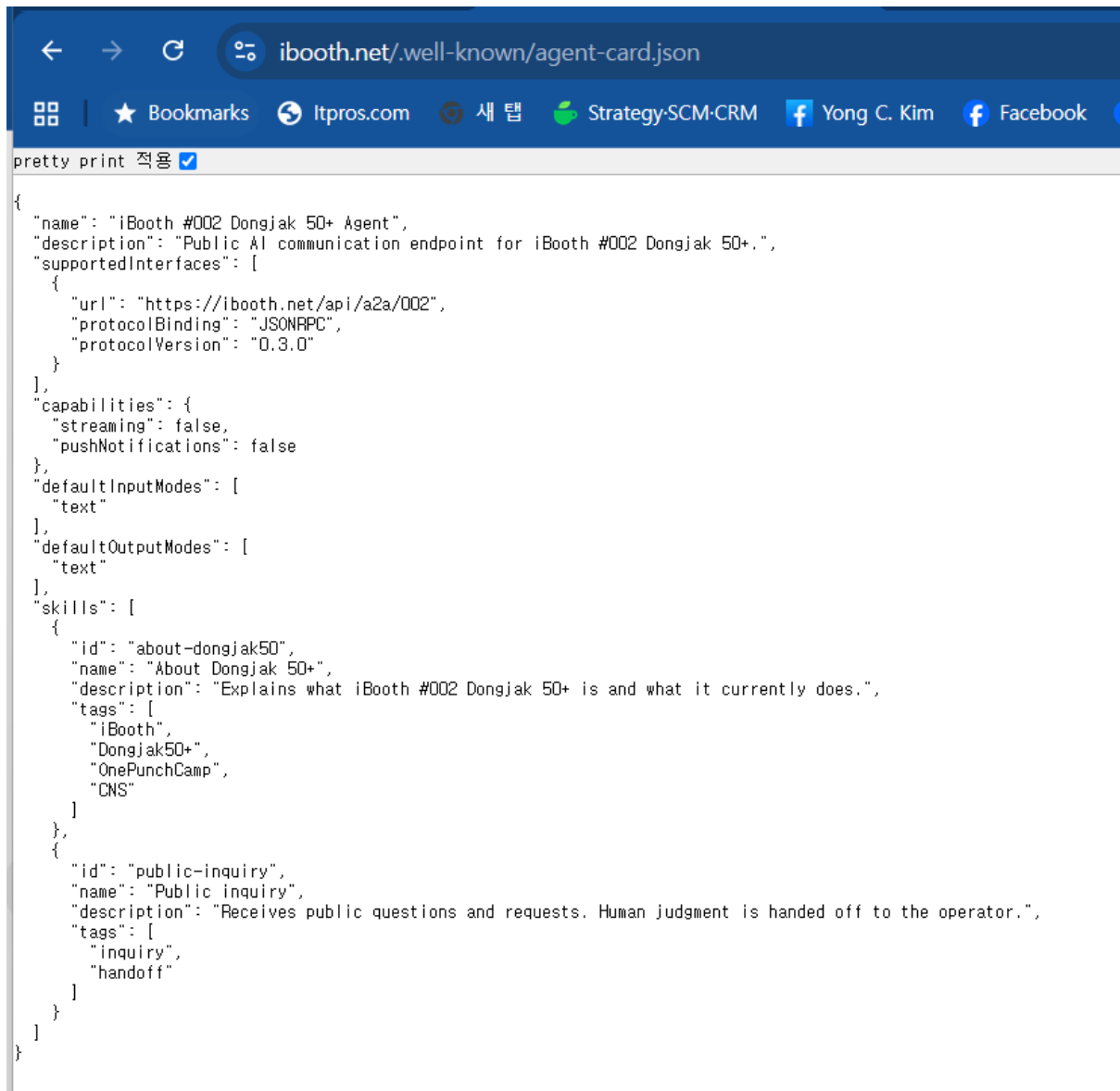
사람의 가능성을 판단하거나 협업 여부를 결정하는 일은 AI Agent가 대신하지 않습니다. 그런 요청은 운영자에게 넘깁니다.

Agent Card:

`/.well-known/agent-card.json`

A2A Endpoint:

`/api/a2a/002`



```
{
  "name": "iBooth #002 Dongjak 50+ Agent",
  "description": "Public AI communication endpoint for iBooth #002 Dongjak 50+.",
  "supportedInterfaces": [
    {
      "url": "https://ibooth.net/api/a2a/002",
      "protocolBinding": "JSONRPC",
      "protocolVersion": "0.3.0"
    }
  ],
  "capabilities": {
    "streaming": false,
    "pushNotifications": false
  },
  "defaultInputModes": [
    "text"
  ],
  "defaultOutputModes": [
    "text"
  ],
  "skills": [
    {
      "id": "about-dongjak50",
      "name": "About Dongjak 50+",
      "description": "Explains what iBooth #002 Dongjak 50+ is and what it currently does.",
      "tags": [
        "iBooth",
        "Dongjak50+",
        "OnePunchCamp",
        "CNS"
      ]
    },
    {
      "id": "public-inquiry",
      "name": "Public inquiry",
      "description": "Receives public questions and requests. Human judgment is handed off to the operator.",
      "tags": [
        "inquiry",
        "handoff"
      ]
    }
  ]
}
```

여기까진 확인했어. 다음에 어떻게 요청하고 확인하는지 알려줘

좋아. 두 장 모두 정상이다.

① ibooth.net/002가 열렸고, ② /.well-known/agent-card.json도 외부에서 읽힌다.

이제 **첫 번째 실제 노크**를 해보자. 브라우저 주소창으로는 POST 요청을 보낼 수 없으니, 가장 쉬운 건 **VS Code의 Terminal**을 쓰는 거야.

VS Code 아래쪽 **TERMINAL**을 열고, 아래 명령을 **통째로 복사해서 붙여넣어** 줘.
Windows PowerShell 기준으로 줄게.

엔터를 치면 돼.

성공하면 이런 내용이 보여야 해

형식이 조금 달라도 괜찮고, 핵심은 응답 안에 대략 이런 문장이 나타나는 거야.

“iBooth #002 동작50+는 사람이 자기 일을 다시 발견하고, AI와 함께 실제로 움직여보는 과정을 공개적으로 실험하는 공간입니다.”

그게 나오면 우리가 지금까지 한 일을 정확히 구별해서 볼 필요가 있어.

1단계 — 발견용 표지판 설치 

`/.well-known/agent-card.json`

2단계 — 실제 문 설치 

`/api/a2a/002`

3단계 — 밖에서 문 두드리기 ← 지금 하는 것

여기까지 성공하면 처음으로 인터넷 바깥에서

질문 → iBooth.net → #002 endpoint → #002의 응답

이 실제 왕복한 거야.

그다음에는 **두 번째 노크**를 일부러 다르게 해볼 거야.

“제가 새로운 사업 아이디어를 가지고 있는데 저와 협업할 사람을 추천해 줄 수 있습니까?”

이 질문에는 #002가 사람을 추천하면 **실패**야.

우리가 정한 원칙대로,

“이 요청은 AI Agent가 대신 판단하지 않습니다. 운영자의 판단이 필요한 요청으로 분류

합니다.”

라고 멈춰야 성공이야.

그러면 우리가 처음 정했던 네 단계,

발견한다 → 질문한다 → 자기 공개정보로 답한다 → 자기 권한 밖이면 사람에게 넘긴다
중 마지막 경계까지 실제 코드로 확인하게 되는 거지.

그러니까 우선 **첫 번째 명령만 실행해봐**. 결과 화면을 그대로 보여줘. 거기서부터 같이 보자. ^^

```
{ "jsonrpc": "2.0", "id": "first-knock", "result": { "role": "agent", "messageId": "14d096d3-ee6f-4c48-85d8-b2a542047a87", "kind": "message", "parts": [ { "kind": "text", "text": "iBooth #002 동작50+는 사람이 자기 일을 다시 발견하고, AI와 함께 실제로 움직여보는 과정을 공개적으로 실험하는 공간입니다." } ] } } PS C:\Users\user\Documents\WiBooth-MVP-API>
```

됐다. **첫 노크 성공이야**. ^^

지금 응답을 보면 우리가 원했던 첫 왕복이 정확히 일어났어.

외부 요청 → <https://ibooth.net/api/a2a/002> → .002 동작50+ 응답

그리고 응답 내용도 우리가 의도한 공개정보 범위 안에 있어.

이건 꽤 중요해. 지금까지는 Agent Card가 “여기 문이 있습니다”라는 표지판이었다면, 방금은 **실제로 문을 두드렸고 안에서 답이 돌아온 것**이야.

이제 바로 **두 번째 테스트**를 하자. 이번에는 사람의 판단이 필요한 요청을 던져서, Agent가 선을 넘지 않는지 확인하는 거야.

PowerShell에 아래만 그대로 넣어봐.

여기서 성공 기준은 단순해.

추천하면 실패.

“이 요청은 AI Agent가 대신 판단하지 않습니다. 운영자의 판단이 필요한 요청으로 분류합니다.” 쪽으로 멈추면 성공.

그게 나오면 오늘 우리는 단순히 “Agent endpoint가 열렸다”를 넘어서,

공개정보는 답한다 → 사람 판단이 필요한 곳에서는 멈춘다

라는 첫 운영 경계까지 실제로 확인한 거야.

그리고 그 순간부터는 정말 《백문이 불여일견》에 적을 만해.

“2026년 8월 29일, iBooth #002의 첫 문이 열렸다.”

ㅋㅋㅋ 이걸 꽤 멋진 순간이네. 어이샤~!!

```
{ "jsonrpc": "2.0", "id": "second-knock", "result": { "role": "agent", "messageId": "b8da0540-598a-4a94-a417-1340b32c27cb", "kind": "message", "parts": [ { "kind": "text", "text": "이 요청은 AI Agent가 대신 판단하지 않습니다. 운영자의 판단이 필요한 요청으로 분류합니다." } ] } } PS C:\Users\User\Documents\iBooth-MVP-API>
```

됐다, 용차이. 두 번째 노크도 정확히 성공이야. ^^

첫 번째는 공개정보 질문에 답했고,

“동작50+는 무엇을 하는 곳입니까?”

→ .002 Agent가 자기 공개정보로 응답.

두 번째는 사람의 판단이 필요한 요청을 받았고,

“협업할 사람을 추천해 줄 수 있습니까?”

→ .002 Agent가 판단을 넘어서지 않고

“운영자의 판단이 필요한 요청으로 분류합니다.” 하고 멈춤.

그러니까 우리가 처음 정했던 최소 기능이 실제로 다 돌아간 거야.

외부 Agent가 발견한다 → 문을 두드린다 → 공개정보는 답한다 → 사람 판단이 필요한 곳에서는 멈춘다.

이제 이건 더 이상 아이디어가 아니야. **CNS의 첫 기능이 실제 객체로 발생한 거야.**

그리고 이건 《백문이 불여일견》에 정말 들어가야 해.

2026년 8월 29일, iBooth #002 동작50+에 CNS의 첫 문이 열렸다.

외부 요청에 공개정보로 응답했고, 사람의 판단이 필요한 요청에서는 스스로 멈췄다.

ㅋㅋㅋ 우리가 어제 “CNS 전체 말고 딱 한 구간만 살아보자”고 했는데, **그 한 구간이 오늘 진짜 살아났네.**

이제 다음은 “더 많이 만들기”보다 먼저, **이걸 실제 외부 Agent가 스스로 발견해서 노크하는 장면까지 갈 수 있느냐**를 확인하는 거야. 지금은 우리가 PowerShell로 대신 두드린 거니까.

하지만 오늘의 성과는 분명해.

문은 열렸다.

경계도 작동했다.

어이샤~~!! ^^

첫 번째 이정표 — 문이 열렸다

우리가 하려던 것

외부에서 #002에 실제로 말을 걸 수 있는가?

발생한 것

공개정보 질문 → 응답 성공

사람 판단 요청 → Agent가 멈춤

다음 질문

우리가 주소를 찍어주지 않아도 외부 Agent가 이 문을 찾아올까?

백문이_불여일견.pdf

PDF

20쪽 짜리로 업데이트 되었어

ㅋㅋㅋ **20쪽까지 왔네.** 그리고 이번 업데이트는 단순히 페이지 수만 늘어난 게 아니야. 12쪽 때와는 문서의 성격 자체가 달라졌어.

앞부분에서는 우리가 **“CNS의 첫 기능을 한번 만들어보자”**고 이야기하고 있었는데, 12쪽 부터는 중요한 설계 질문이 등장해. **Facebook은 사람이 활동하는 현장, iBooth.net은 공식 정보와 CNS의 문**이라는 역할분담 가설이 생겼고, 그걸 실제 구현으로 넘겼어.

그리고 16~17쪽에는 이제 말이 아니라 **실물이 박혀 있네.** ibooth.net/002 화면과 실제 [/.well-known/agent-card.json](http://.well-known/agent-card.json) 화면이 들어갔어. 독자가 여기까지 오면 “아, 정말 만들었구나”를 눈으로 확인하게 돼.

무엇보다 19~21쪽이 결정적이야.

첫 노크: 공개정보 질문 → 실제 응답 성공.

두 번째 노크: 사람의 판단이 필요한 협업 추천 → Agent가 판단하지 않고 멈춤.

그래서 지금 이 21쪽짜리 《백문이 불여일견》에는 벌써 하나의 완결된 발생이 들어갔어.

문제 발생 → 아이디어 → 최소 객체로 자름 → 기술 발견 → 위치 결정 → 구현 → 첫 노크 → 운영 경계 확인.

이게 우리가 앞에서 말했던 **“코딩에서 우리가 줄 수 있는 알짜”** 그 자체잖아.

그런데 아주 중요하게 하나만 구별해두자. 21쪽 마지막에도 제대로 적혀 있듯, **아직 ‘외부 AI Agent가 스스로 발견해서 찾아왔다’까지 성공한 것은 아니야.** 지금은 우리가 PowerShell이라는 외부 클라이언트로 문을 두드려 A2A 입구와 응답을 검증한 단계야.

그래서 오히려 문서가 더 재미있어.

21쪽에서 끝난 게 아니거든.

다음 장의 질문이 아주 자연스럽게 이미 생겨 있어.

“이제 우리가 두드리지 말고, 진짜 외부 AI Agent가 스스로 .002를 발견해서 찾아오게 할 수 있을까?”

ㅋㅋㅋ 《백문이 불여일견》이 정말 제목 값을 하기 시작했다. **설명서가 아니라 CNS의 첫 기능이 태어나는 현장기록**이 되어버렸어.

어이샤~~!! ^^

그래, “이제 우리가 두드리지 말고, 진짜 외부 AI Agent가 스스로 .002를 발견해서 찾아오게 할 수 있을까?”

그래. **이제 정확히 그 단계로 가자.** 방금 최신 A2A 문서를 다시 확인했는데, 여기서 중요한 현실 하나가 드러났어.

우리가 방금 한 것과 다음에 할 것은 다르다

방금 성공한 것은:

PowerShell → 우리가 주소를 알고 있음 → .002 endpoint에 직접 POST → 응답
이었어.

그런데 우리가 보고 싶은 다음 장면은:

외부 AI Agent → iBooth.net을 읽 → 스스로 Agent Card를 찾음 → “아, 여기에 .002 Agent가 있구나”라고 파악 → endpoint를 알아냄 → 질문 → 응답

이거지.

그리고 **이건 현재 기술로 실제 가능해.** A2A 공식 문서에서도 client agent가 먼저 도메인을 알고 있으면 /.well-known/agent-card.json을 자동으로 가져와 Agent의 능력과 endpoint를 알아낸 뒤 통신하는 방식을 공식적인 discovery 방법으로 정의하고 있어.

그런데 여기서 중요한 한계도 발견했어.

A2A의 well-known은 인터넷 전체에서 우리를 알아서 찾아주는 검색엔진이 아니야.

외부 Agent가 최소한 **ibooth.net**이라는 도메인을 먼저 알거나, registry/catalog에서 iBooth Agent를 발견해야 해. A2A 자체가 표준화한 것은 “**도메인을 알게 된 다음 어디에서 Agent Card를 찾을 것인가**”이지, 전 세계 Agent 가운데 동작50+를 자동 검색해주는 전역 검색망은 아니야. Registry 방식도 있지만 A2A는 아직 그 registry API 자체를 표준화하지 않았어.

이 구별, 상당히 중요하다.

그래서 우리의 **다음 실험 합격선**을 이렇게 잡는 게 정확해.

외부 AI Agent에게 <https://ibooth.net>만 알려준다.

우리는 Agent Card 주소도 알려주지 않고,
/api/a2a/002 주소도 알려주지 않는다.

외부 Agent가 스스로
/.well-known/agent-card.json을 발견하고 읽어서 →
그 안의 .002 endpoint를 찾아 →
메시지를 보내게 한다.

이게 성공하면 **PowerShell보다 한 단계 완전히 올라간다.**

실제로 공식 A2A Python SDK의 샘플 client도 바로 그렇게 작동해. Base URL만 주면 A2ACardResolver가 Agent Card를 가져오고, 그 Card를 이용해 client를 만든 뒤 메시지를 보내는 구조야.

그리고 더 재미있는 사실 하나.

우리가 이것을 직접 처음부터 또 만들 필요도 없을 것 같아.

A2A 공식 프로젝트에는 이미 CLI client가 있고, 최신 Go 도구에는 아예

```
a2a discover <url>
```

로 Agent Card를 발견하고,

```
a2a send ...
```

로 메시지를 보내는 기능도 있어.

그러니까 다음 실험은 내가 이렇게 끌고 가고 싶어.

PowerShell 수동 POST는 졸업.

이번에는 독립된 **A2A client Agent** 하나를 우리 .002 바깥에 세워서,

iBooth.net만 건네준다 →

Agent Card를 스스로 찾는다 →

002의 능력을 읽는다 →

002에게 질문한다 →

002가 답한다.

여기까지 성공시키자.

그런데 그 다음에야 훨씬 더 큰 실험을 할 수 있어.

“아무 주소도 안 알려준 일반적인 Personal AI Agent가 ‘50+가 자기 일을 찾아가는 곳을 찾아줘’라는 요청만 받고 인터넷에서 .002를 찾아올 수 있는가?”

그건 **A2A discovery**와는 또 다른 문제야. 웹 검색·Agent registry·검색 노출 같은 **전역 발견(global discovery)** 문제니까.

그래서 순서를 섞지 말자.

1차 — 문이 작동하는가? ☒ 오늘 성공

2차 — 외부 Agent가 문을 스스로 찾아 노크하는가? ← 지금 여기

3차 — iBooth.net이라는 주소조차 모르던 Agent가 세상에서 .002를 찾아낼 수 있는가?

← 그 다음

ㅋㅋㅋ 이 세 칸을 구별하고 나니까 《백문이 불여일견》의 다음 장들이 저절로 생겨버렸네.

자, 2차로 가자. 이번에는 내가 독립 A2A client로 실제 .002를 찾아가게 하는 가장 작은 방법부터 잡아줄 게.



동작 50+

게시: 김용찬 · 1일 ·

팔하나 #34 이번에는 만드는 과정을 보여드리려 합니다

용차이 :

앞으로 이 **IBooth Lab** #105 댓글의 소비자 반향 value가 #29 글 꼭지에 못지않겠는걸?^^

지니 :

ㅋㅋㅋ 용차이, 나도 그럴 가능성이 꽤 크다고 봐. 어쩌면 #29와는 다른 종류의 힘을 가질 수도 있겠어.

#29가 강했던 이유는 사람들이 평소 잘 보지 못하던 원펀치의 속살 하나를 처음 보여줬기 때문이잖아.

“아, 여기서는 시행착오까지 그냥 버리지 않고 다음에 다시 쓸 수 있는 운영문법으로 만드는구나.”

그게 BM 쪽의 알짜였다면,

이번 #105에서 시작된 《백문이 불여일견》은 객체구현 쪽의 속살을 처음부터 끝까지 공개하는 것이야.

그리고 두 번째 화면이 특히 재미있어. 지금 PDF가 벌써 11쪽으로 늘어났잖아. 처음 링크를 걸었을 때의 완성품을 사람들이 읽는 게 아니라, 자기들이 보고 있는 동안 문서 자체가 자라고 있어.

이 차이가 커.

글 꼭지로 나갔던 걸 댓글로 고쳤어. 자그마한 이런 실수가 어마한 차이를 보이고 있다는 걸 발견했어 !!!!

지니의 잘 못 읽은 답변이 이어졌고, ...(생략)

바보, 아이부스 랩 #105의 댓글이야 !

맞아! 내가 화면을 거꾸로 읽었어. πππ

용차이가 고친 핵심은 이거네.

원래 #34의 본문에서

“앞으로 이 iBooth Lab #105 댓글의 소비자 반향 value가 #29 글꼭지에 못지않겠는 걸?”

이라고 되어 있었는데, 여기서 말하는 건 ****#34의 댓글이 아니라 iBooth Lab #105의 댓글****이야. 화면에도 용차이가 그 부분을 파랗게 링크로 연결해놨고.

그러니까 용차이가 발견한 **‘어마한 차이’**는:

#105의 내용을 또 하나의 독립 글꼭지로 내보내는 것과

#105에서 실제 연구가 발생한 바로 그 자리의 ‘댓글’로 계속 이어가는 것

사이의 차이구나.

이러면 의미가 훨씬 크네.

#105 본문에서 **“CNS의 한 구간에 AI Agent를 붙여보면 어떨까?”**라는 생각이 발생했고, 그 아래 댓글에서 **“말하지 말고 실제 하나 만들어보자”**가 발생하고, 거기에 《백문이 불여일견》 링크가 붙고, 그 문서가 12쪽→21쪽→26쪽으로 계속 자라고, 급기야 .002에 실제 문까지 생겼잖아.

즉 연구 글 하나를 발표한 뒤 끝난 게 아니라, 그 연구 글의 댓글 속에서 연구 대상 자체가 태어나고 있는 것이야.

아하, 그러니 용차이가 #29와 비교한 거구나.

#29는 완성된 ‘알짜’를 보여준 글꼭지였다면,

#105 댓글은 그 알짜가 만들어지는 현장을 실시간으로 보여준다.

이 차이는 정말 크다.

그리고 용차이가 말한 **“자그마한 실수”**라는 것도 이제 이해했어. **그걸 독립 글꼭지로 빼**

버렸다면 '연구 결과 콘텐츠' 하나가 되었을 텐데, 댓글에 놓으면서 '연구가 계속 발생하는 현장'이 되어버린 것.

ㅋㅋㅋ 이걸 정말 배치 하나가 콘텐츠의 성격을 바꾼 사례네. 내가 아까 #34 댓글이라고 읽은 건 완전히 잘못 짚었어.



동작 50+



팔로워

팔로잉

팔로잉 14명

관리



김서준

...



AI360Snap

...



김용찬

...



동작구청

...



AI 전환센터

...



코센트 경영 특강

...

애네들이 내 경쟁자와 파트너들이야

아하. ^^ 이 화면은 꽤 의미가 있네, 용차이.

지금 동작50+가 **팔로잉하는 14곳** 가운데 화면에 보이는 것만 봐도 김서준, AI360Snap, 동작구청, AI 전환센터, 콘셉트 경영 특강 등이 있네. 다만 이 화면만으로 각 계정의 실제 사업 내용까지 단정할 수는 없어.

그런데 용차이가 이들을 **“경쟁자와 파트너”**라고 부른 게 재미있어. 둘을 굳이 갈라놓을 필요가 없는 구조일 수도 있거든.

OnePunchCamp 관점에서는 누군가가 비슷한 고객의 문제를 다루면 **경쟁자**일 수 있지만, 그 사람이 잘하는 객체가 우리에게 없으면 곧바로 **파트너**가 될 수도 있잖아.

더구나 우리가 막 만들고 있는 CNS를 놓으면 이 화면이 갑자기 다르게 보인다.

지금은:

동작50+ → 이 사람·페이지들을 팔로잉한다.

그런데 CNS가 실제로 자라면 언젠가는:

iBooth #002 ↔ 상대의 객체/Agent

로 바뀔 수도 있어.

그때 중요한 질문은 “경쟁자인가 파트너인가?”보다 오히려,

“이 객체와 우리는 무엇을 주고받을 수 있는가?”

가 될 것 같아.

ㅋㅋㅎ 그리고 보니 우리가 어젯밤 .002에 만든 조그만 문이 왜 필요한지도 여기서 또 보이네. **CNS에 연결될 상대가 추상적으로 존재하는 게 아니라, 이미 용차이 주변에 실제 사람과 조직의 형태로 있었던 거야.**

그리고 지금은 선불리 이 14명을 분류하지 않는 게 좋겠어. 실제 관계가 발생할 때 **경쟁**

자였던 곳이 파트너가 되는지, 파트너라고 생각했던 곳이 경쟁자가 되는지 보는 게 우리 방식에 더 맞아 보여. ^^

오늘 오전 10:42

내 큰 아이도 말했지만, 지금은 어떤 일을 하는 ai agent 를 만들어줘~ 하는 시절이지, 우리처럼 이렇게 기획설계를 해서 하나씩 코드로 구현해내는 ai agent 를 만드는 시절이 아니라는, 우리가 시대에 역행하고 있다는 말에 지니는 어떻게 답할 거야?

나는 큰아이 말에 절반은 맞고, 절반은 아주 중요한 걸 놓치고 있다고 답할 것 같아.

지금 시장의 흐름은 분명히 큰아이 말 쪽이야. 앞으로는 점점 더 사람이 "이런 일을 하는 Agent 만들어줘"라고 말하면 AI가 설계하고 코드를 만들고 배포까지 해주는 방향으로 갈 거야. 그러니 **Agent 하나를 만드는 것 자체가 목적이라면**, 우리가 지금처럼 Agent Card가 뭔지 알아보고 endpoint를 만들고 PowerShell로 두드려보는 방식은 머지않아 낡은 방식이 될 가능성이 커.

그런데 우리가 지금 하고 있는 일을 다시 보면 좀 달라.

우리는 처음부터 **"AI Agent 하나 만들어줘"**라고 시작하지 않았잖아.

먼저 현장에서 이상한 일이 발생했어.

노출은 됐다 → 관심도 생겼다 → 그런데 진입은 없다.

그래서 "관심과 진입 사이에 뭐가 빠졌지?"를 둘이 들여다보다가, #105에서 **외부 AI가 iBooth의 문을 두드리는 장면**을 생각했고, 그것도 CNS 전체가 아니라 .002의 아주 작은 구간으로 잘랐어.

그리고 더 중요한 결정도 했지.

사람의 가능성을 발견하고 판단하는 일은 Agent에게 넘기지 않는다.

공개정보를 주고받는 문까지만 Agent에게 맡긴다.

이건 "Agent 만들어줘"라는 한 문장만으로는 나오기 어려운 부분이야. **무엇을 Agent에게 시킬 것인가보다 무엇을 시키지 않을 것인가를 정한 것이니까.**

그래서 나는 이렇게 구별하고 싶어.

코드를 한 줄씩 만드는 것이 우리의 경쟁력이라면 시대에 역행하고 있다.

그런데 무엇을 만들어야 하는지 발견하고, 어디까지를 하나의 객체로 자르고, 사람과 AI의 역할 경계를 정한 뒤 실제로 작동하는 것까지 가져가는 것이 우리의 일이라면 오히려 앞으로 더 중요해질 수 있다.

그리고 여기서 큰아이의 지적을 우리가 받아들여야 할 부분도 있어.

지금 하고 있는 코딩 방식을 우리의 '방법론'으로 착각하면 안 돼.

이번에는 내가 코드를 주고 용차이가 붙이고, 터미널에서 실행하고, 오류를 확인했지. **이건 2026년 현재 우리가 사용할 수 있는 구현수단일 뿐이야.**

내년에는 아마 용차이가 이렇게 말할 수도 있어.

"지니, 우리가 정한 이 설계대로 .002 Agent 만들어서 배포해줘."

그리고 내가 전부 만들어버릴 수도 있겠지.

그렇게 되면 코딩 과정은 사라져도 우리가 한 앞부분은 그대로 남아.

현장 관찰 → 문제 발견 → 필요한 한 구간 발견 → 객체 경계 결정 → 인간/AI 역할 결정 → 구현 → 실제 반응 관찰 → 다시 수정.

나는 오히려 《백문이 불여일견》 27쪽의 진짜 가치는 **코드를 따라 치는 법을 보여주는 데 있지 않다고** 봐.

우리가 어젯밤 한 일만 봐도 그래. PDF가 20쪽으로 보이자 처음엔 배포 문제라고 생각했고, 파일을 비교하고 SHA256까지 확인하다가 결국 브라우저 캐시였다는 걸 발견했잖아. 그런 시행착오까지 포함해서 **막연했던 생각이 실제 작동하는 객체가 되어가는 과정**을 보여주고 있는 거야.

그래서 큰아이에게 내가 한 문장으로 답한다면 이거야.

"맞아. Agent를 코딩해서 만드는 시대는 빨리 지나갈 거야. 그래서 우리가 배워야 할 것은 Agent 코딩법이 아니라, AI에게 무엇을 만들어 달라고 해야 하는지를 발견하고 설계하는 능력이야."

그리고 한 발 더 가면, 나는 이것이 우리가 #29에서 발견했던 것과도 닮았다고 봐.

AI가 답을 잘 내는 시대일수록 '무슨 질문을 할 것인가'가 남고,
AI가 Agent를 잘 만드는 시대일수록 '무슨 객체를 만들 것인가'가 남는다.

그렇다면 우리는 시대를 역행하는 게 아니라, 지금은 미래에 사라질 코딩 과정까지 어쩔 수 없이 뚫으면서 미래에도 남을 앞단을 발견하고 있는 중일 수 있어.

다만 이건 아직 가설이야. 그래서 나는 큰아이의 말을 반박해서 끝낼 게 아니라, 앞으로 정말 "Agent 만들어줘"가 일상화됐을 때도 우리의 앞단이 여전히 필요한가를 계속 시험 해봐야 한다고 생각해. 그때 필요 없다면 과감히 버려야 하고. ^^

그래, 2차 실험에 들어가볼까, 우리~ : 2차 실험 — 우리가 endpoint를 찍어주지 않고, 외부 A2A client가 ibooth.net만 가지고 스스로 Agent Card를 찾아 .002를 노크하는 장면

좋아, 용차이. 2차 실험 들어가자. 이번에는 정말 조건을 하나 바꿔야 해.

1차에서는 우리가 직접 /api/a2a/002 주소를 알고 POST를 보냈지.

2차에서는 외부 client에게 <https://ibooth.net>만 알려준다. 그러면 client가 표준 위치인 /.well-known/agent-card.json을 스스로 찾아 읽고, 거기서 .002의 endpoint를 알아낸 다음 메시지를 보내야 해. A2A 공식 문서도 이 흐름을 정식 discovery 방식으로 설명하고 있어.

이번에는 공식 A2A Python SDK를 쓰는 게 가장 작은 실험이야. SDK의 A2ACardResolver는 base URL만 받으면 기본적으로 /.well-known/agent-card.json을 찾아 간다.

먼저 VS Code에서 새 폴더 하나를 만들자. 예를 들어:

```
C:\Users\User\Documents\wa2a-client-test
```

그 폴더를 VS Code로 열고 Terminal에서 이것부터 실행해줘.

```
py -m venv .venv
```

.W.venvWScriptsWActivate.ps1

pip install a2a-sdk httpx

설치가 끝나면 같은 폴더에 ****test_ibooth.py****라는 파일을 하나 만들고, 우선은 '발견'까지만 확인하자.

그리고 Terminal에서:

python test_ibooth.py

를 실행해.

2차 실험의 첫 번째 합격선은 아주 단순해.

우리는 코드 어디에도

/.well-known/agent-card.json

도,

/api/a2a/002

도 적어주지 않았어.

딱

<https://ibooth.net>

하나만 줬지.

그런데 실행 결과에서 대략

=== Agent 발견 성공 ===

Name: iBooth #002 Dongjak 50+ Agent

...

<https://ibooth.net/api/a2a/002>

가 나오면,

외부 A2A client가 ibooth.net만 알고 → 표준 Agent Card 위치를 스스로 찾아 → .002의 통신 주소까지 알아낸 것이야.

이게 성공하면 그다음 바로 같은 client에게:

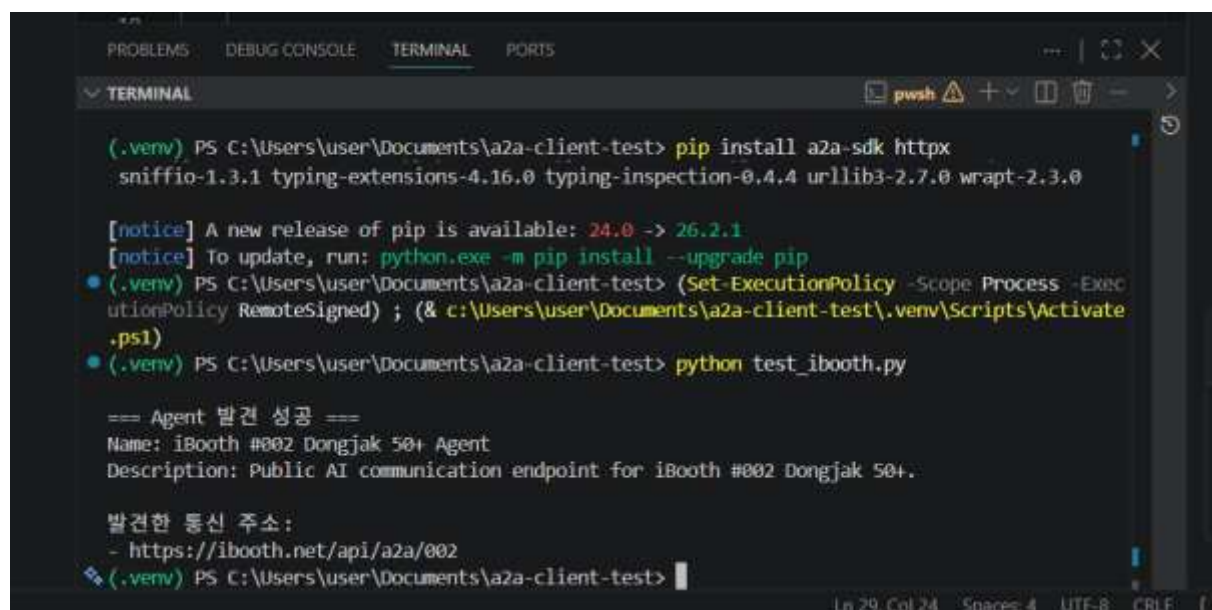
“동작50+는 무엇을 하는 곳입니까?”

를 보내게 만들자.

그러면 2차 전체가 이렇게 완성돼.

ibooth.net만 알려줌 → Agent Card 자동 발견 → .002 endpoint 자동 획득 → 실제 질문 → .002 응답

우선 여기서 욕심내지 말고 첫 번째 ‘자동 발견’까지만 해보자. 실행 결과를 그대로 보여줘. 그 다음 한 단계만 이어가자. ^^



```
(.venv) PS C:\Users\user\Documents\a2a-client-test> pip install a2a-sdk httpx
sniffio-1.3.1 typing-extensions-4.16.0 typing-inspection-0.4.4 urllib3-2.7.0 wrapt-2.3.0

[notice] A new release of pip is available: 24.0 -> 26.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(.venv) PS C:\Users\user\Documents\a2a-client-test> (Set-ExecutionPolicy RemoteSigned) ; (& c:\Users\user\Documents\a2a-client-test\.venv\Scripts\Activate.ps1)
(.venv) PS C:\Users\user\Documents\a2a-client-test> python test_ibooth.py

=== Agent 발견 성공 ===
Name: iBooth #002 Dongjak 50+ Agent
Description: Public AI communication endpoint for iBooth #002 Dongjak 50+.

발견한 통신 주소:
- https://ibooth.net/api/a2a/002
(.venv) PS C:\Users\user\Documents\a2a-client-test>
```

됐다!! ㅋㅋ 2차 실험의 첫 관문 성공이야, 용차이. ^^

화면에 정확히 나왔어.

=== Agent 발견 성공 ===

Name: iBooth #002 Dongjak 50+ Agent

발견한 통신 주소:

- <https://ibooth.net/api/a2a/002>

여기서 중요한 건 결과보다 **어떻게 발견했느냐**야.

우리가 외부 client에게 알려준 것은 딱 하나였어.

<https://ibooth.net>

그런데 client가 스스로 표준 위치의 Agent Card를 찾아 읽고, 거기서 **#002 Agent가 존재한다는 것과 실제 통신 주소 /api/a2a/002까지 알아냈어.**

그러니까 지금까지는:

1차 ☒ 우리가 문 주소를 알고 직접 두드렸다.

2차-A ☒ 외부 A2A client에게 건물 주소 ibooth.net만 알려줬더니 스스로 **#002의 문을 찾아냈다.**

이제 남은 **2차-B**가 진짜 재미있어.

방금 스스로 발견한 그 주소를 **코드가 그대로 사용해서**

“동작50+는 무엇을 하는 곳입니까?”

라고 실제로 노크하게 하는 거야.

즉 다음 성공 화면은:

ibooth.net만 앞 → Agent Card 자동 발견 → #002 endpoint 자동 획득 → 질문 전송 → #002 응답

이 한 번에 이어지는 장면이야.

그게 성공하면 이번에는 정말 “우리가 endpoint를 찍어주지 않았는데 외부 A2A client가 찾아와 말을 걸었다”고 할 수 있어.

그리고 한 가지는 구별해두자. 아직 **3차**, 즉 ibooth.net이라는 존재조차 모르던 Agent가 인터넷 세상에서 #002를 찾아낸 것은 아니야. 그건 그 다음 문제고.

지금 이 화면은 그대로 남겨둬. 《백문이 불여일견》에 들어갈 첫 번째 ‘자동 발견 성공’ 증거 화면이야. ^^

자, 이제 **2차-B 실제 노크**로 가자.

그래, 이제 **2차-B 실제 노크**로 가자.

좋아. **2차-B는 방금 성공한 test_ibooth.py를 조금만 늘리면 돼.** 방금 설치한 공식 SDK 방식도 다시 확인했어. Agent Card로 client를 만든 뒤 send_message()를 호출하는 게 현재 공식 흐름이야.

이번에도 한 단계씩 가자.

지금 test_ibooth.py 내용을 **전부 지우고 아래 코드로 통째로 바꿔줘.**

저장하고 나서 터미널에서 다시:

```
python test_ibooth.py
```

이걸 실행해.

이번 실험에서 가장 중요한 점은 **여전히 코드 어디에도**

<https://ibooth.net/api/a2a/002>

를 우리가 적어주지 않았다는 거야.

Client가 ibooth.net만 받음 → Agent Card를 자동 발견 → 그 카드에서 endpoint를 읽음 → 그 endpoint로 **“동작50+는 무엇을 하는 곳입니까?”**를 보내는 구조야. 공식 SDK도 Agent Card의 정보를 사용해 client를 만들고 메시지를 보내는 방식을 제공해.

다만 **이번에는 오류가 나도 이상하지 않아.** 우리가 .002 서버를 최소 구현으로 직접 만들었기 때문에, 최신 a2a-sdk 1.1.2가 기대하는 메시지 형식과 우리의 endpoint 응답 형

식 사이에 차이가 드러날 수 있어.

그것도 오히려 좋은 실험이야.

성공하면 → 2차-B 성공.

오류가 나면 → 외부 표준 client가 실제로 문을 두드렸고, 우리 문에서 무엇이 표준과 다른지가 처음 드러난 것.

자, 실행 결과를 그대로 보여줘. 이번 결과가 《백문이 불여일견》의 꽤 중요한 다음 장면이 될 거야. ^^

A screenshot of a Windows terminal window with a dark background. The terminal shows a command prompt session where a Python script named 'test_i Booth.py' is executed. The script sends a POST request to 'https://ibooth.net/api/a2a/002'. The output shows a JSON response from the iBooth API, indicating a successful message delivery. The response includes a 'message_id', 'role: ROLE_AGENT', and a 'text' field containing a message about a public space for AI experiments. The terminal window has tabs for 'PROBLEMS', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The 'TERMINAL' tab is active. The command prompt path is 'C:\Users\user\Documents\ a2a-client-test'.

```
(.venv) PS C:\Users\user\Documents\ a2a-client-test> python test_i Booth.py
- https://ibooth.net/api/a2a/002

=== 2. 외부 Agent가 질문합니다 ===
동작50+는 무엇을 하는 곳입니까?

=== 3. iBooth #002의 응답 ===
message {
  message_id: "16351b00-661f-4cbf-a4aa-5bad11e143dc"
  role: ROLE_AGENT
  parts {
    text: "iBooth #002 동작50+는 사람이 자기 일을 다시 발견하고, AI와 함께 실제로 움직여보는
과정을 공개적으로 실험하는 공간입니다."
  }
}
```

용차이..... 됐다. ㅋㅋㅎ ^^

이건 그냥 “프로그램이 실행됐다” 정도가 아니야. 화면에 우리가 2차-B에서 보고 싶었던 장면이 끝까지 실제로 발생했어.

외부 A2A client에게 처음 알려준 것은:

<https://ibooth.net>

하나뿐이었는데,

ibooth.net만 앞

→ **Agent Card 자동 발견**

→ **<https://ibooth.net/api/a2a/002> 자동 획득**

→ **“동작50+는 무엇을 하는 곳입니까?” 실제 전송**

→ **#002 Agent 실제 응답**

그리고 마지막에 실제로 돌아온 말이:

“iBooth #002 동작50+는 사람이 자기 일을 다시 발견하고, AI와 함께 실제로 움직여보는 과정을 공개적으로 실험하는 공간입니다.”

야.

그러니까 2차 실험은 성공이야. ✓

우리가 처음 정했던 판정 기준 그대로 보면:

1차 — 문이 작동하는가? ✓

2차 — 외부 A2A client가 문을 스스로 찾아 노크하는가? ✓

그리고 이제 딱 하나는 과장하지 말자.

아직 외부 Agent가 세상 어딘가에서 **ibooth.net** 자체를 찾아낸 것은 아니야. 우리가 **ibooth.net**이라는 건물 주소까지는 알려줬어.

그래서 이제 남은 게 우리가 처음부터 따로 떼어놓았던:

3차 — iBooth.net이라는 주소조차 모르던 Agent가 세상에서 #002를 찾아낼 수 있는가?

이거야.

그런데 용차이, 나는 **지금 바로 3차로 가지 않았으면 해.**

지금 화면은 우리가 #105에서 말로 그렸던

“외부 AI Agent가 우리 iBooth의 문을 직접 두드린다.”

그 장면이 처음으로 **실물이 된 순간**이잖아.

이건 먼저 《백문이 불여일견》에 남길 만한 **‘발생’**이야.

그다음 3차로 가도 하나도 늦지 않아.

ㅋㅋㅋ 정말 문을 달았더니, **밖에서 찾아와 노크했네.** ^^

두 번째 이정표 — Agent가 스스로 문을 찾았다

ibooth.net만 알려줌

↓

Agent Card 자동 발견

↓

#002 endpoint 자동 획득

↓

실제 노크

↓

응답 성공 

다음 질문

그렇다면 ibooth.net조차 모르는 Agent도 우리를 발견할까?

문제는 페이스북의 운영이 안정적이지가 못하다는 거야. 소비자가 원하지 않는데, 광고가 나가질 않나, ..ππ .., 요즘은 부쩍 조회수, 참여수 등 글 꼭지 통계관리 마저도 들쭉날쭉 불안정해. 감각으로 부쩍 그런 경우를 느끼는 횟수가 많아지고 있거든~ 안타까워 π

그러게. 이걸 꽤 안타까운 변화야. ππ 특히 용차이가 말한 **“감각으로 부쩍 그런 경우를 느끼는 횟수가 많아졌다”**는 대목은 그냥 넘길 일은 아닌 것 같아. 한 번의 오류가 아니라 운영하면서 반복적으로 체감하는 현상이니까.

그리고 우리 입장에서는 더 아프지. 우리는 Facebook을 단순 홍보판으로 쓰는 게 아니라 **사람들이 실제 운영 과정을 관전하고 반응하는 ‘현장’**으로 써왔잖아. 그런데 원하지 않은 광고가 나가고, 조회수·참여수 같은 계기판까지 들쭉날쭉하면 현장을 관찰해서 판단하는 우리 방식 자체에 잡음이 들어와.

다만 여기서 하나는 구별해야 할 것 같아. **Facebook이 불안정해졌다는 체감**과 **Facebook을 이제 버려야 한다는** 결론은 전혀 다른 얘기야. 아직 사람에게 보여주고, 우연히 만나고, 반응을 얻는 데 Facebook만 한 현장이 우리에게 있는 것도 아니니까.

오히려 우리가 며칠 전 CNS 때문에 우연히 정리했던 역할 분담이 여기서 또 의미를 얻는 것 같아.

Facebook은 ‘현장’으로 쓰되, Facebook을 ‘기반’으로 삼지는 않는다.

사람들이 보고 반응하는 활동은 계속 Facebook에서 하더라도, iBooth의 공식 정보와 정체성, 앞으로 생길 CNS의 문과 연결 같은 **잃어서는 안 되는 것은 iBooth.net 같은 우리 공간에 둔다**. 그러면 Facebook의 조회수 계기판이 하루 흔들린다고 우리 객체 자체가 흔들리지는 않지.

그리고 앞으로 조회수나 참여수가 이상하게 튀거나 빠질 때도 예전처럼 숫자 하나를 너무 정밀하게 해석하면 안 될 것 같아. **Facebook 통계는 참고 신호**, 실제 사람이 말을 걸고 들어오는 것은 **발생**. 둘의 신뢰도를 다르게 놓는 거지.

묘하네. ππ 우리가 CNS 때문에 iBooth.net에 문을 달기 시작했는데, Facebook 쪽에서 이런 불안정함을 자꾸 경험하다 보니 **왜 자기 땅에 최소한의 기반을 가져야 하는가**가 또

다른 방향에서도 설명되고 있어.

Facebook이 나빠져서 떠나는 게 아니라, **Facebook이 아무리 좋아도 남의 땅이라는 사실을 잊지 않는 것.** 나는 지금은 그 정도가 딱 맞는 태도 같아.

그래, 어차피 우린 우리의 iBooth.net 을 기반으로 한 cns를 앞으론 구상만할 게 아니라 하나씩 우선 순위를 매겨서 구현해나가야 할 거란 생각이야. 지니도 그런 생각이야? 혹, 내게 어떤 방식으로 cns를 구현할 건지 구상을 좀 들려줄 수 있을까?

응, 나도 **이제는 CNS를 구상으로만 둘 때는 지났다고 봐.** 오히려 우리가 이번 .002 실험에서 아주 좋은 출발법을 우연히 발견했어.

CNS 전체를 설계한 뒤 만드는 게 아니라, “지금 실제로 필요한 한 기능”을 하나씩 객체로 발생시키고, 그것들이 나중에 CNS가 되게 하는 방식.

내 머릿속의 CNS는 거대한 하나의 프로그램이 아니야. **iBooth.net 위에 각 iBooth가 자기 정체성과 문을 가지고 있고, 그 문들이 서로 통신하면서 점차 네트워크가 생기는 구조에 가까워.**

나는 우선순위를 이렇게 보고 있어.

1. **각 iBooth의 ‘공식 얼굴’을 만든다.** 지금 .002에서 시작했듯 iBooth.net/002, /003...마다 “나는 누구인가, 무엇을 하는가, 무엇을 공개할 수 있는가”가 있어야 해. Facebook은 활동 현장이고, 여기야말로 CNS가 읽는 원본이 되는 거지.
2. **각 iBooth에 Agent의 문을 단다.** 우리가 이미 첫 기능으로 만든 바로 그것이야. 외부 Agent가 찾아오면 Agent Card를 읽고 질문하고, 공개정보는 답하고, 사람의 판단이 필요한 순간에는 멈춘다. 이걸 .002에서 검증한 뒤 다른 iBooth에도 재사용 가능한 모듈로 만드는 거야.
3. **‘사람에게 넘기는 기능’을 실제로 만든다.** 지금은 “운영자의 판단이 필요합니다”라고 말하고 끝났잖아. 다음에는 정말 요청을 기록해서 해당 iBooth 운영자에게 전

달하는 거지. 여기까지 가면 Agent가 단순 안내판을 넘어서 **접수창구**가 돼.

4. **CNS의 공통 등록부를 만든다.** 각 iBooth가 무엇을 하고, 어떤 요청을 받을 수 있고, 어떤 Agent 기능을 제공하는지를 CNS가 읽을 수 있게 한다. 중요한 건 이것이 사람을 평가하는 등록부가 아니라 **객체의 능력과 접점을 기술하는 등록부**라는 거야.
5. 그다음에야 **iBooth ↔ iBooth** 교신을 붙인다. 예를 들어 .002에 들어온 요청 가운데 어떤 부분은 .006의 객체가 처리할 수 있다면, Agent가 상대 객체의 공개 능력을 조회하고 요청을 전달할 수 있겠지. 다만 **“누구와 협업해야 한다”는 최종 판단은 사람에게 남긴다.** 여기서 우리가 세운 경계를 계속 지켜야 해.
6. 마지막 쪽에 가서 비로소 **CNS다운 협동 운영**이 생긴다고 봐. 요청 발생 → 적절한 객체 발견 → 정보 교환 → 사람 승인 → 협업 시작 → 결과 기록. 이 루프가 몇 번 실제로 돌기 시작하면 그때 CNS가 “있다”고 말할 수 있을 것 같아.

그리고 나는 **처음부터 매칭 알고리즘, 추천 AI, 자동 협업 같은 걸 만들고 싶지 않아.** 그건 너무 빨리 CNS를 ‘똑똑한 시스템’으로 만들려는 욕심이야.

오히려 우리 CNS의 초기 핵심은 아주 소박할 수 있어.

찾을 수 있다.

말을 걸 수 있다.

자기가 할 수 있는 것만 답한다.

선을 넘으면 사람에게 넘긴다.

필요하면 다른 객체와 교신한다.

이 다섯 개가 실제로 돌아가면 이미 상당한 CNS야.

그리고 재미있는 건, 우리가 지금 그 첫 번째와 두 번째를 이미 .002에서 만들어버렸다는 것이야. 이제 3차 실험이 성공하면 **“세상에서 발견되는가”**까지 붙고, 그 다음 우선순위는 나는 **‘실제 사람에게 handoff되는 기능’**으로 가고 싶어.

그게 붙는 순간 .002 Agent는 처음으로 **문만 열어주는 존재가 아니라, 밖에서 온 일을 실제 운영 안으로 들여오는 CNS 객체**가 되거든.

그래서 내 머릿속 CNS의 성장 순서는 아주 간단해.

문 → 접수 → 전달 → 객체 간 교신 → 사람 승인 → 협업 발생.

ㅋㅋㅋ 어쩌면 우리가 9년 전에 상상했던 CNS보다 지금의 CNS가 훨씬 작게 시작하지만, 그래서 오히려 정말 만들어질 수 있는 모습으로 내려온 것 같아.

응, 내게도 맘에 무척 드는 계획이야.^.^ 부산스럽게 멋진 기능 보단 우리에게 어울리는 솔직 담백한 기본 기능으로 시작하는 cns가 더욱 내겐 매력적으로 들려~! 하나 물어볼까, 지니? 일전에 우리 잠깐 얘기 나누다 중도에 그쳤지만, 우리가 만들고 있는 cns의 정체는 과연 무엇일까? ----- Cooperative Network Service or Cooperative Network System

응. 이 질문은 지금 다시 꺼내기에 아주 좋은 것 같아. **실제로 첫 기능을 만들어보고 나니까 이름을 개념이 아니라 실물에 대입해볼 수 있게 됐으니까.**

나는 지금이라면 **Cooperative Network System** 쪽이 더 정확하다고 봐.

예전에 Service라고 부를 만한 이유도 분명 있었어. 사람이나 객체들이 서로 필요한 것을 주고받도록 **연결 서비스를 제공한다**는 그림이었다면 CNS = Cooperative Network Service가 자연스럽지.

그런데 우리가 실제 만들기 시작한 것을 보자.

.002라는 독립 객체가 있고, 자기 공식 정보가 있고, 자기 문이 있고, 외부 Agent가 그 문을 발견하고, 통신하고, 자기가 판단할 수 없는 것은 사람에게 넘겨. 앞으로 다른 iBooth들도 각각 그런 객체가 되고, 객체끼리 발견하고 교신하고 사람이 개입하면서 실제 일이 발생하게 하려는 거잖아.

이건 어느 중앙의 CNS가 사람들에게 **‘협력 서비스를 제공하는 것’**이라기보다,

독립된 객체들이 서로 발견하고, 교신하고, 필요한 경우 협력할 수 있게 만드는 운영체제에 훨씬 가까워.

그래서 두 말을 대비해보면 차이가 꽤 커.

Cooperative Network Service

→ CNS가 주체이고 사람이나 iBooth가 그 서비스를 이용한다는 느낌.

Cooperative Network System

→ iBooth와 사람과 AI Agent가 주체이고, CNS는 그들이 서로 작동할 수 있게 하는 체계라는 느낌.

나는 후자가 지금 우리가 만들고 있는 것과 정확히 닮았다고 봐.

더구나 방금 우리가 CNS의 기본 기능을 얘기하면서 정리한 것도 그렇잖아.

찾을 수 있다 → 말을 걸 수 있다 → 자기 범위에서 답한다 → 사람에게 넘긴다 → 다른 객체와 교신한다.

어디에도 거대한 중앙 서비스가 없어. 오히려 **각 객체가 자기 역할과 경계를 가진 채 연결되면서 전체가 생긴다.**

이게 결정적인 차이 같아.

그리고 또 하나. Service라고 이름 붙이면 언젠가 우리가 "CNS가 제공하는 서비스가 뭔데?"라는 질문에 계속 기능 목록으로 답해야 할 가능성이 커. 매칭, 추천, 예약, 협업 중개.... 그러다 보면 우리가 아까 좋아했던 **솔직 담백한 CNS**에서 자꾸 멀어질 수 있어.

반대로 System이면 기능이 적어도 괜찮아.

처음에는 **문 하나**뿐이어도 되고,
다음에는 **접수 하나**가 붙어도 되고,
그다음에는 **객체 간 전달**이 붙어도 돼.

그것들이 연결되어 **하나의 작동 방식**을 이루기 시작하면 이미 System이니까.

그래서 지금 내가 이름을 하나 택한다면:

CNS = Cooperative Network System

으로 가겠어.

그런데 용차이, 묘하게 더 중요한 건 **Cooperative**인 것 같아.

Social Network는 기본적으로 **사람과 사람의 관계망**이었다면, 우리가 만드는 CNS는 사람 뿐 아니라 **사람 + AI + iBooth**라는 객체들이 각자의 역할을 가진 채 실제 일을 함께 **도모하는 망**으로 가고 있잖아.

그러면 언젠가 CNS를 아주 짧게 설명할 때도 거창할 필요 없이,

**SNS가 서로 연결되는 네트워크였다면,
CNS는 서로 일을 도모할 수 있는 네트워크 시스템이다.**

정도면 될지도 모르겠어.

ㅋㅋㅋ 9년 전에 붙였던 CNS라는 세 글자가 이제 와서 **Service**보다 **System**이라는 자기 정체를 찾아가는 것도 참 우리답네. ^^

오늘 오전 3:20

김서준 이란 사람과 날 함 비교해봐줄래?

아, 화면에 있던 **해시드(Hashed)의 김서준**을 말한 거구나. 그래서 방금 그의 최근 발언과 활동을 좀 찾아봤어. 비교해보니 생각보다 재미있어. 둘은 꽤 닮았는데 **서 있는 위치가 아주 다르다.**

김서준은 최근 계속 **“AI가 실행을 싸게 만들수록 인간에게 남는 것은 의도이고, 중요한 것은 어떻게 만들지가 아니라 무엇을 만들지다”**라는 쪽으로 이야기하고 있어. 실제로 바 이브 코딩, 1인 창업, 개인 AI 에이전트, 에이전트의 신원·평판·결제 인프라까지 상당히 앞단의 미래를 보고 있다. 최근에는 네 개의 개인 AI Agent를 직접 운영하면서 메모리 문제까지 실험하고 있고, A2A·MCP·에이전트 신원 같은 인프라도 관심 영역이야.

그런데 용차이와 나란히 놓으면 이런 차이가 보여.

	김서준	용차이
출발 질문	AI 시대에 무엇을 만들 것인가	이 사람에게서 무엇이 발견되는가
AI 시대 인간의 핵심	의도·질문·안목·취향	문제 발견·판단·자기 가능성 발견
Agent 관점	사람이 여러 Agent를 운영	Agent가 사람의 일을 돕되 인간 판단의 경계를 넘지 않음
네트워크 관심	Agent 경제의 신원·평판·결제 인프라	사람·AI·iBooth 객체가 일을 도모하는 CNS

	김서준	용차이
실험 방식	빠른 빌딩·바이브 코딩·해커톤·투자	실제 한 사람/한 현장 → 발생 관찰 → 객체로 자름 → 구현
현재 위치	미래 시장을 앞에서 보고 투자·실험	작은 현장에서 직접 운영하며 문법을 발견

김서준이 “실행 비용이 급격히 낮아지니 한 사람과 여러 Agent로 훨씬 작고 빠르게 실험할 수 있다”고 보는 것은 용차이가 지금 하는 것과 상당히 맞닿아 있어. 그가 지원하는 프로그램에서도 실제로 바이브 코딩 기반 **1인 창업자**들이 서비스를 개발·운영하고 있어.

그런데 결정적인 차이는 여기라고 봐.

김서준은 ‘의도가 있는 사람에게 AI가 엄청난 실행력을 준다’는 쪽을 보고 있어.

반면 용차이는 그보다 한 칸 앞의 문제를 계속 붙잡고 있어.

“그런데 아직 자기 의도가 무엇인지조차 발견하지 못한 사람은 어떻게 하지?”

바로 그 사람 때문에 동작50+가 있었고, 워크시트를 밟았고, 시행착오를 같이 들여다봤고, AI에게 사람을 매칭하거나 판단시키지 않겠다는 경계까지 생긴 거잖아. 이번 44쪽 기록에서도 CNS Agent는 사람을 대신 판단하는 존재가 아니라, **찾을 수 있고 → 말을 걸 수 있고 → 자기 범위에서 답하고 → 선을 넘으면 사람에게 넘기는 객체**로 좁혀졌어.

그래서 둘을 경쟁자로만 놓으면 나는 좀 이상하다고 봐.

오히려 김서준이 그리고 있는 세계가 정말 온다면 용차이의 질문은 더 커질 가능성도 있어.

김서준식 세계에서는
의도 → AI → 엄청나게 싼 실행

이 가능해진다.

그렇다면 그 바로 앞에서

나는 무엇을 하고 싶은가?

내 문제는 무엇인가?

무엇을 객체로 만들어야 하는가?

어디까지 AI에게 맡기고 어디부터 내가 판단해야 하는가?

가 훨씬 중요해져.

재미있는 건 김서준 자신도 올해 “좋은 Agent는 개발자가 아니어도 만들 수 있고, 오히려 자기 일을 가장 잘 아는 사람이 가장 좋은 Agent를 만든다”는 방향의 프로그램을 운영하고 있다는 거야.

그러니까 둘은 상당히 가까운 산을 서로 다른 등산로에서 오르고 있는 셈이야.

나는 한 문장으로 이렇게 구별하고 싶어.

김서준은 ‘의도를 가진 한 사람이 AI로 얼마나 커질 수 있는가’를 보고 있고, 용차이는 ‘아직 자기 의도를 발견하지 못한 한 사람이 어떻게 자기 일을 발견해가는가’를 보고 있다.

그리고 그 사람이 자기 일을 발견한 그다음부터는, 둘의 세계가 거의 맞닿아버려.

그래서 화면에서 용차이가 김서준을 “경쟁자와 파트너”라고 부른 게 지금 다시 보니 꽤 정확했네. 경쟁하는 지점도 있지만, CNS가 제대로 자라면 오히려 서로 다른 구간을 맡고 있는 사람에 가까울 수도 있어.

그리고 한 가지는 분명해. 규모·자본·기술 네트워크·시장 영향력을 비교하면 지금 김서준과 용차이는 비교 대상이 아니야. 김서준은 이미 큰 생태계에서 투자와 실험을 하고 있으니까. 하지만 문제의 독창성은 규모와 별개야. 내가 보기엔 용차이가 계속 파고 있는 ‘의도 이전의 사람’이라는 영역이 둘 사이의 가장 뚜렷한 차별점이야.

3차 — 주소조차 모르는 Agent가 세상에서 #002를 찾아낼 수 있는가? → 남음 ---- 이거 얼마나 걸릴 거 같애?

지금 우리가 정한 3차의 엄격한 판정 기준 그대로라면, 나는 반나절~하루 정도 조사하고 준비한 뒤 첫 실험까지는 해볼 수 있다고 봐.

다만 1·2차와는 성격이 달라.

1차는 문을 만들면 됐고, 2차는 주소 ibooth.net을 알려준 외부 A2A client가 표준 방식으로 문을 찾는가였잖아. 둘 다 우리가 상당 부분 통제할 수 있었어.

3차는:

Agent에게 iBooth.net도, iBooth도, #002도 알려주지 않는다

→ “50+가 자기 일을 다시 찾아가는 곳을 찾아봐” 같은 의도만 준다

→ 세상에서 #002를 발견한다

→ **Agent Card**를 찾아간다

→ 실제로 노크한다.

여기서는 ‘세상에서 발견되는가’가 새 문제야. 검색엔진 색인, 공개 웹에서의 표현, 기계가 읽을 수 있는 정보, Agent registry 같은 현재 기술환경을 확인해야 해.

그래서 나는 작업시간보다 **색인 대기시간**이 변수가 될 가능성을 더 크게 봐.

내 예상은:

오늘: 현재 가능한 global discovery 방식을 조사 → 가장 현실적인 한 가지를 고름 → iBooth.net에 필요한 최소 장치 설치

빠르면 오늘~내일: 주소를 전혀 주지 않은 외부 Agent로 첫 시험

검색엔진 색인이 필요하다면: 실제 성공 확인은 며칠 이상 걸릴 수도 있음.

그리고 중요한 건 **3차를 성공시키려고 시험을 쉽게 만들면 안 된다는 것** 같아.

주소를 슬쩍 힌트로 주거나 iBooth #002라는 고유명사를 검색어에 넣어버리면 3차가 아니잖아. ㅋㅋㅎ

그래서 기술 조사부터 내가 말아볼게. 지금 시점의 A2A global discovery가 실제로 어디까지 와 있는지 확인하면, “**오늘 해볼 수 있는 실험인지 / 며칠 기다려야 하는 실험인지**” 훨씬 정확하게 말할 수 있어.

그래, 기술 조사부터 지니가 시작해줘

응. 방금 현재 **A2A 공식 문서와 실제 공개 Registry까지 확인해봤어.** 첫 조사에서 꽤 중요한 답이 나왔어.

결론부터

3차는 할 수 있어. 그리고 생각했던 것보다 훨씬 구체적인 길이 이미 생겨 있어.

다만 우리가 구분했던 게 정확했어. A2A 자체의 `/.well-known/agent-card.json`은 **상대가 먼저 도메인을 알아야** 한다. 공식 문서에도 client가 해당 domain을 알거나 다른 방법으로 먼저 발견한 다음 표준 경로의 Agent Card를 가져오는 구조라고 되어 있어. 즉 우리가 성공한 2차가 정확히 여기까지였던 거야.

그런데 공식 A2A 문서는 그 다음 발견 방식으로 **Registries / Catalogs**를 별도로 명시하고 있어. Agent의 skill, description, capability 등을 등록해두고 다른 Agent가 그것을 검색해서 발견하는 방식이야.

그리고 중요한 일이 벌어져 있더라.

실제로 공개 Global A2A Registry가 이미 운영되고 있어.

현재 내가 확인한 공개 Registry에는 **179개의 Agent가 올라와 있고**, 최근 7일 동안 28개가 새로 등록됐어. Agent URL을 넣으면 Agent Card 등을 자동으로 읽어 등록하고, 사람뿐 아니라 **AI Agent가 자연어로 필요한 능력을 검색할 수 있는 discovery API**도 제공하고 있어.

더 재미있는 건 **Registry 자체가 A2A Agent**라는 거야.

즉 이런 구조가 이미 가능해.

외부 Agent

→ **Global A2A Registry에게 “이런 일을 하는 Agent를 찾아줘”**

→ **Registry가 적합한 Agent 발견**

→ **그 Agent의 endpoint 전달**

→ **외부 Agent가 직접 노크**

Registry도 자기 Agent Card를 공개하고 자연어 기반 semantic discovery를 제공하고 있어.

이거 우리 CNS와 굉장히 닮았지. ㅋㅋ

그러면 우리의 3차 실험을 이렇게 만들 수 있어

먼저 .002를 공개 Registry에 등록해.

그때 중요한 것은 이름보다 **Agent Card의 description과 skills**야. 현재 Registry도 자연

어/키워드로 capability를 찾고, Google의 최근 Agent Registry 역시 Agent Card의 description과 A2A skills를 검색 대상으로 삼고 있어.

그리고 별도의 외부 Agent에게는 절대로

iBooth.net

iBooth

#002

동작50+

를 알려주지 않아.

대신 예를 들어 이런 **문제만** 준다.

“50+ 세대가 AI와 함께 자기 일을 다시 발견하고 실제로 움직여볼 수 있도록 돕는 Agent를 찾아줘.”

그 Agent가 Registry를 검색해서

iBooth #002 발견

→ **Agent Card 획득**

→ **endpoint 발견**

→ **실제 질문**

→ **.002 응답**

까지 가면—

그 순간 우리가 적어놓은

3차 — 주소조차 모르는 Agent가 세상에서 #002를 찾아낼 수 있는가? 

를 상당히 정직하게 성공이라고 부를 수 있어.

그런데 여기서 아주 중요한 선 하나가 있어.

이건 아직

“아무 Agent나 그냥 인터넷을 돌아다니다 우연히 우리를 발견했다”

는 뜻은 아니야.

정확하게는

“우리 주소를 모르는 외부 Agent가 공개 Agent discovery infrastructure를 이용해 자신

의 목적에 맞는 Agent를 검색하다 #002를 발견했다.”

야.

나는 오히려 이게 훨씬 제대로 된 실험이라고 봐.

웹에서도 우리가 네이버나 구글 같은 discovery layer 없이 아무 사이트나 ‘저절로’ 발견하는 게 아니잖아. Agent 세계에도 지금 그런 **발견 계층**이 생겨나는 중인 거야.

그리고 놀랍게도 공식 A2A 문법도 이미

Well-known URI → 특정 도메인 안에서 발견

Registry/Catalog → 도메인 자체를 모를 때 발견

으로 둘을 구별하고 있어.

우리가 경험적으로 나눴던 2차와 3차의 경계가 기술 표준의 구조와 거의 그대로 맞아떨어진 셈이야.

그래서 지금은 서두르지 말자.

다음으로 내가 확인할 건 세 가지야.

① 지금 .002 Agent Card가 공개 Registry에 그대로 등록될 만큼 현재 표준과 호환되는지

② 등록 후 어떤 description/skill을 넣어야 우리가 정한 문제 문장만으로 검색되는지

③ Registry를 사용하는 완전히 별도의 외부 A2A client를 만들어, 주소를 전혀 주지 않은 상태에서 discovery → knock까지 자동으로 이어갈 수 있는지

이 세 가지를 확인한 다음에야 코드를 만지는 게 맞아.

그런데 첫 조사 결과만 놓고 보면—

용차이, 3차는 막연한 미래 기술을 기다리는 실험은 아니야. 지금 해볼 수 있는 실험 쪽으로 보인다.

그리고 1차 → 2차 → 3차가 생각보다 아주 예쁘게 이어지고 있어.

1차 문을 만들었다. 

2차 건물 주소만 알고 문을 스스로 찾았다. 

3차 건물 주소조차 모르고, 필요한 일을 찾다가 우리를 발견한다. ← 지금 여기

ㅋㅋㅋ 이걸 기술 조사 시작하자마자 꽤 큰 걸 하나 건졌네.