Week 3 Assignment

Note App using Roomdb

Dyutin R 20BCG10060

Main Activity

```
package com.project.noteapp.feature_note.presentation

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.ExperimentalAnimationApi
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.tooling.preview.Preview
import androidx.navigation.NavType
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import androidx.navigation.navArgument
import
com.project.noteapp.feature_note.presentation.add_edit_note.AddEditNoteScre
en
import com.project.noteapp.feature_note.presentation.notes.NotesScreen
import com.project.noteapp.feature_note.presentation.util.Screen
import com.project.noteapp.ui.theme.NoteAppTheme
import dagger.hilt.android.AndroidEntryPoint

@AndroidEntryPoint
class MainActivity : ComponentActivity() {
    @ExperimentalAnimationApi
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            NoteAppTheme {
                Surface(
                    color = MaterialTheme.colors.background
                ) {
                    val navController = rememberNavController()
                    NavHost(
                        navController = navController,
                        startDestination = Screen.NotesScreen.route
                    ) {
                        composable(route = Screen.NotesScreen.route) {
                            NotesScreen(navController = navController)
                        }
                        composable(
                            route = Screen.AddEditNoteScreen.route +

"?noteId={noteId}&noteColor={noteColor}",
                            arguments = listOf(
                                navArgument(
                                    name = "noteId"
                                ) {
                                    type = NavType.IntType
                                    defaultValue = -1
                                },
                                navArgument(
                                    name = "noteColor"
                                ) {
                                    type = NavType.IntType
                                    defaultValue = -1
                                },
                            )
                        ) {
                            val color = it.arguments?.getInt("noteColor")
?: -1
                            AddEditNoteScreen(
                                navController = navController,
                                noteColor = color
                            )
                        }
                    }
                }
            }
        }
    }
}
```

AddEditNoteViewModel

```
package com.project.noteapp.feature_note.presentation.add_edit_note

import androidx.compose.runtime.State
import androidx.compose.runtime.mutableStateOf
import androidx.compose.ui.graphics.toArgb
import androidx.lifecycle.SavedStateHandle
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.project.noteapp.feature_note.domain.model.InvalidNoteException
import com.project.noteapp.feature_note.domain.model.Note
import com.project.noteapp.feature_note.domain.use_case.NoteUseCases
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.flow.asSharedFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class AddEditNoteViewModel @Inject constructor(
    private val noteUseCases: NoteUseCases,
    savedStateHandle: SavedStateHandle
) : ViewModel() {

    private val _noteTitle = mutableStateOf(
        NoteTextFieldState(
        hint = "Enter title..."
    )
    )
    val noteTitle: State<NoteTextFieldState> = _noteTitle
    private val _noteContent = mutableStateOf(
        NoteTextFieldState(
        hint = "Enter description"
    )
    )
    val noteContent: State<NoteTextFieldState> = _noteContent

    private val _noteColor =
mutableStateOf(Note.noteColors.random().toArgb())
    val noteColor: State<Int> = _noteColor

    private val _eventFlow = MutableSharedFlow<UiEvent>()
    val eventFlow = _eventFlow.asSharedFlow()

    private var currentNoteId: Int? = null

    init {
        savedStateHandle.get<Int>("noteId")?.let { noteId ->
            if(noteId != -1) {
                viewModelScope.launch {
                    noteUseCases.getNote(noteId)?.also { note ->
                        currentNoteId = note.id
                        _noteTitle.value = noteTitle.value.copy(
                            text = note.title,
                            isHintVisible = false
                        )
                        _noteContent.value = _noteContent.value.copy(
                            text = note.content,
                            isHintVisible = false
                        )
                        _noteColor.value = note.color
                    }
                }
            }
        }
    }

    fun onEvent(event: AddEditNoteEvent) {
        when(event) {
            is AddEditNoteEvent.EnteredTitle -> {
                _noteTitle.value = noteTitle.value.copy(
                    text = event.value
                )
            }
            is AddEditNoteEvent.ChangeTitleFocus -> {
                _noteTitle.value = noteTitle.value.copy(
                    isHintVisible = !event.focusState.isFocused &&
                            noteTitle.value.text.isBlank()
                )
            }
            is AddEditNoteEvent.EnteredContent -> {
                _noteContent.value = _noteContent.value.copy(
                    text = event.value
                )
            }
            is AddEditNoteEvent.ChangeContentFocus -> {
                _noteContent.value = _noteContent.value.copy(
                    isHintVisible = !event.focusState.isFocused &&
                            _noteContent.value.text.isBlank()
                )
            }
            is AddEditNoteEvent.ChangeColor -> {
                _noteColor.value = event.color
            }
            is AddEditNoteEvent.SaveNote -> {
                viewModelScope.launch {
                    try {
                        noteUseCases.addNote(
                            Note(
                                title = noteTitle.value.text,
                                content = noteContent.value.text,
                                timestamp = System.currentTimeMillis(),
                                color = noteColor.value,
                                id = currentNoteId
                            )
                        )
                        _eventFlow.emit(UiEvent.SaveNote)
                    } catch(e: InvalidNoteException) {
                        _eventFlow.emit(
                            UiEvent.ShowSnackbar(
                                message = e.message ?: "Couldn't save note"
                            )
                        )
                    }
                }
            }
        }
    }

    sealed class UiEvent {
        data class ShowSnackbar(val message: String): UiEvent()
        object SaveNote: UiEvent()
    }
}
```

AddEditNoteScreen

```
package com.project.noteapp.feature_note.presentation.add_edit_note

import androidx.compose.animation.Animatable
import androidx.compose.animation.core.tween
import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Save
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.draw.shadow
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.toArgb
```

```kotlin
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavController
import com.project.noteapp.feature_note.domain.model.Note
import
com.project.noteapp.feature_note.presentation.add_edit_note.components.Tran
sparentHintTextField
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.launch

@Composable
fun AddEditNoteScreen(
    navController: NavController,
    noteColor: Int,
    viewModel: AddEditNoteViewModel = hiltViewModel()
) {
    val titleState = viewModel.noteTitle.value
    val contentState = viewModel.noteContent.value

    val scaffoldState = rememberScaffoldState()

    val noteBackgroundAnimatable = remember {
        Animatable(
            Color(if (noteColor != -1) noteColor else
viewModel.noteColor.value)
        )
    }
    val scope = rememberCoroutineScope()

    LaunchedEffect(key1 = true) {
        viewModel.eventFlow.collectLatest { event ->
            when(event) {
                is AddEditNoteViewModel.UiEvent.ShowSnackbar -> {
                    scaffoldState.snackbarHostState.showSnackbar(
                        message = event.message
                    )
                }
                is AddEditNoteViewModel.UiEvent.SaveNote -> {
                    navController.navigateUp()
                }
            }
        }
    }

    Scaffold(
        floatingActionButton = {
            FloatingActionButton(
                onClick = {
                    viewModel.onEvent(AddEditNoteEvent.SaveNote)
                },
                backgroundColor = MaterialTheme.colors.primary
            ) {
                Icon(imageVector = Icons.Default.Save, contentDescription =
"Save note")
            }
        },
        scaffoldState = scaffoldState
    ) {
        Column(
            modifier = Modifier
                .fillMaxSize()
                .background(noteBackgroundAnimatable.value)
                .padding(16.dp)
        ) {
            Row(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(8.dp),
                horizontalArrangement = Arrangement.SpaceBetween
            ) {
                Note.noteColors.forEach { color ->
                    val colorInt = color.toArgb()
                    Box(
                        modifier = Modifier
                            .size(50.dp)
                            .shadow(15.dp, CircleShape)
                            .clip(CircleShape)
                            .background(color)
                            .border(
                                width = 3.dp,
                                color = if (viewModel.noteColor.value ==
colorInt) {
                                    Color.Black
                                } else Color.Transparent,
                                shape = CircleShape
                            )
                            .clickable {
                                scope.launch {
                                    noteBackgroundAnimatable.animateTo(
                                        targetValue = Color(colorInt),
                                        animationSpec = tween(
                                            durationMillis = 500
                                        )
                                    )
                                }

viewModel.onEvent(AddEditNoteEvent.ChangeColor(colorInt))
                            }
                    )
                }
            }
            Spacer(modifier = Modifier.height(16.dp))
            TransparentHintTextField(
                text = titleState.text,
                hint = titleState.hint,
                onValueChange = {
                    viewModel.onEvent(AddEditNoteEvent.EnteredTitle(it))
                },
                onFocusChange = {

viewModel.onEvent(AddEditNoteEvent.ChangeTitleFocus(it))
                },
                isHintVisible = titleState.isHintVisible,
                singleLine = true,
                textStyle = TextStyle(fontSize = 32.sp, fontWeight =
FontWeight.Bold, )
            )
            Spacer(modifier = Modifier.height(16.dp))
            TransparentHintTextField(
                text = contentState.text,
                hint = contentState.hint,
                onValueChange = {
                    viewModel.onEvent(AddEditNoteEvent.EnteredContent(it))
                },
                onFocusChange = {

viewModel.onEvent(AddEditNoteEvent.ChangeContentFocus(it))
                },
                isHintVisible = contentState.isHintVisible,
                textStyle = MaterialTheme.typography.body1,
                modifier = Modifier.fillMaxHeight()
            )
        }
    }
}
```

GetNotes

```
package com.project.noteapp.feature_note.domain.use_case

import com.project.noteapp.feature_note.domain.model.Note
import com.project.noteapp.feature_note.domain.repository.NoteRepository
import com.project.noteapp.feature_note.domain.util.NoteOrder
import com.project.noteapp.feature_note.domain.util.OrderType
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.map

class GetNotes(
    private val repository: NoteRepository
) {

    operator fun invoke(
        noteOrder: NoteOrder = NoteOrder.Date(OrderType.Descending)
    ): Flow<List<Note>> {
        return repository.getNotes().map { notes ->
            when(noteOrder.orderType) {
                is OrderType.Ascending -> {
                    when(noteOrder) {
                        is NoteOrder.Title -> notes.sortedBy {
it.title.lowercase() }
                        is NoteOrder.Date -> notes.sortedBy { it.timestamp
}
                        is NoteOrder.Color -> notes.sortedBy { it.color }
                    }
                }
                is OrderType.Descending -> {
                    when(noteOrder) {
                        is NoteOrder.Title -> notes.sortedByDescending {
it.title.lowercase() }
                        is NoteOrder.Date -> notes.sortedByDescending {
it.timestamp }
                        is NoteOrder.Color -> notes.sortedByDescending {
it.color }
                    }
                }
            }
        }
    }
}
```

Add Note

```
package com.project.noteapp.feature_note.domain.use_case

import com.project.noteapp.feature_note.domain.model.InvalidNoteException
import com.project.noteapp.feature_note.domain.model.Note
import com.project.noteapp.feature_note.domain.repository.NoteRepository

class AddNote(
    private val repository: NoteRepository
) {

    @Throws(InvalidNoteException::class)
    suspend operator fun invoke(note: Note) {
        if(note.title.isBlank()) {
            throw InvalidNoteException("The title of the note can't be
empty.")
        }
        if(note.content.isBlank()) {
            throw InvalidNoteException("The content of the note can't be
empty.")
        }
        repository.insertNote(note)
    }
}
```

DeleteNote

```
package com.project.noteapp.feature_note.domain.use_case

import com.project.noteapp.feature_note.domain.model.Note
import com.project.noteapp.feature_note.domain.repository.NoteRepository

class DeleteNote(
    private val repository: NoteRepository
) {

    suspend operator fun invoke(note: Note) {
        repository.deleteNote(note)
    }
}
```

NoteDao

```
package com.project.noteapp.feature_note.data.data_source

import androidx.room.Dao
import androidx.room.Delete
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import com.project.noteapp.feature_note.domain.model.Note
import kotlinx.coroutines.flow.Flow
```

```
@Dao
interface NoteDao {
    @Query("SELECT * FROM note")
    fun getNotes(): Flow<List<Note>>

    @Query("SELECT * FROM note Where id = :id")
    suspend fun  getNoteById(id: Int): Note?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun  insertNote(note: Note)

    @Delete
    suspend fun deleteNote(note: Note)
}
```

Note

```
package com.project.noteapp.feature_note.domain.model

import androidx.room.Entity
import androidx.room.PrimaryKey
import com.project.noteapp.ui.theme.BabyBlue
import com.project.noteapp.ui.theme.LightGreen
import com.project.noteapp.ui.theme.RedOrange
import com.project.noteapp.ui.theme.RedPink
import com.project.noteapp.ui.theme.Violet

@Entity
data class Note(
    val title: String,
    val content: String,
    val timestamp: Long,
    val color: Int,
    @PrimaryKey val id: Int? = null
){
    companion object{
        val noteColors = listOf(RedOrange, LightGreen, Violet, BabyBlue,
RedPink)

    }
}
class InvalidNoteException(message: String): Exception(message)
```

NoteOrder

```
package com.project.noteapp.feature_note.domain.util

sealed class NoteOrder(val orderType: OrderType) {
    class Title(orderType: OrderType): NoteOrder(orderType)
    class Date(orderType: OrderType): NoteOrder(orderType)
    class Color(orderType: OrderType): NoteOrder(orderType)

    fun copy(orderType: OrderType): NoteOrder {
        return when(this) {
            is Title -> Title(orderType)
            is Date -> Date(orderType)
            is Color -> Color(orderType)
        }
    }
}
```

NoteItem

```
package com.project.noteapp.feature_note.presentation.notes.components

import androidx.compose.foundation.Canvas
import androidx.compose.foundation.layout.*
import androidx.compose.material.Icon
import androidx.compose.material.IconButton
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Text
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Delete
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.geometry.CornerRadius
import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.geometry.Size
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.Path
import androidx.compose.ui.graphics.drawscope.clipPath
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.Dp
import androidx.compose.ui.unit.dp
import androidx.core.graphics.ColorUtils
import com.project.noteapp.feature_note.domain.model.Note

@Composable
fun NoteItem(
    note: Note,
    modifier: Modifier = Modifier,
    cornerRadius: Dp = 10.dp,
    cutCornerSize: Dp = 30.dp,
    onDeleteClick: () -> Unit
) {
    Box(
        modifier = modifier
    ) {
        Canvas(modifier = Modifier.matchParentSize()) {
            drawRoundRect(
                color = Color(note.color),
                size = size,
                cornerRadius = CornerRadius(cornerRadius.toPx())
            )
        }
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(16.dp)
                .padding(end = 32.dp)
        ) {
            Text(
                text = note.title,
                style = MaterialTheme.typography.h6,
                color = MaterialTheme.colors.onSurface,
```

```kotlin
                maxLines = 1,
                overflow = TextOverflow.Ellipsis,
                fontWeight = FontWeight.Bold
            )
            Spacer(modifier = Modifier.height(8.dp))
            Text(
                text = note.content,
                style = MaterialTheme.typography.body1,
                color = MaterialTheme.colors.onSurface,
                maxLines = 10,
                overflow = TextOverflow.Ellipsis
            )
        }
        IconButton(
            onClick = onDeleteClick,
            modifier = Modifier.align(Alignment.BottomEnd)
        ) {
            Icon(
                imageVector = Icons.Default.Delete,
                contentDescription = "Delete note",
                tint = MaterialTheme.colors.onSurface
            )
        }
    }
}
```