Jetpack Compose + ViewModel Agrupar camps de text en un model reactiu

Objectiu
Encapsular lestat dels camps de configuració dins un `ViewModel` i fer que la UI lobservi.

1. Crear el ViewModel

```kotlin
import androidx.lifecycle.ViewModel
import androidx.compose.ui.text.input.TextFieldValue
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow

data class NotificacioConfiguracio(
    val limitNotifications: TextFieldValue = TextFieldValue(),
    val kmsInicial: TextFieldValue = TextFieldValue(),
    val kmsAvis: TextFieldValue = TextFieldValue()
)

class SettingsViewModel : ViewModel() {

    private val _notificacioState = MutableStateFlow(NotificacioConfiguracio())
    val notificacioState: StateFlow<NotificacioConfiguracio> = _notificacioState

    fun onLimitNotificationsChange(value: TextFieldValue) {
        if (value.text.toIntOrNull() != null) {
             _notificacioState.value = _notificacioState.value.copy(limitNotifications =
value)
        }
    }

    fun onKmsInicialChange(value: TextFieldValue) {
        if (value.text.toFloatOrNull() != null) {
            _notificacioState.value = _notificacioState.value.copy(kmsInicial = value)
        }
    }

    fun onKmsAvisChange(value: TextFieldValue) {
        if (value.text.toFloatOrNull() != null) {
            _notificacioState.value = _notificacioState.value.copy(kmsAvis = value)
        }
    }
}
```

2. Proporcionar el ViewModel a la UI

```kotlin
@Composable
fun SettingsContent(
    modifier: Modifier = Modifier,
    viewModel: SettingsViewModel = viewModel()
) {
```

```
    val notificationsEnabled = remember { mutableStateOf(false) }

    Column(
        modifier = modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        NotificationSwitch(R.string.settings_enable_notifications, notificationsEnabled)

        if (notificationsEnabled.value) {
            val state by viewModel.notificacioState.collectAsState()

            ParametritzacioNotificacio(state, viewModel)
            AfegirBtn(state)
        }
    }
}
```

3. Refactoritzar els composables fills

ParametritzacioNotificacio

```kotlin
@Composable
fun ParametritzacioNotificacio(
    state: NotificacioConfiguracio,
    viewModel: SettingsViewModel,
    modifier: Modifier = Modifier
) {
    KmsInicials(state.kmsInicial, viewModel::onKmsInicialChange)
    KmsAvis(state.kmsAvis, viewModel::onKmsAvisChange)
    LimitNotificacions(state.limitNotifications, viewModel::onLimitNotificationsChange)
}
```

Exemple: LimitNotificacions

```kotlin
@Composable
fun LimitNotificacions(
    value: TextFieldValue,
    onValueChange: (TextFieldValue) -> Unit,
    modifier: Modifier = Modifier
) {
    TextField(
        value = value,
        onValueChange = onValueChange,
        label = {
            Text(text = stringResource(id = R.string.settings_limit_notificacions))
        },
        modifier = modifier
    )
}
```

AfegirBtn

```kotlin
@Composable
fun AfegirBtn(state: NotificacioConfiguracio, modifier: Modifier = Modifier) {
    TextButton(
        onClick = {
              Log.d("Info", "${state.limitNotifications.text}, ${state.kmsInicial.text},
${state.kmsAvis.text}")
        },
        content = {
            Text(
                text = stringResource(id = R.string.settings_add_btn_interval),
                style = MaterialTheme.typography.labelSmall
            )
        }
    )
}
```

Beneficis

- Separació de responsabilitats (UI estat lògica)
- Model observable via `StateFlow`
- Facilitat per compartir lestat entre pantalles
- Preparació per testejar la lògica del ViewModel

Propers passos suggerits

- Llistes dintervals dinàmics
- Validació i errors visuals
- Persistència amb DataStore