모바일 앱프로그래밍 2

기술문서

Team2

목차

1. 메인, 데이터셋– 성진용 학우
2. 별자리 운세 – 박형석 학우
3. 오늘의 운세 – 신재원 학우
4. 관상, 셋팅, 탭바 - 조윤상 학우

메인 – 성진용 학우

```
    static var db = DBHelper()
    static var unse = DataSet()
    static var mUser : User? = nil
```

메인이 시작되기 전, View 는 모든 데이터를 받는다.

사용자 데이터가 저장된 데이터 베이스

운세 데이터가 저장된 데이터 저장소

그리고 사용자 본인의 데이터인 유저 데이터가 존재한다.

```
class User {
    var birthYear : Int
    var birthMonth : Int
    var birthDay : Int
    var name : String
    var id : Int

    init(id: Int, name : String, year:Int, month:Int, day:Int)
    {
        birthDay = day
        birthMonth = month
        birthYear = year
        self.name = name
        self.id = id
    }
}
```

유저 데이터는 다음과 같다.

Id 는 데이터베이스를 1 개보다 많이 유지하지 않도록 하기 위함과 동시에, 설정 초기화 시 용이하게 만들기 위해 첨가되었다.

```
import SQLite3

class DBHelper
{
    init()
    {
        db = openDatabase()
        createTable()
    }

    let dbPath: String = "db.sqlite"
    var db:OpaquePointer?
```

데이터베이스 저장소 클래스.

```
func insert(id:Int = 1, name:String, year:Int, month:Int, day:Int)
{
    if read() != nil {
        print("Already insert User Data")
        return
    }

    let insertStatementString = "INSERT INTO user (Id, name, year, month, day) VALUES (?, ?, ?, ?, ?);"
    var insertStatement: OpaquePointer? = nil
```

데이터를 추가할 때 읽어지는 데이터가 존재하면 바로 취소시켜버린다.

```
func read() -> User? {
    let queryStatementString = "SELECT * FROM user;"
    var queryStatement: OpaquePointer? = nil
    var user : User? = nil
```

필요한 데이터는 1 개이므로, 리턴타입을 1 개로 주고, 데이터를 읽는다. 다만 데이터가 없을 수도 있기 때문에 옵셔널로 주었다.

```
func deleteByID(id:Int = 1) {
    let deleteStatementStirng = "DELETE FROM user WHERE Id = ?;"
    var deleteStatement: OpaquePointer? = nil
```

데이터 삭제. Id 를 기준으로 데이터를 삭제한다.

```
override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)
    if ViewController.mUser == nil {
        loadUserData()
    }
}
```

이러한 데이터들을 이용하여 View 를 사용자에게 보여줄 때, 유저 데이터를 확인하여 데이터가 없을 경우에 데이터를 받는다.

```
func loadUserData(){
    ViewController.mUser = ViewController.db.read()
    if ViewController.mUser == nil {
        let vc = storyboard?.instantiateViewController(withIdentifier: "createID") as! CreateIdViewController
        vc.modalPresentationStyle = .fullScreen
        self.present(vc, animated: true)

    }
}
```

데이터베이스로부터 유저 데이터를 Load 했을 때, 데이터베이스에 유저 데이터가 존재하지 않는다면, ID(unique user)를 만들도록 View 를 Change 한다.

버튼 이동은 탭바 컨트롤러의 영역 하에 둘 수 있도록 조절했다.

```
@objc func clickButton(_ sender: Any) {
    let id = (sender as? UITapGestureRecognizer)?.view?.restorationIdentifier
    if id == "별자리 운세"
    {
        tabBarController?.selectedIndex = 1
    }else if id == "오늘의 운세"
    {
        tabBarController?.selectedIndex = 2
    }else if id == "관상" {
        tabBarController?.selectedIndex = 3
    }else{
        tabBarController?.selectedIndex = 4
    }
}
```

이미지 뿐만 아니라, 글자 영역에서도 클릭할 수 있도록, 두가지를 모두 담은 스택뷰 자체에 제스쳐를 넣었고, 제스쳐 인식 시에 id 를 통해 탭을 통한 이동으로 유지할 수 있도록 했다.

별자리 운세 – 박형석 학우

[AstralUnseViewController.swift] 기술구현

AstralUnseViewController.swift 는 오늘의 별자리 별 운세를 알려주는 기능을 구현한 것이다. 테이블 뷰를 통해 화면에 별자리 이미지와 운세를 나타내도록하였다. 우선 수집한 별자리, 별자리 이미지, 운세 내용을 변수를 배열로 생성하여 데이터를 임의로 생성하였다. 데이터 부분은 다음 그림과 같이 DataSet.swift에 따로 구성하였다.

```
var astralString:[String] = ["가보고 싶다고 체크해두었던 와인 바가 있다면 오늘 같은 날 가라. 그동안
    자주 가던 분식집 혹은 음식점보다는 분위기 있는 곳에서 사랑하는 사람과 조촐한 저녁식사를 하는 게
    좋을 듯. 장소의 분위기에 취해 연인사이의 관계가 한 단계 더 발전할 수도 있겠다.", "오늘 당신의
    스케줄은? 비었나? 오늘은 변신의 날로 삼자. 그 오래된 헤어스타일부터 청산해 버리자. 유행을 따를
    거라면 처음부터 끝까지 확실하게 패션리더의 모습으로, 그게 아니라면 좀 평범하게.", "당신 연인의 말
    들어서 손해 날 것 하나도 없는 날이다. 오늘은 당신 연인의 꼭두각시가 되어보자, 하라는 대로 다하고
    연인의 물음에 원하는 대답해주고 당신의 연인도 즐겁겠지만 나름 당신에게도 재미있을 걸?!", "생각
    없이 속 얘기가 술술 나오는 날이니 오늘은 당신의 입에 촘촘한 그물망을 설치해야 하는 날이다.
    무턱대고 믿고 털어놓은 속 얘기가 소문이 날 수 있겠으니 아무에게나 속얘기를 털어놓지 말자. 상처받을
    수 있다.", "당신의 선택에 따라 오늘 하루가 달라질 것이다. 선택이 중요한 날이니, 우물쭈물 하다
    기회를 놓치지 말고 확실한 선택을 하여 행운을 잡아라! 앞뒤 상황 재고 결정하는 것 보다 순간적인 느낌,
```

```
    var astralName:[String] =
        ["양자리","황소자리","쌍둥이자리","게자리","사자자리","처녀자리","천칭자리","전갈자리","사수자리"
        ,"염소자리","물병자리","물고기자리"]

    var astralImage:[String] = ["aries.jpg", "taurus.jpg", "gemini.jpg", "cancer.jpg", "leo.jpg",
        "virgo.jpg", "libra.jpg", "scorpio.jpg", "sagittarius.jpg", "capricorn.jpg", "aquarius.jpg",
        "pisces.jpg"]

var astralUnse = [[String:String]]()
var todayUnse = [[String:String]]()
```

이어서 화면을 구성하기 위해 테이블뷰를 사용하고 메소드를 생성한다.

```swift
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return ViewController.unse.astralUnse.count
}

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
    UITableViewCell {
    let cell = tableView.dequeueReusableCell(
        withIdentifier: "Cell",
        for: indexPath
    )

    let currentRowOfList = ViewController.unse.astralUnse[indexPath.row]

    cell.textLabel?.text = currentRowOfList["astralName"]
    cell.detailTextLabel?.text = currentRowOfList["astralString"]
    cell.imageView?.image = UIImage(named: currentRowOfList["astralImage"]!)

    cell.selectionStyle = .none

    return cell

}
```

func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int 메소드는 데이터의 row 갯수를 반환해주는 메소드이다. 여기서는 별자리의 갯수를 반환해 주도록 구현하였다. 별자리는 12 가지이고 12 개의 데이터로 구성되었기에 12 를 반환하게된다. func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell. 메소드는 구체적인 하나의 row 를 나타내는 table view cell 을 반환하는 메소드이다. cell 안에 데이터를 직접 뿌려주는 일을 수행한다.

```swift
let calendar = Calendar.current
let date = Date()
let currentYear = calendar.component(.year, from: date)
let currentMonth = calendar.component(.month, from: date)
let currentDay = calendar.component(.day, from: date)

for index in 0...astralName.count-1 {
    astralUnse.append( [ "astralName":astralName[index], "astralString":astralString[(index +
        currentDay + (ViewController.mUser!.birthDay)) % astralString.count],
        "astralImage":astralImage[index] ] )
}
```

별자리 이미지는 조건문으로 cell 의 이름이 별자리와 같을 경우 선택된 이미지를 표시해 주도록 구현하였다. 별자리를 표시 할 경우 운세 내용이 날마다 달라져야하기 때문에 위와 같이 운세를 현재 날짜와 개인 생년월일을 시드값으로 모드연산 랜덤 공식을 통해 랜덤으로 append 시킨다.

이후 컨테이너 뷰를 생성한 후 myUnseViewController.swift 부분이 버튼을 클릭할 경우 나올 수 있도록 구현하였다.

우선 별자리 별 시작 날짜와 끝나는 날짜를 입력받는 Constellation 클래스가 포함된 변수를 다음과 같이 구현하였다.

```
var constellations = [Constellation(startDate: "3/21", stopDate: "4/19"),
    Constellation(startDate: "04/20", stopDate: "05/20"),
    Constellation(startDate: "05/21", stopDate: "06/20"),
    Constellation(startDate: "06/21", stopDate: "07/22"),
    Constellation(startDate: "07/23", stopDate: "08/22"),
    Constellation(startDate: "08/23", stopDate: "09/22"),
    Constellation(startDate: "09/23", stopDate: "10/22"),
    Constellation(startDate: "10/23", stopDate: "11/21"),
    Constellation(startDate: "11/22", stopDate: "12/21"),
    Constellation(startDate: "12/22", stopDate: "01/19"),
    Constellation(startDate: "01/20", stopDate: "02/18"),
  Constellation(startDate: "02/19", stopDate: "03/20")]
```

Constellation 클래스에서는 다음과 같이 별자리의 시작 날짜와 끝나는 날짜를 init을 통해

```
class Constellation {
    var startDate: String
    var stopDate: String

    init(startDate: String, stopDate: String) {

        self.startDate = startDate
        self.stopDate = stopDate
    }

    func checkInterval(dateComponent: DateComponents) -> Bool {
        if let date = dateComponent.date, let year = dateComponent.year {

            let start = "\(year)/\(self.startDate)"
            let end = "\(year)/\(self.stopDate)"

            let dateFormatter = DateFormatter()
            dateFormatter.dateFormat = "yy/MM/dd"

            if startDate > stopDate {
                if let start1 = dateFormatter.date(from: start), let end1 = dateFormatter.date(from: "\(year)/12/31"), let start2 =
                    dateFormatter.date(from: "\(year)/01/01"), let end2 = dateFormatter.date(from: end){
                    return DateInterval(start: start1, end: end1).contains(date) || DateInterval(start: start2, end: end2).contains(date)
                }
            } else {
                if let start = dateFormatter.date(from: start), let end = dateFormatter.date(from: end) {
                    return DateInterval(start: start, end: end).contains(date)
                }
            }
        }
        return false
    }
}
```

설정해 준다.

이후 checkInterval 메소드에서 앞서 입력된 별자리의 시작날짜와 끝나는 날짜를 날짜 데이터로 바꾼다. 조건문을 통해 입력받은 시작날짜와 끝나는 날짜의 유효성을 확인하고 유효하다면 생년월일이 어떤 별자리에 해당하는지 DateInterval()을 통해 확인하고 생년월일이 포함된 별자리에 맞는지 체크하도록 한다.

그 뒤 ViewDidLoad()에서 반복문을 통해 별자리의 개수만큼 checkInterval()을 통해 해당하는 별자리일 경우 그 별자리의 이미지와 운세를 화면에 나타날 수 있게 구현하였다. 운세 내용의 경우는 앞선AstralUnseViewController.swift 의 구현과 동일하다.

오늘의 운세 – 신재원 학우

1. 사용자 정보 입력 시 키보드 이슈

```
    /*키보드 외부를 터치하면 키보드가 내려가도록*/
    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?){
                self.view.endEditing(true)
          }
```

- 키보드가 아닌 다른 화면 부분을 터치하는 이벤트가 발생하면 입력을 종료하도록 설정

```
/*키보드의 리턴키를 (엔터) 입력하면 키보드가 내려가도록*/
extension CreateIdViewController {
    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        textField.resignFirstResponder()
        return true
    }
}
```

- 키보드로 입력 중에 리턴 키가 입력되면 입력을 종료하도록 설정

-

2. 오늘의 운세에서 오늘의 날짜도 함께 알려주기 위한 라벨 구현

```
/*라벨에 오늘 날짜를 띄욱 위한 변수*/
let now=NSDate()
let dateFormatter = DateFormatter()
dateFormatter.dateFormat = "yyyy년 MM월 dd일의 운세"
TodayString.text=dateFormatter.string(from: now as Date)
/*오늘의 날짜르 보여주는 라벨의 굴곡 설정*/
TodayString.clipsToBounds = true
self.TodayString.layer.cornerRadius = 15.0
```

- NSDate()로 시스템상 날짜를 스트링으로 불러와 라벨과 연결 후 코너 굴곡을 주어 디자인 처리

3. 오늘의 운세를 정하는 원리

```
class DataSet {
        var zodiacString:[String] = ["드물게 운이 좋은 하루입니다. 하는 일도 순조롭고 모처럼 풍족한 여유도
            누리게 됩니다. 당신의 주위에 사람들이 많이 모여드는 하루가 되겠습니다.","좋은 사람을 얻게 될 운수입니다.
            두 사람 모두 삶의 또 다른 출발선에서 마주치는 상황일 것입니다. 지금 당장은 서로에게 큰 힘이 되어 주지
            못하겠지요.","남달리 큰 포부를 갖게 되는 때입니다. 그러나 기회는 준비 없이는 오지 않는 법이지요. 뜻을
            이루기 위해서는 더 세심한 준비와 무르익은 시기를 기다릴 필요가 있습니다.","곤란을 겪는 가운데서도 희망의
            빛을 발견할 수 있는 운입니다. 당신의 뜻하는 바의 일을 추진하다 보면 다소 어려움에 부딪히기도 하는 하루가
            될 것입니다.","다소 곤란한 상황이 예상되는 하루입니다. 추진하는 일에 들었던 행운의 여신이 잠시 모습을
            감추었습니다. 때로 당혹스럽고 견디기 힘들겠군요.","당신의 뜻을 마음껏 펼쳐 다양한 시도를 해볼 수 있는 드문
            기회가 될 것입니다. 이제까지 준비해 왔던 일이나 계획한 바가 있다면 이 기회에 과감히 도전해
            보십시오.","시련은 혼자 오지 않는다고 합니다. 유난히 엎친 데 덮친 격으로 어려움이 끊이질 않습니다. 가도
            가도 끝이 없는 망망대해를 건너는 심정일 수도 있습니다.","평소에 다른 사람에 대한 세심한 배려로 인해 많은
            사람으로부터 호감을 받고 있습니다. 그것이 오늘, 당신에게 큰 도움이 되겠군요.","인생은 '공수래공수거'라는
            말이 있습니다. 손해를 볼 수도, 때로는 이득을 얻을 수도 있겠지요. 얻은 만큼 잃게 된다는 말도
            있습니다.","뜻하지 않게 어려움에 처하는 상황이 예상되는 하루입니다. 미리 준비가 되어 있는 상태가 아니므로
            순발력 있게 적절한 대처 방안을 마련하기가 쉽지 않을 것입니다.","눈앞의 안개가 시야를 가리는 형국이지만
            어쩔 수 없이 밀고 나아가야 할 처지입니다. 자잘한 근심으로 분주한 하루가 예상됩니다.","많은 사람들의 당신에
            대한 기대가 큽니다. 그러나 부담스러움에 지레 주눅 들지 말고 대담하게 앞장서서 일을 추진하십시오.","매우
            안온한 하루가 예상됩니다. 아직 결정적인 시기를 만나지는 못했을 뿐, 달빛 아래 펼쳐진 풍요로운 들판처럼
            잠재된 당신의 재능은 무한합니다.","큰 산의 정상에 떠오른 태양이 그 위세를 떨치는 형상입니다. 보기 드문
```

```
        let calendar = Calendar.current
        let date = Date()
        let currentYear = calendar.component(.year, from: date)
        let currentMonth = calendar.component(.month, from: date)
        let currentDay = calendar.component(.day, from: date)
```

```
        var pseudoRandomNumber = (currentYear + currentMonth + currentDay +
            ViewController.mUser!.birthYear + ViewController.mUser!.birthMonth +
            ViewController.mUser!.birthDay)
        /*운세를 날짜와 개인 생년월일을 시드값으로 모드연산 수도랜덤 공식을 통해 랜덤으로 append*/

        for index in 0...zodiacName.count - 1{
            pseudoRandomNumber = (pseudoRandomNumber + index) % zodiacName.count

            todayUnse.append( ["zodiacString":zodiacString[pseudoRandomNumber
                ],"zodiacName":zodiacName[index],"zodiacImage":zodiacImage[index]])
        }
```

- 데이터 베이스에 저장된 사용자의 생년월일과 시스템 상의 오늘 날짜를 시드 값으로 의사 난수로 데이터 베이스 상의 운세를 뽑아온다. 운세 내용은 사용자의 생년월일과 오늘의 날짜에 맞추어 랜덤으로 계산되기에 사용자의 정보가 달라지거나 하루가 지나는 등의 값의 변화가 있어야 달라지도록 구현하였다. 가져온 운세 내용을 12간지의 각 띠에 맞추어 할당하여 같이 저장되도록 하였다.

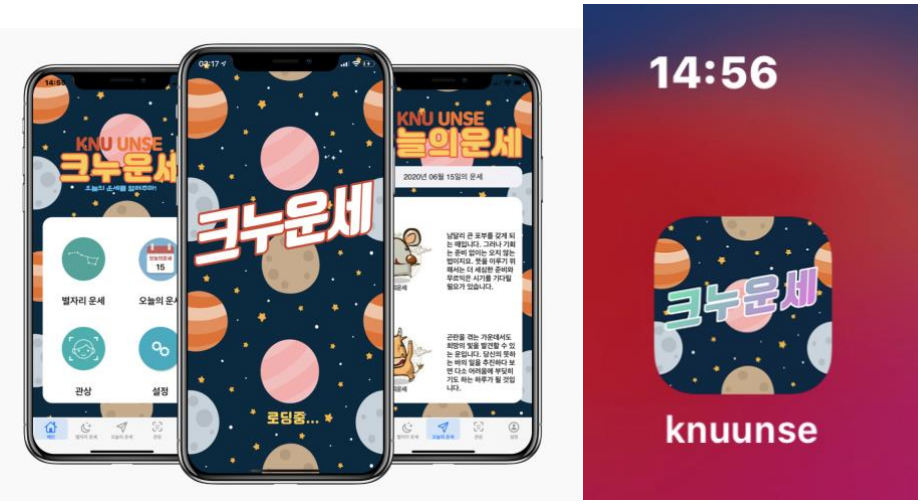
4. 앞에서 정해진 운세를 테이블 뷰로 구현

```
func tableView(_ tableView: UITableView,cellForRowAt indexPath: IndexPath) ->
    UITableViewCell {

        let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for:
            indexPath) as! UnseCellTableViewCell
           /*셀을 추가*/
        let unseCell = ViewController.unse.todayUnse[indexPath.row]

        cell.UnseString.text = unseCell["zodiacString"]
        cell.ZodiacName.text = unseCell["zodiacName"]
        cell.ZodiacImageView.image = UIImage(named: unseCell["zodiacImage"]!)
        cell.selectionStyle = .none

        return cell
}
```

```
    self.unseTableView.layer.cornerRadius = 20.0
    ContentView.layer.cornerRadius = 20.0
    unseTableView.delegate = self
    unseTableView.dataSource = self
}
```

- 하나의 셀에 각각 하나의 띠 이미지와 이름 그리고 운세 내용이 배치가 되도록 구현하였다. 총 12간지이므로 12개의 셀이 완성이 될 것이고, 이 12개의 셀이 하나의 테이블 뷰로 구성되도록 구현하였다.

5. 배경화면 및 각종 아이콘 일러스트 디자인



- 일러스트레이터를 이용하여 각 기능을 잘 표현할 수 있는 일러스트로 전부 제작하였다.

관상 ,설정 – 조윤상

[FaceUnseViewController.swift] WebView **,[GitPage.siwft]** 도 동일

```
override func viewDidLoad() {
    super.viewDidLoad()
        // Do any additional setup after loading the view.
     WebView.load(URLRequest(url: URL(string: "https://yoonsang2.github.io/MobileAPP_iOS_TEAM/")!))

    self.WebView.navigationDelegate = self
```

url 을 URLRequest 통해 캐싱하여 WebView 로 로드해온다.

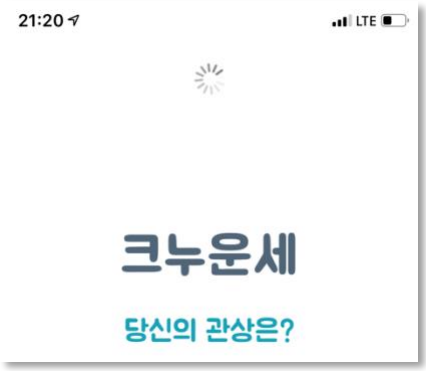
또한 Webview 자기 자신을 델리게이트 하여준다.

```
let refreshControl = UIRefreshControl()
refreshControl.addTarget(self, action: #selector(refreshWebView(_:)), for: UIControl.Event.valueChanged)
WebView.scrollView.addSubview(refreshControl)
WebView.scrollView.bounces = true
```

RefreshControl 객체를 생성하고 target(refreshWebView 함수)을 연결하여준다. 그뒤 WebView 의 ScrollView 에 Action 을 추가하여준다.

```
@objc func refreshWebView(_ sender: UIRefreshControl){
            WebView?.reload()
            sender.endRefreshing()
}
```

Action 이 일어나면 refreshWebView 함수를 실행하고 웹페이지를 새로고침한다.




```
func webView(_ WebView2: WKWebView, didFinish navigation: WKNavigation!) {
        // hide/stop indicator
        indicator.stopAnimating()
}
```

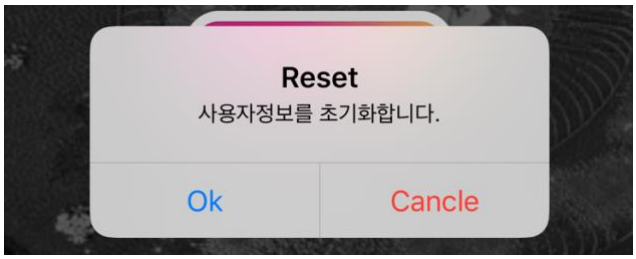
WKViewNavigation 이 로드를 끝냈을때 indicator 가 작동중지 함수를 수행한다.

[SettingProfileViewController.swift]

사용자 정보 표시 라벨의 모서리 처리와 투명도

```swift
//모서리 처리
profile.layer.cornerRadius = 5
profile.layer.masksToBounds = true

// 투명도
self.profile.backgroundColor = UIColor.gray.withAlphaComponent(0.5)
```

AlertController 구현




```swift
@IBAction func resetData(_ sender: Any) {
 // alert 객체 생성
 let alert = UIAlertController(title: "Reset", message: "사용자정보를 초기화합니다.", preferredStyle: UIAlertController.Style.alert)
 //ok action
 let okAction = UIAlertAction(title: "Ok", style: .default) { (action) in
     ViewController.db.deleteByID()
        ViewController.mUser = nil
     self.tabBarController?.selectedIndex = 0
 }
 let cancle = UIAlertAction(title: "Cancel", style: .destructive) { (action) in}

 alert.addAction(okAction) //alert add OkAction
 alert.addAction(cancle) //alert add CancleAction
 present(alert, animated: false, completion: nil)
```

Alert 객체를 생성하고 타이틀과 메시지를 입력하여 준다. 그 후 AlertAction 객체 2 개를 생성후 컨트롤러에 추가하여준다. (action) in 뒤에는 ok 버튼을 눌렀을때의 실행결과로써 db 를 초기화 시켜주고 tabBarcontrroller 에 0 번 페이지로 이동한다.

[TabBarController.swift] -> 스와이프 제스쳐로 화면넘김 구현

```swift
let swipeRight = UISwipeGestureRecognizer(target: self, action: #selector(handleSwipeGesture))
swipeRight.direction = .right
self.view.addGestureRecognizer(swipeRight)

let swipeLeft = UISwipeGestureRecognizer(target: self, action: #selector(handleSwipeGesture))
swipeLeft.direction = .left
self.view.addGestureRecognizer(swipeLeft)
```

좌, 우 스와이프 제스쳐 객체를 생성하여 준다.

```swift
@objc func handleSwipeGesture(_ gesture: UISwipeGestureRecognizer) {

    if gesture.direction == .left {
        if (self.selectedIndex) < 4 { // 슬라이드할 탭바 갯수 지정 (전체 탭바 갯수 - 1)
            animateToTab(toIndex: self.selectedIndex+1)
        }
    } else if gesture.direction == .right {
        if (self.selectedIndex) > 0 {
            animateToTab(toIndex: self.selectedIndex-1)
```

그 후 제스처 핸들 함수를 구축한다. 각 제스처별 페이지 index 를 +-1 씩 증감시켜준다. 범위는 0~4(5 개의 페이지).