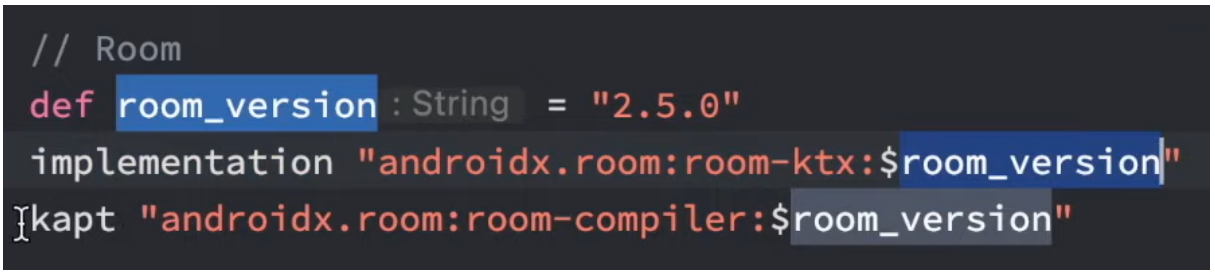
Room

https://www.youtube.com/watch?v=bOd3wO0uFr8&ab_channel=PhilippLackner

Room setup

Add the Room dependencies:

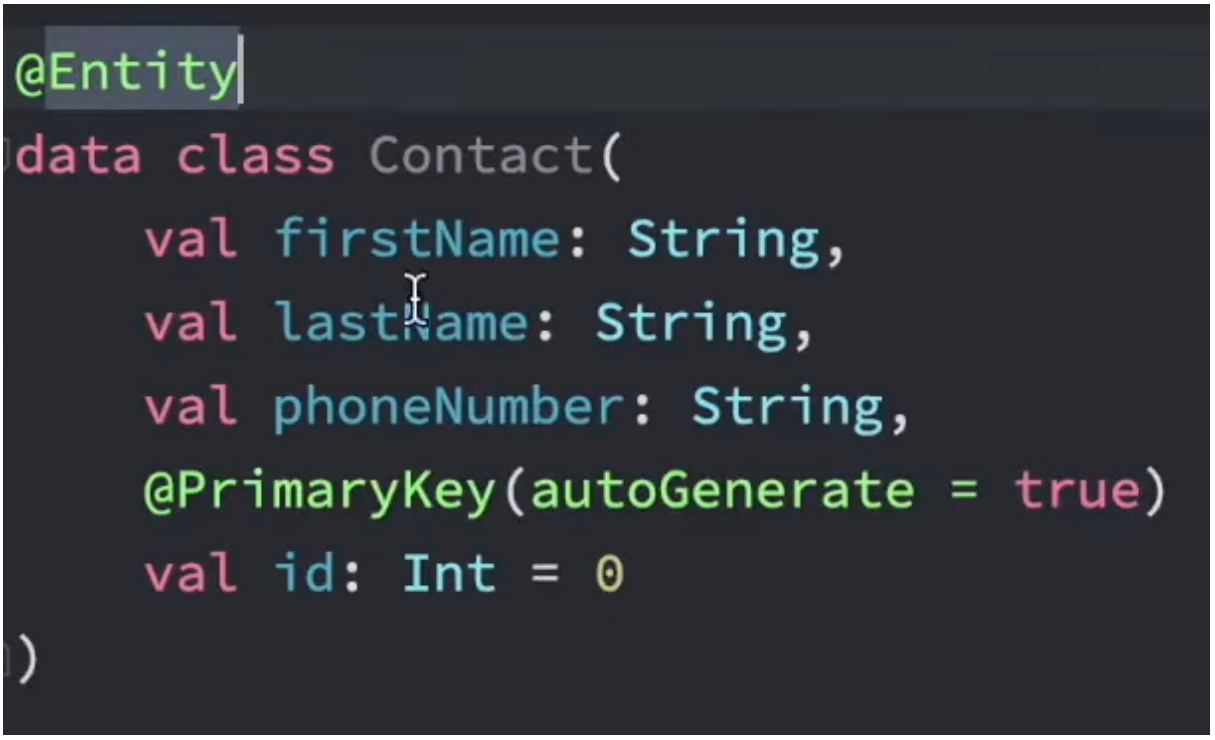


kapt stands for kotlin annotation processing tool.

Also you need to add the “id ‘kotlin-kapt” in your plugins.

In Android Studio → run your app → App inspection → select a process .roomguide → this way you can check the tables of Room in Android Studio.

Start by creating a class:



It has to have the @Entity annotation and a primary key.

Now create the DAO (Data Access Object). It contains the functions to access the data.

```
@Dao
interface ContactDao {

    @Upsert
    suspend fun upsertContact(contact: Contact)

    @Delete
    suspend fun deleteContact(contact: Contact)

    @Query("SELECT * FROM contact ORDER BY firstName ASC")
    fun getContactsOrderedByFirstName(): Flow<List<Contact>>

    @Query("SELECT * FROM contact ORDER BY lastName ASC")
    fun getContactsOrderedByLastName(): Flow<List<Contact>>

    @Query("SELECT * FROM contact ORDER BY phoneNumber ASC")
    fun getContactsOrderedByPhoneNumber(): Flow<List<Contact>>
}
```

@Upsert is a mix between insert and update. It will create a new object if it doesn't exist and it will update it if it exists. You can also use @Insert(onConflict = ...)

The query functions could return a List but we return a Flow to listen to changes on that list.

Last we need to define the Database.

```
@Database(
    entities = [Contact::class],
    version = 1
)
abstract class ContactDatabase: RoomDatabase() {

    abstract val dao: ContactDao
}
```

version is used to let Room know what to do with a new version of the Database.

Use of the Database functions

SortType enum class:

```
enum class SortType {
    FIRST_NAME,
    LAST_NAME,
    PHONE_NUMBER
}
```

Events of the app:

```
sealed interface ContactEvent {
    object SaveContact: ContactEvent
    data class SetFirstName(val firstName: String): ContactEvent
    data class SetLastName(val lastName: String): ContactEvent
    data class SetPhoneNumber(val phoneNumber: String): ContactEvent
    object ShowDialog: ContactEvent
    object HideDialog: ContactEvent
    data class SortContacts(val sortType: SortType): ContactEvent
    data class DeleteContact(val contact: Contact): ContactEvent
}
```

Current state of the app to know what to show and hide:

```
data class ContactState(
    val contacts: List<Contact> = emptyList(),
    val firstName: String = "",
    val lastName: String = "",
    val phoneNumber: String = "",
    val isAddingContact: Boolean = false,
    val sortType: SortType = SortType.FIRST_NAME
)
```

Now create the ViewModel in which we will map all of our states:

```
package com.plcoding.roomguideandroid

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import kotlinx.coroutines.ExperimentalCoroutinesApi
import kotlinx.coroutines.flow.*
import kotlinx.coroutines.launch
```

```
@OptIn(ExperimentalCoroutinesApi::class)
class ContactViewModel(
    private val dao: ContactDao
): ViewModel() {

    private val _sortType = MutableStateFlow(SortType.FIRST_NA
    private val _contacts = _sortType
        .flatMapLatest { sortType ->
            when(sortType) {
                SortType.FIRST_NAME -> dao.getContactsOrdered
                SortType.LAST_NAME -> dao.getContactsOrderedB
                SortType.PHONE_NUMBER -> dao.getContactsOrder
            }
        }
        .stateIn(viewModelScope, SharingStarted.WhileSubscrib

    private val _state = MutableStateFlow(ContactState())
    val state = combine(_state, _sortType, _contacts) { state
        state.copy(
            contacts = contacts,
            sortType = sortType
        )
    }.stateIn(viewModelScope, SharingStarted.WhileSubscribed(

    fun onEvent(event: ContactEvent) {
        when(event) {
            is ContactEvent.DeleteContact -> {
                viewModelScope.launch {
                    dao.deleteContact(event.contact)
                }
            }
            ContactEvent.HideDialog -> {
                _state.update { it.copy(
                    isAddingContact = false
                ) }
            }
            ContactEvent.SaveContact -> {
                val firstName = state.value.firstName
```

```
                val lastName = state.value.lastName
                val phoneNumber = state.value.phoneNumber

                if(firstName.isBlank() || lastName.isBlank()
                    return
                }

                val contact = Contact(
                    firstName = firstName,
                    lastName = lastName,
                    phoneNumber = phoneNumber
                )
                viewModelScope.launch {
                    dao.upsertContact(contact)
                }
                _state.update { it.copy(
                    isAddingContact = false,
                    firstName = "",
                    lastName = "",
                    phoneNumber = ""
                ) }
            }
            is ContactEvent.SetFirstName -> {
                _state.update { it.copy(
                    firstName = event.firstName
                ) }
            }
            is ContactEvent.SetLastName -> {
                _state.update { it.copy(
                    lastName = event.lastName
                ) }
            }
            is ContactEvent.SetPhoneNumber -> {
                _state.update { it.copy(
                    phoneNumber = event.phoneNumber
                ) }
            }
            ContactEvent.ShowDialog -> {
```

```
                _state.update { it.copy(
                    isAddingContact = true
                ) }
            }
            is ContactEvent.SortContacts -> {
                _sortType.value = event.sortType
            }
        }
    }
}
```

Code the UI

Create the the ContactScreen:

```
package com.plcoding.roomguideandroid

import androidx.compose.foundation.clickable
import androidx.compose.foundation.horizontalScroll
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.rememberScrollState
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.material.icons.filled.Delete
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Alignment.Companion.CenterVertical
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

@Composable
fun ContactScreen(
    state: ContactState,
    onEvent: (ContactEvent) -> Unit
```

```
) {
    Scaffold(
        floatingActionButton = {
            FloatingActionButton(onClick = {
                onEvent(ContactEvent.ShowDialog)
            }) {
                Icon(
                    imageVector = Icons.Default.Add,
                    contentDescription = "Add contact"
                )
            }
        },
    ) { _ ->
        if(state.isAddingContact) {
            AddContactDialog(state = state, onEvent = onEvent
        }

        LazyColumn(
            contentPadding = PaddingValues(16.dp),
            modifier = Modifier.fillMaxSize(),
            verticalArrangement = Arrangement.spacedBy(16.dp)
        ) {
            item {
                Row(
                    modifier = Modifier
                        .fillMaxWidth()
                        .horizontalScroll(rememberScrollState
                    verticalAlignment = Alignment.CenterVerti
                ) {
                    SortType.values().forEach { sortType ->
                        Row(
                            modifier = Modifier
                                .clickable {
                                    onEvent(ContactEvent.Sort
                                },
                            verticalAlignment = CenterVertica
                        ) {
                            RadioButton(
```

```
                                selected = state.sortType ==
                                onClick = {
                                    onEvent(ContactEvent.Sort
                                }
                            )
                            Text(text = sortType.name)
                        }
                    }
                }
            }
            items(state.contacts) { contact ->
                Row(
                    modifier = Modifier.fillMaxWidth()
                ) {
                    Column(
                        modifier = Modifier.weight(1f)
                    ) {
                        Text(
                            text = "${contact.firstName} ${co
                            fontSize = 20.sp
                        )
                        Text(text = contact.phoneNumber, font
                    }
                    IconButton(onClick = {
                        onEvent(ContactEvent.DeleteContact(co
                    }) {
                        Icon(
                            imageVector = Icons.Default.Delet
                            contentDescription = "Delete cont
                        )
                    }
                }
            }
        }
    }
}
```

Create the AddContact dialog:

```
package com.plcoding.roomguideandroid

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.material.AlertDialog
import androidx.compose.material.Button
import androidx.compose.material.Text
import androidx.compose.material.TextField
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp

@Composable
fun AddContactDialog(
    state: ContactState,
    onEvent: (ContactEvent) -> Unit,
    modifier: Modifier = Modifier
) {
    AlertDialog(
        modifier = modifier,
        onDismissRequest = {
            onEvent(ContactEvent.HideDialog)
        },
        title = { Text(text = "Add contact") },
        text = {
            Column(
                verticalArrangement = Arrangement.spacedBy(8.
            ) {
                TextField(
                    value = state.firstName,
                    onValueChange = {
                        onEvent(ContactEvent.SetFirstName(it)
                    },
```

```
                    placeholder = {
                        Text(text = "First name")
                    }
                )
                TextField(
                    value = state.lastName,
                    onValueChange = {
                        onEvent(ContactEvent.SetLastName(it))
                    },
                    placeholder = {
                        Text(text = "Last name")
                    }
                )
                TextField(
                    value = state.phoneNumber,
                    onValueChange = {
                        onEvent(ContactEvent.SetPhoneNumber(i
                    },
                    placeholder = {
                        Text(text = "Phone number")
                    }
                )
            }
        },
        buttons = {
            Box(
                modifier = Modifier.fillMaxWidth(),
                contentAlignment = Alignment.CenterEnd
            ) {
                Button(onClick = {
                    onEvent(ContactEvent.SaveContact)
                }) {
                    Text(text = "Save")
                }
            }
        }
    )
}
```

Initialize the Database and code the MainActivity

You should use dependency injection to manage all the dependencies using Dagger-Hilt. For simplicity it isn't done in that way in this project. This is the code of the MainActivity:

```
package com.plcoding.roomguideandroid

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.viewModels
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material.MaterialTheme
import androidx.compose.material.Surface
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelProvider
import androidx.room.Room
import com.plcoding.roomguideandroid.ui.theme.RoomGuideAndroi

class MainActivity : ComponentActivity() {

    private val db by lazy {
        Room.databaseBuilder(
            applicationContext,
            ContactDatabase::class.java,
            "contacts.db"
        ).build()
    }
```

```
    private val viewModel by viewModels<ContactViewModel>(
        factoryProducer = {
            object : ViewModelProvider.Factory {
                override fun <T : ViewModel?> create(modelCla
                    return ContactViewModel(db.dao) as T
                }
            }
        }
    )

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            RoomGuideAndroidTheme {
                val state by viewModel.state.collectAsState()
                ContactScreen(state = state, onEvent = viewMo
            }
        }
    }
}
```

Learn more

Clone the repo at https://github.com/philipplackner/RoomGuideAndroid/tree/master and try to implement the following:

- [ ] Details of one contact by clicking on a contact (add things like address, email address).
- [ ] Also you could try to improve the UI.