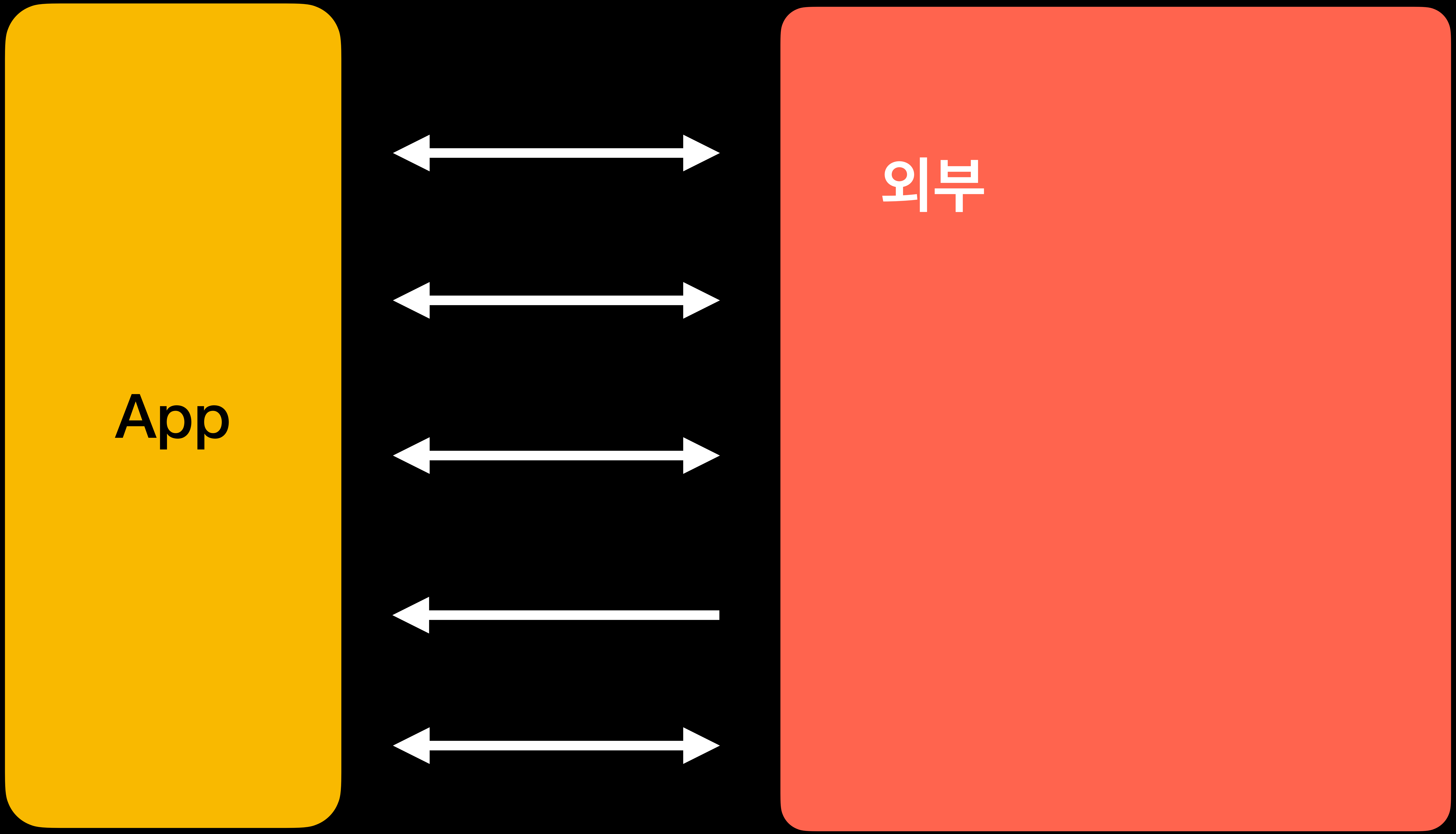
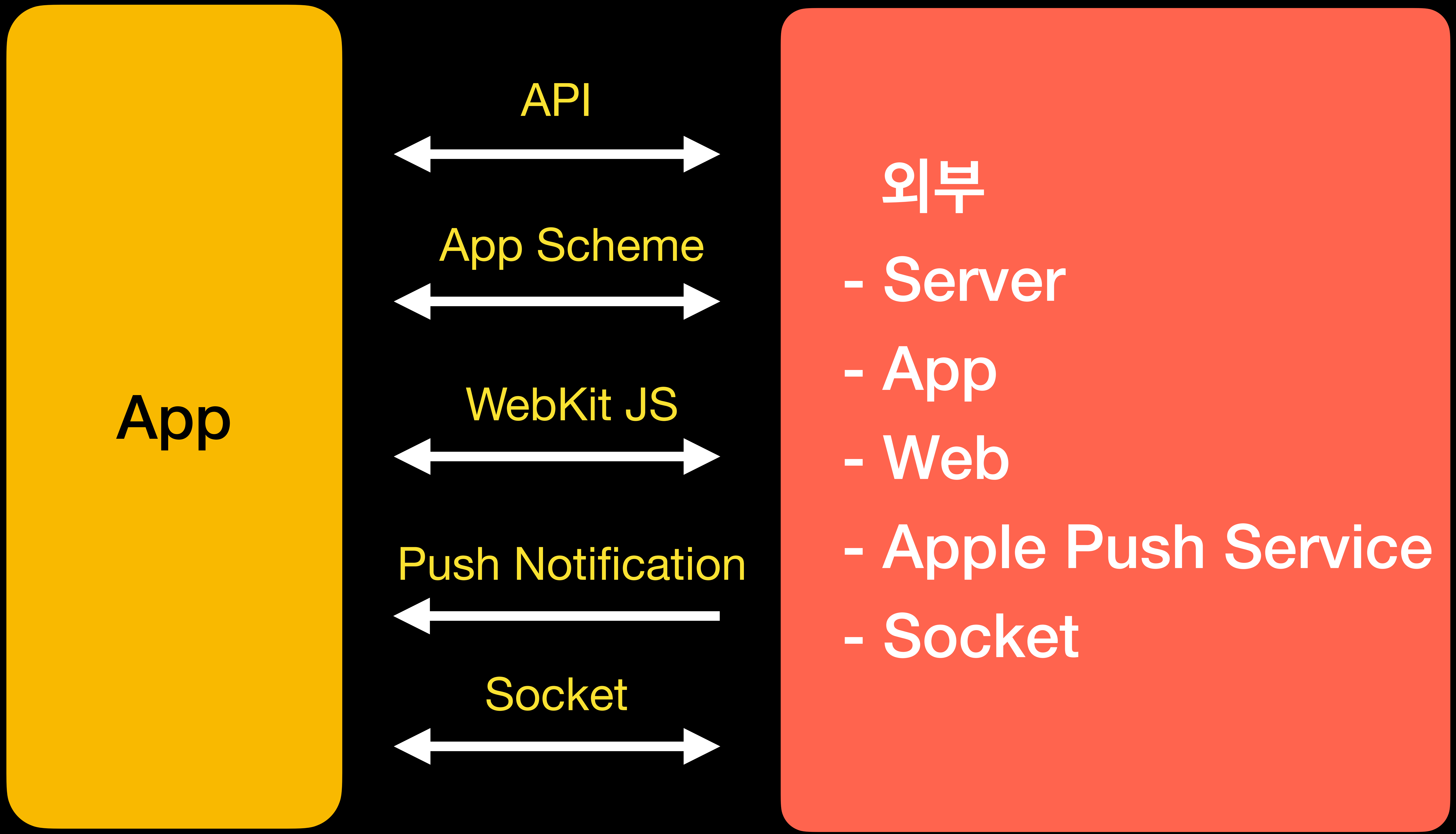
동적 데이터을 대응하는 코드 작성하기

Plugin 패턴을 활용하기

안정민


App
외부


App
API
App Scheme
WebKit JS
Push Notification
Socket
외부
- Server
- App
- Web
- Apple Push Service
- Socket

- 외부 서비스와 앱 간의 통신은 다양함
- 외부 서비스와 앱 간에는 약속한 데이터를 전달
  - 전달하고 받을 데이터의 구조화
    - API는 URL, Parameter, Response 등 송수신에 필요한 데이터 구조화 작업
    - App Scheme, Javascript Handler 등에서 전달받은 데이터의 처리는 구조화가 되어 있지 않음

```swift
// WKScriptMessageHandler 프로토콜 메서드 구현
func userContentController(_ userContentController: WKUserContentController, didReceive message: WKScriptMessage) {
    // 메시지의 이름과 body 추출
    guard
        message.name == "actionHandler",
        let messageBody = message.body as? [String: Any],
        let action = messageBody["action"] as? String
    else { return }

    // Action에 따라 처리하는 switch 문
    switch action {
    case "loading": loading(body: messageBody)
    case "openCard": openCard(body: messageBody)
    case "payment": payment(body: messageBody)
    case "log": log(body: messageBody)
    default: break
    }
}

// loading Action을 처리하는 함수
func loading(body: [String: Any]) {
    guard
        let value = body["show"] as? Bool
    else { return }

    switch value {
    case true: showLoading()
    case false: hideLoading()
    }
}

// payment Action을 처리하는 함수
func payment(body: [String: Any]) {
    guard
        let id = body["paymentId"] as? String,
        let info = body["paymentInfo"] as? [String: String]
    else { return }

    cardPayment(paymentId: id, paymentInfo: info)
}
```

```swift
// WKScriptMessageHandler 프로토콜 메서드 구현
func userContentController(
    _ userContentController: WKUserContentController,
    didReceive message: WKScriptMessage
) {
    // 메시지의 이름과 body 추출
    guard
        message.name == "actionHandler",
        let messageBody = message.body as? [String: Any],
        let action = messageBody["action"] as? String
    else { return }

    // Action에 따라 처리하는 switch 문
    switch action {
    case "loading": loading(body: messageBody)
    case "openCard": openCard(body: messageBody)
    case "payment": payment(body: messageBody)
    case "log": log(body: messageBody)
    default: break
    }
}
```

```swift
// loading Action을 처리하는 함수
func loading(body: [String: Any]) {
    guard
        let value = body["show"] as? Bool
    else { return }

    switch value {
    case true: showLoading()
    case false: hideLoading()
    }
}

// payment Action을 처리하는 함수
func payment(body: [String: Any]) {
    guard
        let id = body["paymentId"] as? String,
        let info = body["paymentInfo"] as? [String: String]
    else { return }

    cardPayment(paymentId: id, paymentInfo: info)
}
```

```swift
struct WKScriptMessageActionHandler {
    let closure: (_ body: [String: Any], _ webView: WKWebView?) -> Void
}


class WebViewManager {
    private let actionHandlers: [String: WKScriptMessageActionHandler]
    var webView: WKWebView?

    init(actionHandlers: [String: WKScriptMessageActionHandler] = [:]) {
        self.actionHandlers = actionHandlers
    }

    // WKScriptMessageHandler 프로토콜 메서드 구현
    func userContentController(
        _ userContentController: WKUserContentController,
        didReceive message: WKScriptMessage
    ) {
        // 메시지의 이름과 body 추출
        guard
            message.name == "actionHandler",
            let messageBody = message.body as? [String: Any],
            let action = messageBody["action"] as? String
        else { return }

        actionHandlers[action]?.closure(messageBody, webView)
    }
}
```

```swift
let loadingHandler = WKScriptMessageActionHandler {
    [weak self] body, webview in
    guard
        let self,
        let value = body["show"] as? Bool
    else { return }

    switch value {
    case true: showLoading()
    case false: hideLoading()
    }
}

let paymentHandler = WKScriptMessageActionHandler {
    [weak self] body, webview in
    guard
        let self,
        let id = body["paymentId"] as? String,
        let info = body["paymentInfo"] as? [String: String]
    else { return }

    cardPayment(paymentId: id, paymentInfo: info)
}

WebViewManager(actionHandlers: ["loading": loadingHandler,
                                "payment": paymentHandler])
```

- 모든 도메인과의 결합이 된 만능 WebView가 만들어지지 않도록 작업 필요
- 웹페이지에서 받은 ActionHandler를 직접 제어문을 통해 다루지 않아야함
  - WebView를 사용하는 서비스, 도메인에서 Action과 Handler를 주입
  - WebView는 WebView로서의 역할을 제한해야 함.

구조화

```swift
struct WKScriptMessageActionHandler {
    let closure: (_ body: [String: Any], _ webView: WKWebView?) -> Void
}


class WebViewManager {
    private let actionHandlers: [String: WKScriptMessageActionHandler]
    var webView: WKWebView?

    init(actionHandlers: [String: WKScriptMessageActionHandler] = [:]) {
        self.actionHandlers = actionHandlers
    }

    // WKScriptMessageHandler 프로토콜 메서드 구현
    func userContentController(
        _ userContentController: WKUserContentController,
        didReceive message: WKScriptMessage
    ) {
        // 메시지의 이름과 body 추출
        guard
            message.name == "actionHandler",
            let messageBody = message.body as? [String: Any],
            let action = messageBody["action"] as? String
        else { return }

        actionHandlers[action]?.closure(messageBody, webView)
    }
}
```

```swift
let loadingHandler = WKScriptMessageActionHandler {
    [weak self] body, webview in
    guard
        let self,
        let value = body["show"] as? Bool
    else { return }

    switch value {
    case true: showLoading()
    case false: hideLoading()
    }
}

let paymentHandler = WKScriptMessageActionHandler {
    [weak self] body, webview in
    guard
        let self,
        let id = body["paymentId"] as? String,
        let info = body["paymentInfo"] as? [String: String]
    else { return }

    cardPayment(paymentId: id, paymentInfo: info)
}

WebViewManager(actionHandlers: ["loading": loadingHandler,
                                "payment": paymentHandler])
```

```swift
struct WKScriptMessageActionHandler {
    let closure: (_ body: [String: Any], _ webView: WKWebView?) -> Void
}

class WebViewManager {
    private let actionHandlers: [String: WKScriptMessageActionHandler]
    var webView: WKWebView?

    init(actionHandlers: [String: WKScriptMessageActionHandler] = [:]) {
        self.actionHandlers = actionHandlers
    }

    // WKScriptMessageHandler 프로토콜 메서드 구현
    func userContentController(
        _ userContentController: WKUserContentController,
        didReceive message: WKScriptMessage
    ) {
        // 메시지의 이름과 body 추출
        guard
            message.name == "actionHandler",
            let messageBody = message.body as? [String: Any],
            let action = messageBody["action"] as? String
        else { return }

        actionHandlers[action]?.closure(messageBody, webView)
    }
}
```

```swift
let loadingHandler = WKScriptMessageActionHandler {
    [weak self] body, webview in
    guard
        let self,
        let value = body["show"] as? Bool
    else { return }

    switch value {
    case true: showLoading()
    case false: hideLoading()
    }
}

let paymentHandler = WKScriptMessageActionHandler {
    [weak self] body, webview in
    guard
        let self,
        let id = body["paymentId"] as? String,
        let info = body["paymentInfo"] as? [String: String]
    else { return }

    cardPayment(paymentId: id, paymentInfo: info)
}

WebViewManager(actionHandlers: ["loading": loadingHandler,
                                "payment": paymentHandler])
```

```swift
protocol JSInterfacePluggable {
    var action: String { get }
    func callAsAction(_ message: [String: Any], with: WKWebView)
}
```

```swift
protocol JSInterfacePluggable {
    var action: String { get }
    func callAsAction(_ message: [String: Any], with: WKWebView)
}


/// Supervisor class responsible for loading and managing JS plugins.
class JSInterfaceSupervisor {
    var loadedPlugins = [String: JSInterfacePluggable]()

    init() {}
}
```

```swift
extension JSInterfaceSupervisor {
    /// Loads a single plugin into the supervisor.
    func loadPlugin(_ plugin: JSInterfacePluggable) {

        let action = plugin.action
        guard loadedPlugins[action] == nil else {
            assertionFailure("\(action) action already exists. Please
check the plugin.")
            return
        }
        loadedPlugins[action] = plugin
    }
}

extension JSInterfaceSupervisor {
    /// Resolves an action and calls the corresponding plugin with a message and web view.
    func resolve(
        _ action: String,
        message: [String: Any],
        with webView: WKWebView) {

        guard
            let plugin = loadedPlugins[action], plugin.action == action
        else {
            assertionFailure("Failed to resolve \(action): Action is not
loaded. Please ensure the plugin is correctly loaded.")
            return
        }
        plugin.callAsAction(message, with: webView)
    }
}
```

```swift
protocol JSInterfacePluggable {
    var action: String { get }
    func callAsAction(_ message: [String: Any], with: WKWebView)
}


/// Supervisor class responsible for loading and managing JS plugins.
class JSInterfaceSupervisor {
    var loadedPlugins = [String: JSInterfacePluggable]()

    init() {}
}
```

```swift
private let supervisor = JSInterfaceSupervisor()

// WKScriptMessageHandler 프로토콜 메서드 구현
func userContentController(
    _ userContentController: WKUserContentController,
    didReceive message: WKScriptMessage
) {
    guard
        message.name == "actionHandler",
        let messageBody = message.body as? [String: Any],
        let action = messageBody["action"] as? String,
        let webView
    else { return }

    supervisor.resolve(action, message: messageBody, with: webView)
}
```

```swift
extension JSInterfaceSupervisor {
    /// Loads a single plugin into the supervisor.
    func loadPlugin(_ plugin: JSInterfacePluggable) {

        let action = plugin.action
        guard loadedPlugins[action] == nil else {
            assertionFailure("\(action) action already exists. Please
check the plugin.")
            return
        }
        loadedPlugins[action] = plugin
    }
}

extension JSInterfaceSupervisor {
    /// Resolves an action and calls the corresponding plugin with a message and web view.
    func resolve(
        _ action: String,
        message: [String: Any],
        with webView: WKWebView) {

        guard
            let plugin = loadedPlugins[action], plugin.action == action
        else {
            assertionFailure("Failed to resolve \(action): Action is not
loaded. Please ensure the plugin is correctly loaded.")
            return
        }
        plugin.callAsAction(message, with: webView)
    }
}
```

```swift
class LoadingJSPlugin: JSInterfacePluggable {
    let action = "loading"

    func callAsAction(
        _ message: [String: Any],
        with webView: WKWebView) {
        guard
            let result = Parser(message)
        else { return }

        closure?(result.info, webView)
    }

    func set(_ closure: @escaping (Info, WKWebView) -> Void) {
        self.closure = closure
    }

    private var closure: ((Info, WKWebView) -> Void)?
}
```

```swift
extension LoadingJSPlugin {
    struct Info {
        let uuid: String
        let isShow: Bool
    }
}

private extension LoadingJSPlugin {
    struct Parser {
        let info: Info

        init?(_ dictonary: [String: Any]) {
            guard
                let uuid = dictonary["uuid"] as? String,
                let body = dictonary["body"] as? [String: Any],
                let isShow = body["isShow"] as? Bool
            else { return nil }

            info = .init(uuid: uuid, isShow: isShow)
        }
    }
}
```

```swift
class PaymentJSPlugin: JSInterfacePluggable {
    let action = "payment"

    func callAsAction(
        _ message: [String: Any],
        with webView: WKWebView) {
        guard
            let result = Parser(message)
        else { return }

        closure?(result.info, webView)
    }

    func set(_ closure: @escaping (Info, WKWebView) -> Void) {
        self.closure = closure
    }

    private var closure: ((Info, WKWebView) -> Void)?
}

extension PaymentJSPlugin {
    struct Info {
        let uuid: String
        let paymentAmount: Int
        let paymentTransactionId: String
        let paymentId: String
        let paymentGoodsName: String
    }
}
```

```swift
private extension PaymentJSPlugin {
  struct Parser {
    let info: Info
      init?(_ dictonary: [String: Any]) {
        guard
            let uuid = dictonary["uuid"] as? String,
            let body = dictonary["body"] as? [String: Any],
            let paymentAmount = body["amount"] as? Int,
            let paymentTransactionId = body["transactionId"] as? String,
            let paymentId = body["paymentId"] as? String,
            let paymentGoodsName = body["paymentGoodsName"] as? String
        else { return nil }

        info = .init(
          uuid: uuid,
          paymentAmount: paymentAmount,
          paymentTransactionId: paymentTransactionId,
          paymentId: paymentId,
          paymentGoodsName: paymentGoodsName
        )
    }
  }
}
```

```swift
class ViewController: UIViewController {
    private let webViewManager = WebViewManager()

    override func viewDidLoad() {
        super.viewDidLoad()

        initPlugins()
    }

    private func initPlugins() {
        let loadingPlugin = LoadingJSPlugin()
        let paymentPlugin = PaymentJSPlugin()
        loadingPlugin.set { info, webView in
            print("LoadingPlugin :", info)
        }
        paymentPlugin.set { info, webView in
            print("PaymentJSPlugin :", info)
        }

        webViewManager.set(plugins: [loadingPlugin, paymentPlugin])
    }
}
```

- Action Handler를 분석하고, 직접 다루게 되면 만능 WebView가 탄생
- Plugin으로 Action을 구조화하며, WebView에서 등록된 Plugin을 호출하여 해당 Action의 책임을 가지지 않도록 함.
- 각 Plugin은 Action, 데이터 유효성 검증을 구현하여 Plugin을 검증할 수 있으며, Plugin이 호출되었을 때의 응답값을 예상할 수 있음
- Plugin 방식은 웹페이지의 Javascript, 앱 푸시, 챗의 데이터, 웹 소켓 등의 동적 데이터를 유연하게 대응할 수 있음

데모

QnA