Anàlisi i millora de l'estructura en Kotlin amb Jetpack Compose

Anàlisi del codi

L'estructura proporcionada utilitza Jetpack Compose per crear una interfície d'usuari en Android amb Kotlin. El codi està ben modularitzat, amb components `Composable` com `SettingsContent`, `NotificationSwitch`, `ParametritzacioNotificacio`, i altres, que estructuren la interfície amb `Column` i `Row`. L'ús de `Modifier` és adequat en la majoria dels casos, i la gestió de l'estat amb `remember { mutableStateOf() }` és correcta per mantenir la reactivitat.

Problemes potencials

- **Objecte `Configuracio`:** Es crea un objecte `Configuracio(0f, 0f, 0)` a `ParametritzacioNotificacio`, però no s'utilitza. Això pot ser un error o codi incomplet.
- **Validació d'entrada:** Els `TextField` permeten qualsevol entrada, però sembla que s'esperen valors numèrics (p. ex., quilòmetres). No hi ha validació per assegurar entrades vàlides.
- **Botó `AfegirBtn`:** El `TextButton` té un `onClick` buit, cosa que indica que la funcionalitat no està implementada.
- **Modificadors buits:** Alguns `Modifier` són passats com a paràmetres però no s'utilitzen, cosa que pot ser confusa.
- **Espaiat i estil:** L'ús de `Spacer` no és consistent, i l'aspecte visual podria millorar-se amb més espaiat i estil.

Funcionalitat

El codi mostra una interfície amb un `Switch` que activa/desactiva camps de text i un botó. Tanmateix, no hi ha lògica per processar o emmagatzemar els valors dels `TextField`, com guardar-los en una base de dades o utilitzar-los per a notificacions.

Suggeriments de millora

Validació d'entrada als `TextField`

Per assegurar que els camps `kmsInicial`, `kmsAvis`, i `limitNotifications` acceptin només números, es pot utilitzar `keyboardType = KeyboardType.Number` i validar les entrades a `onValueChange`.

```
@Composable
fun KmsInicials(kmsInicial: MutableState<TextFieldValue>, modifier: Modifi
    Row(
        verticalAlignment = Alignment.CenterVertically,
        modifier = modifier.fillMaxWidth()
    ) {
        TextField(
            value = kmsInicial.value,
            onValueChange = { newValue ->
                if (newValue.text.isEmpty() || newValue.text.toFloatOrNull
                    kmsInicial.value = newValue
                }
            },
            label = { Text(text = stringResource(id = R.string.settings_km
            keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.
            modifier = Modifier.fillMaxWidth()
        )
    }
}
```

Optimització de l'objecte `Configuracio`

L'objecte `Configuracio` hauria d'actualitzar-se amb els valors dels `TextField`. Es pot definir com una `data class` i utilitzar els valors introduïts.

```
data class Configuracio(
    val limitNotifications: Float,
    val kmsInicial: Float,
    val kmsAvis: Int
)

@Composable
fun ParametritzacioNotificacio(
    limitNotifications: MutableState<TextFieldValue>,
    kmsInicial: MutableState<TextFieldValue>,
    kmsAvis: MutableState<TextFieldValue>,
    modifier: Modifier = Modifier
) {
    val configuracio = Configuracio(
        limitNotifications = limitNotifications.value.text.toFloateHTy

% Continuant el document LaTeX
FloatOrNull() ?: 0f,
        kmsInicial = kmsInicial.value.text.toFloatOrNull() ?: 0f,
        kmsAvis = kmsAvis.value.text.toIntOrNull() ?: 0
    )
    Column(modifier = modifier.fillMaxWidth()) {
        KmsInicials(kmsInicial)
        Spacer(modifier = Modifier.height(8.dp))
        KmsAvis(kmsAvis)
        Spacer(modifier = Modifier.height(8.dp))
        LimitNotificacions(limitNotifications)
    }
}
```

Implementació del botó `AfegirBtn`

El botó `AfegirBtn` hauria de tenir una acció significativa, com guardar els valors de `Configuracio`.

```
@Composable
fun AfegirBtn(onClick: () -> Unit, modifier: Modifier = Modifier) {
    Row(
        modifier = modifier.fillMaxWidth(),
        horizontalArrangement = Arrangement.End
    ) {
        TextButton(
            onClick = onClick,
            enabled = true
        ) {
            Text(
                text = stringResource(id = R.string.settings_add_btn_inter
                style = MaterialTheme.typography.labelSmall
            )
        }
    }
}
```

Gestió de l'estat amb ViewModel

Es recomana utilitzar un `ViewModel` per gestionar l'estat de manera centralitzada, especialment si es necessita persistir les dades.

```
class SettingsViewModel : ViewModel() {
    var notificationsEnabled by mutableStateOf(false)
    var limitNotifications by mutableStateOf(TextFieldValue(""))
    var kmsInicial by mutableStateOf(TextFieldValue(""))
    var kmsAvis by mutableStateOf(TextFieldValue(""))

    fun updateConfiguracio(limit: String, inicial: String, avis: String) {
        limitNotifications = TextFieldValue(limit)
        kmsInicial = TextFieldValue(inicial)
        kmsAvis = TextFieldValue(avis)
    }
}
```

Millora de l'espaiat i estil

S'ha d'aplicar un espaiat consistent amb `Spacer` i utilitzar `modifier.fillMaxWidth()` als `TextField` per millorar l'aspecte visual.

Accessibilitat

Afegir `contentDescription` als elements interactius com el `Switch` per millorar l'accessibilitat.

Codi complet millorat

A continuació, es presenta el codi complet amb les millores aplicades:

```
@Composable
fun SettingsContent(viewModel: SettingsViewModel = viewModel()) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        NotificationSwitch(
            labelId = R.string.settings_enable_notifications,
            notificationsEnabled = viewModel.notificationsEnabled
        )
```

```
        Spacer(modifier = Modifier.height(8.dp))

        if (viewModel.notificationsEnabled.value) {
            ParametritzacioNotificacio(
                limitNotifications = viewModel.limitNotifications,
                kmsInicial = viewModel.kmsInicial,
                kmsAvis = viewModel.kmsAvis
            )
            AfegirBtn(
                onClick = {
                    viewModel.updateConfiguracio(
                        viewModel.limitNotifications.value.text,
                        viewModel.kmsInicial.value.text,
                        viewModel.kmsAvis.value.text
                    )
                }
            )
            Spacer(modifier = Modifier.height(8.dp))
        }
    }
}

@Composable
fun NotificationSwitch(
    labelId: Int,
    notificationsEnabled: MutableState<Boolean>,
    modifier: Modifier = Modifier
) {
    Row(
        verticalAlignment = Alignment.CenterVertically,
        modifier = modifier.fillMaxWidth()
    ) {
        Text(
            text = stringResource(id = labelId),
            style = MaterialTheme.typography.labelSmall
        )
        Spacer(modifier = Modifier.weight(1f))
        Switch(
            checked = notificationsEnabled.value,
            onCheckedChange = { notificationsEnabled.value = it },
            modifier = Modifier.padding(horizontal = 8.dp),
            contentDescription = stringResource(id = labelId)
        )
    }
}

@Composable
fun ParametritzacioNotificacio(
    limitNotifications: MutableState<TextFieldValue>,
    kmsInicial: MutableState<TextFieldValue>,
    kmsAvis: MutableState<TextFieldValue>,
    modifier: Modifier = Modifier
) {
    Column(modifier = modifier.fillMaxWidth()) {
        KmsInicials(kmsInicial)
        Spacer(modifier = Modifier.height(8.dp))
        KmsAvis(kmsAvis)
        Spacer(modifier = Modifier.height(8.dp))
        LimitNotificacions(limitNotifications)
    }
}

@Composable
fun LimitNotificacions(limitNotifications: MutableState<TextFieldkleValue>
    Row(
        verticalAlignment = Alignment.CenterVertically,
        modifier = modifier.fillMaxWidth()
    ) {
        TextField(
            value = limitNotifications.value,
```

```
            onValueChange = { newValue ->
                if (newValue.text.isEmpty() || newValue.text.toFloatOrNull
                    limitNotifications.value = newValue
                }
            },
            label = { Text(text = stringResource(id = R.string.settings_li
            keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.
            modifier = Modifier.fillMaxWidth()
        )
    }
}

@Composable
fun KmsAvis(kmsAvis: MutableState<TextFieldValue>, modifier: Modifier = Mo
    Row(
        verticalAlignment = Alignment.CenterVertically,
        modifier = modifier.fillMaxWidth()
    ) {
        TextField(
            value = kmsAvis.value,
            onValueChange = { newValue ->
                if (newValue.text.isEmpty() || newValue.text.toIntOrNull()
                    kmsAvis.value = newValue
                    Log.d("DEBUG", "Quilòmetres avís: ${kmsAvis.value.text
                }
            },
            label = { Text(text = stringResource(id = R.string.settings_km
            keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.
            modifier = Modifier.fillMaxWidth()
        )
    }
}

@Composable
fun KmsInicials(kmsInicial: MutableState<TextFieldValue>, modifier: Modifi
    Row(
        verticalAlignment = Alignment.CenterVertically,
        modifier = modifier.fillMaxWidth()
    ) {
        TextField(
            value = kmsInicial.value,
            onValueChange = { newValue ->
                if (newValue.text.isEmpty() || newValue.text.toFloatOrNull
                    kmsInicial.value = newValue
                }
            },
            label = { Text(text = stringResource(id = R.string.settings_km
            keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.
            modifier = Modifier.fillMaxWidth()
        )
    }
}

@Composable
fun AfegirBtn(onClick: () -> Unit, modifier: Modifier = Modifier) {
    Row(
        modifier = modifier.fillMaxWidth(),
        horizontalArrangement = Arrangement.End
    ) {
        TextButton(
            onClick = onClick,
            enabled = true
        ) {
            Text(
                text = stringResource(id = R.string.settings_add_btn_inter
                style = MaterialTheme.typography.labelSmall
            )
        }
    }
}
```

```
}
```

Conclusió

L'estructura original és vàlida i funcional, però es poden aplicar millores per fer-la més robusta i usable:

1. Afegir validació d'entrada als `TextField` per acceptar només números.
2. Implementar la lògica del botó `AfegirBtn` per processar els valors.
3. Utilitzar un `ViewModel` per gestionar l'estat de manera centralitzada.
4. Millorar l'espaiat i l'estil per a una millor experiència d'usuari.
5. Revisar l'ús de l'objecte `Configuracio` per assegurar-ne la funcionalitat.