

Smart-Commerce

제 1 장 서론

제 2 장 HomeViewController

2.1 addContentView

2.2 removeContentView

제 3 장 MenuButtonSectionView

3.1 MenuButtonSectionViewModel

제 1 장 서론

지난 프로젝트에서 Escaping Closure, Notification Center를 구현하면서 비동기 통신 과정을 명확하게 추론하는것에 한계를 느꼈다. 또한 UI 구현 코드와 데이터 통신 코드가 무분별하게 섞여 가독성을 저하시키는 문제점을 파악했다. 이를 보완하고자 RxSwift와 MVVM 디자인 패턴을 학습했으며, 본 프로젝트는 학습한 기술을 적용하여 문제점을 개선하는데 첫 번째 목적을 둔다.

또한 현재 애플리케이션 시장에서 전반적으로 사용되는 UI를 벤치마킹하여 고객 친화적인 인터페이스에대한 견문을 넓히는데 두 번째 목적을 둔다.

제 2 장 HomeViewController

HomeViewController에서 구현된 기능은 SubComponent인 MenuButtonSectionView에 등록된 Button들이 눌릴 때마다 버튼의 id 값에 해당되는 ViewController를 HomeViewController 화면에 띄우는 것이다.

이때 HomeViewController에 ViewController를 Embed 하는 작업이 필요하며 이를 `addContentView` 함수에 정의하였다.

2.1 addContentView

```
private func addContentView(_ index: Int) {  
    let contentView = contentViewList[index]  
  
    addChild(contentView)  
    contentView.view.frame = containerView.frame  
    containerView.addSubview(contentView.view)  
    contentView.view.snp.makeConstraints {  
        $0.edges.equalToSuperview()  
    }  
    contentView.didMove(toParent: self)  
}
```

`addContentView` 함수에서 Parameter로 받은 `index`를 사용하여 미리 저장해둔 `contentViewList`에서 해당 ViewController를 가져온다.

가져온 ViewController를 HomeViewController.`addChild` 메소드를 통해 부모-자식 관계를 설정한다.

여기서 `addChild` 메소드를 사용하는 이유는 ViewController들은 `children` 배열 프로퍼티를 갖고 있는데, 이 프로퍼티는 UINavigationController처럼 child를 관리하기 쉽도록 내부적으로 제공하는 프로퍼티이다.

따라서 `addChild`를 통해 HomeViewController의 `children` 프로퍼티에 `contentView`를 넣기 위해 사용되는 것이다.

위 잡업을 해야 `addSubview`를 통해 containerView에 `contentView.view`를 추가해서 View를 띄울 수 있다.

또한 `contentView` 입장에서는 언제 HomeViewController에 추가될지 모르기 때문에 `didMove(toParent:)` 메소드를 통해 추가 및 삭제되는 시점을 알려주도록 한다.

이때 명시적으로 `didMove`를 호출하는 이유는 `addChild` 메소드에서 내부적으로 `willMove`만 호출하기 때문이며, 추후에 나올 `removeContentView`에서 `willMove`만 호출하는 이유는 `removeFromParent`에서 `didMove`만 호출되기 때문이다.

위 작업을 마치면 MenuButtonSectionView에 등록된 버튼이 눌릴 때마다 원하는 View가 HomeViewController 위에 띄워진다. 하지만 여기서 문제점은 사용자는 버튼을 무수하게 누를 수 있고 그때마다 새로운 contentView의 인스턴스들이 호출돼서 HomeViewController 위에 뜨게 된다. 화면상으로는 문제가 없어 보이지만 이는 심각한 메모리 누수를 발생시킨다. 이 문제의 해결 방안은 `removeContentView` 함수에 정의하였다.

2.2 removeContentView

```
private func removeContentView(_ index: Int) {  
    let contentView = contentViewList[index]  
  
    contentView.willMove(toParent: nil)  
    contentView.removeFromParent()  
    contentView.view.removeFromSuperview()  
}
```

`removeContentView` 함수에는 `addContentView`에서와 마찬가지로 contentView에게 HomeViewController로부터 삭제되기 전에 `willMove(toParent:)` 메소드를 통해 알려준다.

그 후 `removeFromParent()` 메소드를 통해 HomeViewController와의 부모-자식 관계를 삭제하기 위해 `removeFromParent` 메소드를 사용하여 앞서 언급한 HomeViewController의 `children` 배열 프로퍼티에서 해당 ViewController를 제거하는 작업을 갖는다.

제 3 장 MenuButtonSectionView

MenuButtonSectionView에서 주요 기능은 등록된 버튼을 탭 했을때 탭된 버튼의 id 값을 HomeViewController에 넘기는 것이다.

3.1 bind

MenuButtonSectionView.bind에서는 MenuButtonSectionViewModel에서 .just로 한 번에 넘긴 menuDataList 배열을 onNext로 받아 각각이데의 id 값과 일치하는 menuButtonList 인덱스와 일치하는 버튼들을 세팅해주고 있다.

