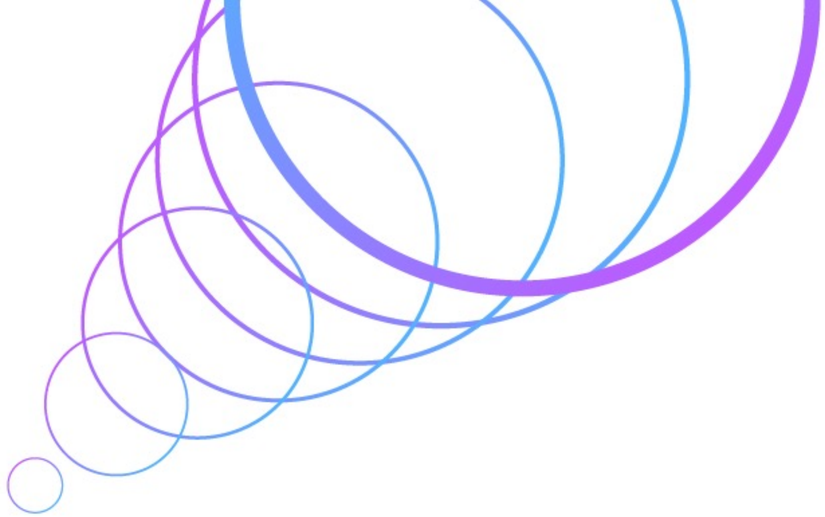


DEVIEW 2021

Android 앱 개발자는 왜 자진해서 서버 개발자가 되었나

정언 NAVER

세션 소개

클로바앱의 점진적 배포와 호환성 관리

- “Android 앱 개발자는 왜 자진해서 서버 개발자가 되었나”
- 발표자의 본업은 Android 앱 클라이언트 개발입니다.
- 그런데 이제 iOS 앱 코드 리뷰, 웹 앱 개발, 서버 개발과 운영 역할을 곁들인…

이런 분들을 위해 준비했습니다.

- 저는 그래도 서비스 개발이 어렵고 재미있어요
- 그런데 그 중 많이 어려운 부분이 정말 너무 힘드네요

세션 소개

면책 조항: 이런 내용을 기대하신다면 실망을 보전해 드릴 수 없습니다.

- ‘Android에서 서버로?’ 일당백 풀스택 개발자의 탄생
- ‘굳이 분야를 바꿔?’ 존재하지 않던 것을 만든 난세의 영웅담
- “점진적 배포?” 개발/검증/운영 비용을 0으로 만든 마법의 솔루션
- "호환성 관리?” 99.9% 성공한 리팩토링

변명하자면, ‘첨단기술보다는 적정기술’

차례

1. 클로바앱이 쏘아올린 작은 공

1.1: 클로바 어시스턴트 서비스

DEVIEW 2021

음성 기반 (Voice UI / Auditory UI) 대화형 인공지능

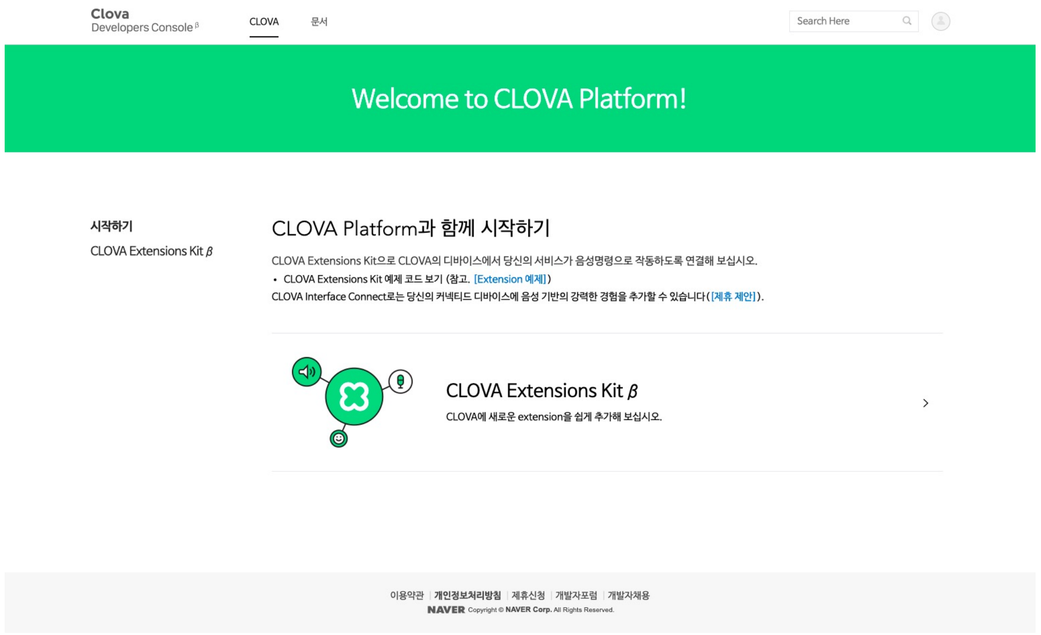
- 날씨, 교통상황 등 생활밀착형 정보 + 음악, 오디오북 등 미디어 + 스마트홈 + …

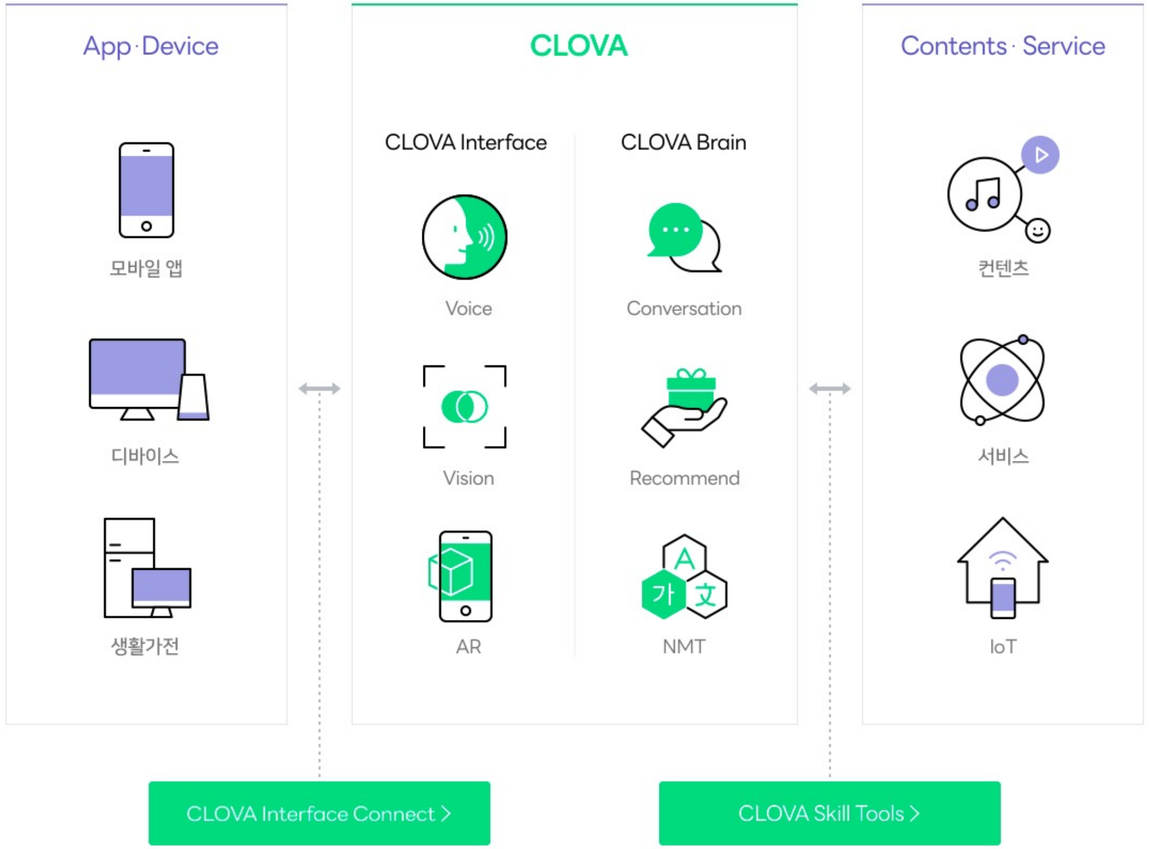
https://clova.ai/

1.1: 클로바 어시스턴트 서비스

DEVIEW 2021

"CIC" 사이드의 '클라이언트'와
"CST"/"CEK" 사이드 '스킬', '익스텐션'





https://developers.naver.com/console/clova/
https://clova.ai/

1.2: 클로바앱의 역할

DEVIEW 2021

Voice UI 기반의 클로바 서비스를 관통하는 유일한 Graphical UI

- 음량 조절 등, 엔드유저 입장에서 GUI가 익숙하고 간편한 부분부터
- Bluetooth 페어링, 음악 서비스 로그인 등 VUI로 만들기엔 난해한 부분까지

음량 얼마까지 설정할 수 있어?

음량은 0부터 15까지 조절하실 수 있습니다.

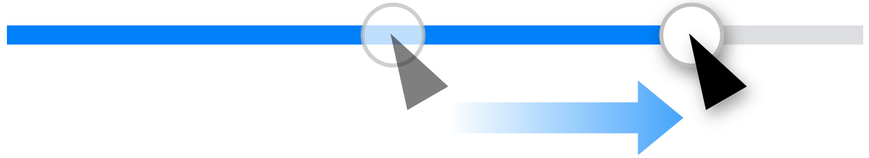
음량 12로 설정해.

음량 12는 무척 큰 소리예요. 변경을 원하시나요?

그래.

… ✓

vs.

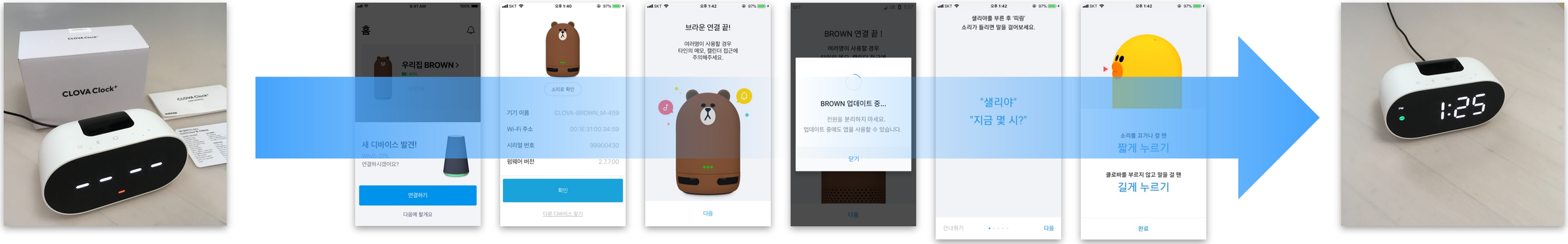


* 간단하게는 “음량 80 퍼센트로 설정해.”라고 말할 수 있습니다.

1.2: 클로바앱의 역할

DEVIEW 2021

최종 이용자 관점에서, 클로바 서비스에 대한 진입점이기도 함



"Out-of-box Experience" (OOBE)

1.3: 출시, 기능개선, 서포트… – 기기 제어

DEVIEW 2021

음량 얼마까지 설정할 수 있어?

음량은 0부터 15까지 조절하실 수 있습니다.

음량 12로 설정해.

음량 12는 무척 큰 소리예요. 변경을 원하시나요?

그래.

… ✓

vs.

클로바앱에서 이 부분이 동작하는 과정을 알아봅시다

1.3: 출시, 기능개선, 서포트… – 기기 제어

기반 기술: 자연어 이해NLU, 음성 인식ASR, 음성 합성TTS 등

NLU: natural language understanding; NLG: - generation

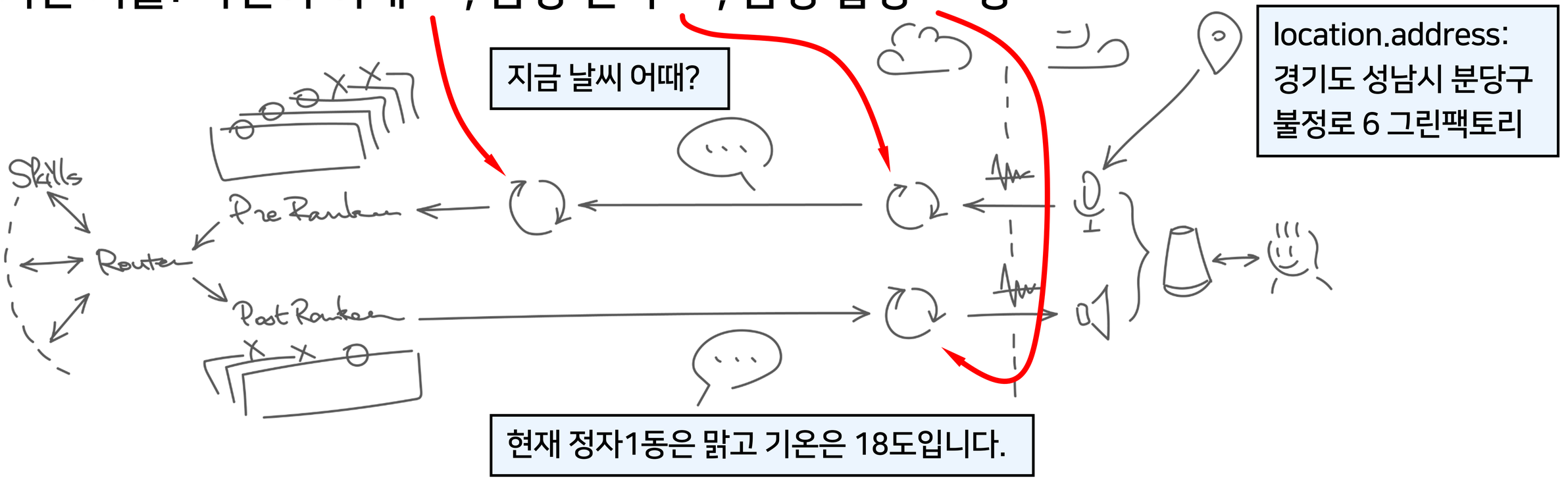
ASR: automatic speech recognition

TTS: text-to-speech (i.e. speech synthesis)

1.3: 출시, 기능개선, 서포트… – 기기 제어

DEVIEW 2021

기반 기술: 자연어 이해NLU, 음성 인식ASR, 음성 합성TTS 등



NLU: natural language understanding; NLG: - generation

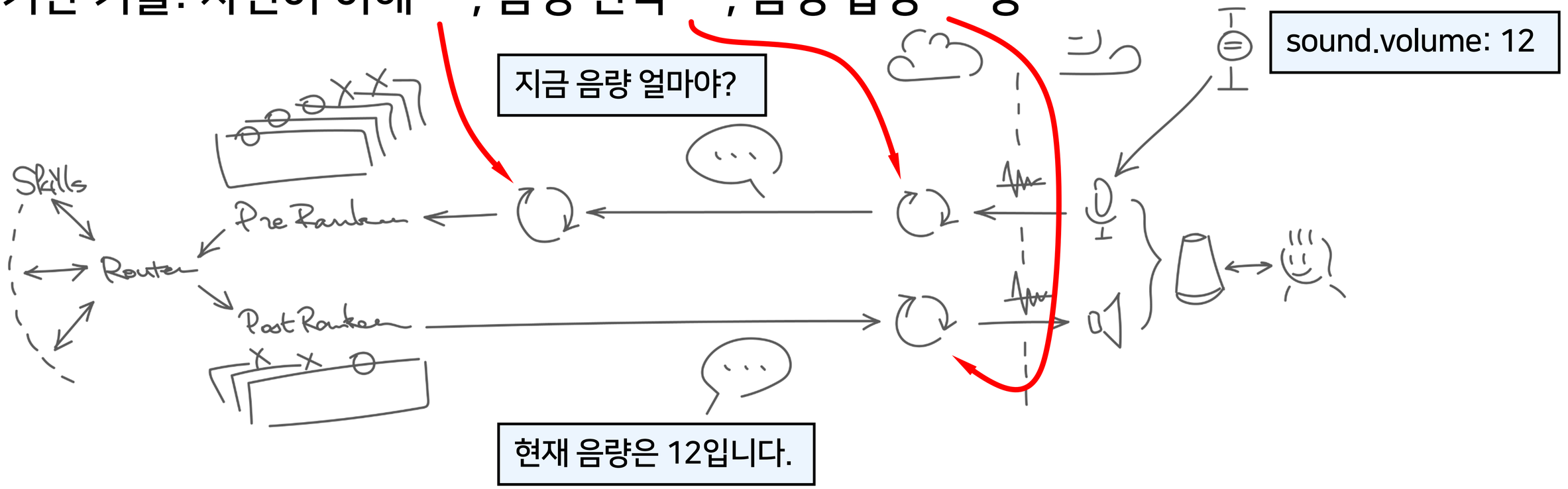
ASR: automatic speech recognition

TTS: text-to-speech (i.e. speech synthesis)

1.3: 출시, 기능개선, 서포트… – 기기 제어

DEVIEW 2021

기반 기술: 자연어 이해NLU, 음성 인식ASR, 음성 합성TTS 등



NLU: natural language understanding; NLG: - generation

ASR: automatic speech recognition

TTS: text-to-speech (i.e. speech synthesis)

1.3: 출시, 기능개선, 서포트… – 기기 제어

AI 핵심기술은 클라우드에 있지만,
서비스 본성은 사물인터넷IoT 내지는 가상물리시스템CPS

- 기기의 상태(음량, 미디어 재생 여부 등)에 대한 신뢰가능한 원본은 기기 자체에 있음
- 서버에서 부분적으로 이를 추적해 사본을 관리하지만, 도메인마다 형상과 전략이 다름
  (예: 음악의 경우, CP와 정산이 필요하므로 불일치 사례에 특히 민감)

클로바앱은 클로바 디바이스와 동등한 클로바 CIC 클라이언트로,
디바이스의 상태를 표시할 때 서버를 거쳐 기기 상태를 직접 조사하는 방식을 채택

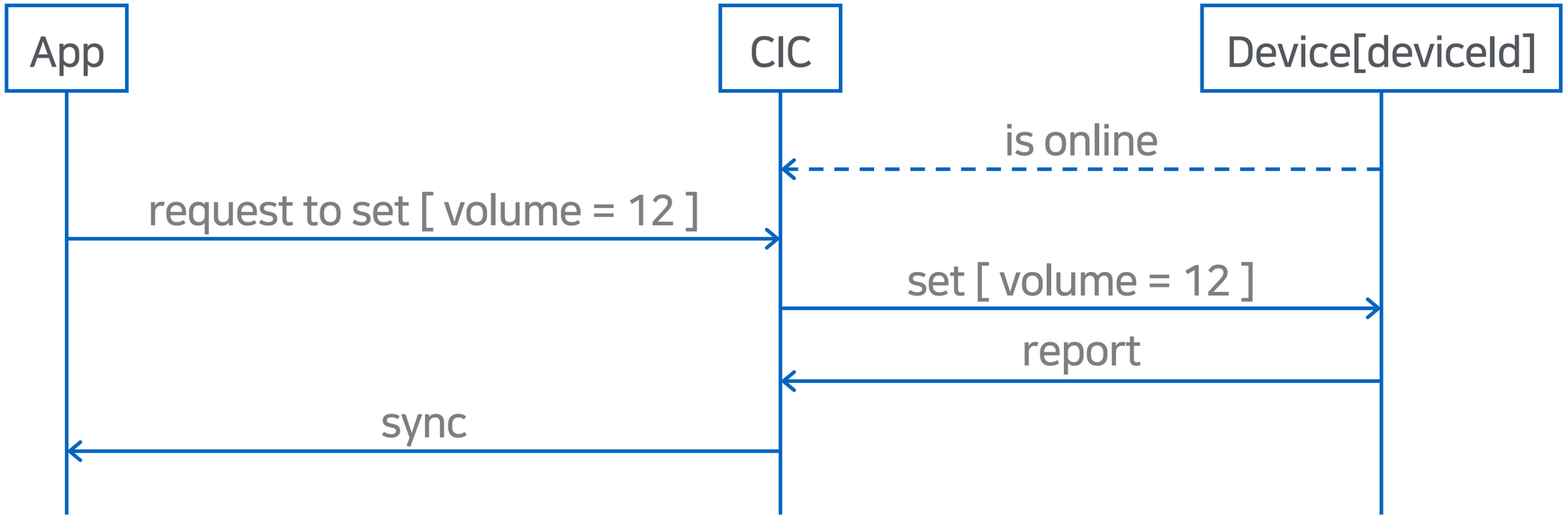
IoT: internet of things
CPS: cyber-physical system
CP: content provider
MVOT: multiple versions of truth; SSOT: single source of -

** 기기device 한정 MVOT. 장치appliance의 경우,
대체로 가전제품 제조사 IoT 플랫폼이 SSOT입니다.

1.3: 출시, 기능개선, 서포트… – 기기 제어

클로바앱은 클로바 디바이스와 동등한 클로바 CIC 클라이언트로,
디바이스의 상태를 표시할 때 서버를 거쳐 기기 상태를 직접 조사하는 방식을 채택

➔ 기능 도메인이 추가되면, 상태 관리 전략에 따라 앱에서 로직을 추가 작성해야 함



* 음량의 경우, 가장 간단한 상태 관리

1.3: 출시, 기능개선, 서포트… – 기기 제어

DEVIEW 2021

꾸준한 기기 출시 / 서드파티 기기 지원 포함 40종 이상 (앱 기준)

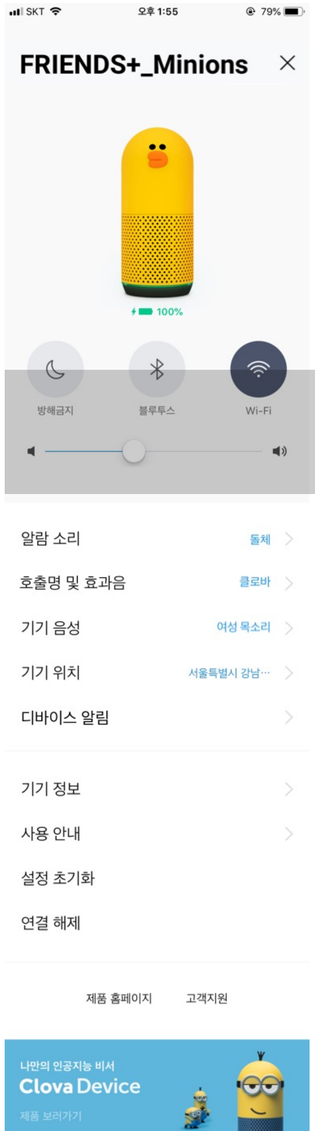
클로바앱 2.0 클로바앱 3.0

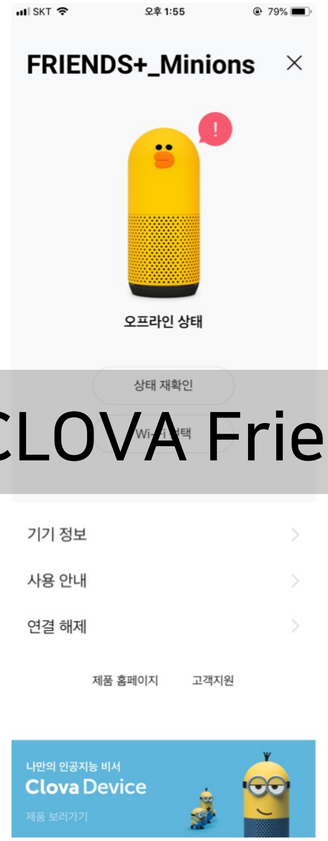
2017	2018	2019	2020	2021
WAVE Friends B/S	F mini B/S F Minions	ON+ (KR) DESK (JP)	Clock, Lamp	Clock+2

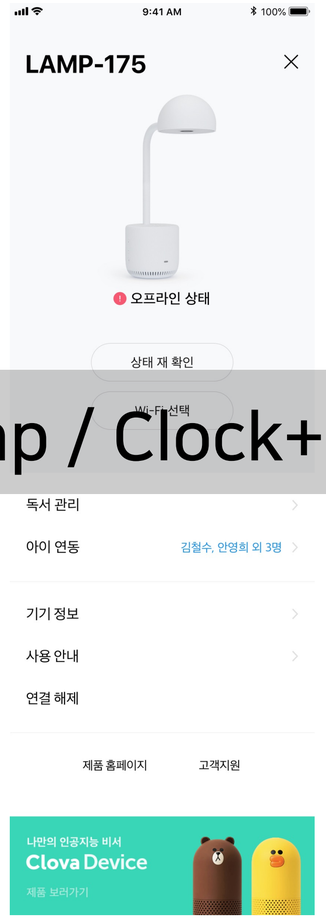

1.3: 출시, 기능개선, 서포트… – 기기 제어

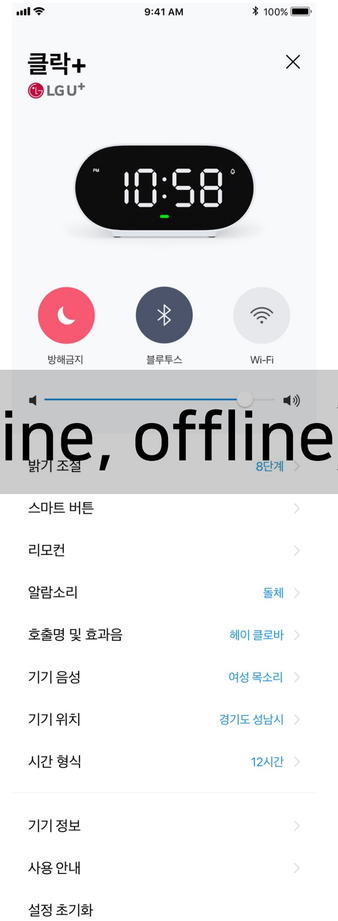
DEVIEW 2021

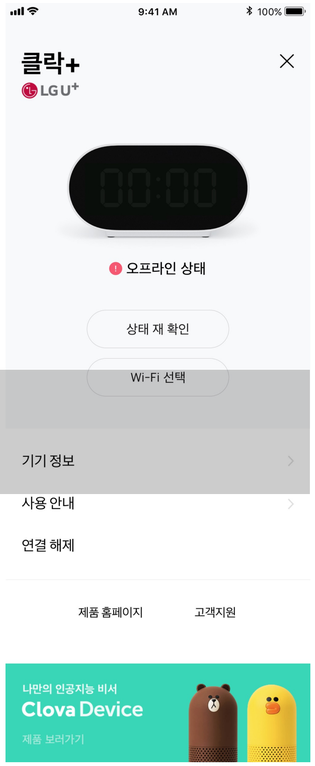
CLOVA Friends SALLY / Lamp / Clock+ (online, offline)



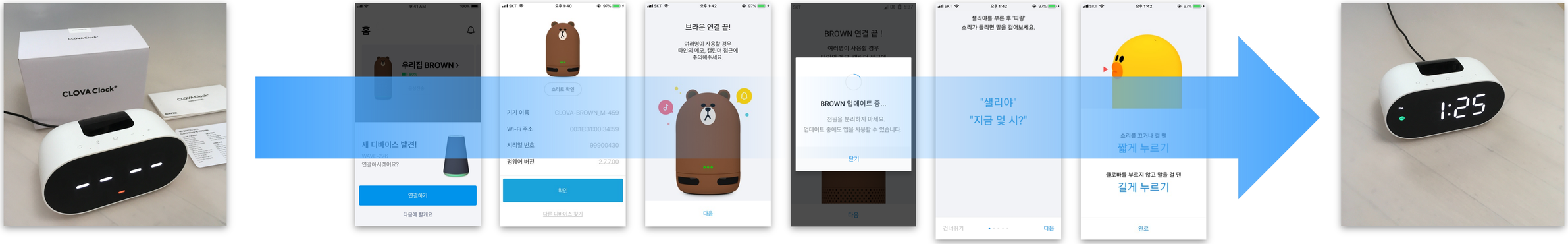








1.4: 출시, 기능개선, 서포트… – 기기 OOBE



Q: 이 부분에 대해 서버가 알고 관리할 필요가 있을까요?

1.4: 출시, 기능개선, 서포트… – 기기 OOBE

Q: 이 부분에 대해 서버가 알고 관리할 필요가 있을까요?

A: 원칙적으로 알면 좋겠지만, 서버사이드의 버저닝 범위는:

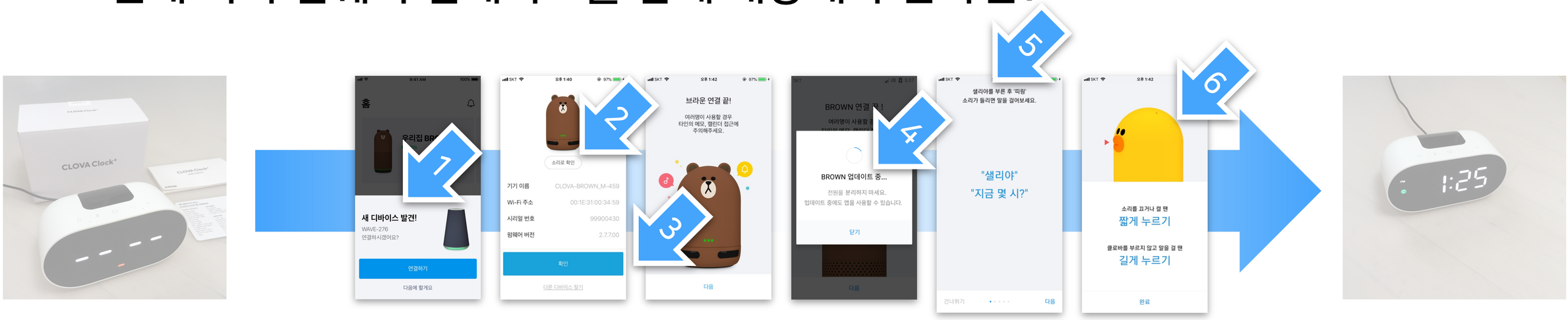
- 플랫폼 버전 (클라이언트별 스킬 서비스 바인딩 등, 도메인별 로직 문제 마찬가지)
- 각 스킬 서비스 버전
- 머신러닝 기반 컴포넌트의 모델 서빙 버전 …

기기 OOBE는 기기가 새로 출시될 때 앱에 한번
하드코딩하고 마는 쪽이 상대적으로 저비용이고, 그렇게 해 왔음

1.4: 출시, 기능개선, 서포트… – 기기 OOBE

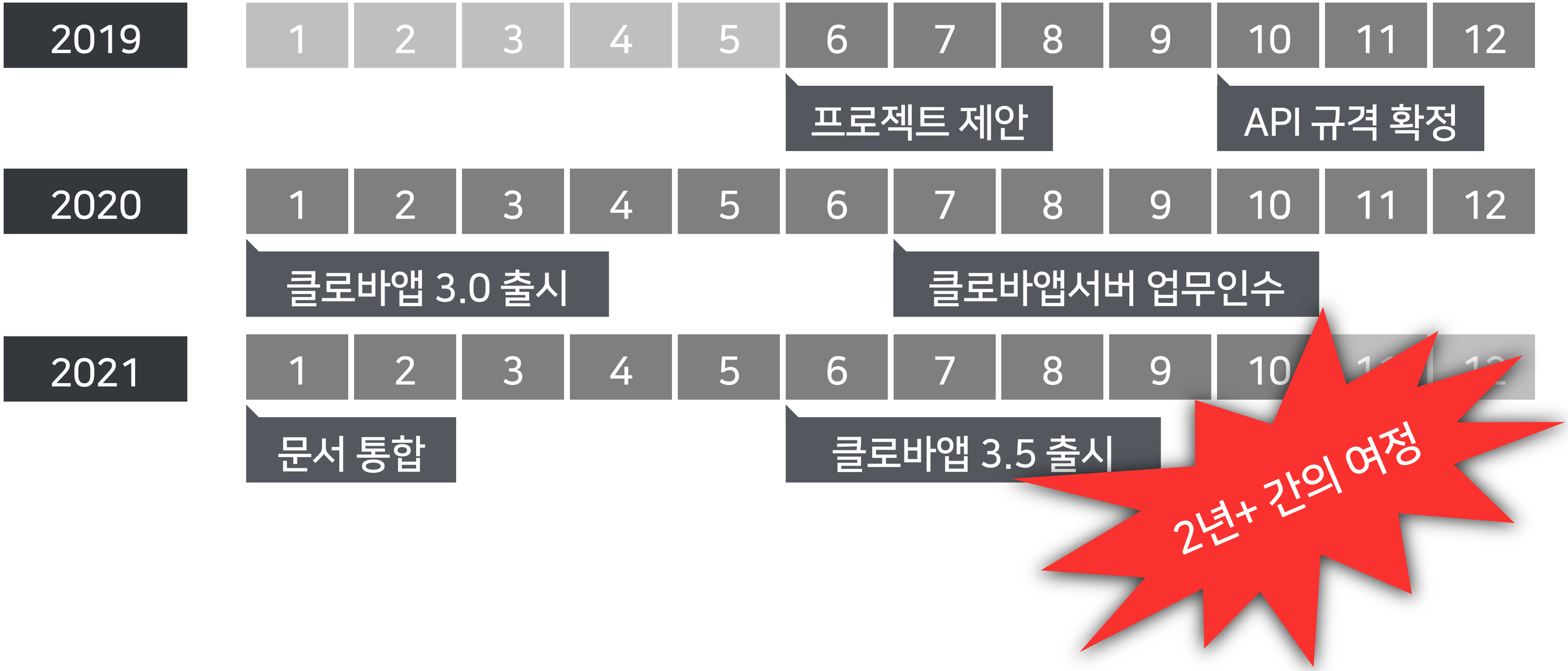
기기 OOBE는 기기가 새로 출시될 때 앱에 한번
하드코딩하고 마는 쪽이 상대적으로 저비용이고, 그렇게 해 왔음

➔ 그런데 기기 펌웨어 업데이트를 함께 대응해야 한다면?



2. 과거부터 현재까지 눈덩이 굴리기

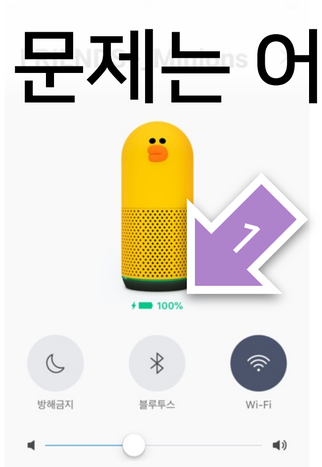
2.1: 탐험의 시작

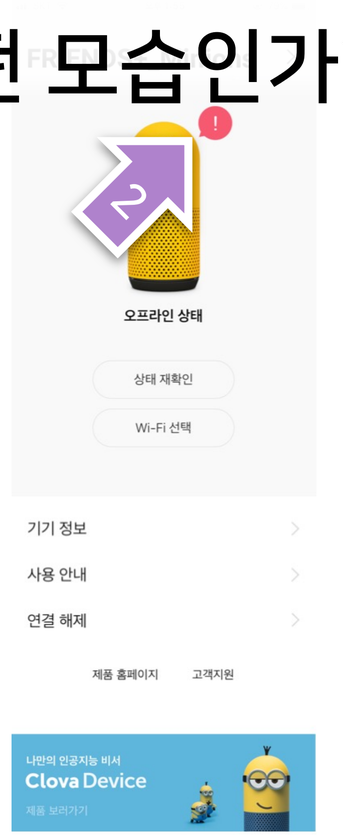

2019
1 2 3 4 5 6 7 8 9 10 11 12
프로젝트 제안
API 규격 확정
2020
1 2 3 4 5 6 7 8 9 10 11 12
클로바앱 3.0 출시
클로바앱서버 업무인수
2021
1 2 3 4 5 6 7 8 9 10
문서 통합
클로바앱 3.5 출시
2년+ 간의 여정

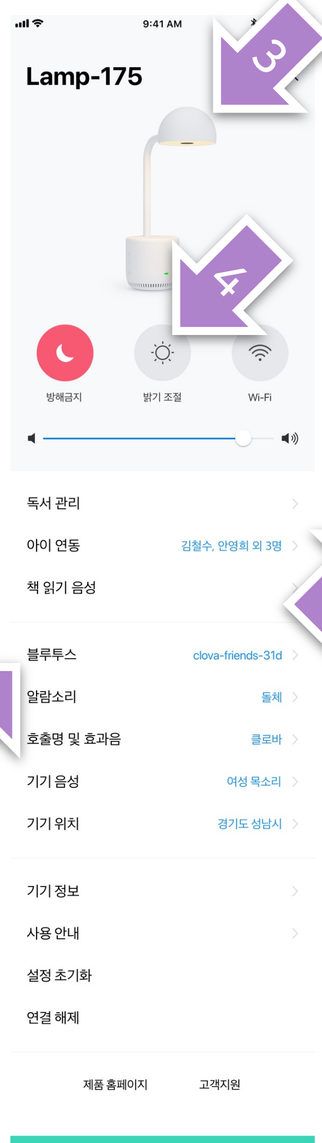
2.1: 탐험의 시작

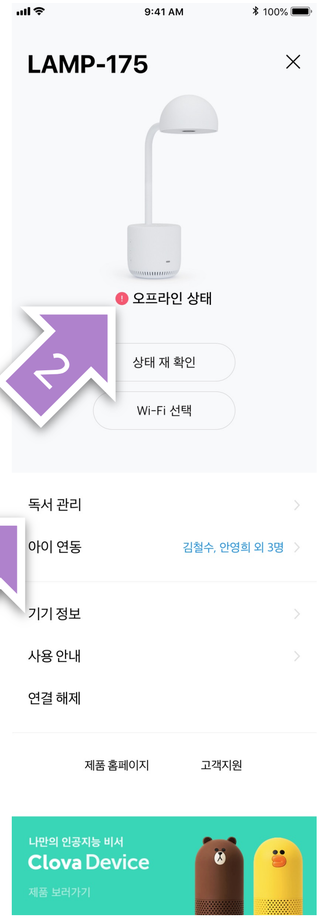
DEVIEW 2021

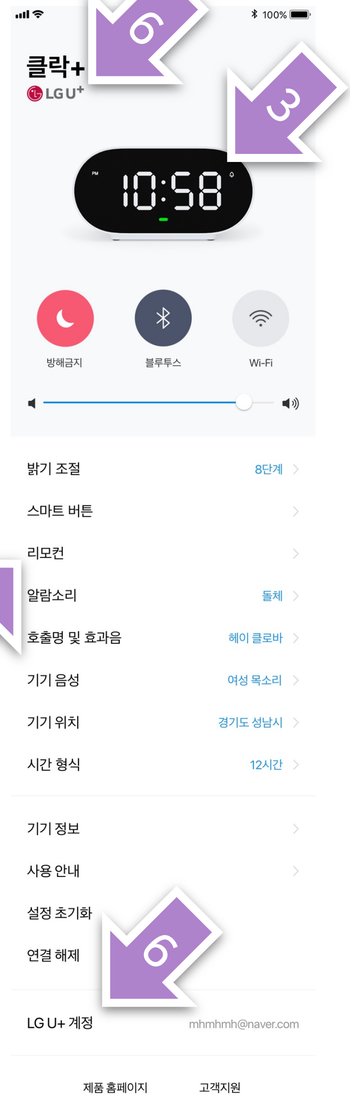
문제는 어떤 모습인가?

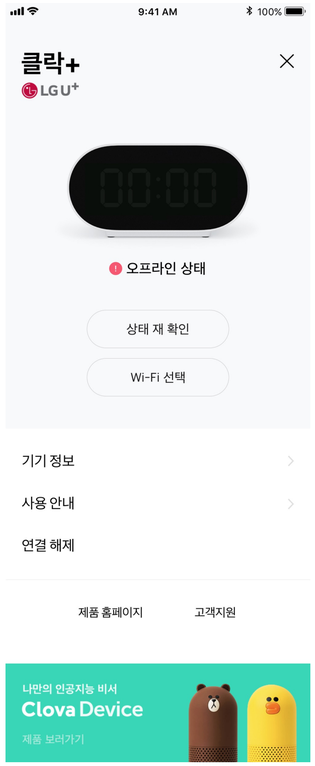












2.1: 탐험의 시작

DEVIEW 2021

문제는 어떤 모습인가?

- 클로바앱 바이너리는 하나
- 독립변수: 디바이스, 서비스 국가, OS 언어, 단말 해상도, 단말 픽셀 밀도
- 종속변수: 소스 코드 곳곳의 정수 값, 실수 근삿값 (FP), (형식) 문자열, 이미지
- 이후 독립변수가 추가되거나, 종속변수가 추가되더라도, 포맷의 스키마 변화를 최소화하여 한 포맷으로 미래의 앱 개발에도 유연하게 대응할 수 있어야 함

FP: floating point

2.1: 탐험의 시작

문제는 어떤 모습인가? **해법은 어떤 형태가 되어야 할 것인가?**

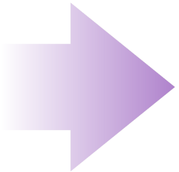
- 클로바앱 바이너리는 하나
- 독립변수: 디바이스, 서비스 국가, OS 언어, 단말 해상도, 단말 픽셀 밀도
- 종속변수: 소스 코드 곳곳의 정수 값, 실수 근삿값 (FP), (형식) 문자열, 이미지
- 이후 독립변수가 추가되거나, 종속변수가 추가되더라도, 포맷의 스키마 변화를 최소화하여 한 포맷으로 미래의 앱 개발에도 유연하게 대응할 수 있어야 함
- **변수들의 이름을 사전 정의, 독립변수들로부터 종속변수를 얻어낼 수 있는 DBMS (!)**

2.1: 탐험의 시작

문제는 어떤 모습인가? 해법은 어떤 형태가 되어야 할 것인가?

	numbers	texts	images
Service country	Yes	Yes	No
User language	Yes	Yes	Partial
CLOVA device	Partial	Yes	Yes
# pixels & density	Partial	No	Yes

* “No”: mostly no; “Partial”: possibly yes.



```
booleans:
  _typeSpec: boolean
  _levelSpec:
    - deviceSpecies
    - country
    - deviceFirmwareVersion
  _keySpec:
    _mergeDefaults:
      - DeviceSpecies.AnyDevice
    _fallbackPreference:
      - DeviceSpecies.UnknownDevice
  _match:
    - _matchArgs:
        - deviceDefaultName
      _matchStrategy: startWith
      _matchParams:
        DeviceSpecies.AmicaWave: WAVE
        DeviceSpecies.ClovaWave: CLOVA-WAVE-
```

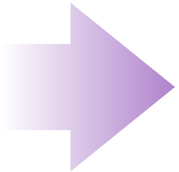
2.1: 탐험의 시작

DEVIEW 2021

문제는 어떤 모습인가? 해법은 어떤 형태가 되어야 할 것인가?

	numbers	texts	images
Service country	Yes	Yes	No
User language	Yes	Yes	Partial
CLOVA device	Partial	Yes	Yes
# pixels & density	Partial	No	Yes

* “No”: mostly no; “Partial”: possibly yes.



```
booleans:
  DeviceSpecies.AmicaWave:
    anycountry:
      anyversion:
        Device.SomeImagesAreMiniSized: false
        DeviceInit.CanCheckProtocolVersion: false
  DeviceSpecies.ClovaWave:
    anycountry:
      anyversion:
        Device.SomeImagesAreMiniSized: false
        DeviceInit.CanCheckProtocolVersion: true
  DeviceSpecies.ClovaFriendsBrown:
    anycountry:
      anyversion:
        Device.SomeImagesAreMiniSized: false
        DeviceInit.CanCheckProtocolVersion: true
```

2.1: 탐험의 시작

제약 만족 문제 CSP 방식 (like Prolog, SMT-LIB)

- 데이터에 변수 간의 관계를 기술, 선행 변수에 대해 후행 변수를 얻는 방법을 포함
- 독립 변수를 확보해 가며 종속 변수의 범위를 줄여 나가는 식으로 사용

지적된 문제들

- 이 방식의 접근이 대부분의 개발자에게는 지나치게 생소함
- "이런 모양의 데이터를 본다면, 사람들은 이 데이터로부터
  어떤 결과가 나와야 하는지 추적할 수 없을 것"

CSP: constraint satisfaction problem

2.1: 탐험의 시작

게다가 초창기 목표는:

- 기기 관련 코드 전수조사, 기기별 분기 되고 있는 값들을 추출해 서버에서 제공
- 이 값들에 대한 웹 기반 운영 도구를 포함하는 업무 프로세스 확립
- 앱은 서버 대응해서, 신기능 수준 테스팅을 전면적으로 거쳐 클로바앱 3.0 출시
- 그대로라면 클로바앱 4.0에서는 장치별 분기 코드도 제거할 수 있지 않을까?
- 그렇게 생각하던 시기가 있었습니다.

2.1: 탐험의 시작


```
{
  "devices": [
    {
      "name": "CLOVA-BROWN-",
      "properties": [ ... ],
      "scenes": { ... }
    },
    ...
  ]
}
```

3.0 릴리즈 시점의 데이터

- 기기에 따라 달라지는 값임을 구조에 드러냄
- 일반화된 DB 구조 제거
- DBMS 툴링 없이 당분간 손으로 작성
- 데이터 읽는 인간 (프로그래머) 친화적 구조
- 그만큼 구체적인 피드백도 많아짐

어레이보다는 이걸 키로 하는 오브젝트가 낫지 않겠나?

이것만 넘겨주면 되지 않겠나?

2.1: 탐험의 시작

DEVIEW 2021

```
{
  "devices": [
    {
      "name": "CLOVA-BROWN-",
      "properties": [ ... ],
      "scenes": { ... }
    },
    ...
  ]
}
```

DeviceProfile

이것만 넘겨주면 되지 않겠나?

BasesProfile

```
{
  "bases": { ... },
  "devices": [
    {
      "name": "CLOVA-BROWN-",
      "filters": {
        "defaultNamePrefixes": [ "CLOVA-BROWN-" ],
        "clientNames": [ "FRIENDS_BROWN" ]
      },
      "properties": [ ... ],
      "scenes": { ... }
    },
    ...
  ]
}
```

어레이보다는 이걸 키로 하는 오브젝트가 낫지 않겠나?

잘 설계된 시행착오

2.2: 주요 기술결정 Pros & Cons

최신 언어 툴링 기반으로 JSON 파서 유형 작성하기로 약속

- Android: Kotlin (data class or enum class) + Moshi
- iOS: Swift Codable (struct or enum)

양 클라이언트에서 쉽게 대응 가능한 범위 내에서 JSON 유형을 정의

- 이름은 일괄 camelCase로, Kotlin enum class 코드에서도 camelCase로 정의
- object 유형을 하위 유형으로 확장할 수 있지만 이때 한 필드를 string으로 모든 하위 유형에 고정 (Swift: CodingKey, Kotlin: PolymorphicJsonAdapterFactory)

* '확장 대 수정' 문제가 대두되었고 시간이 상당히 흐른 후에 결론을 내게 됩니다.

2.2: 주요 기술결정 Pros & Cons

JSON Schema

손도 깔끔

vs. TypeScript d.ts

- 명세가 간결하고, IDE 지원이 있고, 직관적 합 유형 / 곱 유형에 상속 개념도 공존
- 수치 범위나 RegExp 명세가 불가능하고 (per 2019) JSON validation 방법이 난해

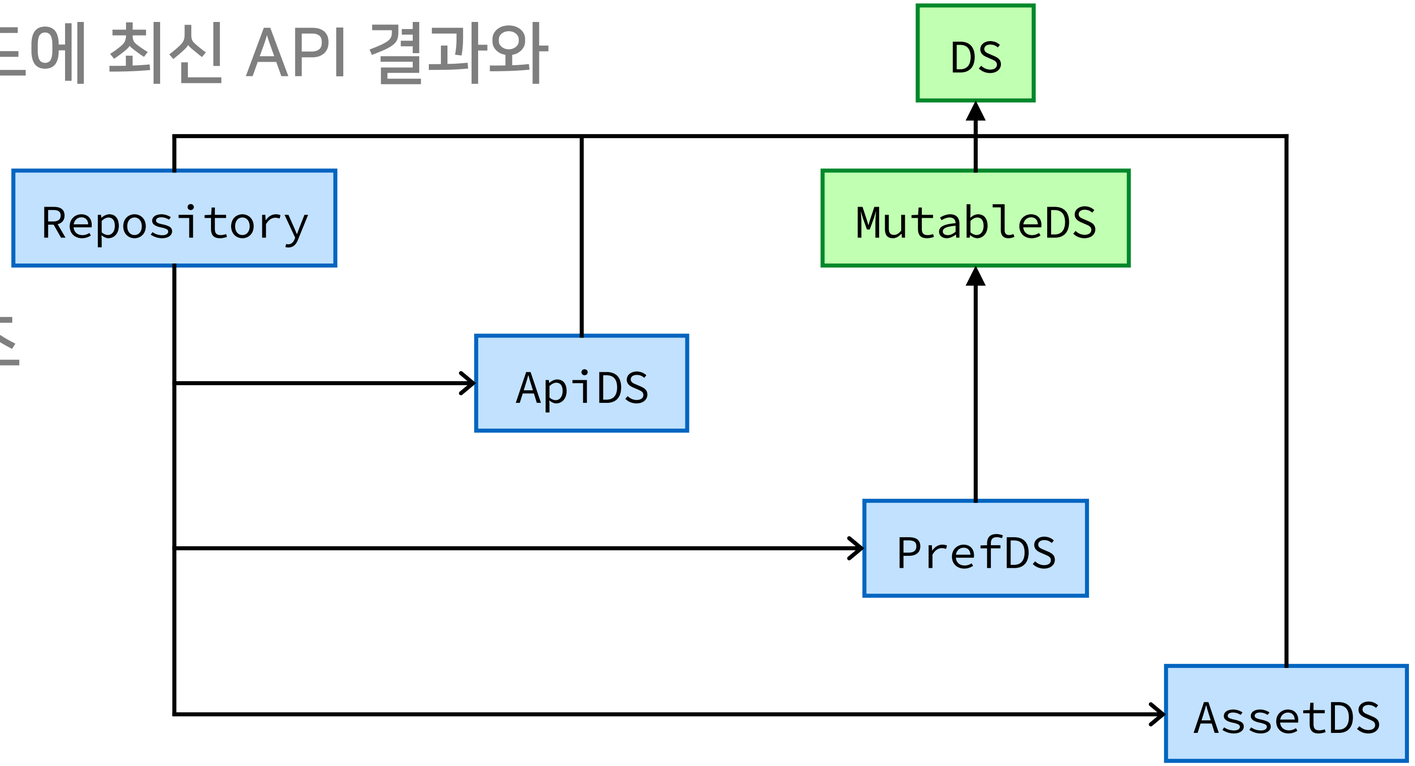
vs. 자체 제작 유형 명세 언어

- 더 읽기 좋게, JSON 유형 정의 과정에서 정한 규칙으로 간소화한 파서를 생성 가능
- 문서, IDE 지원, 파서 생성기 디버깅 등에 업무가 매몰될 위험성이 있음

2.2: 주요 기술결정 Pros & Cons

클라이언트 DataSource & Repository 설계 (Kotlin 기준)

- 앱은 네트워크 유실 상황에서도 기본 동작을 해야 함
- 최소 동작을 보장하기 위해 앱 빌드에 최신 API 결과와 같은 데이터를 포함 (asset)
- 이후 수시로 업데이트 (api)
- 앱이 저장해 두고 (pref) 쓰는 구조



2.2: 주요 기술결정 Pros & Cons

DEVIEW 2021

서버 개발자의 서포트 최소화

- 과제 초입에는 이 사람이 서버 개발자가 아니었음
- 툴링 없이 수기 JSON 파일 작성 + NGINX 정적 서빙 구축해서 프로토타이핑
- 이후 JSON 파일을 서버에 등록하는 식으로 업무흐름 변화를 최소화, 실질에 집중

2.3: 점진적 배포 개념 수립

기존에 구축한 Constraints DB 파일을 ‘revision 1’,
클로바앱 3.0에 서빙 시작한 JSON 포맷을 ‘revision 2’로 정함

- ‘기기 OOBE’ 지원 기기 업데이트부터 지원 시작
- 앱 클라이언트 모든 버전에 거의 같은 JSON 파일 제공
- 치명적인 문제가 있을 경우에만 서버가 서포트
- 기 구축 r1 DB 기반으로 r2 데이터 생성 계획
- r2 API 확장으로 ‘기기 제어’ 업데이트 지속

```
{
  ...,
  "scenes": {
    "deviceInit": { ... },
    "deviceGuide": { ... }
  }
}
```

2.3: 점진적 배포 개념 수립

DEVIEW 2021

그런데, 그 후 Clock+ 출시

- 앱 3.0, 3.1: ‘기기 OOBE’ 지원 기기 업데이트 가능 (기술적) / ‘기기 제어’는 불가능
- “이용자가 ‘기기 제어’를 전혀 할 수 없다면,
  ’기기 OOBE’를 제공한다고 할 수 없는데요…”
- 앱 3.0, 3.1: ‘기기 OOBE’ 지원 기기 추가 배제 결정
  (UX 요구사항에 따라 배포시 배제, 복잡도 폭증 시작)

[스포일러] 앱 3.2부터 ‘기기 OOBE’ 지원 기기 추가

```
{
  ...,
  "scenes": {
    "deviceInit": { ... },
    "deviceGuide": { ... }
  }
}
```

2.3: 점진적 배포 개념 수립

복잡도 폭증; 근본적 질문:

신규 기기를 '기기 제어' 지원하려면 앱 하드코딩이 필수이고,
과거 앱 버전에서 '기기 제어' 기능을 온전히 지원할 수 없는데,
앱 버전 이후 출시된 신규 기기를 '기기 OOBE'라도 지원한다는 것은
실현 불가능한 상상이었나?

2.3: 점진적 배포 개념 수립

DEVIEW 2021

동작 여부 체크리스트 vs. 상호운용성 매트릭스

device	Bluetooth availability
Friends	fw > 1.8.2
Clock+	always
App	always

vs.

Bluetooth availability	App 3.4	App 3.5
Friends	fw > 1.8.2	
Clock+	always	
Clock+2	never	always

어떻게?

2.3: 점진적 배포 개념 수립

DEVIEW 2021

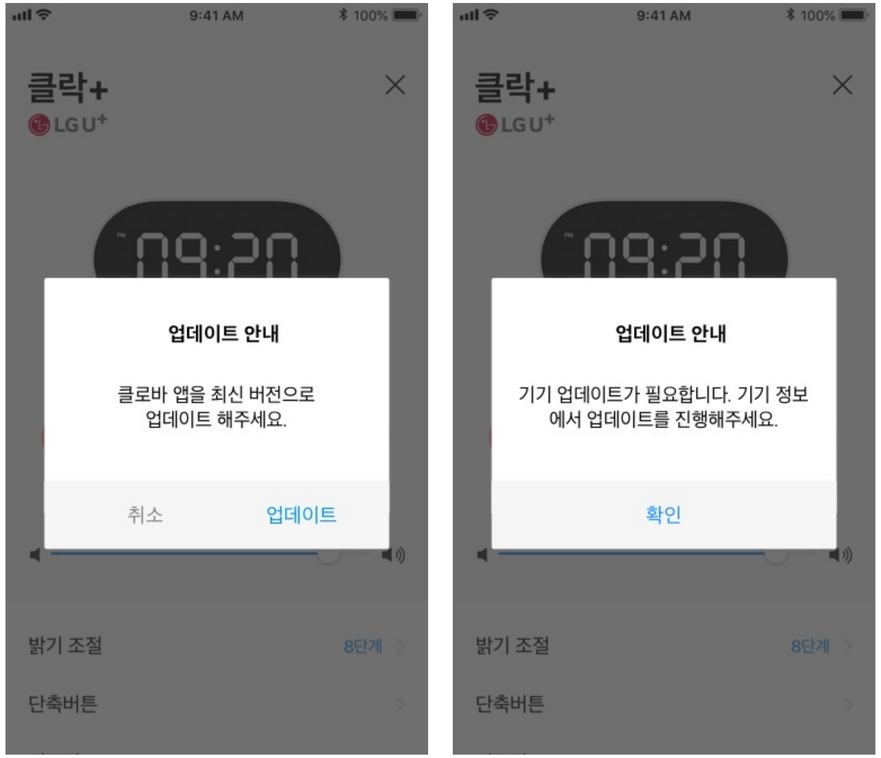
‘기기 제어’ 세부 기능의 진입점을 통제

앱이 받는 데이터에 기능별 기기 버전 조건 포함

```
{
  "scenes": {
    "deviceInit": { ... },
    "deviceGuide": { ... },
    "deviceDetails": {
      "menu": { ... },
      ...
    }
  }
}
```

기능 구현 정상 진입

앱에/기기에 구현이 없는 경우



2.3: 점진적 배포 개념 수립

DEVIEW 2021

‘iref’ (invocation reference)

- <scene-domain>.<subscene-name> 꼴의 기능 참조
- scene-domain에 따라 기능 실행 맥락이 필수 전제되므로
  웹 문서 href 따위에 쓰이는 URL과는 아예 다르게 정의
- ‘r2’ amendment level (r2a0, r2a1, …)
- 기기 버전은 검사해야 하지만, 앱 버전은 매번 스스로 검사하지 않는 게 합리적인 편

```
[
  ...,
  "deviceDetails.info",
  ...
]
```

2.3: 점진적 배포 개념 수립

DEVIEW 2021

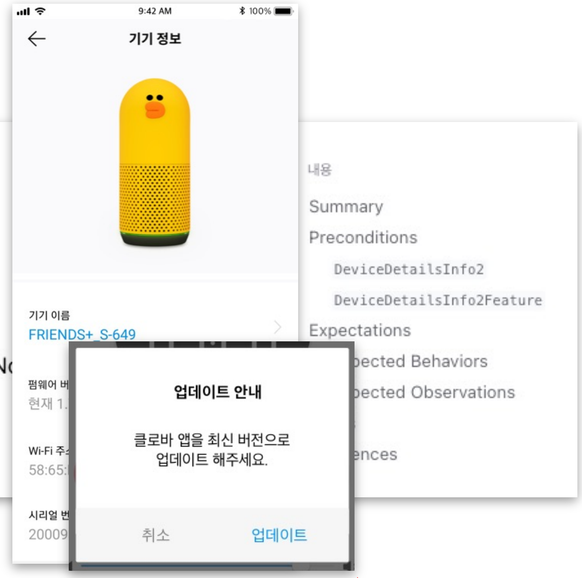
info avail	App 3.4	App 3.5
Clock+	always	
Clock+2	never	always

Client Element: `deviceDetails.info2`

다음 앱 요소를 기술합니다.

- `deviceDetails.info2` (UI scene, introduced since r2a5)
- `deviceDetails.info` (UI scene, introduced since r2a2, deprecated since r2a5) [N

Summary



App 3.4 implements r2a4

```
{ (Clock+), ?: [ ..., "deviceDetails.info", ... ] },
{ (Clock+2), ?: [ ..., "deviceDetails.info2", ... ] },
...
```

backport + remain

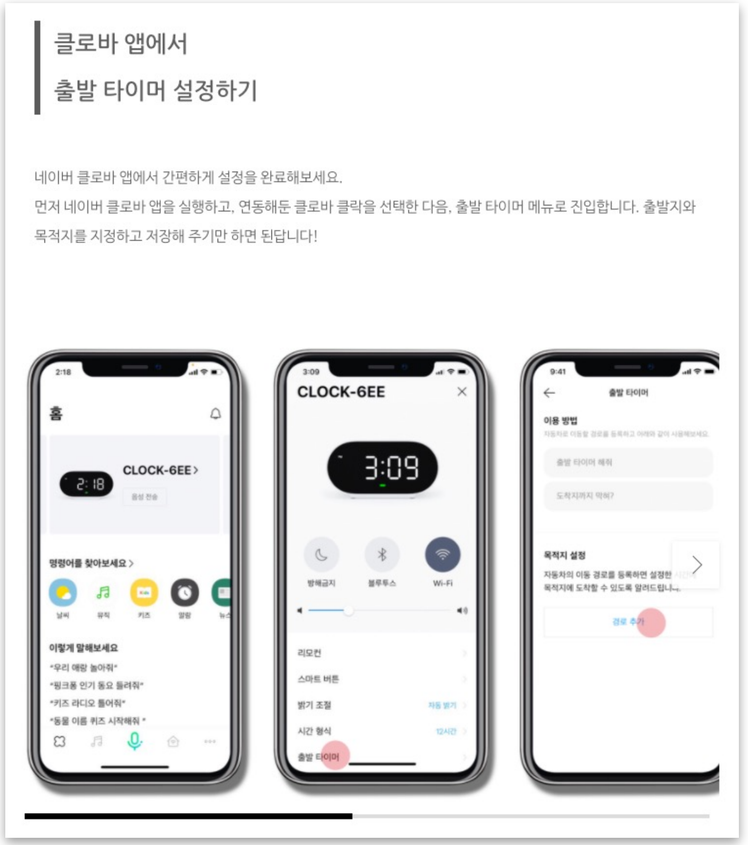
App 3.5 implements r2a5

```
{ (Clock+), ?: [ ..., "deviceDetails.info2", ... ] },
{ (Clock+2), ?: [ ..., "deviceDetails.info2", ... ] },
...
```

2.4: 주요 업데이트 사례

[Case 1] Clock+ 출발 타이머 기능 오픈

- 클로바앱 3.2는 이미 출발 타이머 구현 포함 2020-06
  (r2a2 iref “deviceDetails.clockDepartureTimer”)
- 펌웨어 버전에 따라 메뉴 진입 차단 필요
- 클로바앱 3.3에서 버전 경계 규격만 추가 대응 2020-08
  (r2a3 iref “deviceDetails.clockDepartureTimer2”)
- 같은 앱으로도 CBT 버전 디바이스에만 메뉴 노출되도록



2021-02-18 [업데이트 소식] 출발 타이머, 늦지 않게 집을 나설 시간을 알려드려요
https://m.blog.naver.com/clova_ai/222247944407

2.4: 주요 업데이트 사례

DEVIEW 2021

[Case 2] Clock+2 출시 대응

- 상응: 클로바앱 3.5
- 앱 3.2 (‘기기 제어’ 지원 기기 업데이트 가능 최초 버전) 이후 최초로 기존 앱 코드베이스 전무한 신규 기기
- ‘기기 제어’ 일부 기능은 여전히 기기별 분기를 포함, 앱 3.4 이하 앱의 경우 Clock+2에서 해당 세부기능 진입 차단하고 앱 3.5로 업데이트 유도

3. 그 다음 단계의 주제들

3.1: 끝, 없는 탐험

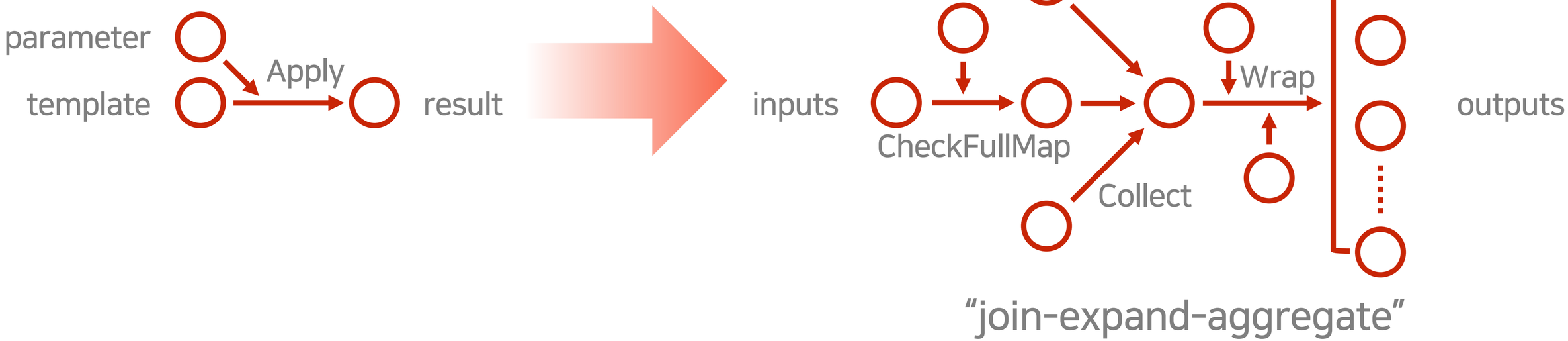
빈번한 iref 다운그레이드

- 다양한 정책: remain, backport, drop
- 커버리지 증가
- 사람의 실수

3.1: 끝, 없는 탐험

확장과 리팩토링

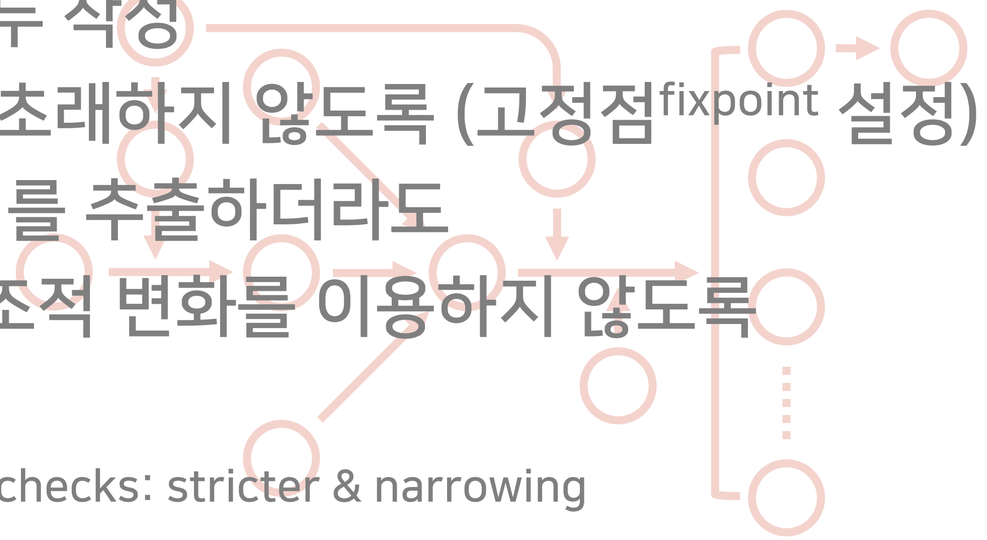
소스 데이터 분할, 상호 참조, 무결성 검증 활용



3.1: 끝, 없는 탐험

코드 신뢰: 데이터 기준으로 생각하기

- 소스 데이터를 변환할 땐 왕복round-trip 코드 모두 작성
- 구조적 변화가 API로 제공할 결과값의 변화를 초래하지 않도록 (고정점fixpoint 설정)
  → 소스 데이터의 형식이 변하더라도, 파라미터를 추출하더라도
- API로 제공할 결과값을 변경하고자 한다면 구조적 변화를 이용하지 않도록 (불변량invariant 설정)
- 유형 점검은 엄격하게, 강화하는 방향으로 type checks: stricter & narrowing



3.1: 끝, 없는 탐험

확장 대 수정 extension vs. modification

- enum (string) 유형이나 object 하위 유형을 확장할 수 있도록 했으나,
  최종 이용자를 위한 앱이 각 기능 구현에서 모든 확장에 어떻게든 대응하도록 하고
  이를 정상 동작으로 규정하기엔 상당히 어려움이 있음
- 앱이 정상 동작으로 대응한 확장과 아닌 것에 따라,
  서버가 일부 확장을 수정으로 인식하고 수정을 피해서 적절한 정책을 구사해야 함
- 그래서 remain, backport or drop

3.2: 약속된 자유의 개척지

DEVIEW 2021

점진적 배포 가능한 지점이 많아지고, 배포 요청도 많아지고

- 이젠 적정 수준에서 자유도를 공유해야 하지 않을까?
- 서버와 구성 설정이 있다면, 내부 운영 도구가 있는게 인지상정

3.2: 약속된 자유의 개척지

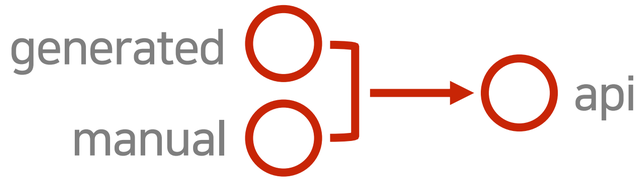
이번에도 초기 목표 설정을 실패하고 시작

- 소스 데이터 유닛 UI를 전부 만들고
- join-expand-aggregate 및 검증 규칙을 서버에 투입하고
- 완성된 결과값이 앱에서 어떻게 보일지 미리보기 UI까지 만들어서
- 개발자 개입이 불필요한 운영도구 시대 도래

3.2: 약속된 자유의 개척지

DEVIEW 2021

어드민 UI: 빠른 시제품 fast prototyping 접근



- 서드파티 기기는 주로 메뉴 고정, 기기 이미지 URL만 변함
- 기기 이미지 URL만 가져오면 이런 서드파티 기기를 등록할 수 있게 하자
- “이미지 URL을 입력하면, 바로 실서비스에 반영된다구요…? (난처)”
- 일반적으로 운영 도구 이용자는 거의 최종 이용자와 비슷한 수준으로 생각해야 함

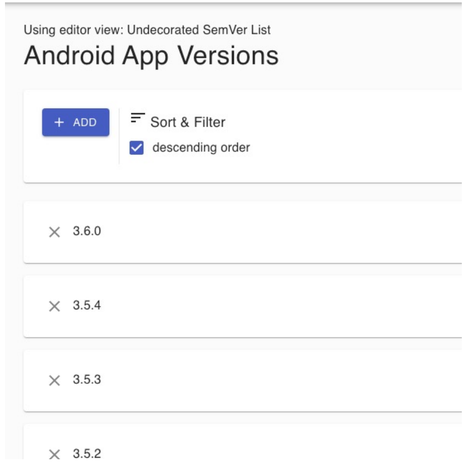
3.2: 약속된 자유의 개척지

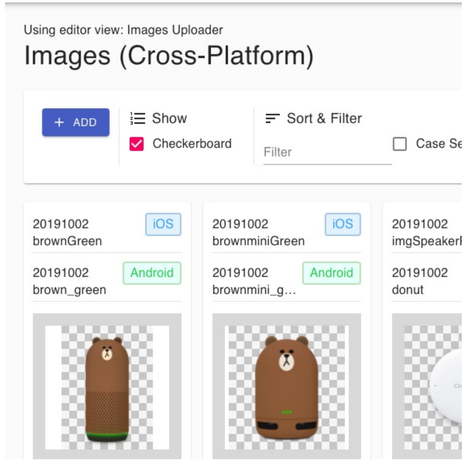
DEVIEW 2021

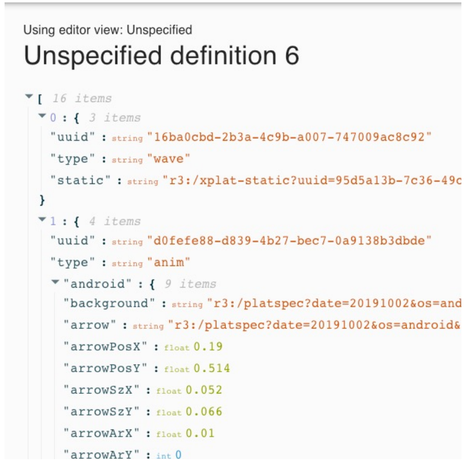
어드민 UI: 점진적 유형화 gradual typing 접근

- 최종 이용자에게 데이터의 유형은 곧 데이터 조작 UI의 자유도
- 소스 데이터 유닛 UI를 운영 도구 이용자에게 제공
- 운영 도구에서 만들어진 데이터: 검증, API 제공할 결과 생성, 배포 → 개발자가 수행
- 소스 데이터의 유형은 확장과 리팩토링 과정에서 늘어 가고…

* 실무 도입되지 않은 시스템 스크린샷 포함







3.2: 약속된 자유의 개척지

DEVIEW 2021

기존 소스 데이터에서 이미 존재하는 오류가 발견되기도 함

프로덕션에서 정상 동작할 결과를 내는 한, 어떤 건 검증 단계에서 실패 처리하긴 무리

```
user@host r3 % magick identify $(find . -type f -name sallymini_green.png | sort)
./images-20191002/android/hdpi/sallymini_green.png PNG 413x437 413x437+0+0 8-bit sRGB 23897B 0.000u 0:00.000
./images-20191002/android/xhdpi/sallymini_green.p  550x583 550x583+0+0 8-bit sRGB 38372B 0.000u 0:00.000
./images-20191002/android/xxhdpi/sallymini_gree G 825x875 825x875+0+0 8-bit sRGB 66156B 0.000u 0:00.000
./images-20191002/android/xxxhdpi/sallymini_gr  NG 1100x1166 1100x1166+0+0 8-bit sRGB 82190B 0.000u 0:00.000
user@host r3 % magick identify $(find . -type f  ame minionsmini_green.png | sort)
./images-20191002/android/hdpi/minionsmini_green.png PNG 413x437 413x437+0+0 8-bit sRGB 20830B 0.000u 0:00.000
./images-20191002/android/xhdpi/minionsmini_green.p  550x583 550x583+0+0 8-bit sRGB 34517B 0.000u 0:00.000
./images-20191002/android/xxhdpi/minionsmini_gree G 825x875 825x875+0+0 8-bit sRGB 60340B 0.000u 0:00.000
./images-20191002/android/xxxhdpi/minionsmini_gr  NG 1100x1166 1100x1166+0+0 8-bit sRGB 90290B 0.000u 0:00.000
user@host r3 % magick identify $(find . -type f -n  e device_on.png | sort)
./images-20200408/android/hdpi/device_on.png PNG 413x438 413x438+0+0 8-bit sRGB 156999B 0.000u 0:00.000
./images-20200408/android/xhdpi/device_on.p  550x584 550x584+0+0 8-bit sRGB 269245B 0.000u 0:00.000
./images-20200408/android/xxhdpi/device_on  G 825x876 825x876+0+0 8-bit sRGB 499297B 0.000u 0:00.000
./images-20200408/android/xxxhdpi/device  NG 1100x1168 1100x1168+0+0 8-bit sRGB 375330B 0.000u 0:00.000
user@host r3 %
```

모범적인 정상 값

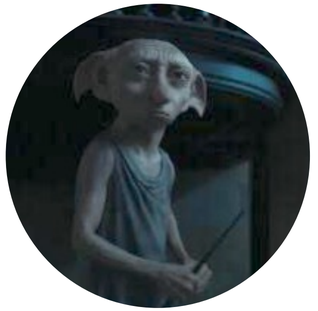
유형 오류로 실패 처리하고 싶지만,
그럭저럭 정상 값

3.2: 약속된 자유의 개척지

DEVIEW 2021

꿈은 가깝지만 현실적인 목표는 멀고 과정은 험한 편

- 자유 (그런데 자유로운 집요정의 자유)
- 그러나 만약 이 작업을 시작하지도 않았다면?
- 원래 알던 현실의 문제들이지만, 데이터로 만들고 보니 더 정확히 보임



“Dobby is a free elf!” Harry Potter and the Chamber of Secrets

3.3: FAQ & Conclusion

DEVIEW 2021

자주 들은 질문

- 정말 이걸 혼자 다 하셨나요? → 거의 그렇습니다
- 일이 계속 바뀌는데 힘들진 않으신가요? → 당연히 쉽지 않습니다

3.3: FAQ & Conclusion

자주 들은 질문 2

RN, Flutter 같은 프레임워크를 잘 쓰면 안되나요? ➔

- 이 주제에 있어서는 그럴듯해 보입니다만;
- 단말 오디오, 연락처, BLE 등 여러 데이터 소스를 적절히 다루기 쉽지 않고,
  각 OS 최신 API를 따라가야 하는 일도 자주 발생합니다.
  (개인정보, 백그라운드, 미디어, 푸시 알림 …)
- 각 OS SDK가 제공하는 Kotlin, Swift 언어와 개발 환경 이점을 포기하기 어렵습니다.

RN: React Native
BLE: Bluetooth Low Energy

3.3: FAQ & Conclusion

자주 들은 질문 3

관계형 데이터베이스RDB처럼 일부 검증이 내장된 형식을 사용한다면 소스 데이터나 API 결과값 검증에 손을 덜 수 있지 않나요? ➔

- 관계 대수적 제약으로 표현하기 힘든데 검증해 줘야 하는 것이 많습니다.
- 데이터 유형은 다양하고 수량은 적어서 RDBMS의 이점을 보지 못합니다.
- 점진적 유형화 접근에 대해서는 스키마 없는 데이터가 오히려 낫습니다. (like NoSQL)

3.3: FAQ & Conclusion

자주 들은 질문 4

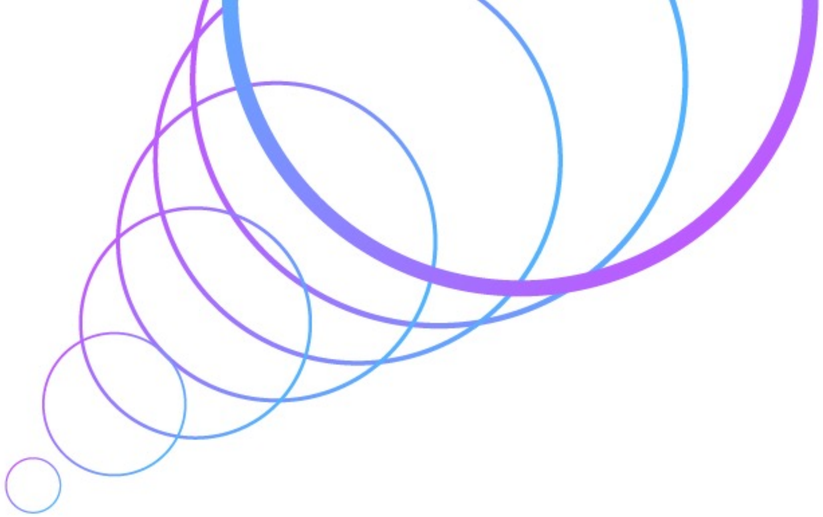
그래도 서버 방식으로 데이터를 잘 다뤄야 이용자에 따라 응답을 달리해 개발 데모나 A/B 테스팅을 수행할 수 있지 않나요? ➔

- 맞습니다. 조만간 그렇게 할 수 있으면 좋겠습니다.
- 어느 정도는 현재 이 부분에서도 workaround를 구사하고 있습니다.
- 각 앱 개발자들이 변화를 주도할 수 있는 workflow를 크게 제약하지 않고 편의를 제공하는 방법을 고안하고 있습니다.

3.3: FAQ & Conclusion

“Android 앱 개발자는 왜 자진해서 서버 개발자가 되었나”

- 결국 그들에게 필요했던 API 서비스: 앱에서 필요한 JSON 파일을 (거의) 정적 서빙
- 체계화, 유지보수, 배포 안정성 관리 등: 꼭 서버사이드 파이프라인일 필요 없음
- 운영 도구: 앱 시대의 이용자가 원하는 것은 개발자들의 상식보다 훨씬 다양하고,
  기존 협업 도구들의 범위는 제한적, 문서는 장황해짐, 작업 규칙은 소실됨
- 앱 디자이너, 기획자와 긴밀하게 소통하기 위해 좋은 도구를 만들어 가는 도전



Thanks for watching!