المدرسة العليا للتكنولوجيا – الصويرة
ⵜⵉⵎⵙⵙⵓⵔⵉⵜ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵏ ⵜⵙⵙⵏⵉⵡⵉⵏ - ⵎⵓⵍⵉⵔⴰ
ÉCOLE SUPÉRIEURE DE TECHNOLOGIE - ESSAOUIRA

DUT : Génie Informatique

Cours de Développement Mobile avec Kotlin et Jetpack Compose

Pr. Lamia ZIAD
École Supérieure de Technologie d'Essaouira

Table des matières

1 Introduction à Jetpack Compose

1.1 Qu'est-ce que Jetpack Compose ?

Jetpack Compose est le toolkit moderne de Google pour créer des interfaces utilisateur natives sur Android. Il utilise une approche déclarative avec Kotlin.

Avantages :

— **Déclaratif** : Décrivez ce que vous voulez, pas comment le faire

— **Concis** : Moins de code que XML

— **Intuitif** : Code Kotlin pur

— **Compatibilité** : Fonctionne avec les vues existantes

1.2 Configuration de l'environnement

```
// build.gradle.kts (Module app)
android {
    buildFeatures {
        compose = true
    }

    composeOptions {
        kotlinCompilerExtensionVersion = "1.5.4"
    }
}

dependencies {
    implementation("androidx.compose.ui:ui:1.5.4")
    implementation("androidx.compose.material3:material3:1.1.2")
    implementation("androidx.activity:activity-compose:1.7.2")
}
```

2 Premiers Pas avec Compose

2.1 Structure de base

```
// MainActivity.kt
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MyAppTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    Greeting("Android")
                }
            }
        }
    }
}

@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
```

```
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}
```

2.2 Composants de base

```
@Composable
fun BasicComponents() {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        // Texte
        Text(
            text = "Bonjour Compose!",
            style = MaterialTheme.typography.headlineMedium
        )

        Spacer(modifier = Modifier.height(16.dp))

        // Bouton
        Button(
            onClick = { /* Action */ }
        ) {
            Text("Cliquez-moi")
        }

        // Champ de texte
        var text by remember { mutableStateOf("") }
        TextField(
            value = text,
            onValueChange = { text = it },
            label = { Text("Entrez du texte") }
        )
    }
}
```

3 Gestion d'État dans Compose

3.1 State et Recomposition

```
@Composable
fun Counter() {
    // State avec remember
    var count by remember { mutableStateOf(0) }

    Column(
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(
            text = "Compteur: $count",
            style = MaterialTheme.typography.headlineMedium
        )
```

```
13
14          Button(
15              onClick = { count++ }
16          ) {
17              Text("Incrementer")
18          }
19      }
20  }
```

3.2 State Hoisting

```
1   @Composable
2   fun CounterScreen() {
3       var count by remember { mutableStateOf(0) }
4
5       Counter(
6           count = count,
7           onIncrement = { count++ },
8           onDecrement = { count-- }
9       )
10  }
11
12  @Composable
13  fun Counter(
14      count: Int,
15      onIncrement: () -> Unit,
16      onDecrement: () -> Unit
17  ) {
18      Row(
19          verticalAlignment = Alignment.CenterVertically
20      ) {
21          Button(onClick = onDecrement) {
22              Text("-")
23          }
24
25          Text(
26              text = "$count",
27              modifier = Modifier.padding(16.dp)
28          )
29
30          Button(onClick = onIncrement) {
31              Text("+")
32          }
33      }
34  }
```

4 Layout et Modifiers

4.1 Layouts principaux

```
1   @Composable
2   fun LayoutExamples() {
3       // Column - disposition verticale
4       Column {
5           Text("Element 1")
6           Text("Element 2")
```

```
        Text("Element 3")
    }

    // Row - disposition horizontale
    Row {
        Text("Element A")
        Text("Element B")
        Text("Element C")
    }

    // Box - superposition
    Box(
        contentAlignment = Alignment.Center
    ) {
        Image(
            painter = painterResource(R.drawable.background),
            contentDescription = null
        )
        Text("Texte superpose")
    }
}
```

4.2 Modifiers courants

```
@Composable
fun ModifierExamples() {
    Text(
        text = "Texte stylise",
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp)
            .background(Color.Blue)
            .clip(RoundedCornerShape(8.dp))
            .border(1.dp, Color.Black)
            .clickable { /* Click action */ }
    )
}
```

5 Listes et Grids

5.1 LazyColumn - Liste défilante

```
data class Item(val id: Int, val name: String)

@Composable
fun ItemList(items: List<Item>) {
    LazyColumn {
        items(items) { item ->
            ItemRow(item = item)
        }
    }
}

@Composable
fun ItemRow(item: Item) {
    Card(
```

```
        modifier = Modifier
            .fillMaxWidth()
            .padding(8.dp),
        elevation = CardDefaults.cardElevation(4.dp)
    ) {
        Row(
            modifier = Modifier.padding(16.dp),
            verticalAlignment = Alignment.CenterVertically
        ) {
            Text(
                text = item.name,
                style = MaterialTheme.typography.bodyLarge,
                modifier = Modifier.weight(1f)
            )
            Icon(
                imageVector = Icons.Default.ArrowForward,
                contentDescription = null
            )
        }
    }
}
```

5.2 LazyVerticalGrid

```
@Composable
fun PhotoGrid(photos: List<Photo>) {
    LazyVerticalGrid(
        columns = GridCells.Adaptive(minSize = 128.dp)
    ) {
        items(photos) { photo ->
            Image(
                painter = rememberImagePainter(photo.url),
                contentDescription = null,
                modifier = Modifier
                    .aspectRatio(1f)
                    .padding(4.dp)
                    .clip(RoundedCornerShape(8.dp))
            )
        }
    }
}
```

6 Navigation avec Compose

6.1 Configuration de la navigation

```
// build.gradle.kts
dependencies {
    implementation("androidx.navigation:navigation-compose:2.7.0")
}

// Navigation graph
@Composable
fun MyApp() {
    val navController = rememberNavController()

```

```
    NavHost(
        navController = navController,
        startDestination = "home"
    ) {
        composable("home") {
            HomeScreen(
                onNavigateToProfile = {
                    navController.navigate("profile")
                }
            )
        }
        composable("profile") {
            ProfileScreen(
                onBack = { navController.popBackStack() }
            )
        }
    }
}
```

6.2 Écrans et navigation

```
@Composable
fun HomeScreen(onNavigateToProfile: () -> Unit) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text("Page d'accueil")

        Button(
            onClick = onNavigateToProfile
        ) {
            Text("Voir le profil")
        }
    }
}

@Composable
fun ProfileScreen(onBack: () -> Unit) {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        IconButton(onClick = onBack) {
            Icon(Icons.Default.ArrowBack, "Retour")
        }

        Text("Profil utilisateur")
    }
}
```

7 Architecture MVVM avec Compose

7.1 ViewModel et State

```
// ViewModel
class UserViewModel : ViewModel() {
    private val _userState = mutableStateOf(UserState())
    val userState: State<UserState> = _userState

    fun loadUser(userId: String) {
        viewModelScope.launch {
            _userState.value = _userState.value.copy(
                isLoading = true
            )

            try {
                val user = userRepository.getUser(userId)
                _userState.value = _userState.value.copy(
                    user = user,
                    isLoading = false
                )
            } catch (e: Exception) {
                _userState.value = _userState.value.copy(
                    error = e.message,
                    isLoading = false
                )
            }
        }
    }
}

data class UserState(
    val user: User? = null,
    val isLoading: Boolean = false,
    val error: String? = null
)
```

7.2 Utilisation dans Compose

```
@Composable
fun UserProfileScreen(
    viewModel: UserViewModel = hiltViewModel()
) {
    val state by viewModel.userState.collectAsState()

    when {
        state.isLoading -> {
            CircularProgressIndicator()
        }
        state.error != null -> {
            ErrorMessage(state.error!!)
        }
        state.user != null -> {
            UserDetails(state.user!!)
        }
        else -> {
            // Etat initial
        }
```

```
    }
}

@Composable
fun UserDetails(user: User) {
    Column {
        Text(text = user.name)
        Text(text = user.email)
        // ...
    }
}
```

8 Thèmes et Material Design 3

8.1 Configuration du thème

```
@Composable
fun MyAppTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    content: @Composable () -> Unit
) {
    val colorScheme = when {
        darkTheme -> DarkColorScheme
        else -> LightColorScheme
    }

    MaterialTheme(
        colorScheme = colorScheme,
        typography = Typography,
        content = content
    )
}

private val LightColorScheme = lightColorScheme(
    primary = Purple40,
    secondary = PurpleGrey40,
    tertiary = Pink40
    // ... autres couleurs
)

private val DarkColorScheme = darkColorScheme(
    primary = Purple80,
    secondary = PurpleGrey80,
    tertiary = Pink80
    // ... autres couleurs
)
```

9 Accès aux Données

9.1 Room Database

```
// Entity
@Entity
data class Task(
    @PrimaryKey val id: Int,
```

```
    val title: String,
    val completed: Boolean = false
)

// DAO
@Dao
interface TaskDao {
    @Query("SELECT * FROM task")
    fun getAll(): Flow<List<Task>>

    @Insert
    suspend fun insert(task: Task)

    @Update
    suspend fun update(task: Task)
}

// Repository
class TaskRepository(private val taskDao: TaskDao) {
    fun getAllTasks(): Flow<List<Task>> = taskDao.getAll()

    suspend fun addTask(task: Task) = taskDao.insert(task)
}
```

9.2 Intégration avec Compose

```
@Composable
fun TaskListScreen(viewModel: TaskViewModel = hiltViewModel()) {
    val tasks by viewModel.tasks.collectAsState(initial = emptyList())

    LazyColumn {
        items(tasks) { task ->
            TaskItem(
                task = task,
                onToggle = { viewModel.toggleTask(task) }
            )
        }
    }
}

@Composable
fun TaskItem(task: Task, onToggle: () -> Unit) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        Checkbox(
            checked = task.completed,
            onCheckedChange = { onToggle() }
        )

        Text(
            text = task.title,
            modifier = Modifier.weight(1f),
            textDecoration = if (task.completed) {
                TextDecoration.LineThrough
            } else {
```

```
                TextDecoration.None
            }
        )
    }
}
```

10 Projet Pratique : Application Todo

10.1 Structure complète

```
// MainActivity
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            TodoAppTheme {
                TodoApp()
            }
        }
    }
}

// Navigation
@Composable
fun TodoApp() {
    val navController = rememberNavController()

    NavHost(navController, "tasks") {
        composable("tasks") {
            TaskListScreen(
                onAddTask = {
                    navController.navigate("add_task")
                }
            )
        }
        composable("add_task") {
            AddTaskScreen(
                onBack = { navController.popBackStack() }
            )
        }
    }
}
```

10.2 Écran de liste des tâches

```
@Composable
fun TaskListScreen(
    viewModel: TaskViewModel = hiltViewModel(),
    onAddTask: () -> Unit
) {
    val tasks by viewModel.tasks.collectAsState()

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("Mes Taches") }
```

```
            )
        },
        floatingActionButton = {
            FloatingActionButton(onClick = onAddTask) {
                Icon(Icons.Default.Add, "Ajouter")
            }
        }
    ) { padding ->
        if (tasks.isEmpty()) {
            EmptyState()
        } else {
            LazyColumn(modifier = Modifier.padding(padding)) {
                items(tasks) { task ->
                    TaskItem(
                        task = task,
                        onToggle = { viewModel.toggleTask(task) },
                        onDelete = { viewModel.deleteTask(task) }
                    )
                }
            }
        }
    }
}
```

11 Testing avec Compose

11.1 Tests UI

```
// Test de composant
@Test
fun shouldDisplayInitialText() {
    composeTestRule.setContent {
        MyAppTheme {
            Greeting("Test")
        }
    }

    composeTestRule
        .onNodeWithText("Hello Test!")
        .assertExists()
}

// Test d'interaction
@Test
fun counterShouldIncrementWhenButtonClicked() {
    composeTestRule.setContent {
        Counter()
    }

    composeTestRule.onNodeWithText("Compteur: 0").assertExists()

    composeTestRule.onNodeWithText("Incrementer").performClick()

    composeTestRule.onNodeWithText("Compteur: 1").assertExists()
}
```

12 Performance et Bonnes Pratiques

12.1 Optimisations

- **Utilisez `remember`** : Pour éviter les recalculs coûteux
- **Évitez les lambdas instanciées dans la recomposition** : Utilisez `rememberUpdatedState`
- **Utilisez `Lazy lists`** : Pour les longues listes
- **Profitez du preview** : Pour développer rapidement

12.2 Composable Previews

```
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
    MyAppTheme {
        Greeting("Android")
    }
}

@Preview(
    showBackground = true,
    widthDp = 360,
    heightDp = 640
)
@Composable
fun TaskListPreview() {
    MyAppTheme {
        TaskListScreen(onAddTask = {})
    }
}
```

13 Annexe : Cheatsheet Compose

Composant	Usage
`Text()`	Affichage de texte
`Button()`	Bouton cliquable
`TextField()`	Champ de saisie
`Image()`	Affichage d'image
`Column()`	Layout vertical
`Row()`	Layout horizontal
`Box()`	Superposition
`LazyColumn()`	Liste défilante
`Scaffold()`	Structure d'écran

Modifier	Effet
`.fillMaxSize()`	Remplit l'espace disponible
`.padding()`	Ajoute de l'espace
`.background()`	Change la couleur de fond
`.clickable()`	Rend cliquable
`.size()`	Définit la taille
`.weight()`	Poids dans Row/Column