


”

 **설명:** `viewModel.toggleSelectedDay(it)`


kotlin






복사



편집


```
DailyEditScreen( onDismissRequest = onDismissRequest,
onCompleteRequest = { onCompleteRequest(it) }, uiState = uiState,
selectedDays = uiState.selectedDays ) {
viewModel.toggleSelectedDay(it) } //  ViewModel 메서드 직접 참조 제거
```

-  무슨 의미인가?
 - `DailyEditScreen` 은 `onDaySelected` 콜백을 받는데, `{ viewModel.toggleSelectedDay(it) }` 를 전달하면 요일 선택 시 `ViewModel` 의 상태가 업데이트됨.
 - 즉, 사용자가 요일을 클릭하면 `toggleSelectedDay(day)` 가 실행되어 `uiState.selectedDays` 가 변경됨.
-  왜 `ViewModel` 메서드를 직접 참조하지 않았나?
 - `DailyEditScreen` 내부에서 `ViewModel` 을 직접 참조하면 재사용성이 떨어지고 `Preview` 에서 테스트하기 어려움.
 - 함수(`onDaySelected`)로 분리하면 UI와 로직을 더 유연하게 연결할 수 있음.

 **문제:** 날짜 선택 기능을 `uiState` 에서 처리하고 싶은데, 현재 상태가 원하는 방식이 아님

현재 코드 흐름

1. `DailyEditRoute` 에서 `viewModel.dailyEditUiState` 를 `uiState` 로 사용
2. `DailyEditScreen` 에서 `selectedDays` 를 `uiState.selectedDays` 로 전달
3. `onDaySelected` 로 `ViewModel` 의 `toggleSelectedDay(day)` 를 호출하여 상태 업데이트

 하지만 `ViewModel` 을 통해 상태를 관리하는 것이 아니라, `DailyEditScreen` 내부에서 직접 상태를 관리하고 싶은 거라면?

 `selectedDays` 를 `rememberSaveable` 로 관리하는 방식으로 바꾸면 됨.

✅ 개선 방향

1. **ViewModel**이 아니라 **DailyEditScreen** 내부에서 **selectedDays** 관리
2. **UI**에서 **selectedDays** 를 직접 변경하고, **ViewModel**과 별도로 관리
3. 화면이 종료될 때(**onCompleteRequest**)만 **ViewModel**을 갱신하도록 변경

💡 개선된 코드

kotlin



```

@Composable internal fun DailyEditScreen( onDismissRequest: () ->
Unit, onCompleteRequest: (String, Set<DayOfWeek>) -> Unit, // ✅
selectedDays도 함께 전달 uiState: DailyEditUiState, ) { var daily by
rememberSaveable { mutableStateOf(uiState.dailyExpand.content) } var
selectedDays by rememberSaveable {
mutableStateOf(uiState.selectedDays) } // ✅ 내부에서 관리 var time:
LocalTime? by rememberSaveable { mutableStateOf(null) } var
isTimePickerDialogShow by remember { mutableStateOf(false) } val
daysOfWeek = remember { DayOfWeek.values() } Scaffold( topBar = {
AppBar( title = { Text(stringResource(title_daily)) }, actions = {
Image( painter = painterResource(R.drawable.ic_close_memory_card),
contentDescription = "close", modifier = Modifier .padding(end =
20.dp) .noRippleClickable(onClick = onDismissRequest) ) } ) },
bottomBar = { DefaultButton( onClick = { onCompleteRequest(daily,
selectedDays) }, // ✅ selectedDays도 함께 전달 enabled =
daily.isNotBlank(), modifier = Modifier .fillMaxWidth()
.padding(start = 20.dp, end = 20.dp, bottom = 8.dp) ) {
Text(stringResource(btn_complete_daily)) } } ) {
scaffoldPaddingValues -> Column( modifier = Modifier
.padding(scaffoldPaddingValues) .height(IntrinsicSize.Max)
.background( color = MaterialTheme.colorScheme.background, shape =
RoundedCornerShape(10.dp) ) .padding(top = 24.dp, bottom = 14.dp,
start = 16.dp, end = 16.dp) ) { Text( text =
stringResource(text_per_daily), fontFamily = PretendardFontFamily,
color = BL, fontSize = 18.sp, fontWeight = FontWeight.Medium,
modifier = Modifier.padding(top = 32.dp, bottom = 8.dp) ) Box(
modifier = Modifier.padding(horizontal = 20.dp) ) {
DefaultTextField( value = daily, onValueChange = { if (it.length <=
200) { daily = it } }, singleLine = false, keyboardOptions =
KeyboardOptions(imeAction = ImeAction.None), modifier =
Modifier.height(IntrinsicSize.Max) ) } Text( text =
stringResource(select_week_days), fontFamily = PretendardFontFamily,
color = BL, fontSize = 18.sp, fontWeight = FontWeight.Medium,
modifier = Modifier.padding(top = 32.dp, bottom = 8.dp) ) Row(
horizontalArrangement = Arrangement.SpaceEvenly, modifier = Modifier

```

```
.fillMaxWidth() .height(40.dp) ) { daysOfWeek.forEach { day ->
FilterChip( selected = selectedDays.contains(day), // ✅ 직접 UI에서 상
태 관리 onClick = { selectedDays = selectedDays.toMutableSet().apply {
if (contains(day)) remove(day) else add(day) // ✅ 토글 } }, label =
{ Text(day.getDisplayName(TextStyle.SHORT, Locale.KOREAN)) },
modifier = Modifier.padding(horizontal = 2.dp), colors =
FilterChipDefaults.filterChipColors( selectedContainerColor =
Color.Green ) ) } } } }
```

✅ 변경 내용 및 장점

변경점	기존 방식	개선된 방식
selectedDays 관 리 주체	ViewModel (uiState.selectedDays)	DailyEditScreen 내부 (rememberSaveable)
요일 선택 시 상태 업 데이트	viewModel.toggleSelectedDay(day) 호 출	UI에서 직 접 selectedDays 변경
ViewModel 업데이 트 시점	즉시 반영 (toggleSelectedDay)	완료 버튼 클릭 시 (onCompleteRequest)
UI와 ViewModel 간 결합도	강하게 연결됨	느슨한 연결 (유연한 구조)

✅ 결론: 어떤 방식이 더 좋은가?

- 뷰모델이 UI 상태를 강하게 관리해야 하는 경우
→ 기존 방식(toggleSelectedDay(day)) 유지
- UI에서 상태를 관리하고, 필요할 때만 ViewModel을 갱신하는 경우
→ 개선된 방식(rememberSaveable) 사용

💡 *****"UI에서 사용자가 선택한 데이터를 보관하고, 최종적으로 ViewModel에 저장하는 구조"*****
가 더 적절해 보임.

이제 `selectedDays` 가 UI에서 독립적으로 관리되므로, Preview에서도 정상적으로 동작할 거야!



너는 2번째 방식이 더 좋아보인다는거야?

네, 2번째 방식(`onDaySelected` 를 `DailyEditScreen` 에서 직접 전달하는 방식)이 더 좋은 선택이라고 생각해요. 그 이유를 설명할게요.

🔍 두 가지 접근 방식 비교

1 ViewModel의 메서드를 `DailyEditScreen` 에서 직접 호출하는 방식

```
kotlin
FilterChip( selected = selectedDays.contains(day), onClick = {
viewModel.toggleSelectedDay(day) } // ViewModel 직접 참조 )
```

✅ 장점:

- `DailyEditScreen` 내부에서 상태 변경을 바로 `ViewModel`에 위임할 수 있음.

❌ 단점:

- `ViewModel`을 직접 참조하므로 `DailyEditScreen` 이 `ViewModel`에 강하게 의존함 (테스트하기 어려워짐).
- `DailyEditScreen` 이 재사용성이 떨어짐 (다른 UI에서 사용하려면 `ViewModel`을 변경해야 함).

2 `onDaySelected` 를 `DailyEditScreen` 의 파라미터로 전달하는 방식

```
kotlin
FilterChip( selected = selectedDays.contains(day), onClick = {
onDaySelected(day) } // ViewModel 참조 대신 콜백을 사용 )
```

✅ 장점:

- DailyEditScreen 이 ViewModel을 직접 참조하지 않음 → 재사용성 증가
- DailyEditScreen 을 테스트하기 쉬움 (ViewModel 없이 단독으로 동작 가능)
- 부모 Composable (DailyEditRoute)에서 UI 상태를 더 쉽게 관리할 수 있음
- ✖ 단점:
 - DailyEditRoute 에서 UI 상태 관리를 해야 해서 약간의 추가 코드가 필요할 수도 있음

🔧 UI 상태(uiState)에서 관리하는 게 더 나을까?

너가 고민한 것처럼 ****날짜 선택 기능을 uiState 에서 관리하는 게 좋은가?***에 대해서도 살펴볼게.

사실 selectedDays 를 uiState 에서 관리하는 것은 충분히 좋은 접근 방식이야. 하지만 지금 방식도 충분히 괜찮아!

🔪 현재 방식 (onDaySelected 사용)과 UI 상태에서 관리하는 방식 비교

방식	장점	단점
onDaySelected 를 사용	DailyEditScreen 이 ViewModel에 의존하지 않음, 재사용성 증가	DailyEditRoute 에서 selectedDays 상태를 관리해야 함
UI 상태 (uiState.selectedDays)에서 관리	모든 상태를 ViewModel이 관리하여 일관성 유지	DailyEditScreen 이 ViewModel에 의존하게 됨

👉 내 추천:

지금 방식(2번째 방식)은 재사용성이 높고, 테스트하기 쉬우며, UI와 로직을 분리하는 점에서 더 좋음!

하지만 만약 상태 관리가 복잡해지고 여러 UI에서 공유해야 한다면 uiState 에서 관리하는 것도 좋은 선택이 될 수 있어.

🔧 개선할 방향

- 1 현재 DailyEditScreen 이 uiState 를 너무 많이 직접 사용하고 있음

- `uiState.dailyExpand.content` 를 `rememberSaveable` 로 감싸면서 직접 `daily` 를 관리하는데, UI 상태를 `ViewModel`에서만 관리하도록 하면 더 일관성이 좋아짐.
- 즉, `daily` 상태를 `DailyEditScreen` 내부에서 변경하는 게 아니라 `ViewModel` 을 통해 변경하는 방식이 더 좋을 수도 있음.

2 시간 선택 (time 상태)도 `ViewModel` 에서 관리하는 게 더 좋을 수도 있음

- `DailyEditScreen` 내에서 `time` 을 `rememberSaveable` 로 관리하는데, `ViewModel` 이 관리하면 회전(rotations) 등에서 상태 유지가 더 쉬워짐.

📌 최종 결론

- 날짜 선택 기능 (`onDaySelected`)을 `ViewModel`이 아니라 `DailyEditRoute` 에서 관리하는 현재 방식이 더 좋음
- 다만, `daily` 와 `time` 상태를 `ViewModel`에서 관리하도록 하면 개선 가능성이 있음

너의 프로젝트 구조나 방향에 따라 더 맞는 방식을 선택하면 될 것 같아! 😊

