

딥링크 솔루션 직접 만들어보기

JSConf 2022

최수범, Subeom Choi, 0xWOF @AB180

발표자 소개

최수범, Subeom Choi, 0xWOF

- Software Engineer, SDK Team Lead @AB180
- Working on...
 - Airbridge - 웹애통합 마케팅 광고 성과 분석 솔루션
 - iOS SDK, Web SDK - 광고 성과 데이터 수집
 - React Native SDK, Cordova SDK, Flutter SDK - iOS SDK, Android SDK Wrapping
 - Deeplink SDK - 딥링크 기능
- Interested in...
 - Privacy Preserving Advertising - 개인정보보호를 보장하는 광고기법
 - Apple SKAdNetwork, PCM
 - Google Privacy Sandbox



목차

1. 딥링크란 무엇인가
2. 딥링크 솔루션 미리보기
3. HTTP 딥링크 (공식 명칭은 아니지만 편의상 사용합니다.)
4. Scheme 딥링크 (공식 명칭은 아니지만 편의상 사용합니다.)
5. 정리

이해를 돋기위해 엄밀하지 못한 정보가 포함되어 있습니다.

1. 딥링크란 무엇인가

“딥링크는 뭐고, 왜 필요하고, 솔루션은 왜 필요할까”

하이퍼링크란 무엇인가?

“웹의 특정화면을 연결하는 링크 (고리)”

딥링크란 무엇인가?

“앱의 특정화면을 연결하는 링크”

하이퍼링크 vs 딥링크

	하이퍼링크	딥링크
이동 시	서비스가 변화	앱이 변화
작동여부	서비스를 제공하는 한 작동함	앱이 설치되어 있어야 함
신뢰성	언제나 일관적으로 작동	그때그때 다름
비용	기본적으로 사용가능	추가 개발 필요

Why 딥링크

- 앱의 사용자 경험이 좋아짐
- 앱의 사용자 이탈율을 줄여줌

=> 사용자에게 앱 사용을 유도하기 위해서

- 앱은 더 편리한 기능을 제공할 수 있음
- 앱은 더 많은 사용자의 정보를 얻을 수 있음
 - 더 좋은 상품을 만드는데 활용
 - 더 관심가질 상품을 제공하는데 활용

=> 높은 매출

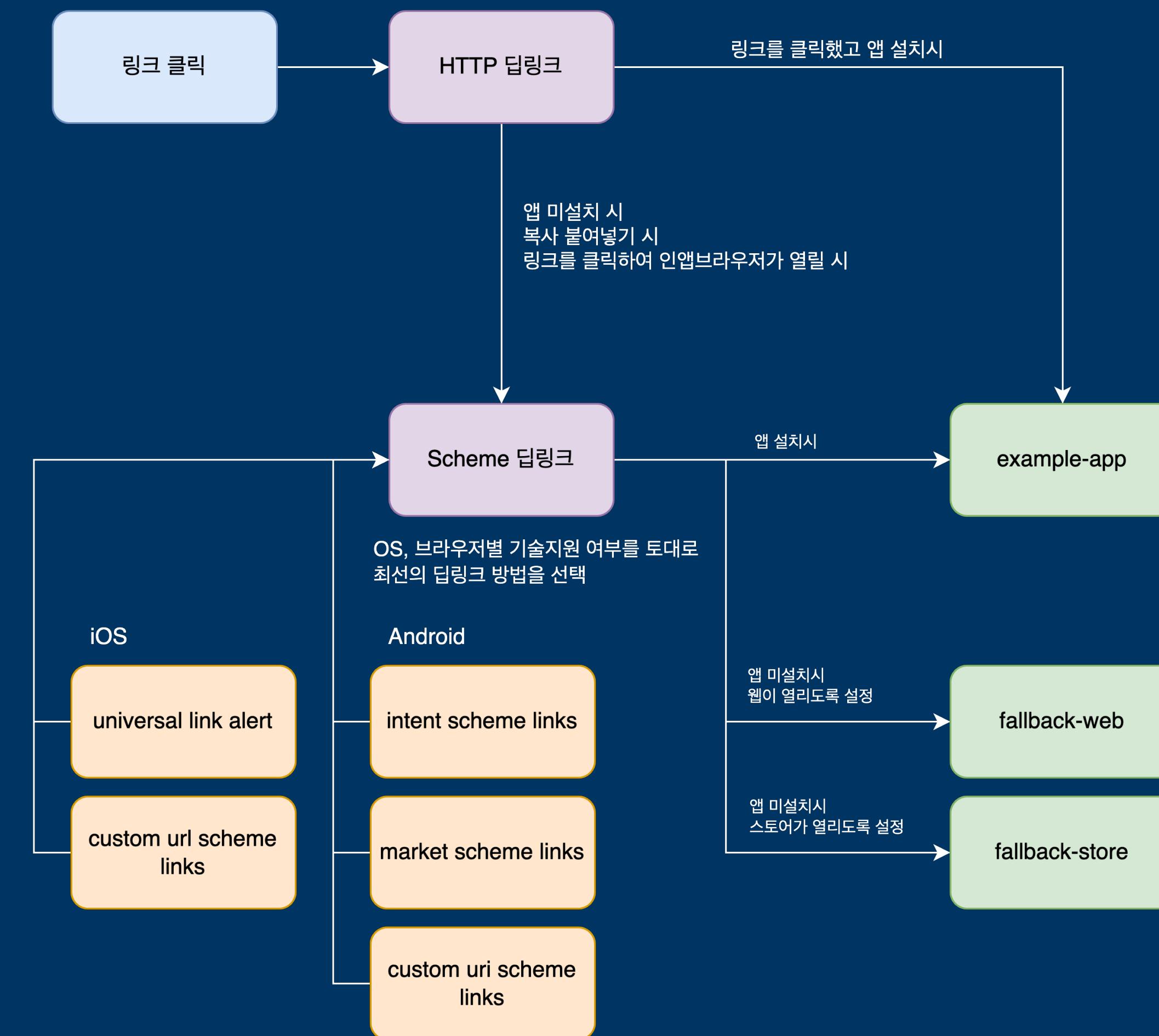
딥링크의 문제를 해결하자

- 작동여부
 - 문제: 앱이 설치되어 있어야만 작동한다.
 - 해결: 앱이 설치되어 있지 않을 때, 스토어나 웹으로 이동하는 기능을 개발
- 신뢰성
 - 문제: 그때그때 다르게 동작한다.
 - 해결: 상황마다 최적화된 딥링크 기술, 방법을 적용
- 비용
 - 문제: 추가 개발이 필요
 - 해결: 솔루션화 하여 제공

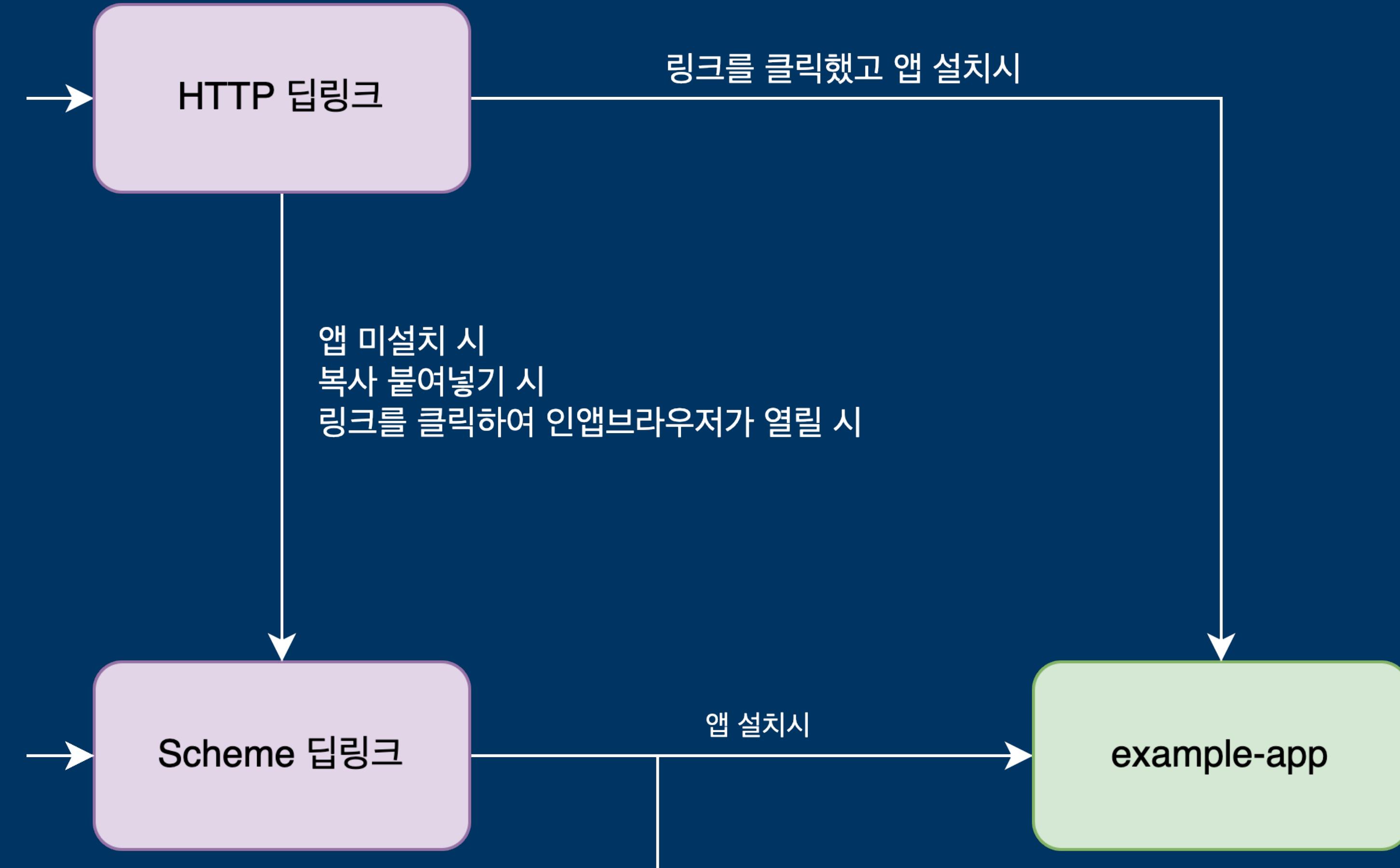
신뢰성: 가장 큰 문제, 집중할 문제

- OS 가 iOS 인가, Android 인가
- 앱이 설치되어 있는가, 설치되어 있지 않은가
- 앱이 설치되어 있지 않을 때 스토어로 이동하고 싶은가, 웹으로 이동하고 싶은가
- 딥링크를 브라우저에서 열었는가, 앱에서 열었는가, 앱의 인앱브라우저에서 열었는가
- 딥링크를 클릭했는가, 복사-붙여넣기 했는가, Javascript 로 리다이렉트 했는가
- 딥링크를 Short Link 서비스로 리다이렉트 했는가, 아닌가
- ...
- 경우의수 = BOOM!

2. 딥링크 솔루션 미리보기

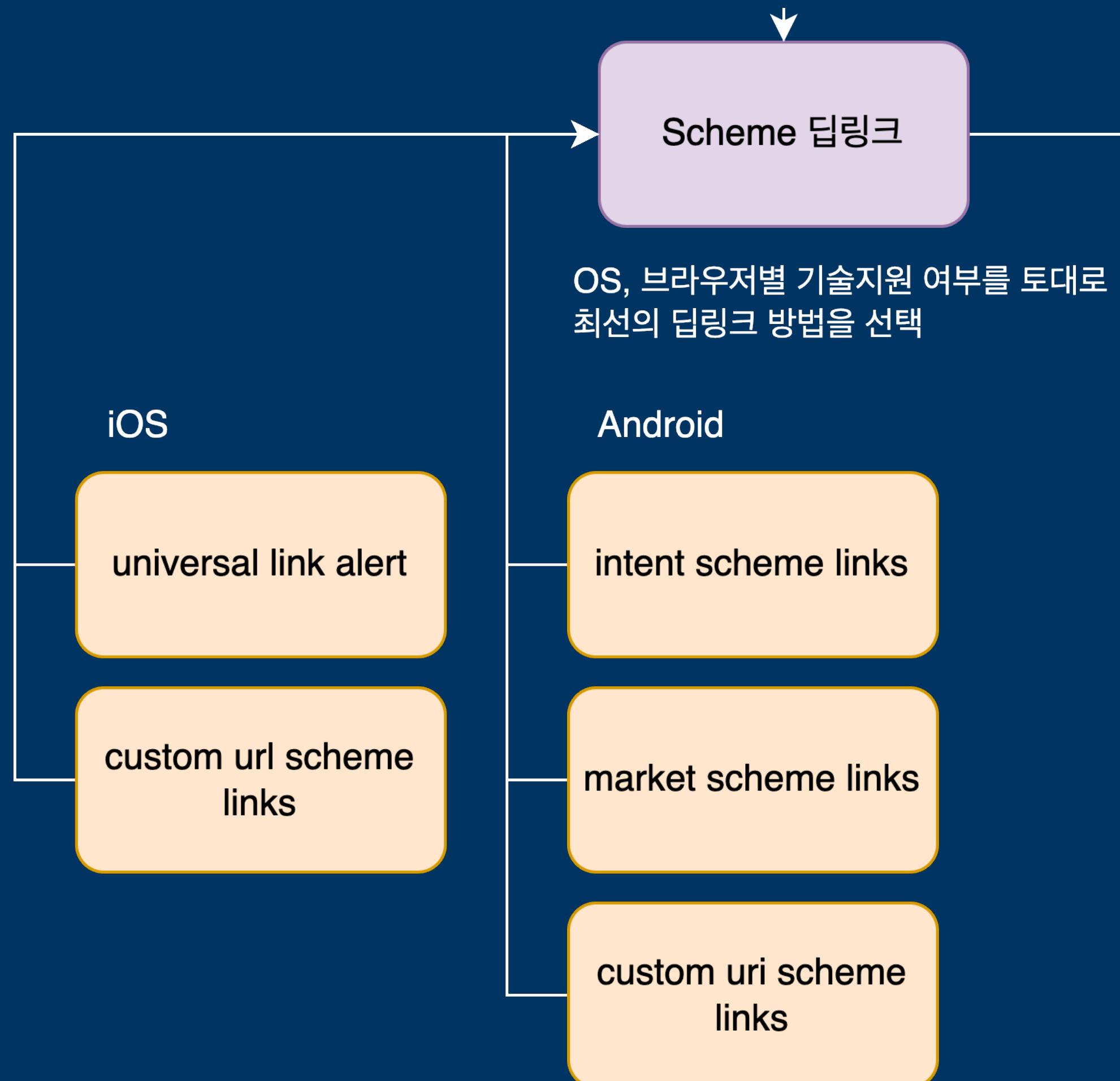


HTTP 딥링크



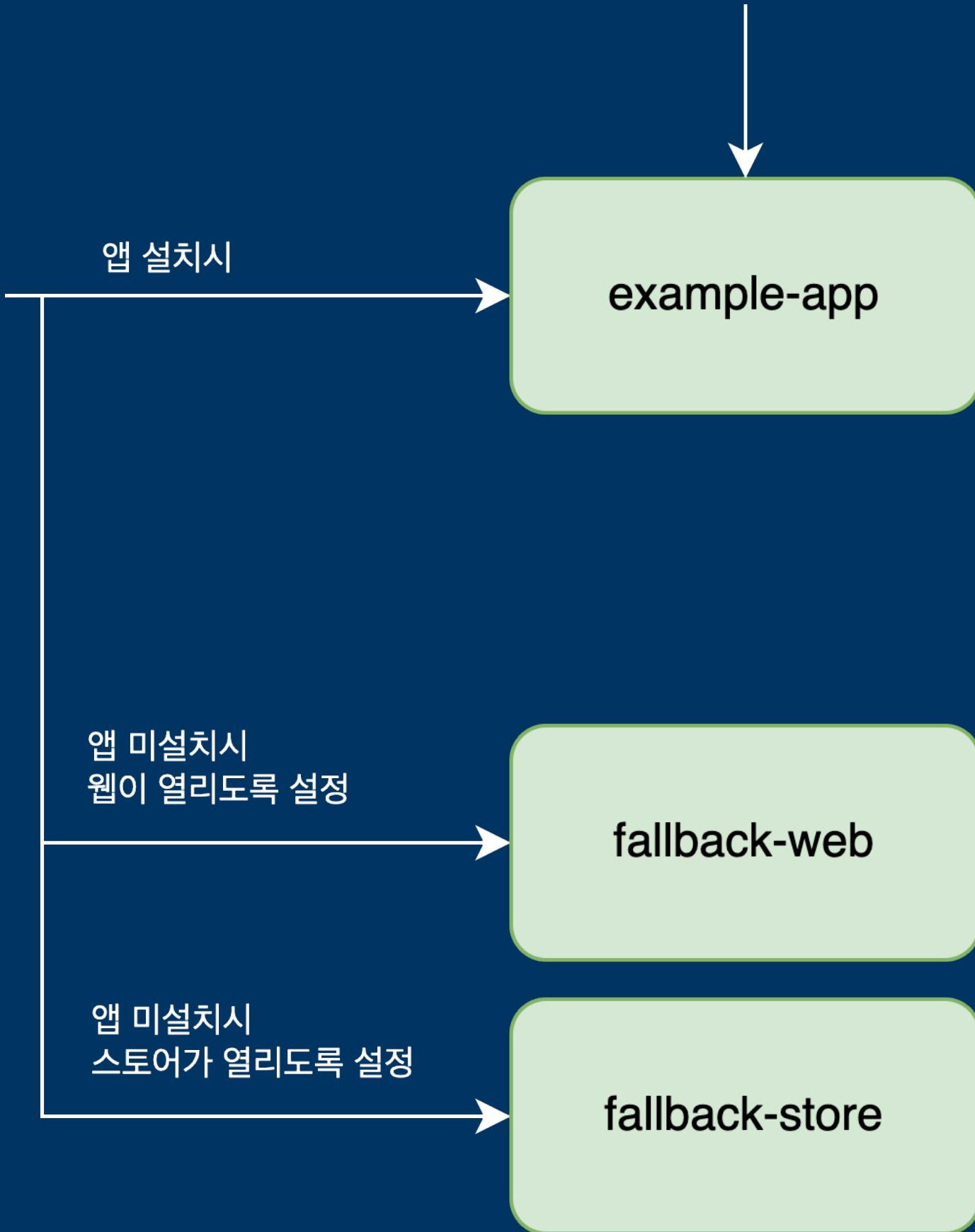
- HTTP 딥링크를 먼저 시도
- 성공시 => 앱이 열림
- 실패시 => Scheme 딥링크 시도

Scheme 딥링크, 최적화의 일반화, 예외적 최적화



- 시나리오별 기술지원 여부 고려
- 이에 따라 최적의 딥링크 기술 선택
- 시나리오별 예외적 최적화
- 할 수 있는 모든 것을 시도

원하는 결과

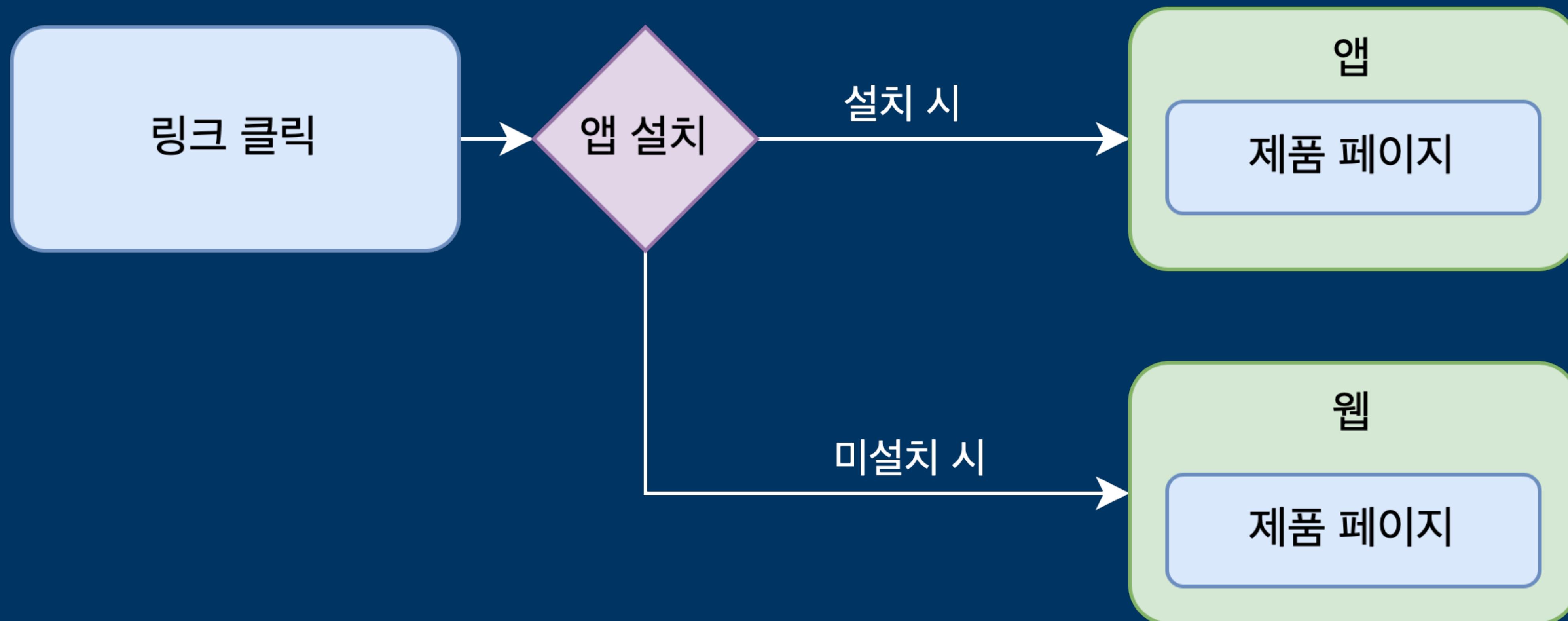


- 앱이 설치되어 있으면
 - 앱이 열림
- 앱이 설치되어 있지 않으면
 - 스토어가 열리거나
 - 웹이 열림

3. HTTP 딥링크

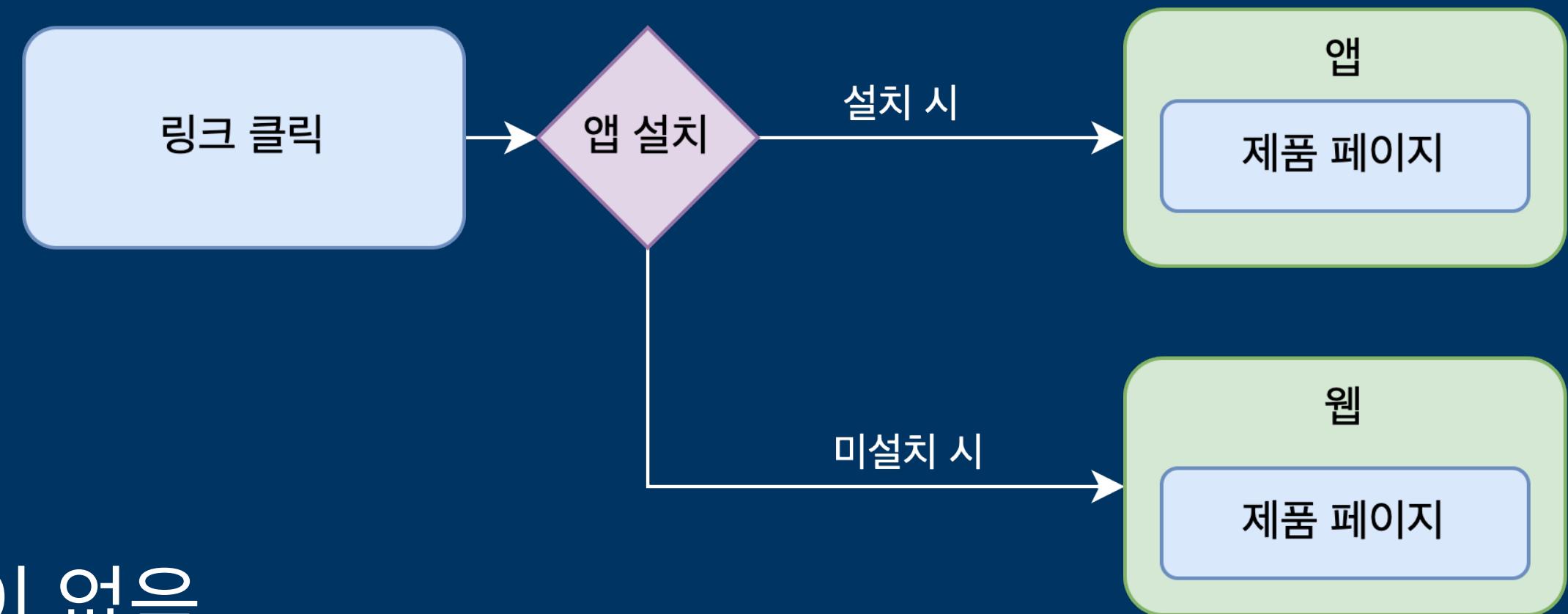
“차세대 딥링크 기능, 하지만 큰 제약, 완벽한 대체를 못함”

3. HTTP 딥링크 - 구조도



HTTP 딥링크의 특성

- “`https://~~~`” 의 웹 URL 형태
- 소유권 인증을 통해 보안성 좋음
 - 도메인 주인만 그 도메인으로 열릴 앱을 정함
 - 앱에서도 그 도메인을 사용할 것이라고 명시함
 - OS 입장에서 어떤 앱을 열어야 할지 모르는 상황이 없음
- 앱이 설치된 경우, 앱이 열림
- 앱이 설치안된 경우, 웹 URL 이 열리게 됨
- 유저 클릭이 보장되어야만 작동함
- iOS 공식명칭: Universal Links
- Android 공식명칭: App Links



딥링크 테스트 앱 준비

- expo init example-app
- expo prebuild (expo go 를 통해서는 테스트할 수 없음)
- Linking 을 이용해서 딥링크로 앱이 열리면 Alert 로 URL 을 표시하도록 개발

```
import { Linking } from 'react-native'

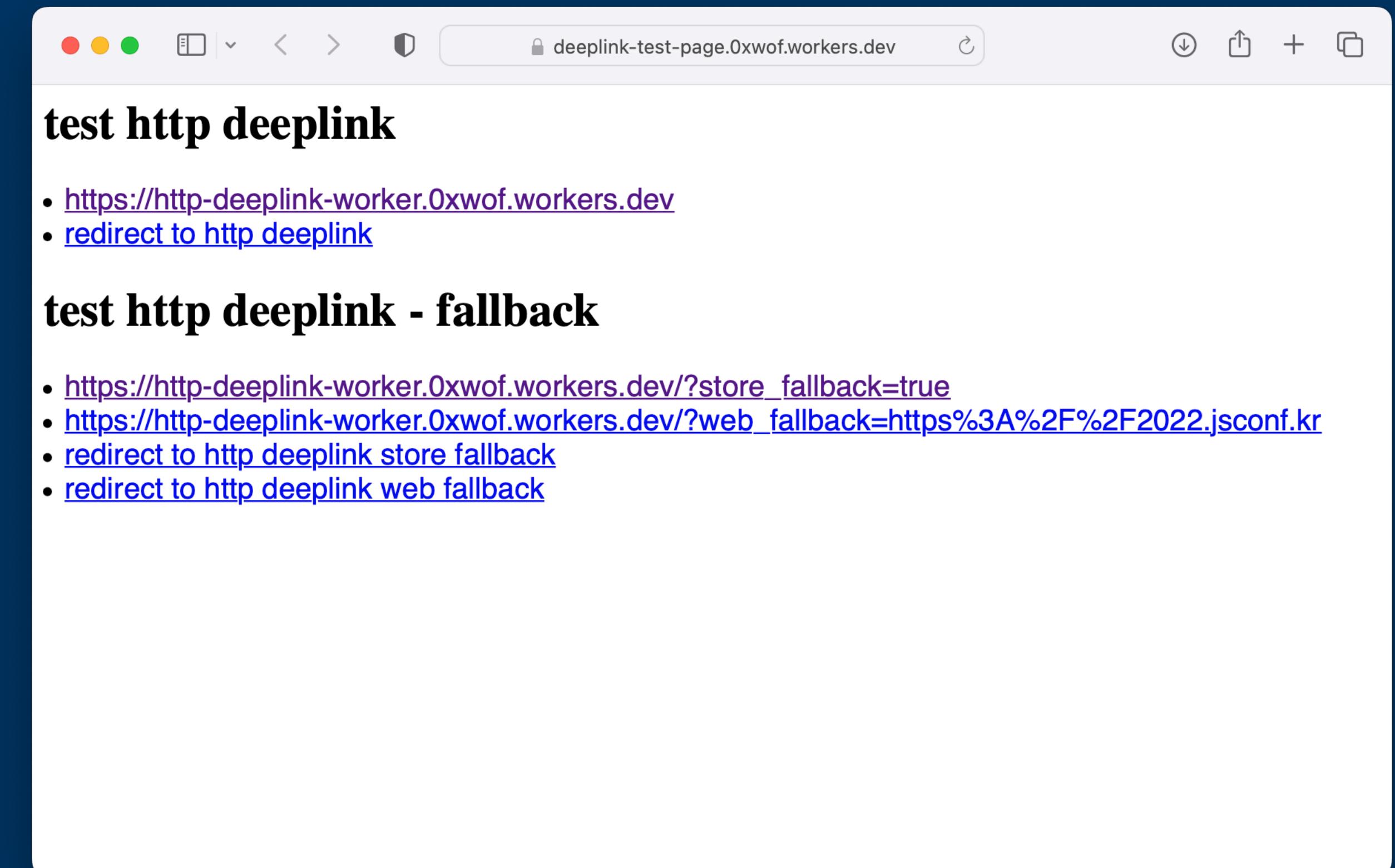
Linking.getInitialURL().then((url) => {
  if (url != null) {
    alert(url)
  }
})

Linking.addEventListener('url', ({ url }) => {
  alert(url)
})
```

딥링크 테스트 페이지 서버 준비

- <https://github.com/0xWOF/deeplink-solution-session-jsconf-2022>
- 위의 deeplink-test-page 를 가져와서 source/constant/cloudflare.ts 의 subdomain 을 변경하여 배포해주세요.
- npm run deploy

딥링크 테스트 페이지 서버 준비



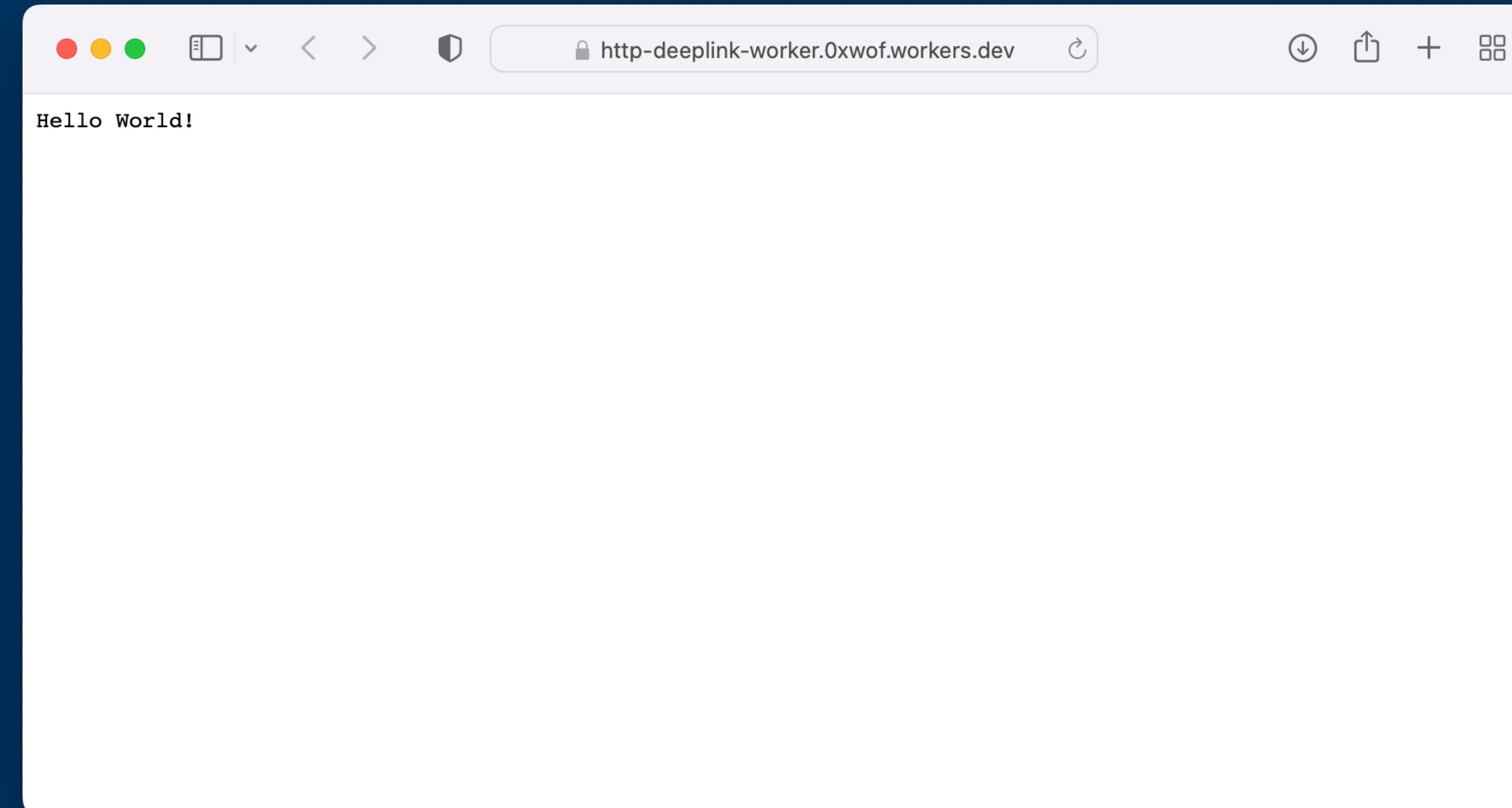
딥링크 솔루션 서버 준비

- wrangler init http-deeplink-worker
- cd http-deeplink-worker
- mv src source
- mv source/index.ts source/module.ts

딥링크 솔루션 서버 준비

- wrangler.toml 파일의 아래 내용을 수정해주세요.
 - main = “source/module.ts”
- 아래 명령어를 통해 배포해주세요.
 - npm run deploy

딥링크 솔루션 서버 준비 - 테스트



JSConf 2022 딥링크 솔루션 직접 만들어보기

HTTP 딥링크 - 소유권 증명 - iOS, Server

- “<https://~~~/.well-known/apple-app-site-association>”
- 위 URL에 아래 JSON을 호스팅해서 어떤 앱을 열어야 하는지 명시

```
{  
    "applinks": {  
        "apps": [],  
        "details": [{  
            "appID": "<developer team id>.<app bundle id>",  
            "paths": ["*"]  
        }]  
    }  
}
```

HTTP 딥링크 - 소유권 증명 - iOS, Server

- source/constant/application.ts

```
const APPLICATION = {  
  ios: {  
    teamID: '<your team id>',  
    bundleID: 'dev.wof.deeplinksolutionsession.exampleapp',  
  },  
}  
  
export { APPLICATION }
```

HTTP 딥링크 - 소유권 증명 - iOS, Server

- source/service/well_known_service.ts

```
import { APPLICATION } from '../constant/application'

const createDependency = () => {}

createDependency.createWellKnownService = () => ({
    APPLICATION,
})

const createWellKnownService = (): WellKnownService => {
    const { APPLICATION } = createDependency.createWellKnownService()

    const renderAppleAppSiteAssociation = ...

    return {
        renderAppleAppSiteAssociation,
    }
}

type WellKnownService = {
    renderAppleAppSiteAssociation: () => string
}

export { createWellKnownService, createDependency }
export type { WellKnownService }
```

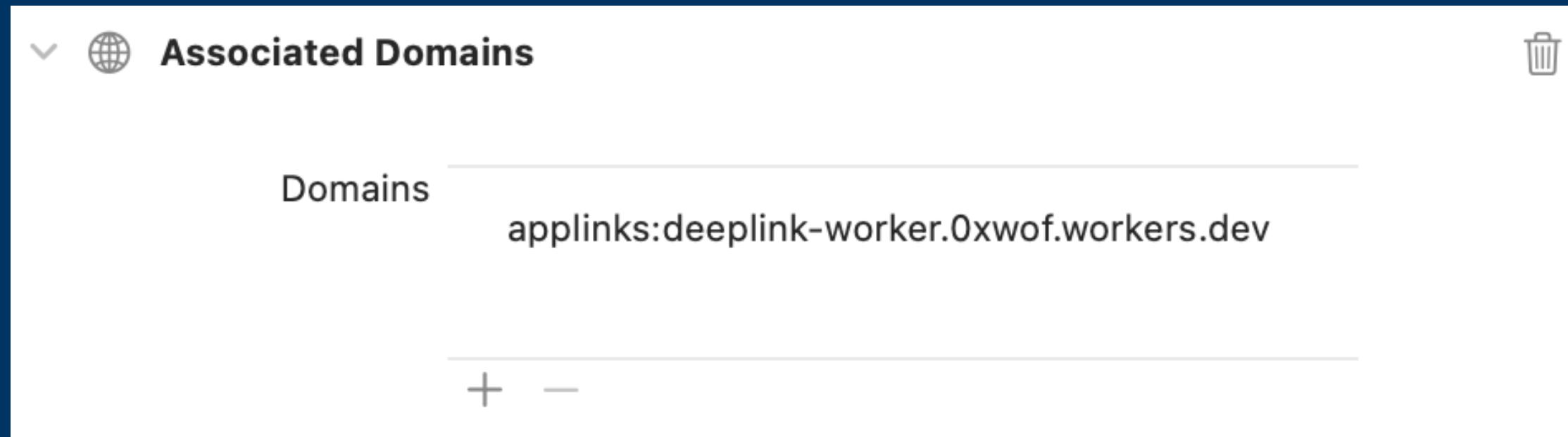
HTTP 딥링크 - 소유권 증명 - iOS, Server

- source/service/well_known_service.ts

```
const renderAppleAppSiteAssociation = () => JSON.stringify({  
    applinks: {  
        apps: [],  
        details: [{  
            appID: `${APPLICATION.ios.teamID}.${APPLICATION.ios.bundleID}`,  
            paths: ['*'],  
        }],  
    },  
})
```

HTTP 딥링크 - 소유권 증명 - iOS, App

- Xcode 의 Associated Domains 설정을 통해 어떤 도메인으로 앱이 열리는지 설정
- 설정된 도메인이 해당 앱을 연다는 증명을 호스팅하고 있으면 동작하게 됨
- 도메인을 여러개 설정하는 경우, 여러개중 하나라도 호스팅이 안되면 모두 동작하지 않음
 - 앱이 열리지 않음



- 테스트 앱 (Expo) 에서는 app.json 에 아래 JSON 을 추가하는 것으로 설정

```
"ios": {  
  "supportsTablet": true,  
  "bundleIdentifier": "dev.wof.deeplinksolutionsession.exampleapp",  
  "associatedDomains": [  
    "applinks:http-deeplink-worker.<your sub domain>.workers.dev"  
  ]  
},
```

HTTP 딥링크 - 테스트 - iOS



JSConf 2022 딥링크 솔루션 직접 만들어보기

HTTP 딥링크 - 소유권 증명 - Android, Server

- “<https://~~~/.well-known/assetlinks.json>”
- 위 URL에 아래 JSON을 호스팅해서 어떤 앱을 열어야 하는지 명시

```
[{
  "relation": [
    "delegate_permission/common.handle_all_urls"
  ],
  "target": {
    "namespace": "android_app",
    "package_name": "<app package name>",
    "sha256_cert_fingerprints": [
      "<sha256 of keychain>"
    ]
  }
}]
```

HTTP 딥링크 - 소유권 증명 - Android, Server

- source/constant/application.ts

```
const APPLICATION = {
  android: {
    package: 'dev.wof.deeplinksolutionsession.exampleapp',
    fingerprint: '<your fingerprint>',
  },
}

export { APPLICATION }
```

HTTP 딥링크 - 소유권 증명 - Android, Server

- source/service/well_known_service.ts

```
const createWellKnownService = (): WellKnownService => {
    const { APPLICATION } = createDependency.createWellKnownService()

    ...

    const renderAssetLinks = ...

    return {
        ...
        renderAssetLinks,
    }
}

type WellKnownService = {
    ...
    renderAssetLinks: () => string
}

export { createWellKnownService, createDependency }
export type { WellKnownService }
```

HTTP 딥링크 - 소유권 증명 - Android, Server

- source/service/well_known_service.ts

```
const renderAssetLinks = () => JSON.stringify([
  relation: [
    'delegate_permission/common.handle_all_urls',
  ],
  target: {
    namespace: 'android_app',
    package_name: APPLICATION.android.package,
    sha256_cert_fingerprints: [
      APPLICATION.android.fingerprint,
    ],
  },
})])
```

HTTP 딥링크 - 소유권 증명 - Android, App

- AndroidManifest.xml 에 Intent Filter 설정을 통해 어떤 도메인으로 앱이 열리는지 설정
- 설정된 도메인이 해당 앱을 연다는 증명을 호스팅하고 있으면 동작하게 됨
- 도메인 여러개 설정하는 경우, 여러개중 하나라도 호스팅이 안되면 모두 동작하지 않음
 - 앱은 열리지만 중복이 없음이 보장되지 않음 (중복시 앱을 열기전 Alert 창이 표시됨)

```
<intent-filter android:autoVerify="true">
    <action android:name="android.intent.action.VIEW"/>
    <data android:scheme="https" android:host="http-deeplink-worker.<subdomain>.workers.dev"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
```

HTTP 딥링크 - 소유권 증명 - Android, App

- 테스트앱 (Expo) 에서는 app.json 에 아래 JSON 을 추가하는 것으로 설정

```
"intentFilters": [{}  
  "autoVerify": true,  
  "action": "VIEW",  
  "data": {  
    "scheme": "https",  
    "host": "http-deeplink-worker.<subdomain>.workers.dev"  
  },  
  "category": [  
    "BROWSABLE",  
    "DEFAULT"  
  ]  
]
```

HTTP 딥링크 - 테스트 - Android



HTTP 딥링크 - Fallback 구현

- source/constant/store.ts

```
const STORE = {  
    ios: 'https://apps.apple.com/app/id375380948',  
    android: 'market://details?  
id=dev.wof.deeplinksolutionsession.exampleapp',  
}  
  
export { STORE }
```

HTTP 딥링크 - Fallback 구현

- source/utility/user_agent.ts

```
const check_iOS = (userAgent: string) => (
  /(iOS) (\d+)(?:[-_. ](\d+))?(?:[-_. ](\d+))?/.test(userAgent)
  || /(CPU OS|iPhone OS|CPU iPhone|CPU iPhone OS) (\d+)(?:[-_. ](\d+))?(?:[-_. ](\d+))?/.test(userAgent)
  || /(iPhone|iPad|iPod|iOS;)/.test(userAgent)
)

const check_Android = (userAgent: string) => (
  /(Android|Adr) (\d+)(?:[-_. ](\d+))?(?:[-_. ](\d+))?/.test(userAgent)
  || /(Android)/.test(userAgent)
)
```

HTTP 딥링크 - Fallback 구현

- source/utility/user_agent.ts

```
const createUserAgent = (userAgent: string): UserAgent => {
  const os = (
    check_iOS(userAgent) ? 'ios'
    : check_Android(userAgent) ? 'android'
    : 'other'
  )

  return {
    os,
  }
}

...

type UserAgent = {
  os: OS
}

type OS = 'ios' | 'android' | 'other'

export { createUserAgent }
export type { UserAgent, OS }
```

HTTP 딥링크 - Fallback 구현

- source/service/fallback_service.ts

```
import { STORE } from '../constant/store'
import { createUserAgent } from '../utility/user_agent'

const createDependency = () => {}

createDependency.createFallbackService = () => ({
    STORE,
})

const createFallbackService = (): FallbackService => {
    const { STORE } = createDependency.createFallbackService()

    const renderStoreFallback = ...
    const renderWebFallback = ...

    return {
        renderStoreFallback,
        renderWebFallback,
    }
}

type FallbackService = {
    renderStoreFallback: (userAgent: string) => string
    renderWebFallback: (url: string) => string
}

export { createFallbackService, createDependency }
export type { FallbackService }
```

HTTP 딥링크 - Fallback 구현

- source/service/fallback_service.ts

```
const renderStoreFallback = (userAgent: string) => `<script>
  var a = document.createElement('a');
  a.href = `${createUserAgent(userAgent).os == 'ios' ? STORE.ios
    : STORE.android}`;
  a.click();
</script>`
```

HTTP 딥링크 - Fallback 구현

- source/service/fallback_service.ts

```
const renderWebFallback = (fallback: string) => `<script>
  var a = document.createElement('a');
  a.href = '${fallback}';
  a.click();
</script>`
```

HTTP 딥링크 - 최종 테스트 - iOS



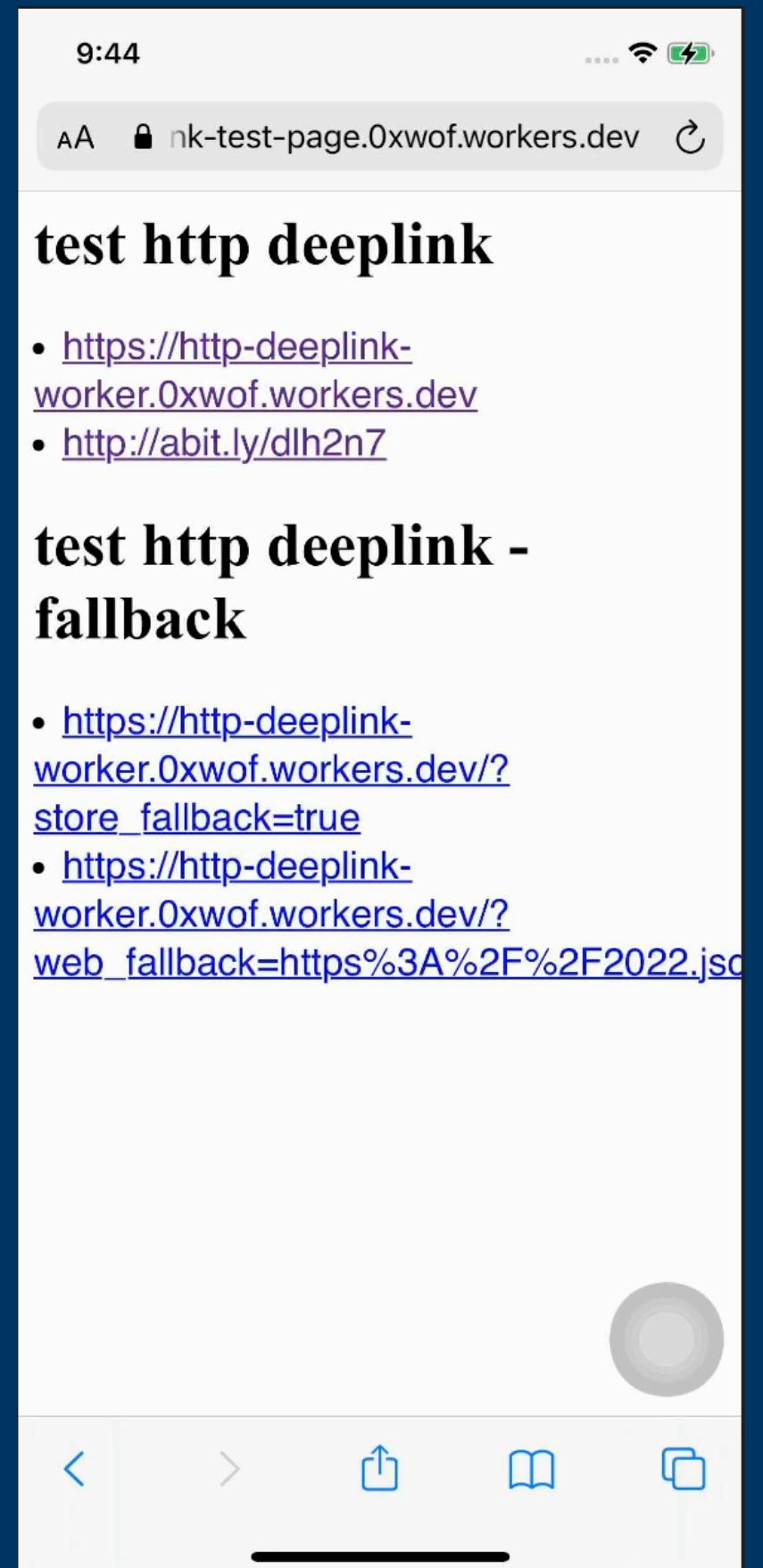
앱 미설치시, Store Fallback



앱 미설치시, Web Fallback



앱 설치시, Store Fallback



앱 설치시, Web Fallback

HTTP 딥링크 - 최종 테스트 - Android



앱 미설치시, Store Fallback



앱 미설치시, Web Fallback



앱 설치시, Store Fallback



앱 설치시, Web Fallback

HTTP 딥링크 - 한계 - 새 탭에서 열기

- HTTP 딥링크는 “새 탭에서 열기”를 클릭해서 열면 앱이 설치되어 있어도 앱이 열리지 않음
- 1번이라도 “새 탭에서 열기”로 HTTP 딥링크를 열면, 그 도메인에 대해서 OS 내부적으로 checking 됨
- checking 된 도메인은 이후, “클릭”으로 열어도 앱이 설치되어 있어도 앱이 열리지 않게 됨
- checking 된 도메인은 다시 앱이 설치된 상태에서 “[앱 이름]에서 열기”로 열면, 그 checking 이 해제됨
- 아래 대화의 주요한 원인 중 하나 (ㅜㅜ...)
 - OO님, 딥링크로 앱이 안열려요!
 - 네? 제 기기에서는 잘 되는데요...



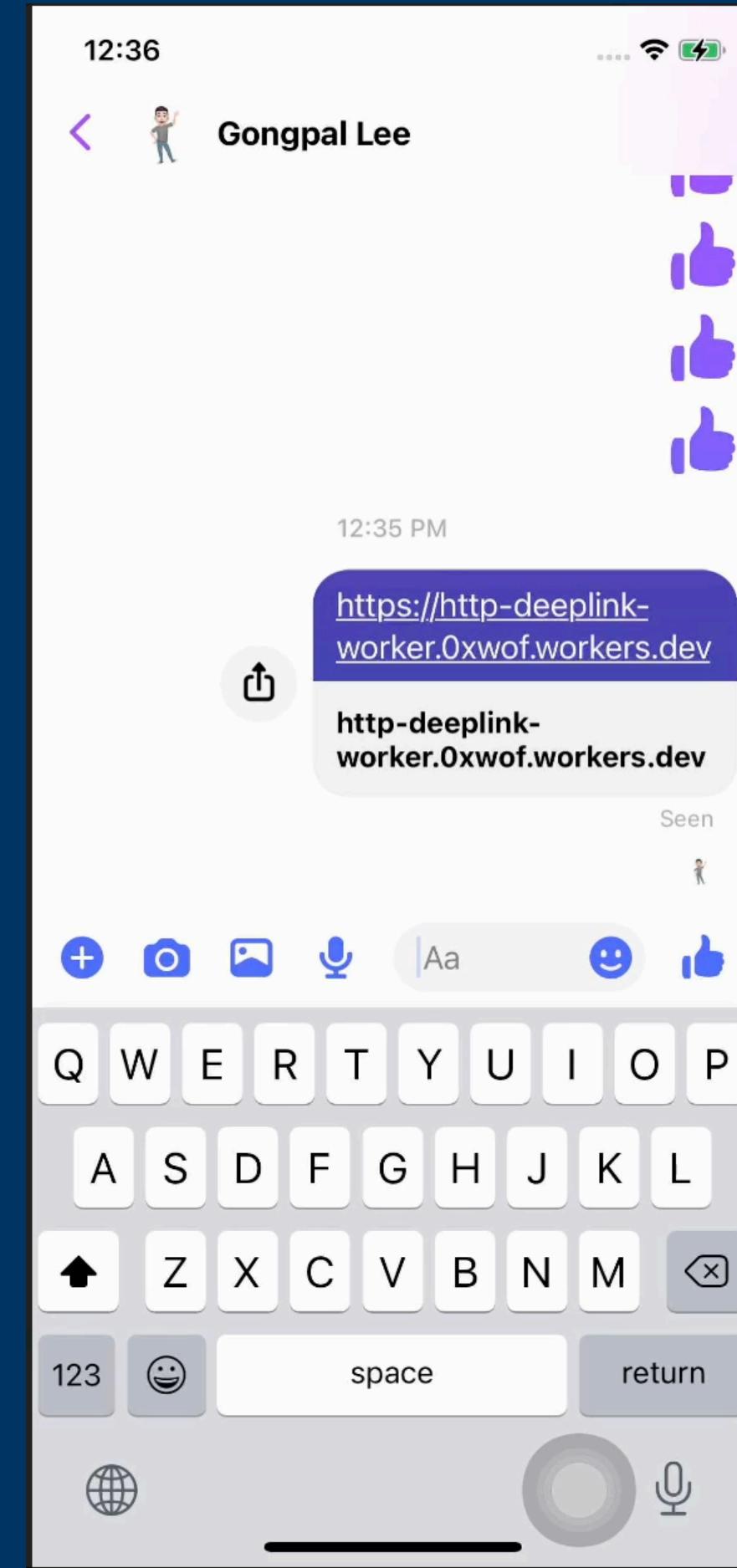
HTTP 딥링크 - 한계 - 복사 붙여넣기

- HTTP 딥링크는 “복사 붙여넣기”로 여는 경우 앱이 설치되어 있어도 앱이 열리지 않음
- 하지만 “새 탭에서 열기”처럼 OS 상에서 checking은 안됨. 이후 “클릭”으로 여는 경우 앱이 열림
- 보통의 경우, 딥링크를 웹페이지에 포함시키고 “클릭”해서 테스트하지 않고, 주소창에 “복사 붙여넣기”해서 테스트함
 - 오해의 원인이 되는 경우가 많음



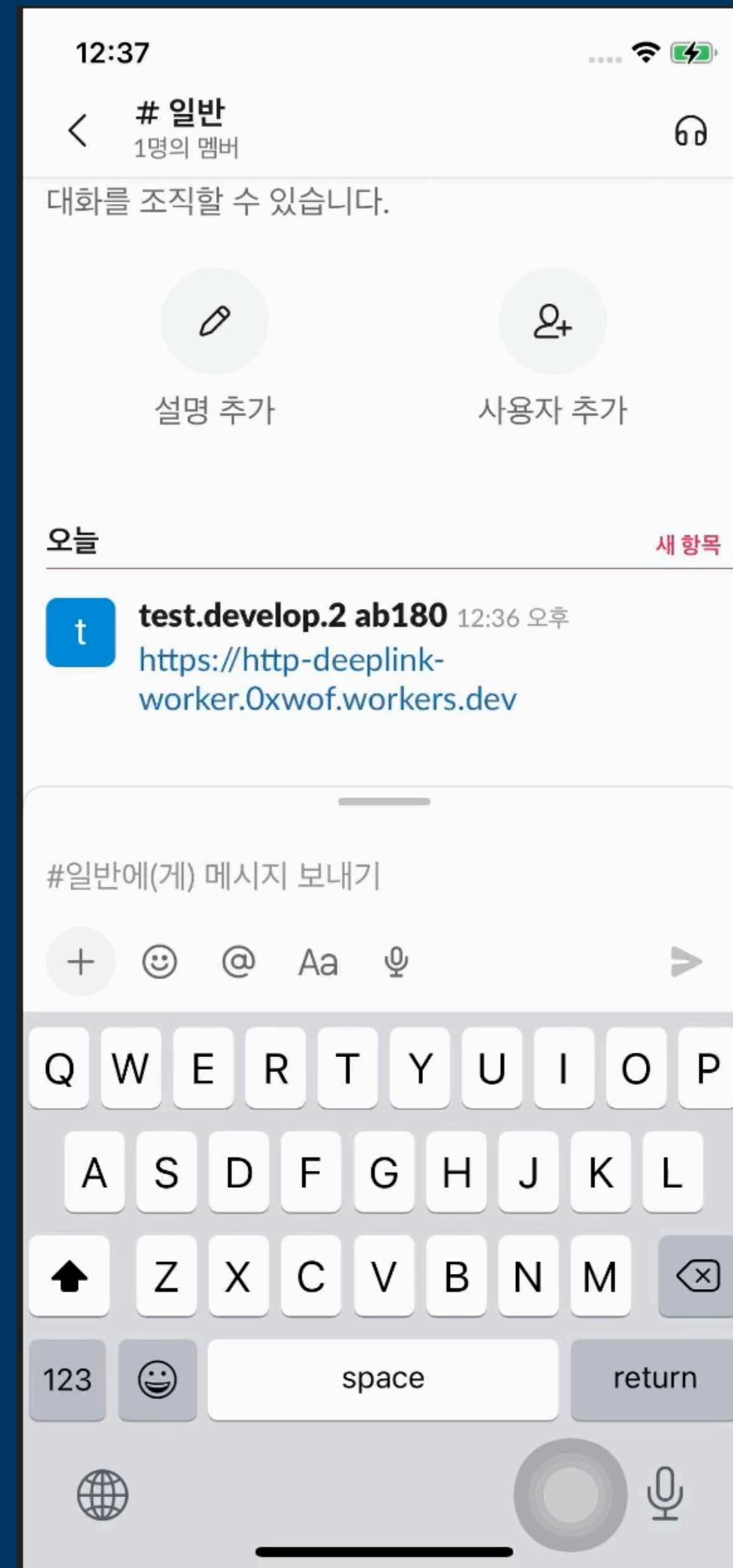
HTTP 딥링크 - 한계 - 인앱브라우저 열림

- HTTP 딥링크는 클릭된 영역(앱)과, 열리는 영역(인앱브라우저)이 다른 경우 앱이 설치되어 있어도 앱을 열지 않음
- 이 문제로 인해 HTTP 딥링크를 눌러 인앱브라우저가 열리는 경우에는 동작하지 않음
- 다음 앱들이 이 경우에 해당함
 - Facebook
 - Facebook Messenger
 - Kakaotalk
 - ...



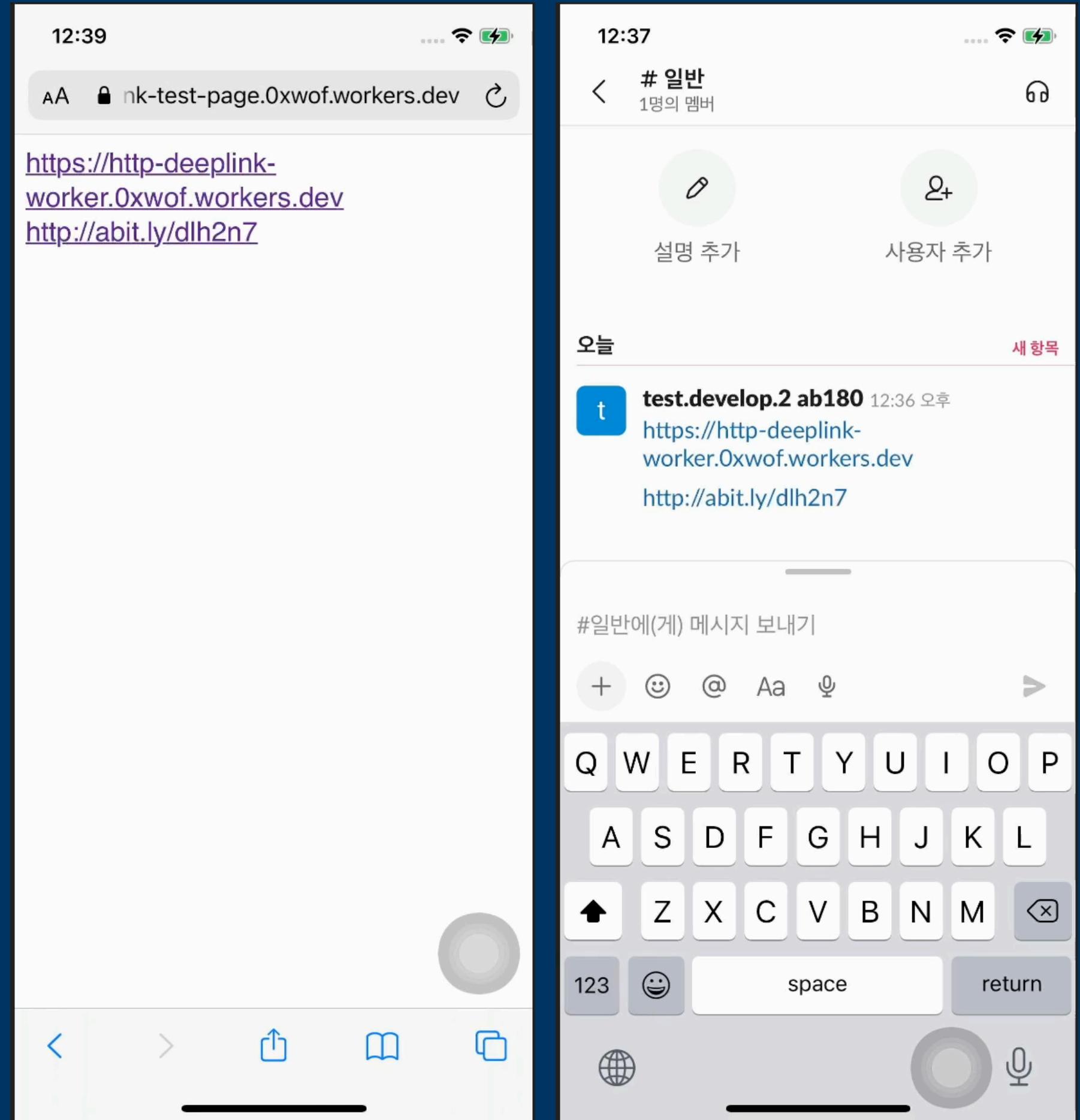
HTTP 딥링크 - 한계 - 인앱브라우저 열림

- 하지만 앱에 따라서 인앱브라우저로 링크를 열기전에 HTTP 딥링크인지 확인하도록 개발하는 경우가 있음 (앱에서 HTTP 딥링크를 열도록)
- 이 경우에는 HTTP 딥링크는 앱을 열고, 아니면 인앱브라우저가 열리는 방식으로 동작하게 됨
- 다음 앱들이 이에 해당함
 - Slack
 - Gmail
 - ...



HTTP 딥링크 - 한계 - 리다이렉트

- 여러가지 이유로 HTTP 딥링크를 Short Link 서비스로 감싸는 경우가 존재함
- 이 경우, 브라우저에서는 정상동작함
- 그리고 HTTP 딥링크를 먼저 확인하지 않는 앱에서 인앱브라우저 열리는 경우도 동일하게 동작함
- 하지만 HTTP 딥링크를 먼저 확인하는 앱에서 인앱브라우저 열리는 경우, 원래 앱이 열리는 것이 열리지 않게 됨 (앱에서 먼저 HTTP 딥링크인지 확인하는데 Short Link 가 되어서 도메인이 달라져서 확인이 불가능해지게 됨)



HTTP 딥링크 - 한계 - Android - 리다이렉트

- iOS 는 내장 WebView 가 기본적으로 HTTP 딥링크를 지원함
- 하지만 Android 는 내장 WebView 에 별도의 개발을 해야만 HTTP 딥링크가 지원됨
- 이로인해 Android 는 HTTP 딥링크가 지원 안 될 가능성이 상대적으로 높음

HTTP 딥링크 - 정리

HTTP 딥링크가
지원되지 않을 수 있음

HTTP 딥링크	iOS	Android
브라우저	●	▲
앱 (HTTP 딥링크 확인O)	▲	▲
앱 (HTTP 딥링크 확인X)	✗	✗
앱의 인앱브라우저 (...)	●	▲

Short Link 가 되면
작동하지 않음

클릭한 영역과
열리는 영역이 달라서

4. Scheme 딥링크

“구세대 딥링크 기능, 중복에 취약, 호환성과 UX 를 위해 필요”

4. Scheme 딥링크 - 특성

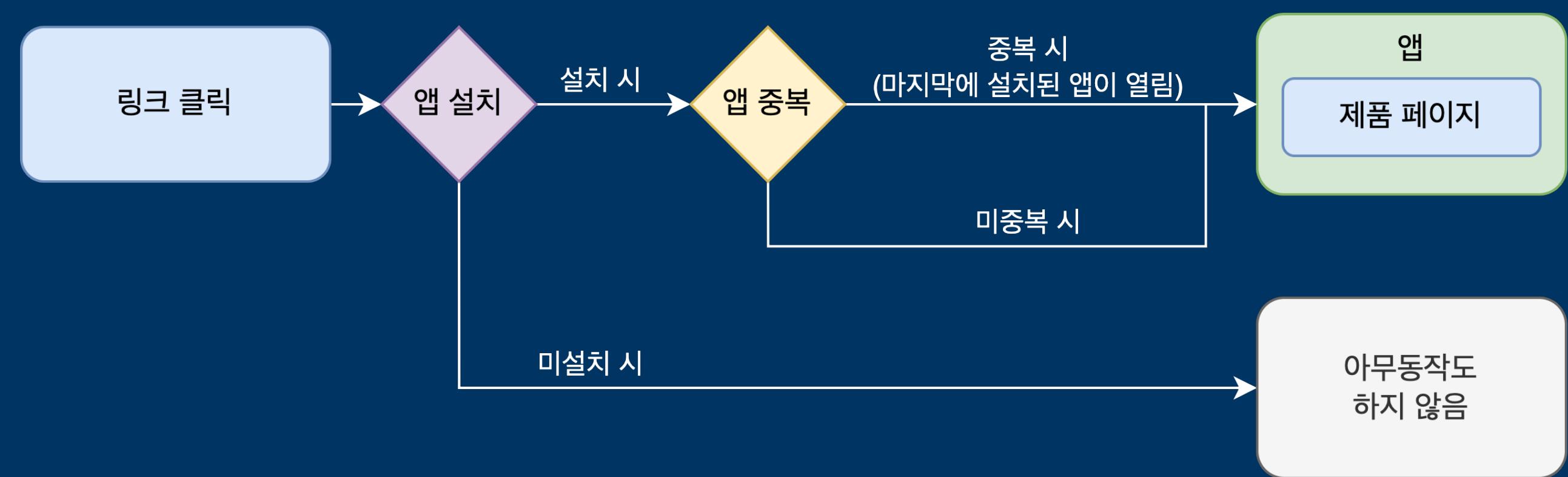
- “scheme://~~~” 의 URL 형태
 - 앱마다 scheme 이 다름 (예: googlechromes://~~~)
- 소유권인증 기능이 없어서 중복된 앱이 하나의 scheme 을 사용할 수 있음
 - 앱이 중복되는 경우, iOS 는 마지막에 설치된 앱이 열림
 - Android 는 선택창이 표시되고 선택된 앱이 열림 (이 선택창이 딥링크 UX 에 영향을 줍니다.)
- 앱이 설치된 경우, 앱이 열림
- 앱이 설치안된 경우, 종류에 따라 다르게 동작함
- 높은 확률로 유저 클릭이 보장되지 않아도 작동함
- 다양한 기술과 구현방법이 존재하고, 그것들이 각 앱, 인앱브라우저마다 다르게 동작할 수 있음

4. Scheme 딥링크 - 종류

- iOS
 - Custom URL Scheme
- Android
 - Custom URI Scheme
 - Market Scheme
 - Intent Scheme

4. Scheme 딥링크 - Custom URL/URI Scheme

- 소유권 보장 기능이 없어서 앱이 중복되면 마지막에 설치된 앱이 열림
- 앱이 설치되어 있으면 앱이 열림
- 아니면 아무동작도 하지 않음
- 이 특성을 이용해서 Javascript를 통해 Fallback을 직접 구현하는 것 가능함



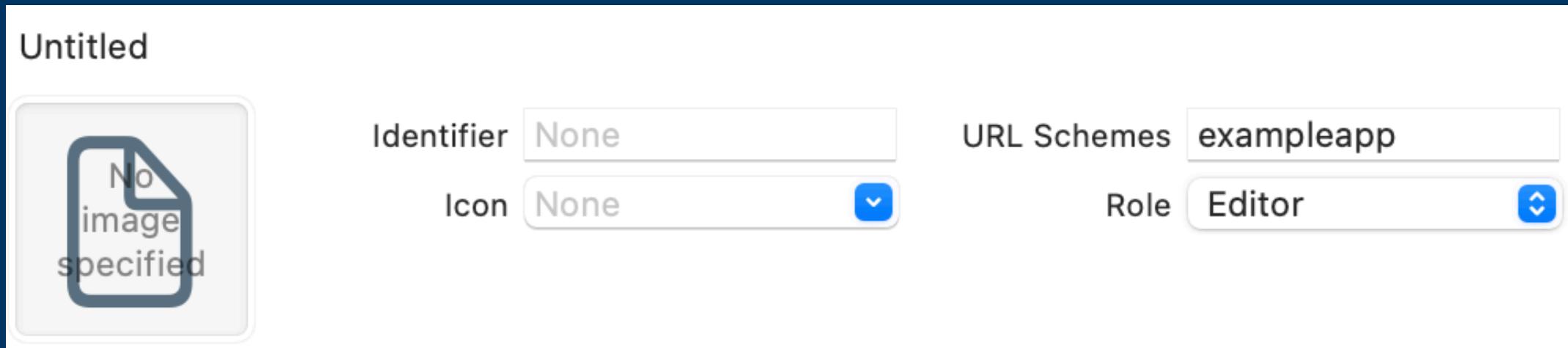
4. Scheme 딥링크 - Custom URL/URI Scheme

- 앱이 설치되어 있다면
 - 1번째 라인에 의해 앱이 열림
 - 2번째 라인에 의해 3번째 라인이 지연됨
 - 3번째 라인이 실행되기 이전에 프로세스 실행권한이 브라우저에서 앱으로 전환되었기 때문에 3번째 라인이 실행되지 않음
- 앱이 설치되어 있다면
 - 1번째 라인에 의해 앱이 열리지 않음
 - 2번째 라인에 의해 3번째 라인이 지연됨
 - 3번째 라인이 실행되면서 스토어가 열림

```
location.replace('scheme://deeplink?page=1')
setTimeout(() => {
    location.replace('market://details?
id=dev.wof.deeplinksolutionsession.exampleap
p')
}, 100)
```

4. Scheme 딥링크 - Custom URL/URI Scheme

- Xcode 의 Info 설정을 통해 어떤 Scheme 을 가진 URL 로 앱이 열리는지 설정 (iOS)



- AndroidManifest.xml 의 Intent Filter 설정을 통해 어떤 Scheme 을 가진 URL 로 앱이 열리는 설정 (Android)

```
<intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <data android:scheme="exampleapp"/>
</intent-filter>
```

- 테스트앱 (Expo) 에서는 app.json 에 scheme key-value 를 추가하는 것으로 설정

```
"scheme": "exampleapp",
```

4. Scheme 딥링크 - Custom URL/URI Scheme - Server 에 적용

- wrangler init scheme-deeplink-worker
- cd scheme-deeplink-worker
- mv src source
- mv source/index.ts source/module.ts

4. Scheme 딥링크 - Custom URL/URI Scheme - Server 에 적용

- wrangler.toml 파일의 아래 내용을 수정해주세요.
 - main = “source/module.ts”
- 아래 명령어를 통해 배포해주세요.
 - npm run deploy

4. Scheme 딥링크 - Custom URL/URI Scheme - Server 에 적용

- http-deeplink-worker 의 constant/store.ts, utility/user_agent.ts 복제
- constant/application.ts

```
const APPLICATION = {  
  ios: {  
    scheme: 'exampleapp',  
  },  
  android: {  
    scheme: 'exampleapp',  
    package: 'dev.wof.deeplinksolutionsession.exampleapp',  
  },  
}  
  
export { APPLICATION }
```

4. Scheme 딥링크 - Custom URL/URI Scheme - Server 에 적용

- service/custom_url_scheme_service.ts

```
import { STORE } from '../constant/store'
import { createUserAgent } from '../utility/user_agent'

const createDependency = () => {}

createDependency.createCustomURLSchemeService = () => ({
    STORE,
})

const createCustomURLSchemeService = (): CustomURLSchemeService => {
    const { STORE } = createDependency.createCustomURLSchemeService()

    const renderStoreFallback = ...

    const renderWebFallback = ...

    return {
        renderStoreFallback,
        renderWebFallback,
    }
}

type CustomURLSchemeService = {
    renderStoreFallback: (deeplink: string, userAgent: string) => string
    renderWebFallback: (deeplink: string, fallback: string) => string
}

export { createCustomURLSchemeService }
export type { CustomURLSchemeService }
```

4. Scheme 딥링크 - Custom URL/URI Scheme - Server 에 적용

- service/custom_url_scheme_service.ts

```
const renderStoreFallback = (deeplink: string, userAgent: string) => `<script>
  var a = document.createElement('a');
  a.href = '${deeplink}';
  a.click();
  setTimeout(function () {
    var a = document.createElement('a');
    a.href = `${createUserAgent(userAgent).os == 'ios' ? STORE.ios
      : STORE.android
    }`;
    a.click();
  }, 100);
</script>
`;
```

4. Scheme 딥링크 - Custom URL/URI Scheme - Server 에 적용

- service/custom_url_scheme_service.ts

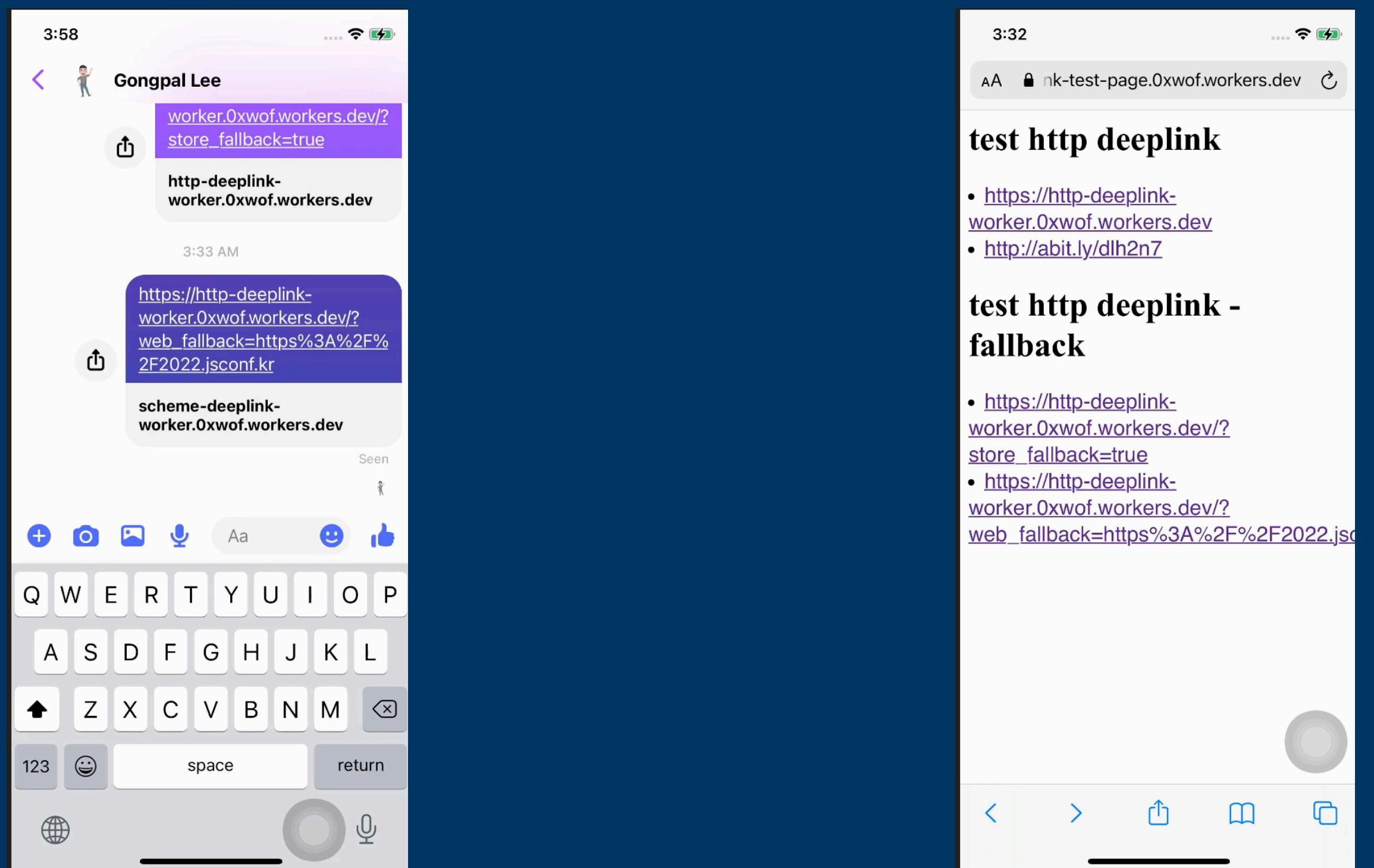
```
const renderWebFallback = (deeplink: string, fallback: string) => `<script>
  var a = document.createElement('a');
  a.href = '${deeplink}';
  a.click();
  setTimeout(function () {
    var a = document.createElement('a');
    a.href = '${fallback}';
    a.click();
  }, 100);
</script>`;
```

4. Scheme 딥링크 - Custom URL/URI Scheme - 테스트

- iOS - 인앱브라우저가 열리는 경우, 복사 붙여넣기 등 동작하도록 개선
- 하지만 여러 케이스가 존재함
 - iOS Safari Store Fallback => 앱이 설치되어 있으면, 앱과 스토어가 동시에 열림
 - Android Facebook Messenger Store Fallback => 앱이 설치되어 있으면 앱이 열리고, 뒤로가기 버튼을 누르면 스토어가 표시됨
 - Android Chrome 복사 붙여넣기 Store Fallback => 아무동작도 하지 않게 됨
 - Android Chrome 복사 붙여넣기 Web Fallback => 앱이 설치되어 있어도 Fallback 으로 이동함

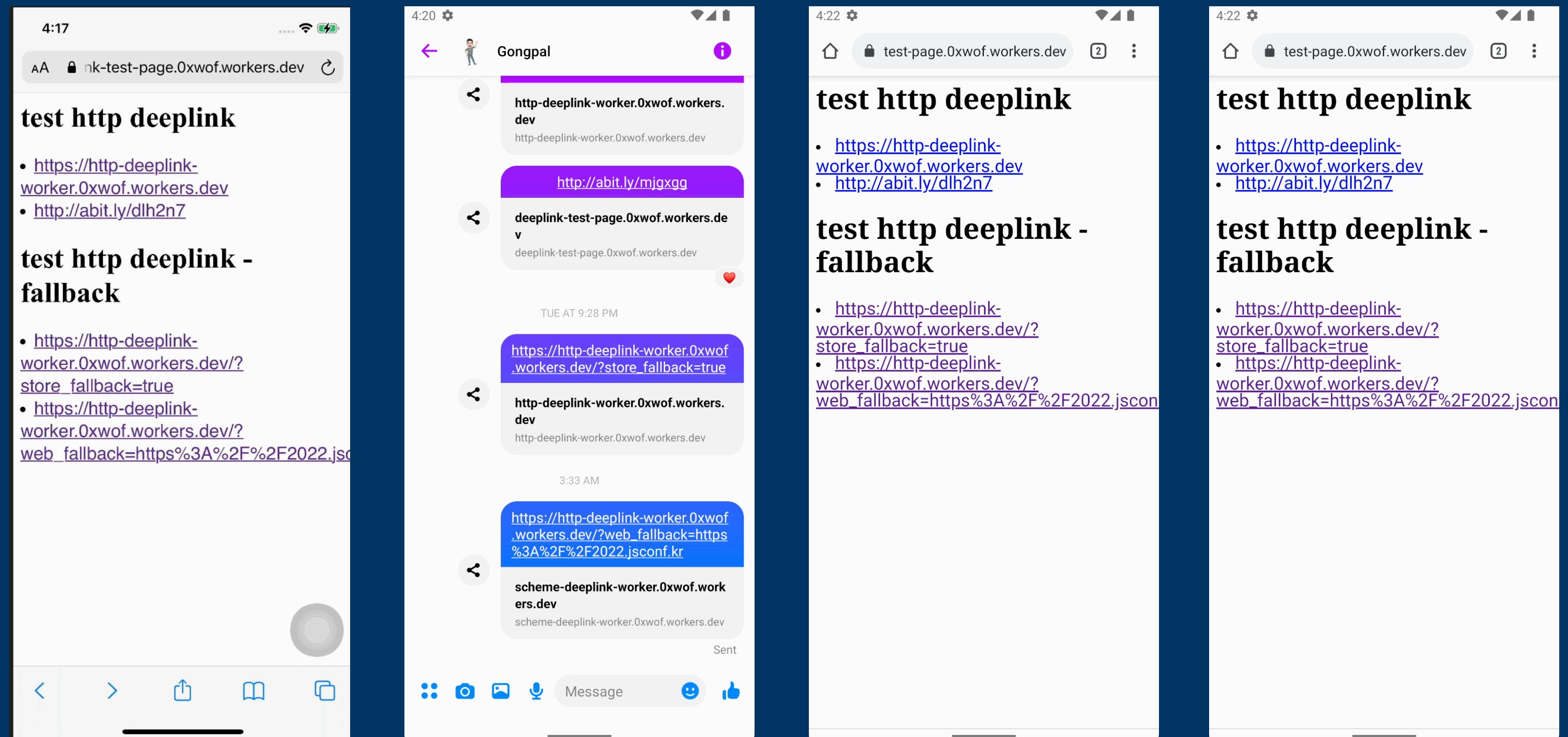
4. Scheme 딥링크 - Custom URL/URI Scheme - 테스트

- 개선된 경우



4. Scheme 딥링크 - Custom URL/URI Scheme - 테스트

- 에러 케이스



4. Scheme 딥링크 - 개선하기 - iOS

- 에러 케이스
 - iOS Safari Store Fallback => 앱이 설치되어 있으면, 앱과 스토어가 동시에 열림
- 문제의 원인 => iOS Safari에서 Scheme 방법이 제대로 동작하지 않음
- 해결 => iOS Safari에서 Scheme 방법을 사용하지 말자
 - => iOS Safari에서는 HTML로 Alert을 그려서 유저가 클릭을 하면 HTTP 딥링크로 리다이렉트 하자 (그럼 HTTP 딥링크의 “클릭” 조건도 달성 가능)

4. Scheme 딥링크 - 개선하기 - iOS

- constant/support.ts

```
const SUPPORT = {
  ios: {
    safari: { scheme: false },
    other: { scheme: true },
  },
} as const

export { SUPPORT }
```

4. Scheme 딥링크 - 개선하기 - iOS

- service/universal_links_alert_service.ts

```
const createUniversalLinksAlertService = (): UniversalLinksAlertService => {
  const render = ...
  return {
    render,
  }
}

type UniversalLinksAlertService = {
  render: (currentURL: string) => string
}

export { createUniversalLinksAlertService }
export type { UniversalLinksAlertService }
```

4. Scheme 딥링크 - 개선하기 - iOS

- service/universal_links_alert_service.ts
- license/sweetalert.md 에 라이센스를 명시

```
const render = (currentURL: string) => `<meta name="viewport" content="width=device-width, initial-scale=1">
<script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js"></script>
<script>
    window.onload = function () {
        swal('Move to other page...').then(function () {
            var a = document.createElement('a');
            a.href = `${currentURL.replace('://scheme-deeplink-worker', '://http-deeplink-worker') +
'&directFallback=true'}`;
            a.click();
        });
    };
</script>
`;
```

4. Scheme 딥링크 - 개선하기 - iOS

- utility/user_agent.ts

```
type UserAgent = {  
  os: OS  
  app: App  
}  
  
type OS = 'ios' | 'android' | 'other'  
type App = 'safari' | 'other'
```

```
const check_iOS_Safari = (userAgent: string) => (  
  /(Mac OS X).+Version\/.+Safari\//.test(userAgent)  
)
```

```
const app = (  
  check_iOS_Safari(userAgent) ? 'safari'  
  : 'other'  
)  
  
return {  
  os,  
  app,  
}
```

4. Scheme 딥링크 - 개선하기 - iOS - 테스트



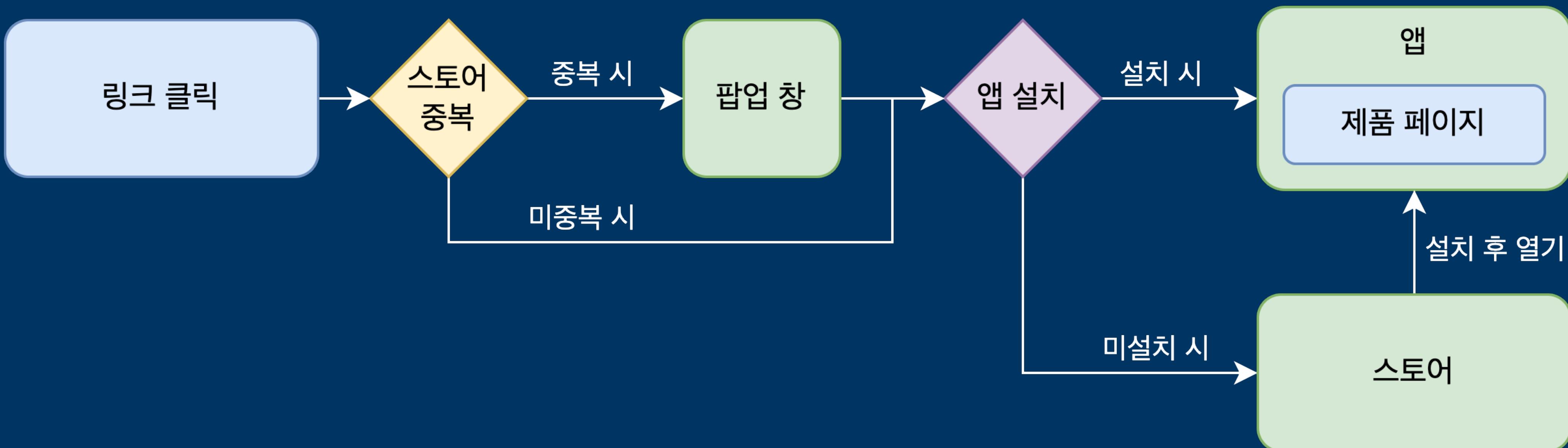
4. Scheme 딥링크 - 개선하기 - Android

- 에러 케이스
 - Android Facebook Messenger Store Fallback => 앱이 설치되어 있으면 앱이 열리고, 뒤로가기 버튼을 누르면 스토어가 표시됨
 - 해결 => Android에서는 Custom URL/URI Scheme 방식이 제대로 동작하지 않는 것으로 보임
 - => 다른 방법으로 딥링크해야 함

4. Scheme 딥링크 - Market Scheme

- scheme: market
- host: details
- query:
 - key: id, value: package
 - key: url, value: custom uri scheme url
- 설명: id 값에 해당하는 앱이 있으면 앱이 url 값으로 열리고 (딥링크) 아니면 해당하는 앱의 스토어 화면이 열림
- 특성: 스토어가 여러개인 경우, 선택창이 표시됨

4. Scheme 딥링크 - Market Scheme



```
market://details?id=dev.wof.deeplinksolutionsession.exampleapp&url=exampleapp%3A%2F%2F
```

4. Scheme 딥링크 - Intent Scheme

- scheme: intent
- host, path, query, ...: custom uri scheme url 의 값
- fragment:
 - #Intent;
 - scheme={custom uri scheme url 의 scheme};
 - package={앱의 package};
 - (optional) S.browserFallbackUrl={앱이 설치안되어 있을 때 이동할 url};
 - end;
- 설명: 앱이 있으면 앱을 custom uri scheme url 으로 열고 아니면 아무동작하지 않거나, S.browserFallbackUrl 이 있으면 그 url 로 이동함
- 특성: 선택창이 표시되지 않음
- 특성: 앱마다 S.browserFallbackUrl 을 지원하지 않는 경우가 있음
- 특성: 앱마다 앱이 설치되어 있지 않으면 무조건 스토어가 열리는 경우가 있음

4. Scheme 딥링크 - Intent Scheme

- Example
 - 앱을 열고 싶은 URL: exampleapp://main
 - Intent Scheme URL: intent://main#Intent;scheme=exampleapp;package=dev.wof.deeplinksolutions.exampleapp;end;

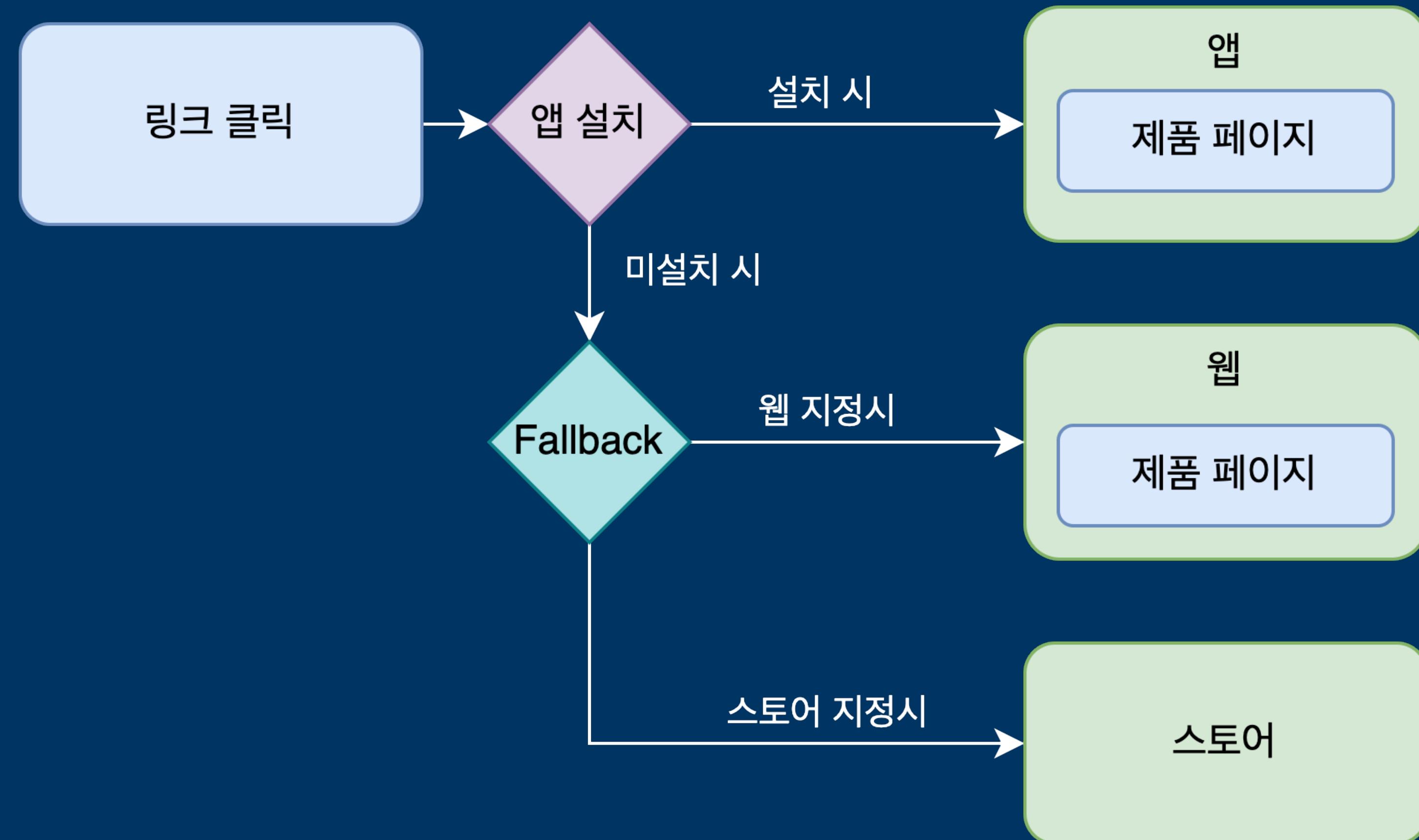
4. Scheme 딥링크 - Intent Scheme

- 응용: Store Fallback 하는 Intent Scheme URL
 - 앱을 열고 싶은 URL: exampleapp://main
 - Intent Scheme URL:
 - intent://details
 - ?id=dev.wof.deeplinksolutionsession.exampleapp
 - &url=exampleapp%3A%3F%3Fmain
 - #Intent;
 - scheme=market;
 - package=com.android.vending;
 - end;
- 설명: Google Play Store 를 Intent Scheme URL 로 지정해서 열고, 앱이 설치되어 있을 때 딥링크로 열리도록 설정

4. Scheme 딥링크 - Intent Scheme

- 응용: Web Fallback 하는 Intent Scheme URL
 - 앱을 열고 싶은 URL: exampleapp://main
 - Web Fallback URL: https://2022.jsconf.kr
 - Intent Scheme URL:
 - intent://main
 - #Intent;
 - scheme=exampleapp;
 - package=dev.wof.deeplinksolutionsession.exampleapp;
 - S.browserFallbackUrl=https%3A%2F%2F2022.jsconf.kr;
 - end;
- 설명: 테스트앱을 지정해서 열고, 앱이 없으면 Web Fallback URL로 이동

4. Scheme 딥링크 - Intent Scheme



4. Scheme 딥링크 - 개선하기 - Android

- constant/support.ts

```
android: {  
    facebook_messenger: { scheme: false, market: true, intent: false },  
    other: { scheme: false, market: true, intent: true },  
},
```

4. Scheme 딥링크 - 개선하기 - Android

- utility/user_agent.ts

```
const app = (
  check_iOS_Safari(userAgent) ? 'safari'
  : check_Facebook_Messenger(userAgent) ? 'facebook_messenger'
  : 'other'
)
```

```
const check_Facebook_Messenger = (userAgent: string) => (
  /\[(FBAN\//(?:MessengerLiteForiOS|MessengerForiOS)|FB_IAB\//(?:MESSENGER|Orca-Android))/.test(userAgent)
)
```

4. Scheme 딥링크 - 개선하기 - Android

- service/market_scheme_service.ts

```
import { APPLICATION } from '../constant/application'

const createDependency = () => {}

createDependency.createMarketSchemeService = () => ({
    APPLICATION,
})

const createMarketSchemeService = (): MarketSchemeService => {
    const { APPLICATION } = createDependency.createMarketSchemeService()

    const renderStoreFallback = ...

    return {
        renderStoreFallback,
    }
}

type MarketSchemeService = {
    renderStoreFallback: (deeplink: string) => string
}

export { createMarketSchemeService }
export type { MarketSchemeService }
```

4. Scheme 딥링크 - 개선하기 - Android

- service/market_scheme_service.ts

```
const renderStoreFallback = (deeplink: string) => `<script>
  var a = document.createElement('a');
  a.href = `${`market://details` +
    `?id=${APPLICATION.android.package}` +
    `&url=${encodeURIComponent(deeplink)}`}
  `;
  a.click();
</script>`
```

4. Scheme 딥링크 - 개선하기 - Android

- service/intent_scheme_service.ts

```
import { APPLICATION } from '../constant/application'

const createDependency = () => {}

createDependency.createIntentSchemeService = () => ({
    APPLICATION,
})

const createIntentSchemeService = (): IntentSchemeService => {
    const { APPLICATION } = createDependency.createIntentSchemeService()

    const renderStoreFallback = ...

    const renderWebFallback = ...

    return {
        renderStoreFallback,
        renderWebFallback,
    }
}

type IntentSchemeService = {
    renderStoreFallback: (deeplink: string) => string
    renderWebFallback: (deeplink: string, fallback: string) => string
}

export { createIntentSchemeService }
export type { IntentSchemeService }
```

4. Scheme 딥링크 - 개선하기 - Android

- service/intent_scheme_service.ts

```
const renderStoreFallback = (deeplink: string) => `<script>
  var a = document.createElement('a');
  a.href = `${`intent://details` +
    `?id=${APPLICATION.android.package}` +
    `&url=${encodeURIComponent(deeplink)}` +
    `#Intent;` +
    `scheme=market;` +
    `package=com.android.vending;` +
    `end;`;
  }`;
  a.click();
</script>`;
```

4. Scheme 딥링크 - 개선하기 - Android

- service/intent_scheme_service.ts

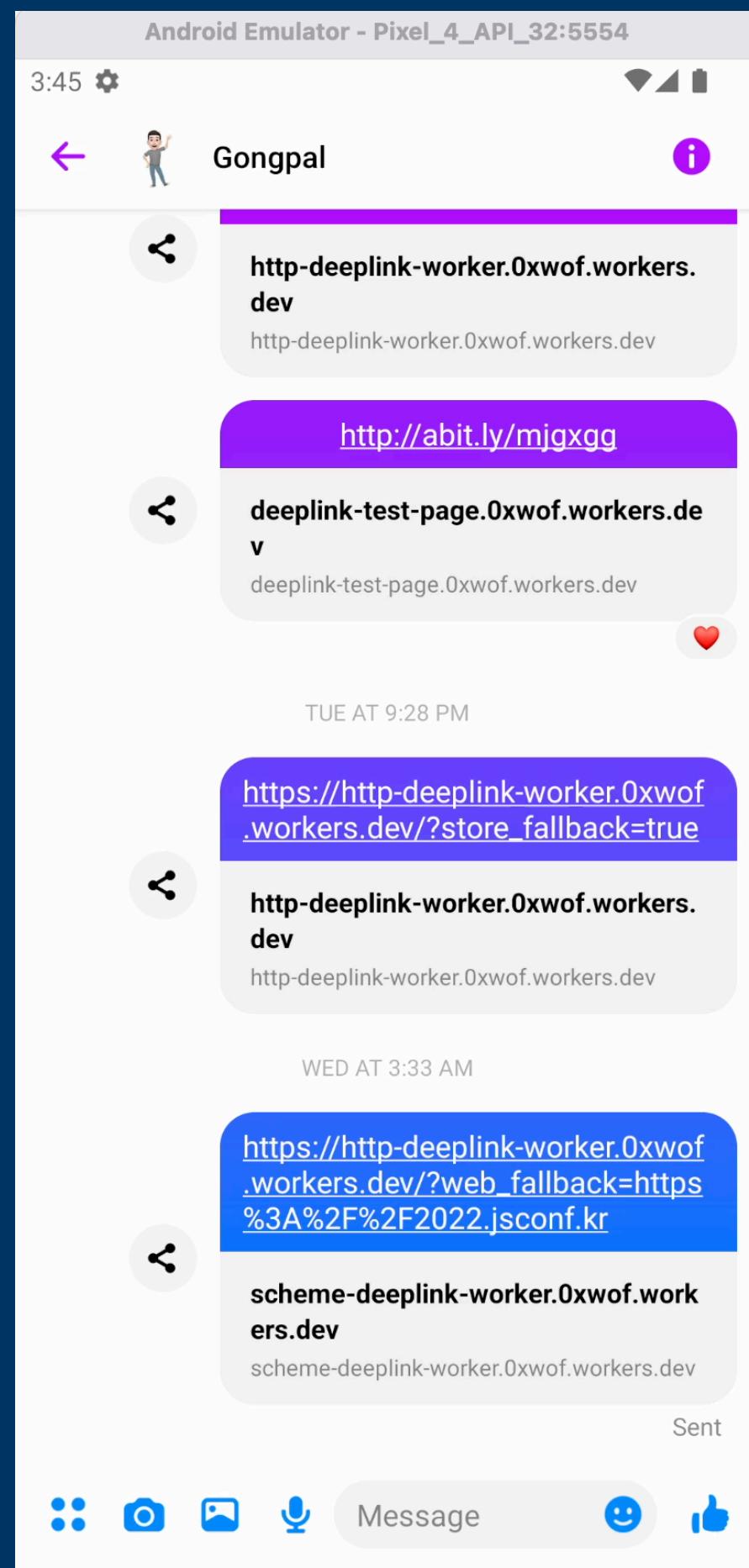
```
const renderWebFallback = (deeplink: string, fallback: string) => {
  const deeplinkURL = new URL(deeplink)

  return `
    <script>
      var a = document.createElement('a');
      a.href = `${deeplink.replace(/^[^:]*/\:/, 'intent:')
      + '#Intent;';
      + `scheme=${deeplinkURL.protocol};`;
      + `package=${APPLICATION.android.package};`;
      + `S.browser_fallback_url=${encodeURIComponent(fallback)};`;
      + `end;`;
      }`;
      a.click();
    </script>
  `
```

4. Scheme 딥링크 - 개선하기 - Android - 테스트



4. Scheme 딥링크 - 개선하기 - Android - 테스트



JSConf 2022 딥링크 솔루션 직접 만들어보기

4. Scheme 딥링크 - 개선하기 - Android

- 에러 케이스
 - Android Chrome 복사 붙여넣기 Store Fallback => 아무동작도 하지 않게 됨
 - Android Chrome 복사 붙여넣기 Web Fallback => 앱이 설치되어 있어도 Fallback 으로 이동함
- 원인 => Android Chrome 은 정책상 복사 붙여넣기로 URL 을 열면, 다음 유저 클릭까지 모든 딥링크 시도가 실패함
- 해결 => 딥링크 시도를 하기 이전에 HTML Alert 를 표시해서 유저 클릭을 강제하기
 - => 하지만 이것은 UX 를 희생하는 방법 => 타협이 필요함
 - => Store Fallback 은 앱이 설치되어 있을 때 스토어로 이동하는 것은 많이 어색함
 - => 딥링크를 시도하고, HTML Alert 를 그리는 방식으로 해결
 - => Web Fallback 은 앱이 설치되어 있을 때 웹으로 이동하는 것은 많이 어색하지 않음
 - => 비정상 동작을 감수

4. Scheme 딥링크 - 개선하기 - Android

- utility/user_agent.ts

```
const check_Chrome = (userAgent: string) => (
  /(Chrome|CrMo|CriOS)\//.test(userAgent)
)
```

4. Scheme 딥링크 - 개선하기 - Android

- service/android_chrome_store_fallback_service.ts

```
import { APPLICATION } from '../constant/application'

const createDependency = () => {}

createDependency.createAndroidChromeStoreFallbackService = () => ({
  APPLICATION,
})

const createAndroidChromeStoreFallbackService = (): AndroidChromeStoreFallbackService => {
  const { APPLICATION } = createDependency.createAndroidChromeStoreFallbackService()

  const renderStoreFallback = ...

  return {
    renderStoreFallback,
  }
}

type AndroidChromeStoreFallbackService = {
  renderStoreFallback: (deeplink: string) => string
}

export { createAndroidChromeStoreFallbackService }
export type { AndroidChromeStoreFallbackService }
```

4. Scheme 딥링크 - 개선하기 - Android

- service/chrome_store_fallback_service.ts

```
const renderStoreFallback = (deeplink: string) => `<meta name="viewport" content="width=device-width, initial-scale=1">
<script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js"></script>
<script>
    window.onload = function () {
        var deeplink = `${`intent://details` +
            `?id=${APPLICATION.android.package}` +
            `&url=${encodeURIComponent(deeplink)}` +
            `#Intent;` +
            `scheme=market;` +
            `package=com.android.vending;` +
            `end;`}`;
    };

    var a = document.createElement('a');
    a.href = deeplink;
    a.click();
    swal('Move to other page...').then(function () {
        var a = document.createElement('a');
        a.href = deeplink;
        a.click();
    });
};

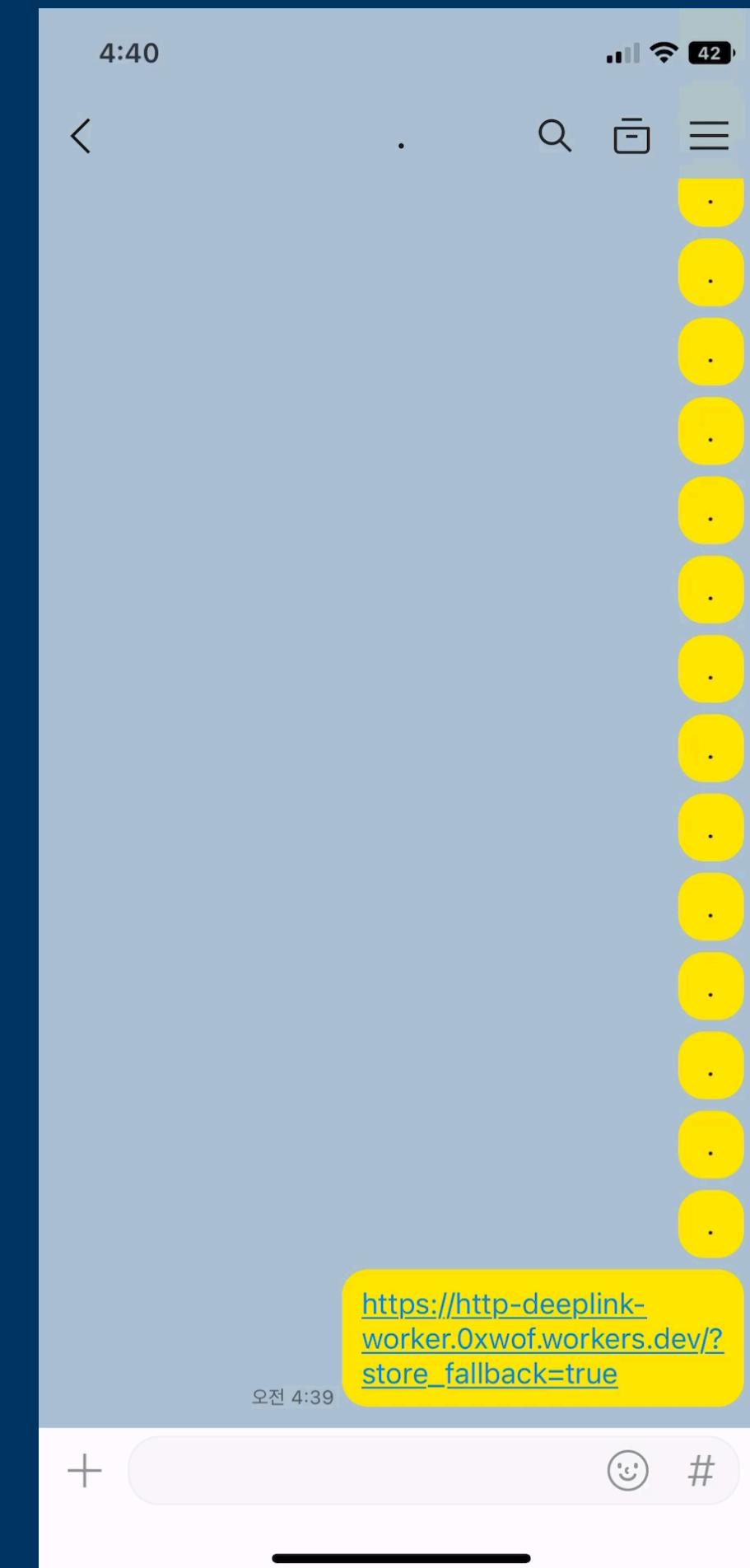
</script>`;
```

4. Scheme 딥링크 - 개선하기 - Android - 테스트



4. Scheme 딥링크 - iOS 예외상황

- iOS 카카오톡에서 앱이 설치되어 있을 때 Store Fallback 딥링크를 누르면 스토어가 열리고 Alert 창이 표시됨. Alert 창은 “취소”를 눌러도 “열기”를 눌러도 아무동작도 하지 않음. “열기”를 누르면 다음 부터는 정상동작하고, “취소”를 누르면 문제상황이 반복함
- iOS 는 앱 A 에서 앱 B 를 Custom URL Scheme URL 로 최초로 열면 앱을 열기전에 Alert 표시하여 사용자의 의사를 물어봄. 그런데 이 Alert 은 (스토어로 이동해서) 앱 A 가 아닌 앱에서 선택되면 앱 B 를 열지 않음. 이 Alert 은 한번이라도 “열기”가 선택되면 더 이상 표시되지 않음. (이는 앱 A 아닌 앱에서 선택되어도 유효함)
- 더 이상 표시되지 않는 것은 앱A-앱B 쌍에 대해서 유효하고, 앱 A-앱C 쌍에 대해서는 유효하지 않음. 유효한 쌍을 우회하지 않게 하기 위해서는 iOS 를 재설치해야함
- 해당 문제는 최초1회만 발생한다는 사실 때문에 해당 문제를 감수하는 경우도 존재함. 이 문제를 감수하고 싶지 않다면 constant/support.ts 의 ios 에 “kakaotalk: { scheme: false }” 를 추가하거나, “other: { scheme: false }” 를 추가하는 것으로 해결가능
 - kakaotalk 에 대해서만 설정하는 이유는, 앱에서 따라서 스토어로 이동하지 않고, 앱 내에서 스토어 View 를 표시하는 앱은 위 문제를 격지 않기 때문 (예시: iOS Chrome)



4. Scheme 딥링크 - 그리고 수 많은 예외상황들...

- android facebook
- ios chrome
- ios facebook messenger
- ios instagram
- ios kakaotalk
- ...

5. 정리

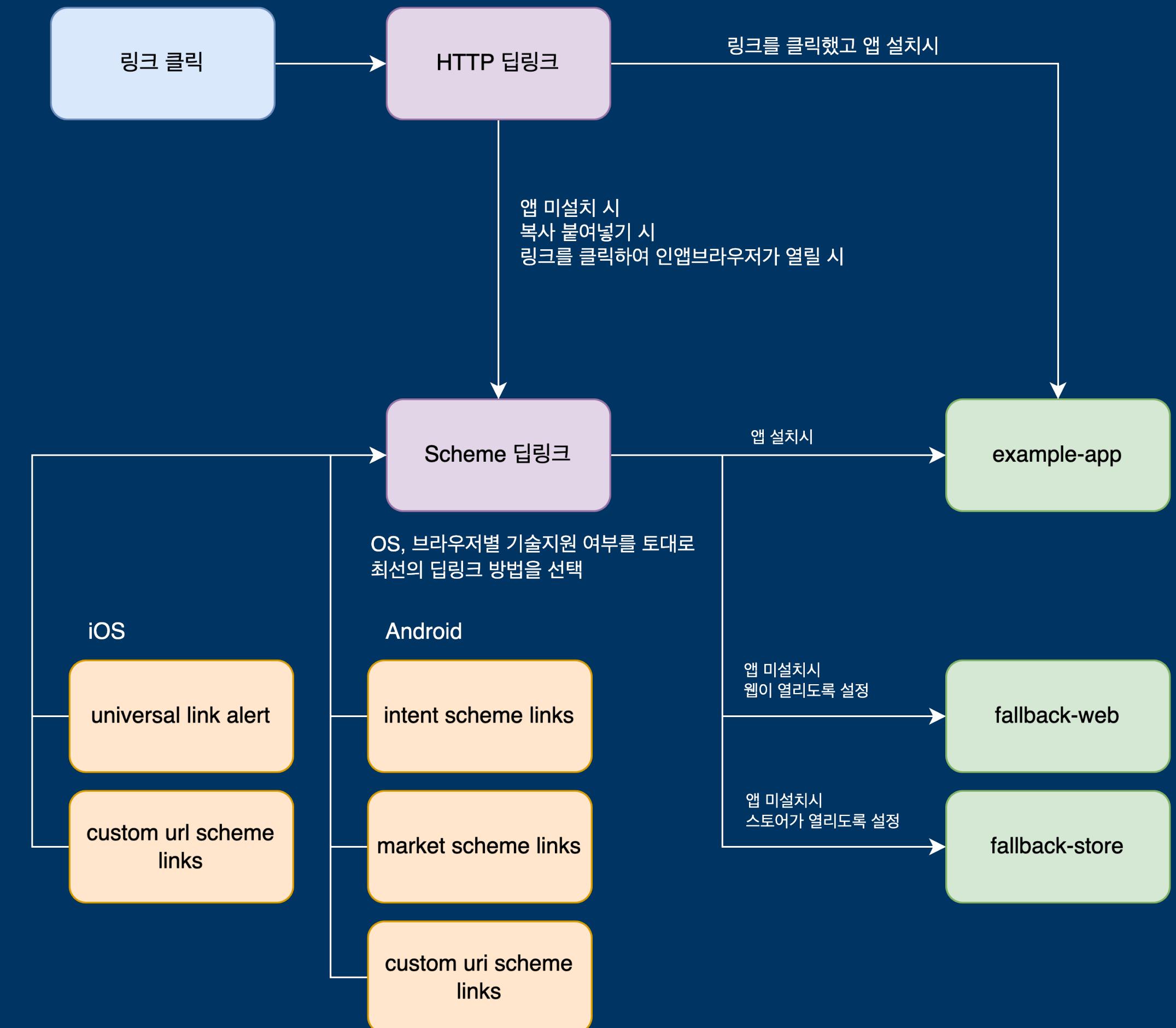
“딥링크는 사서 드시는걸 추천합니다.”

5. 정리

- HTTP 딥링크
 - 소유권이 확실하고, 중복될 수 없음
 - “유저 클릭”이 보장되어야 하기 때문에 동작하지 않는 경우가 많다.
 - 동작하지 않으면 Scheme 딥링크를 통해 보완한다.
- Scheme 딥링크
 - 소유권 기능등이 없어서 보안적으로 떨어진다.
 - “유저 클릭”을 요구하지 않아서 HTTP 딥링크로 앱을 열수 없는 경우에도 앱을 열 수 있다.
 - OS, 앱, 클릭방법에 따라 시나리오 많이 갈리며 수 많은 예외상황을 처리해야 한다.
 - 실시간으로 예외상황이 늘어난다.

5. 정리

- 링크가 클릭되면 먼저 HTTP 딥링크를 시도
- 실패시 Scheme 딥링크로 재시도
 - OS, 앱의 지원여부에 따라 최적화
 - 예외상황에 따라 별도의 방법 사용
- 결론적으로 앱이 설치되어 있으면 앱으로
- 아니면 Fallback 으로 이동시킴
 - Web Fallback or Store Fallback



5. 정리

- 딥링크 솔루션
 - 역할: HTTP 딥링크, Scheme 딥링크, 예외처리를 OS, 앱, 상황별로 최적의 조합으로 사용하여 가장 나은 UX 를 제공할 수 있도록 함
 - 필요성: 너무 많은 예외처리, 조합개수로 인해 큰 노력을 필요로 함.
 - 기술적 과제:
 - 각 OS, 앱별로 최적의 딥링크 방법을 찾고, 예외처리를 통해 딥링크 UX 를 향상시킨다.
 - 원활한 의사소통을 위해 어떤 조합에서 딥링크가 어떻게 동작하는지에 대해 쉽게 설명할 수 있는 시스템을 갖춘다. (example: <https://abit.ly/airbridge-deeplink>)

감사합니다!

subumm1@gmail.com
<https://github.com/0xWOF>

JSConf 2022 딥링크 솔루션 직접 만들어보기