Smart-Commerce

제 1 장 서 론

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 인터페이스에 대한 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 index를 사용하여 미리 저장해둔 contentViewList에서 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 NavigationController처럼 child를 관리하기 쉽도록 내부적으로 제공하는 프로퍼티이다.

따라서 addChild를 통해 HomeViewController의 children 프로퍼티에 contentView를 넣기 위해 사용되는 것이다.

위 잡업을 해야 addSubView를 통해 containerView에 contentView.view를 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 didMove를 호출하는 이유는 addChild 메소드에서 내부적으로 willMove만 호출하기 때문이며, 추후에 나올 removeContenView에서 willMove만 호출하는 이유는 removeFromParnet에서 didMove만 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. 화면상으로

는 문제가 없어 보이지만 이는 심각한 메모리 누수를 발생시키다. 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 removeFromParent메소드를 사용하여 앞서 언급한 HomeViewController의 chilren 배열 프로퍼티에서 해당 ViewController를 제거하는 작업을 갖는다.

2.3 HomeViewModel

```
init() {
    let selectedMenuIndex = menuButtonSectionViewModel.selectedMenu
        .map { $0.id }
        .startWith(0)

    self.selectedMenu = selectedMenuIndex
        .asDriver(onErrorJustReturn: 0)
}
```

HomeViewModel 에서는 MenuButtonSectionVIewModel에서 선택된 버튼에대한 정보(MenuData)를 감싼 Evnet를 받는다.

이때 View 에서는 해당 버튼에대한 id 값만 필요하므로 map Operator를 사용해서 필터링 했다. 또한 화면이 처음 나타날 때 가장 첫 번째 화면(추천) 나오길 원하므로 startWith(0)으로 설정해서 selectedMenuIndex Observable로 선언했다.

id를 방출하는 selectedMenuIndex Observable은 HomeViewController에서 Event를 방출할 때마다 addContentView, removeContentView를 통해 UI가 변경된다. 위 작업은 Main Thread에서 실행되는 것을 보장해야하므로 asDrive를 설정하여 selectedMenu로 선언했다.

제 3 장 MenuButtonSectionView

MenuButtonSectionView 에서는 버튼 등록(recommend, ranking, style, salse, evnet), 등록된 버튼이 눌릴때마다 해당 버튼이 선택 됐음을 알리는 Text 변화, 눌린 버튼의 정보(MenuData)를 MenuButtonSectionView 에 넘기는 세 가지 기능을 구현했다.

3.1 setButton

```
viewModel.menuDataList
    .drive(onNext: { [weak self] menuDataList in
        menuDataList.forEach { self?.menuButtonList[$0.id].setButton($0) }
    }).disposed(by: disposeBag)
```

MenuButtonSectionViewModel.menuDataList 에서 방출되는 배열형태의 Evnet를 Subscribe 해서 방출된 Event의 배열을 forEach문을 통해 미리 설정해둔 muneButtonList에 담긴 UIButton들과 매칭시켜 setButton() 함수를 통해 버튼을 등록했다.

3.2 menuButtonTapped

```
menuButtonList.forEach { button in
    button.rx.tap
        .map { button.tag }
        .bind(to: viewModel.menuButtonTapped)
        .disposed(by: disposeBag)
}
```

각각의 버튼들의 눌렸을 때 Event를 방출 해서 MenuButtonSectionVIewModel.menuButtonTa 에 전달할 수 있도록 menuButtonList 각가의 요소에 UIButton.rx.tap 을 설정했다. 이때 필요한 정보는 각각의 버튼의 tag이므로 map Operator 를 통해 필터링했다.

3.2 isSelect

```
viewModel.selectedMenuIndex
    .drive(onNext: { [weak self] in
        self?.isSelect($0)
    }).disposed(by: disposeBag)
```

MenuButtonSectionViewModel.selectedMenuI 에서 방출되는 Evnet를 Subscribe해서 전달되는 index 값을 isSelect 함수의 argument로 넘겨준다.

```
private func isSelect(_ index: Int) {
    menuButtonList.forEach {
        if $0.tag == index {
            $0.setTitleColor(.black, for: .normal)
            $0.titleLabel?.font = .systemFont(ofSize: 15.0, weight: .bold)
        } else {
            $0.setTitleColor(.gray, for: .normal)
            $0.titleLabel?.font = .systemFont(ofSize: 15.0, weight: .light)
        }
    }
}
```

isSelcte(index:)는 parameter를 통해 menuButtonList의 index에 해당하는 MenuButton을 활성화 시키고 나머지는 비활성화 시킨다.