[WorksMobile 동계인턴십]

[개인 보고서]

[Mobile Dev1 성지훈]

The table of contents

1. 개요

현재 사내 WORKS 및 네이버웍스 애플리케이션에 사용되는 구성원 일정 뷰는 PC 구성원 일정 뷰와 달리 구성원들의 일정을 한눈에 보기 어렵다는 단점과 구성원 일정을 일간 단위로만 볼 수 있다는 점을 개선하고 추후 다른 사용자도 쉽게 사용할 수 있도록 구성원 일정뷰를 커스텀뷰로 제작하여 오픈소스 라이브러리로 제작하는 것을 목표로 하며 라이브러리에 대한 사용방법 및 샘플 애플리케이션을 제작하여 사용자에게 제공하는 것을 최종 목표로 한다.

2. 구조

2.1 라이브러리 구조

라이브러리에 사용되는 커스텀뷰는 Vertical LinearLayout 으로 구성되어 있으며 내부에 FrameLayout 이 있다. FrameLayout 을 사용한 이유는 구성원 일정을 그리기 위한 mainRecyclerView 위에 이름만을 표기하는 nameRecyclerView 를 겹쳐서 놓아야 하기 때문이다.
이처럼 리사이클러뷰를 두가지로 나누어 사용하는 이유는 커스텀뷰가 상하좌우 스크롤이 가능해야 하며 모바일 화면이 PC 에 비해 작은 것을 고려하여 좌우 스크롤시 좌측 헤더가 따라서 스크롤 되어 화면에서 사라지는 것을 방지하고 좌측헤더를 고정시켜야 하기 때문이다. 또한 구성원 일정뷰는 Vertical 한 RecyclerView 이므로 좌우로 스크롤이 불가능하다. 따라서 mainRecyclerView 를 HorizontalScrollView 로 감싸는 형태로 구성하였다.

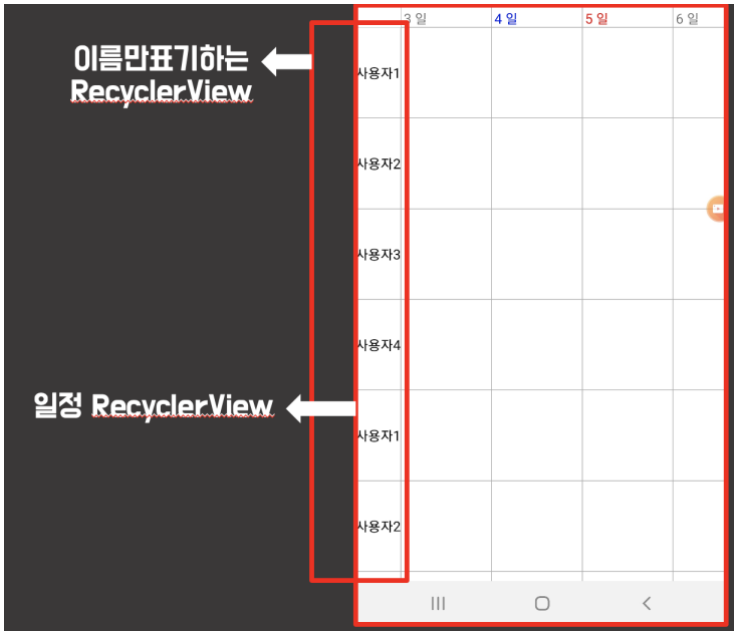

그림 1

그림 1 에서 좌측 이름만 표기하는 RecyclerView 는 구성원 일정뷰를 표기하는 RecyclerView 가 좌우로 스크롤되어도 다른 뷰이기 때문에 항상 고정되어 있는 상태이다. 실제로 좌측 헤더를 따로 고정시킨 것은 아니지만 사용자가 볼 때 불편함이 없도록 아이디어를 생각하여 구현한 방법이다. 또한 상단 헤더 고정을 위해 mainRecyclerView 에서 ItemDecoration 을 추가하여 상단 헤더를 기존 RecyclerView 위에 다시 그려 상단헤더가 고정되어 있는 것 처럼 보이도록 구현하였다. 따라서 좌우 스크롤 시 좌측 사용자의 이름헤더는 고정되어 있으며 상하 스크롤 시 시간 헤더는 고정되어 있는 형태이다. 아래는 커스텀뷰의 구조이다.

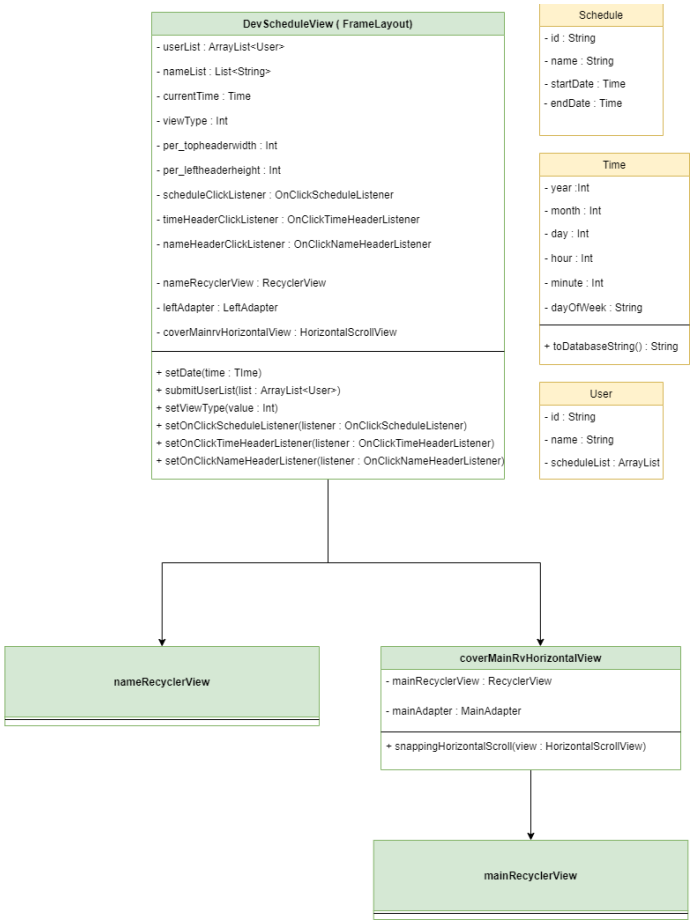

그림 2

3. 구현과정

3.1 Library

라이브러리는 크게 mainview, mainrv, leftrv, interface, model패키지로 구분되어 있으며 공통적으로 사용할 메소드는 singleton object로 Utils.kt에 구현되어 있다.

3.1.1 DevScheduleView

mainview 에는 DevScheduleView 인 메인 커스텀뷰가 구현되어 있다. 커스텀뷰인 DevScheduleView 는 FrameLayout 으로 구성한다.

먼저 사용자가 xml 에서 커스텀뷰를 추가하여 사용할 경우, 사용자는 viewType 과 visibleNumberOfMembers 2 가지의 attribute 를 설정할 수 있다. viewType 은 사용자가 커스텀뷰를 일간/주간으로 설정할 수 있고, visibleNumberOfMembers 는 한 화면에 표기할 사용자의 수를 지정하는 attribute 이다.

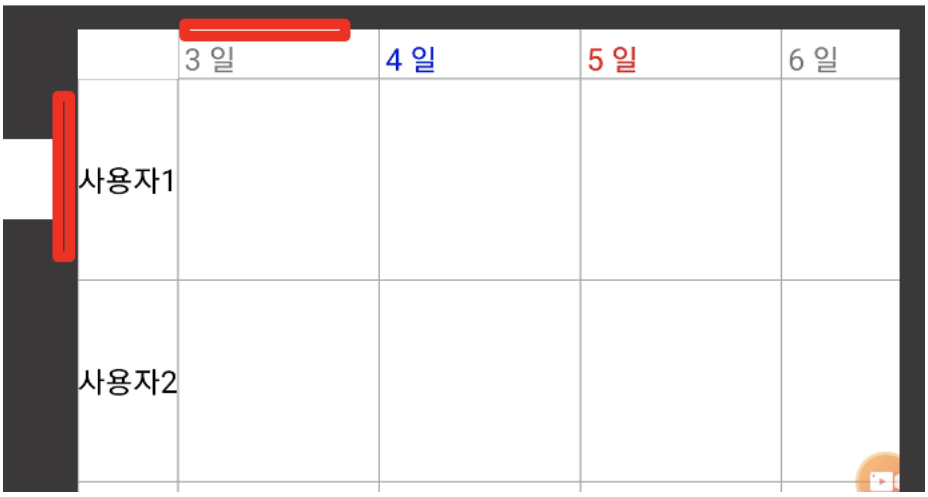

그림 3

현재 그림 3 은 주간 뷰에서 사용하는 모습이다. 사용자가 viewType 을 1 로 설정한 모습이며 viewType 이 0 인경우에는 그림 4 와 같이 시간 헤더가 별도로 표기된다.

그림 4

아래는 커스텀뷰 내의 각 메소드의 세부적인 사항이다.

setupDefaultListener() : 커스텀뷰의 내부에 사용되는 스크롤 리스너들을 설정하는 메소드이다. 내부에는

1. globalLayoutListener : 리스너를 사용한 이유는 뷰가 그려지고, 뷰의 높이를 측정하여 사용자 1 명이 차지하는 높이를 저장하기 위한 리스너이다. ViewTreeObserver 의 OnGlobalLayoutListener 는 레이아웃에 변경이 일어날때 호출된다. 25**dp** 를 뺀 이유는 상단 헤더의 높이가 25dp 이기 때문이다.

```
globalLayoutListener = object : ViewTreeObserver.OnGlobalLayoutListener {
    override fun onGlobalLayout() {
        perLeftHeaderHeight =
            (this@DevScheduleView.height - getDP( value: 25)) / visibleNumberOfMembers
    }
}
```

2. dateChangeListener : 일간, 또는 주간뷰에서 좌측 끝, 우측 끝으로 스크롤될 때 변경된 날짜를 전달받을 수 있는 리스너이다. 사용자가 외부에서 setOnDateChangeListener 로 날짜의 변경을 전달받을 수 있다.

```
dateChangeListener = object : OnDateChangeListener {
    override fun OnDateChange(date: Time) {
        //dateChangeListener
    }
}
```

3. pageChangeListener : 페이징을 사용할 때 다음 요청할 페이지를 전달한다. 외부에서 setOnPageChangeListener 로 설정하고 전달받은 페이지로 데이터를 setList 로 추가하는 방법을 사용하여 페이징을 구현할 수 있다.

```
pageChangeListener = object : OnPageChangeListener {
    override fun onPageChange(page: Int) {
        //pageChangeListener
    }
}
```

4. scheduleLongClickListener : 일정을 LongClick 할때의 동작이 구현되어있다. 기본적으로 LongClick 된 일정 뷰를 받아 drag and drop 을 실행하도록 구현되어 있다
5. scheduleClickListener : 일정을 클릭할때의 동작이 구현되어 있다. 기본적으로 빈 일정 클릭과 존재하는 일정 클릭 두가지로 나누어지며 존재하는 일정을 클릭한 경우, 해당 일정을 가진 사용자의 position 과 schedule 의 index 를 전달한다..

6. nameHeaderClickListener : 이름헤더 클릭에 대한 확장 애니메이션에 대한 기능이 구현되어 있다. 자세한 기능은 아래에 명시되어 있다.
7. timeHeaderClickListener : 상단 시간헤더 클릭 시 동작에 대한 기능이다. 사용자가 setOnTimeHeaderClickListener 를 통해 동작을 정의할 수 있다.

measureHeight() : 사용자가 설정한 한 화면에 보이는 사용자수에 맞춰 각 아이템의 높이를 설정하는 globalLayoutListener 를 rootView 에 추가한다.

setupAnimator() : 모바일 화면에서는 PC 구성원 일정뷰의 주간 일정처럼 일정을 쌓아가는 형태로 구현하는 경우, 일정의 개수가 최대 2~3 개까지만 볼 수 있다는 점을 감안하여 이를 해결하기 위해 ValueAnimator 를 사용하였다. 좌측 헤더 클릭시 slideAnimator 를 사용하여 좌측 헤더와 해당 포지션의 메인 리사이클러뷰 아이템의 높이를 현재 높이의 2 배로 늘리도록 구현하였으며 cancelAnimator 의 경우 사용자가 헤더를 한번더 클릭하거나 다른 사용자 이름을 클릭할 경우 기존의 증가했던 높이를 다시 되돌리는 ValueAnimator 이다.

setupGetWidthView() : 좌우 스크롤 시, 사용자가 지정한 topheaderwidth 에 맞추어 스크롤 하기 위해서 사용자가 지정한 정확한 width 만큼의 스크롤 값을 측정해야한다. 이때 측정되는 width 값을 측정하기 위해 지정한 크기에 맞는 View 를 생성하고 이 뷰(getWidthView)의 width 값을 측정하여 좌우 스크롤시 칸 단위로 스크롤 될 수 있도록 Snapping 을 제어한다.

setupHorizontalScrollView() : 메인 리사이클러뷰를 감싸기 위한 HorizontalScrollView 의 기본 파라미터를 설정하고 프레임 레이아웃 내에 추가한다.

setupMainRecyclerView() : mainRecyclerView 를 생성하고 리사이클러뷰의 어댑터 설정 및 리사이클러뷰를 설정한다. mainRecyclerView 는 LinearLayoutManager 를 사용하며 상단 헤더 고정을 위해 ItemDecoration 을 사용하고, 어댑터에서 onBindViewHolder 를 반복적으로 호출하지 않도록 setItemViewCacheSize()를 사용하고 리사이클러뷰를 coverMainHorizontalView 에 추가한다. 또한 paging 을 감지하기 위한 커스텀 EndlessRecyclerViewScrollListener 를 mainRecyclerView 에 추가한다.

setupNameRecyclerView() : 좌측 헤더를 위한 리사이클러뷰를 생성하고 파라미터를 설정한다. 리사이클러뷰의 어댑터 설정 및 리사이클러뷰를 설정하며 메인 리사이클러뷰와 마찬가지로 setItemViewCacheSize()를 사용한다. 마찬가지로 커스텀한 EndlessRecyclerViewScrollListener 를 추가한다.

setupHeaderSpaceView() : ItemDecoration 을 통해 상단 헤더를 고정시키는 것은 실제로 해당위치에 0 번 포지션의 아이템이 존재하는것이 아니기 때문에 클릭이 불가능하다. 따라서 프레임레이아웃 위에 동일한 크기의 투명한 뷰를 추가하고 클릭 리스너를 설정한다.

setupTopSpaceView() : 좌측상단에 비에있는 위치에 구분을 위한 뷰를 추가한다.

setupObserveScroll() : 두개의 리사이클러뷰는 상,하 스크롤 시 함께 움직여야 하므로 scrollListenerFromMainRv 라는 OnScrollListener 를 생성하고 mainRecyclerView 가 상,하로 스크롤 되는 경우 nameRecyclerView 도 함께 스크롤 되도록 nameRecyclerView.scrollBy()를 이용하여 스크롤한다. 추후에 화면을 갱신할 때 스크롤의 위치를 저장하기 위해서 saveScrollY 라는 값을 두어 스크롤의 위치를 저장하며 생성된 리스너는 onDetachFromWindow 에서 제거된다. 또한 nameRecyclerView 에 ItemTouchListener 를 추가하여 터치이벤트를 제어한다. 사용자가 입력한 터치이벤트가 Action_Down 일 경우에만 이벤트를 전달하고 그 이외의 이벤트는 Intercept 한다. 따라서 nameRecyclerView 를 직접적으로 스크롤 하는 것을 방지하고 터치이벤트만 가능하도록 구현하였다.

setupSnappingEffect() : HorizontalScrollView 에서 ActionUP 또는 ActionCancel 이벤트가 발생할 때 스크롤위치를 재조정 하여 한칸 단위로 스크롤 되는 효과를 설정한다. 먼저 scrollX 값을 받아오고, 사용자가 설정한 한 칸의 width 값을 가져온다. 이 값을 이용하여 스크롤의 위치를 재조정하여 한칸씩 스크롤 되는 효과를 준다.

setOnClickScheduleListener(),
setOnClickTimeHeaderListener(),
setOnClickNameheaderListener(),

setOnPageChangeListener(), setOnDateChangeListener : 사용자가 외부에서 리스너를 별도로 설정하고자 할 때 리스너를 등록할 수 있는 메소드이다. 각각 스케줄클릭, 상단 시간헤더 클릭, 좌측 이름헤더 클릭 시 동작을 정의할 수 있다.

setDate() : 사용자가 외부에서 날짜를 설정할 수 있다. Time 객체를 파라미터로 받으며 커스텀뷰 생성 이후 필수로 날짜를 설정해야 한다.

submitUserList() : List<User>객체를 파라미터로 받으며 사용자가 일정 또는 사용자 갱신 시 사용하는 메소드이다.

resetRv() : 사용자가 viewtype 을 변경하거나 userList 갱신 시, 동적으로 그린 뷰를 제거하고 다시 그리기 위해 사용하는 메소드이다. mainRecyclerView 에 있는 하위 뷰를 mainRecyclerView.removeAllViewsInLayout() 메소드로 지우고 다시 생성한다. 만약 헤더 확장애니메이션이 실행되어 있는 상태인 경우, 애니메이션을 취소하고 높이를 기존 상태로 되돌린다.

startExpandAnimation() : nameRecyclerView 의 터치이벤트 발생 시, 해당 포지션의 헤더 크기를 늘리는 애니메이션을 실행한다. 단, nameRecyclerView 이외에 mainRecyclerView 도 함께 이벤트를 실행해야 하기 때문에 TouchEvent 가 발생한 포지션의 값으로 findViewHolderForAdapterPosition 메소드를 이용하여 포지션에 맞는 뷰를 불러온 뒤, 각 뷰의 높이를 기존 뷰에서 2 배로 늘리는 애니메이션을 실행하는 메소드이다.

cancelAndExpandAnimation() : startExpandAnimation 과 반대로 현재 늘어나 있는 포지션의 뷰를 다시 원래 크기로 되돌리는 애니메이션과 클릭된 포지션의 높이를 2 배로 늘리는 애니메이션을 함께 시작한다. animatorSet.playTogether, animatorSet.start 로 확장, 축소 애니메이션을 동시에 수행한다.

setupInfiniteHorizontalScroll() : HorizontalScrollView 의 ViewTreeObserver 를 이용해 스크롤의 변화를 감지하고, 주간/일간을 구분하여 스크롤의 위치가 처음 또는 마지막에 도달하면 다음 일정을 불러오도록 설정한다. 여기서 dateChangeListener 의 onDateChange 메소드를 실행하여 외부에서 날짜의 변화를 전달받도록 한다.

loadLaterWeekSchedule(), loadLaterDaySchedule() : 스크롤의 위치가 끝에 도달했을때 다음 일정을 로딩하기 위한 메소드이다. 먼저 동기화되어있던 두 리사이클러뷰의 스크롤리스너를 지우고 현재 날짜를 변경한다. 그리고 리사이클러뷰의 뷰를 다시그린다. 이때 현재 확장 애니메이션이 적용된 아이템의 리스트를 어댑터에 전달하고, 확장 애니메이션이 적용된 포지션의 아이템은 높이를 2 배로 설정한다. scrollTo 가 제대로 동작하지 않는 경우를 대비해 postDelayed 를

사용하였으며 일정이 로딩되면서 저장되어있던 상, 하 스크롤 위치로 mainRecyclerView 를 이동한다. 이후, 다시 스크롤 동기화를 위한 리스너를 추가한다.

loadPriorWeekSchedule(), loadPriorDaySchedule() : 스크롤의 위치가 처음에 도달했을때 이전 일정을 로딩하기 위한 메소드이다. 먼저 동기화되어있던 두 리사이클러뷰의 스크롤리스너를 지우고 현재 날짜를 변경한다. 그리고 리사이클러뷰의 뷰를 다시그린다. 이때 현재 확장 애니메이션이 적용된 아이템의 리스트를 어댑터에 전달하고, 확장 애니메이션이 적용된 포지션의 아이템은 높이를 2 배로 설정한다. 이후 일정 로딩과 마찬가지로 scrollTo 가 제대로 동작하지 않는 경우를 대비해 postDelayed 를 사용하였으며 일정이 로딩되면서 저장되어있던 상, 하 스크롤 위치로 mainRecyclerView 를 이동한다. 이후, 다시 스크롤 동기화를 위한 리스너를 추가한다.

resetCurrentPage() : 사용자가 화면을 초기화하거나 다시 로딩이 필요한경우, 불러왔던 페이징 정보를 초기화시킨다.

setList() : 페이징을 사용하는 경우, submitList 가 아닌 setList 로 다음에 추가할 유저리스트를 갱신할 수 있다.

resetScroll() : mainRecyclerView 의 스크롤 위치를 처음으로 이동시키며 HorizontalScrollView 의 스크롤을 뷰타입에 따라 알맞은 위치로 이동시킨다.

setupDragListener() : 커스텀뷰에 DragListener 를 추가한다. 드래그 상태에 따라 좌측 상단에 현재 드래그중인 위치가 표기되며 내부에는 커스텀뷰를 좌표로 보고 현재 드래그 위치를 구하는 로직이 구현되어 있다. 그 이외에 드래그상태에서 스크롤이 가능하도록 구현되어 있다.

countPositionY() : 드래그 상태에서 현재 드래그중인 사용자의 위치를 구하는 메소드이다. 현재 커스텀뷰에는 확장되어 있는 포지션이 있으므로, 이를 별도로 고려하여 현재 Y 좌표를 구하는 메소드이다.

setupIndicator() : 뷰 드래그시 좌측 상단에 표기되는 드래그의 위치를 표기하기 위한 IndicatorView 의 텍스트를 변경하는 메소드이다.

actionDrop() : 드래그앤 드롭 상태에서 일정을 Drop 할때 동작을 구현한 메소드이다. 먼저, 현재 드롭된 일정의 정보를 currentSchedule 에 저장하고 일정의 원래 시작 시간과 현재 Drop 한 시간의 차이를 구한다. 이후 viewType 에 따라 일정을 Drop 한 시간으로 이동시키며 scheduleDropListener.onScheduleDrop 메소드를 통해 변경된 일정에 대한 정보(변경된 유저의

ID 와 변경된 일정)을 전달한다. 매번 사용자가 뷰를 새로고침할 필요가 없도록 내부 리스트에서 기존 일정을 삭제하고 새롭게 옮긴 일정을 리스트에 추가하게 된다. 그후 화면을 갱신하는 updateView 를 실행한다.

updateView() : 변경된 일정을 적용시키는 메소드이다.

3.1.2 mainRecyclerView

mainrv 패키지는 메인 리사이클러뷰 관련 클래스가 모여있는 패키지이다.

- **MainAdapter** : mainRecyclerView 의 어댑터이며 DiffUtil 을 이용한 ListAdapter 를 사용한다. 패키지 내에 UserDiffCallback 으로 아이템의 동일 여부를 비교하며 유저 고유의 Id 와 name, scheduleList 가 모두 일치하는 경우에만 같은 아이템으로 판단한다.
- 또한 실제 사용자가 저장한 각 User 의 position 은 1 부터 시작하며 0 번째 포지션은 시간 헤더로 고정되어 있다. 따라서 2 가지타입의 뷰홀더를 별도로 제작하여 포지션별로 다른 뷰홀더를 바인딩하였다. getItemViewType 메소드를 오버라이딩 하여 position 이 0 인 경우에만 뷰타입을 0 으로 지정한 상태이다.
- **StickyHeaderItemDecoration :** ItemDecoration 을 상속하여 화면 위에 0 번째 포지션의 시간 헤더를 다시 그려주는 동작을 위한 ItemDecoration 이다. 이것으로 상단 헤더가 고정되는 효과를 구현하였다.
- **ItemContent :** content 패키지 내에는 mainRecyclerView 에서 쓸 ContentViewHolder 와 뷰로 사용할 ItemContent, 뷰 내부에 일정을 그릴 ScheduleBinder 가 있다. 뷰로 사용할 ItemContent 는 기본적으로 LinearLayout 이며 뷰타입에 따라 내부 구조가 달라진다. 일정을 겹쳐서 표기하는것을 고려해야하는 일간 뷰의 경우 내부에 FrameLayout 이 있으며 주간뷰의 경우는 각 날짜에 맞춰 Vertical LinearLayout 이 추가되어 있다. 총 14 일간의 일정을 그리기 위해 14 개의 LinearLayout 이 있다.
- **ScheduleBinder** 에서 사용자의 일정과 현재 날짜에 맞춰 동적으로 일정을 그린다. 일정은 TextView 로 구성되어 있다. 사용자의 일정을 그리기 위한 알고리즘은 뷰 타입에 따라 다르게 설계하였다.
- **ItemHeader :** header 패키지 내에는 상단 시간헤더를 표기할 HeaderViewHolder 와 뷰로 사용할 ItemHeader 가 있다. ItemHeader 는 LinearLayout 이며 뷰 타입에 따라 시간 텍스트를 추가하거나 날짜 텍스트를 추가한다. 주간 뷰의 경우 calendar 를 통해 요일을 판별하여 토요일인 경우 textColor 를 blue 로 지정, 일요일인 경우 textColor 를 red 로 지정하였으며 일간 뷰의 경우 0~23 시 까지의 뷰가 추가된다.

일간 뷰에서 일정을 그리는 방법

- 먼저 사용자의 스케줄목록을 순회하며 해당 시간에 맞는 뷰를 그리게 된다. 이 때, 뷰는 ItemContent 에 미리 생성해 놓은 FrameLayout 위에 TextView 를 그리게 된다. 일정의 Case 를 총 4 가지 경우로 분류한다.

  1. 일정의 시작날짜, 종료날짜가 모두 현재 날짜를 벗어나지 않는 경우
     TextView 의 margin, width 를 시작시간 및 종료시간에 맞춰 설정한다. 먼저, 시작시간과 종료시간을 비교하여 사용자가 설정한 per_topheaderwidth 와 곱한 값으로 width 를 지정한다. 시작시간 * per_topheaderwidth 로 marginstart 를 지정한다. 이렇게하여 하루 이내의 일정에 대한 처리를 구현하였다.
  2. 일정의 시작날짜가 현재날짜이고 종료날짜가 현재날짜보다 이후인 경우
     TextView 의 margin 은 1 번 케이스와 마찬가지로 시작시간에 맞춰 설정한다. 하지만 종료시간을 따로 비교할 필요 없이 width 를 MATCH_PARENT 로 지정한다.
  3. 오늘 날짜가 일정의 시작 날짜 이후이고 종료날짜 이전인 경우
     즉 오늘 날짜에선 해당 일정이 종일 일정인 경우이다. 이 경우에는 더 간단하게 margin 없이 width 를 MATCH_PARENT 로 지정한다.
  4. 오늘 날짜가 일정의 시작 날짜 이후이고 종료날짜가 오늘 날짜인 경우
     이 경우에는 width 를 종료시간 * per_topheaderwidth 로 지정하기만 하면 된다. 결과적으로 0 시~종료시간 까지가 해당 일정의 영역으로 지정되며 사용자에게 표기된다.

일간 뷰에서 겹치는 일정에 대한 처리

겹치는 일정은 Case에 관계없이 처리해야 하기때문에 모든 케이스에 공통적으로 적용된다. 알고리즘은 직접 구상하여 구현하였으며 아래와 같이 처리한다. 먼저 시간순으로 정렬된 일정에서 일정을 채울때, 해당 일정이 그려진 위치를 schedulePosition으로 저장한다. schedulePosition[i] = t라고 하면 i번째 위치에 t번째 스케줄이 위치하고 있다는 뜻이다. 따라서 일정을 추가하기 전, 일정들과의 시간을 비교해 현재 추가하고자하는 일정의 시작시간을 기준으로, 가장 상단에 비어있는 포지션이 몇번인지 찾고, 해당위치에 일정을 추가하는 방식이다. 이렇게 구성하면 일정이 절대 겹쳐서 그려지지 않는다. 일정의 높이는 기본적으로 사용자 한명이 차지하는 높이의 절반이지만, 사용자가 한화면에 너무 많은 사용자를 보도록 한 경우에는 상황에 맞게 동적으로 조절된다.

그림 5

일정이 매우 많이 겹치는 경우에도 아래와 같이 일정을 확인할 수 있다.

그림 6

주간 뷰에서 일정을 그리는 방법

주간뷰는 기존 ItemContent 에서 기존 뷰 내에 14 개의 LinearLayout 을 추가하고 이를 List 로 구성해놓았다. 이 레이아웃 리스트의 이름은 week_layoutList 이며 주간 뷰에서 현재 날짜와 일정의 시작, 종료날짜를 적절히 비교하여 알맞은 Index 의 week_layoutList 에 일정을 쌓아나가는 방법으로 구현하였다. 주간 뷰도 총 5 개의 Case 로 구성되어 있다. 단 주간뷰의 경우 케이스별로 해당 날짜의 일정이 종일인 경우가 있고, 해당 날짜가 일정의 종료일만을 포함하거나 시작일만을 포함하는 경우가 있다. 현재 PC 에서는 이러한 경우를 아래와 같이 처리한다.

그림 7

현재 일정은 1 월 31 일 22:00~ 2 월 2 일 10:00 까지의 일정이다. 시작일만을 포함하는 일정의 경우 해당 일정의 시작시간만을 표기하고, 사이에 끼어 종일에 해당하는 일정은 시간 대신 "종일"이라는 텍스트를 표기하고, 종료일에는 종료시간만을 표기하도록 되어있다. 이에맞춰 라이브러리에도 같은 방법을 적용하기 위한 알고리즘을 구상하고 구현하였다.

1. 일정이 하루 이내의 일정이며 그 시간이 현재 날짜로부터 14 일 이내로 있는 경우

2. 일정이 하루를 넘어가며 시작, 종료 시간이 모두 14 일 내에 있는 경우

3. 일정이 하루를 넘어가며 시작 시간은 14 일 내에 있지만 종료시간이 14 일을 벗어나는 경우

4. 일정이 하루를 넘어가며 시작 시간이 현재날짜 이전이며 종료시간이 14 일 내에 있는 경우

5. 일정의 시작, 종료시간의 차이가 15 일 이상이며 현재 날짜가 일정 사이에 있는 경우

먼저, 기본적인 뷰를 그리는 방법은 모두 동일하다. 현재 날짜와 해당 날짜의 차이를 Difference 라고 한다면 미리 정의해놓은 week_layoutList 의 Difference 인덱스에 텍스트뷰를 추가하는 것이다. 단, 케이스에 따라 텍스트를 시작 시간만 표기하거나, 종일로 표기하거나, 종료시간만 표기하는 것이다. 하루 이상의 일정의 경우, for 문을 통해 해당하는 일정에 맞춰 모두 일정 텍스트뷰를 추가한다. 예를들어, 그림 7 에서 보았던 3 일에 걸친 일정은 현재 구현한 뷰에서 그림 8 과 같이 구현된다.

그림 8

다만 일정이 너무 많아 화면에 다 보이지 않는 경우, 일정의 개수를 …+개수로 표기하도록 구현하였다. 최대 일정의 개수는 사용자가 설정한 한 화면에 볼 수 있는 사용자의 수에 따라 변경되며 일정이 임계점이상인 경우 더이상 그리지 않고 초과되는 일정의 개수를 overScheduleNum 배열에 추가하고, 마지막에 화면에 표기한다.

	14일
1	00:02~05:38 일정7
	01:40~02:27 일정1
	02:41~07:22 일정5
	03:04~07:55 일정3
	09:12~11:28 일정10
	... +6

그림 9

일정 클릭 리스너

일정 클릭 리스너는 기본적으로 mainRecyclerView 에서 mainAdapter 의 파라미터로 ViewHolder 까지 전달된다. 기본적으로 ContentViewHolder 는 View.OnClickListener 를 상속받아 onClick 메소드를 오버라이딩 하여 구현한다. 일정을 클릭할 때 동적으로 그린 TextView 마다 setOnClickListener(this)를 통해 리스너를 명시해야 하므로 이를 담을 scheduleListView = ArrayList<TextView>()를 선언하고, 뷰를 동적으로 그릴때마다 리스트에 추가한다.

```
override fun onClick(v: View) {
    if(v in scheduleListView){
        onClickScheduleListener.onClickSchedule( position: adapterPosition-1,map.getOrDefault(scheduleListView.indexOf(v), defaultValue: -1))
    }
    else{
        onClickScheduleListener.onClickEmptySchedule()
    }

}
```

그림 10

만약 클릭 이벤트가 발생한 뷰가 scheduleListView 에 있다면, onClickScheduleListener 의 onClickSchedule 메소드를 실행한다. 이때, adapterPosition 과 해당 유저의 schedule 에서 클릭한 schedule 의 index 를 전달한다. map 은 ConcurrentHashMap<Int, Int>로 구성되어 있으며 현재 추가한 scheduleListView 에서의 Index 로 유저의 scheduleList 의 Index 를 찾기 위해 사용된다.

이 값을 전달받고 스케줄의 id 를 조회하고자 한다면 userList[position].scheduleList[scheduleidx]로 해당 스케줄에 접근할 수 있다.

클릭 이벤트가 발생한 뷰가 scheduleListView 에 포함되지 않았다면, 빈 스케줄을 클릭한 이벤트이므로 onClickScheduleListener.onClickEmptySchedule 메소드를 실행한다.

3.1.3 nameRecyclerView(좌측 헤더)

leftrv 패키지 내에 좌측 헤더 관련 리사이클러뷰의 클래스들을 정리하였다. nameRecyclerView 는 기존 userList 에서 filter{it.name}을 이용하여 이름만을 리스트로 추출하여 사용한다.

- **LeftAdapter** : nameRecyclerView 의 어댑터이며 DiffUtil 을 이용한 ListAdapter 를 사용한다. 패키지 내에 NameDiffCallback 으로 아이템의 동일 여부를 비교하며 유저의 name 만 비교하여 같은 아이템인지 여부를 판단한다.
- **ItemName** : name 패키지 내에는 이름을 표기할 NameViewHolder 와 뷰로 사용할 ItemName 이 있다. ItemName 은 LinearLayout 이며 이름을 표기할 TextView 를 하위 뷰로 가지고 있다.

3.1.4 model

라이브러리에서 사용하는 데이터 모델은 크게 3 가지 이다. 시간을 표기하기위한 Time 객체, 스케줄의 정보를 담은 Schedule 객체, 사용자의 정보를 담은 User 객체가 이에 해당한다.

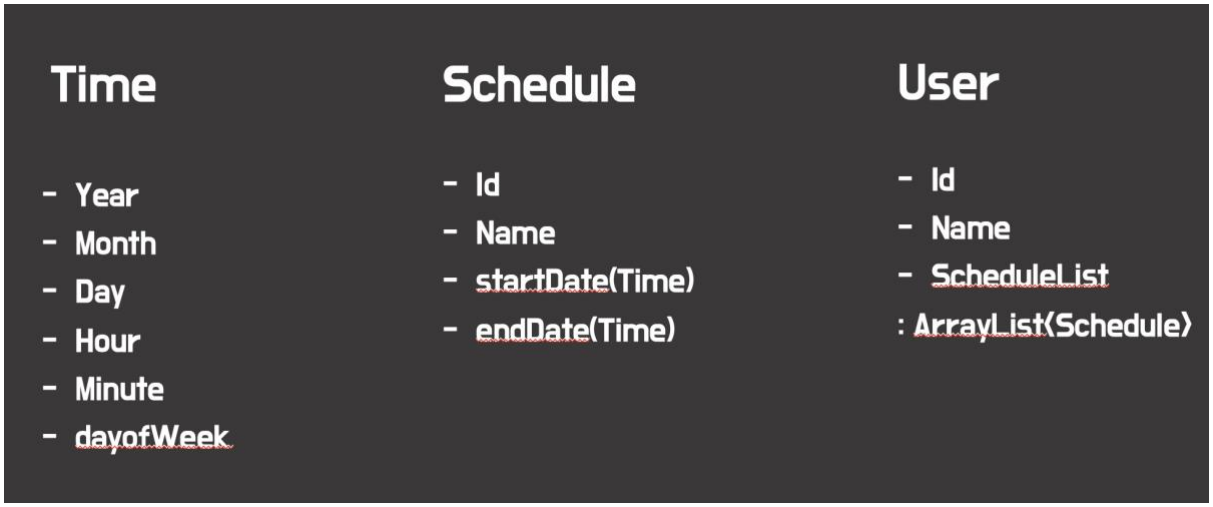

그림 11

3.2 라이브러리 사용법

샘플앱은 라이브러리를 사용하는 한 예시를 제작한 애플리케이션이다. 샘플앱은 Room DB 를 사용하여 로컬에 있는 데이터를 활용하며, MVVM 과 Repository 패턴을 활용하여 구현하였다. 샘플앱은 크게 홈화면, 일정 추가화면, 일정 상세 화면, 개발자화면으로 구성되어 있다.

샘플앱을 구성하는 순서는 아래와 같다.

1. 먼저 라이브러리를 프로젝트에 Import 한다.

2. layout 내에 DevScheduleView 를 추가한다..

```
<com.example.library.mainview.DevScheduleView
                    android:id="@+id/devschedule_view"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
                    app:visibleNumberOfMembers="5"
                    app:viewType="0"/>
```

visibleNumberOfMembers 는 한 화면에 나타낼 사용자의 수이며 viewType 은 0(일간뷰), 1(주간뷰)를 나타내는 값이다.

3. 현재 보여줄 일정의 날짜를 먼저 설정해야 하며 setDate(Time)으로 설정할 수 있다. 기본적으로 뷰 바인딩, 또는 데이터 바인딩 사용을 권장한다.

Time 객체는 연, 월, 일, 시, 분, 요일로 구성된 객체이다.

```
binding.devscheduleView.setDate(Time(2022,1,25,10,30,"FRI")
```

4. 좌, 우 무한스크롤을 사용한다면 스크롤 시 변경된 날짜를 받을 dateChangeListener 를 등록한다.

```
binding.devscheduleView.setOnDateChangeListener(object : OnDateChangeListener {
    override fun OnDateChange(date: Time) {
        //시간 업데이트
    }
})
```

5. devScheduleView.submitUserList, setList 로 User 및 ScheduleList 를 등록한다..

타입은 ArrayList<User> 이며. 사용자의 필요에 따라 내장 데이터베이스를 이용하거나, 외부 API 를 이용하여 적절히 사용한다..

DevScheduleView 에 유저리스트를 등록하는 방법은 페이징을 사용하는 방법, 모든 유저 정보를 한꺼번에 등록하는 방법으로 나눌 수 있다.

* 모든 유저 정보를 한번에 등록하는 경우

기존 설명과 같이 setDate(Time)으로 먼저 날짜를 설정합니다.

User(id, name, ArrayList<Schedule>)을 담은 ArrayList<User> 객체를 binding.devscheduleView.submitList(userList)로 전달하면 해당되는 스케줄을 화면에 자동으로 그리게 된다.

화면에 돌아왔을 때, 자동으로 새로고침이 필요한 경우에 기존의 유저리스트를 로컬에서 저장하고 있다면 아래와 같이 사용한다.

```
override fun onResume(){
    binding.devscheduleView.submitUserList(ArrayList(userList))
    super.onResume()
}
```

userList 가 로컬에 저장되어 있지 않은 경우 binding.devscheduleView.getUserList()를 통해 현재 등록되어 있는 UserList 를 얻을 수 있다..

* 페이징을 사용하는 경우

마찬가지로 setDate(Time)으로 먼저 날짜를 설정한다.

커스텀뷰에는 페이징을 사용하여 데이터를 추가할 때, 현재 스크롤값에 따라 다음 요청할 페이지를 전달받을 수 있는 리스너가 등록되어 있으며 아래와 같이 사용할 수 있으며.
전달받은 페이지에 맞춰 binding.devscheduleView.setList(userList)를 추가한다.

```
binding.devscheduleView.setOnPageChangeListener(object : OnPageChangeListener{
    override fun onPageChange(page: Int) {
        // 필요한 만큼의 데이터를 로드하고, setList(userList)를 통해 추가
```

```
    }
})
```

MVVM 을 사용하는 경우 아래와 같이 필요에 따라 데이터를 로드하고, LiveData 의 변경에 맞춰 setList 를 실행한다.

```
binding.devscheduleView.setOnPageChangeListener(object : OnPageChangeListener{
    override fun onPageChange(page: Int) {
        viewModel.loadUserListPaging(page)
    }
})

viewModel.pagingUserList.observe(this,{
    binding.devscheduleView.setList(it)
})
```

마찬가지로 화면에 돌아올때 마다 새로고침하고자 하는경우, 아래와 같이 코드를 추가한다.

```
override fun onResume(){
    binding.devscheduleView.submitUserList(ArrayList())
    binding.devscheduleView.resetCurrentPage()
    super.onResume()
}
```

6. 드래그앤 드롭을 사용하고자 한다면 변경된 일정을 받아야 하므로 아래와 같이 리스너를 설정한다. 변경된 일정을 받아 내부, 외부 데이터를 업데이트 한다.

```
binding.devscheduleView.setOnScheduleDropListener(object : OnScheduleDropListener {
    override fun onScheduleDrop(changeUserId: String, schedule: Schedule) {
        viewModel.modifySchedule(changeUserId, schedule)
    }
})
```

4. Class Diagram

4.1 Library Structure

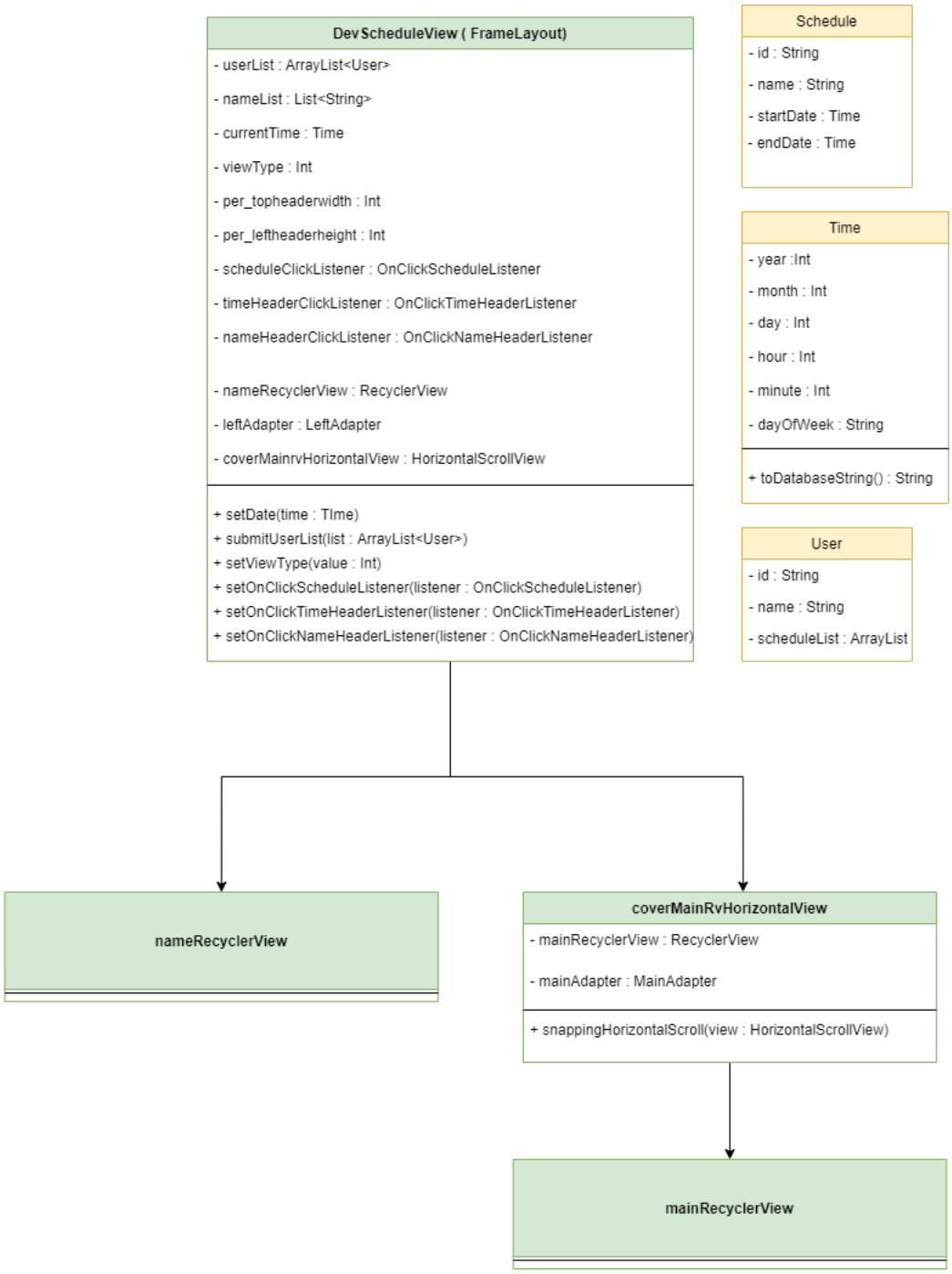

그림 12

4.2 nameRecyclerView

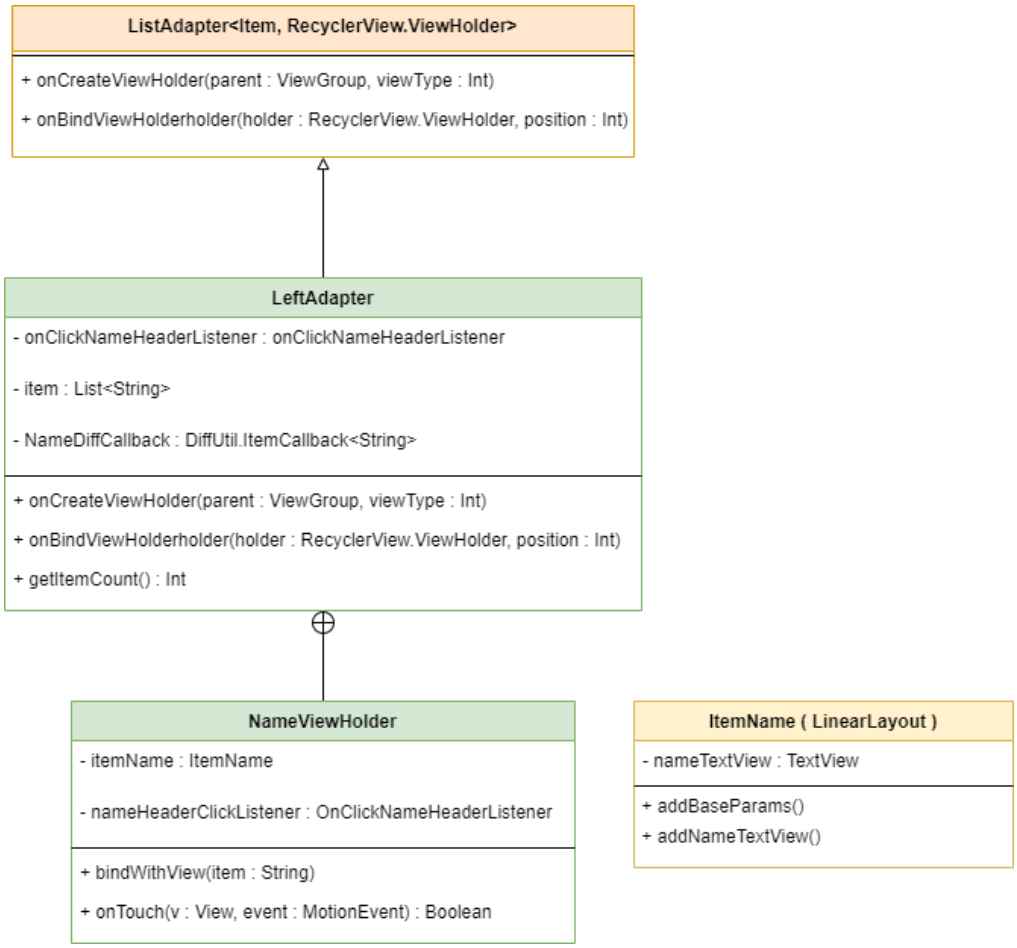

그림 13

4.3 mainRecyclerView

그림 14

5. 결론

프로젝트를 진행함에 있어 개발이 가장 중요하다고 생각했습니다. 학부생 수준에서의 프로젝트는 기획과 요구사항, MD 산정이 전체 프로젝트에서 차지하는 부분이 적었습니다. 하지만 8주간의 인턴십, 그중 약 5주간의 프로젝트 개발 과정을 겪으며 개발이 차지하는 비중이 생각보다 적다고 느꼈습니다. 개발만큼이나 기획이 중요한 부분이며 실력 있는 개발자는 기획적인 부분과 기능 설계 등 개발뿐 아니라 하나의 기능을 개발하기 위한 모든 과정에서 도움을 줄 수 있어야 한다고 느꼈습니다. 이런 관점에서 제가 진행한 과제는 사용자의 입장에서 최대한 편리하게 사용할 수 있도록 기능을 설계하고 라이브러리를 사용하는 개발자의 입장에서 번거로움이 없도록 제작하는 것을 목표로 하였습니다. 하지만 개발 과정에서 많은 이슈가 있었고 기존 계획과 달라진 부분도 많이 존재했습니다. 대부분의 문제를 스스로 해결하려 하였으나 혼자서 해결하기 어려운 부분도 많이 존재했습니다.

프로젝트 개발을 시작하고, 멘토님의 말씀을 듣고 하루의 업무를 시작하기 전, 오늘 해야 할 업무를 오전, 오후로 나누고 미리 계획을 적은 후 개발을 진행하였습니다. 하루 업무가 끝날 때 실제로 개발한 오전, 오후의 업무를 기록하고 계획과 어떤 부분이 다른지, 시간이 더 소요된 부분은 어디인지 기록하고 비교하며 개발을 진행하였습니다. 이러한 습관은 개발을 일정 내에 마무리하고 진행한 업무를 다시 돌아보는 데 큰 도움이 되었습니다.
주니어 개발자인 제가 웍스모바일에서 인턴으로 근무하며 하나의 프로젝트를 설계부터 최종 발표까지 성공적으로 마무리할 수 있었던 이유는 좋은 멘토님과 리더님께서 잘 이끌어주신 덕분이라고 생각합니다. 웍스모바일의 업무를 위한 지원과 복지 및 자유로운 근무환경과 멘토님, 리더님과도 개발에 대한 의견을 자유롭게 주고받을 수 있는 수평적 문화는 개발자로서 성장하는 첫걸음에 있어 가장 좋은 시너지를 낼 수 있었습니다. 제가 작성한 코드에 대한 코드 리뷰, 멘토님과 리더님의 피드백을 받으며 혼자 개발하면서 신경 쓰지 못했던 부분들에 대해 점검할 수 있었습니다.

비록 코로나19로 인해 모든 것이 비대면으로 진행되었지만 앞으로 개발자로서 성장해나가는 과정에서 웍스모바일에서 진행한 인턴십에서 얻은 지식, 그리고 개발에 대한 관점과 생각은 어디에서도 얻을 수 없는 값진 경험이라고 생각합니다. 이런 좋은 기회를 주신 웍스모바일 동료분들께 감사드리고 앞으로 더 실력 있는 개발자가 되기 위해 끊임없이 노력하고 앞으로 나아갈 것입니다.