Giới thiệu về Kotlin JetPack Compose

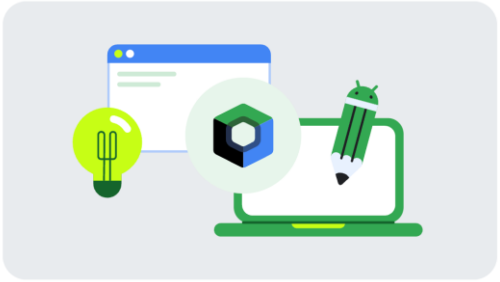
Nội dung

- Giới thiệu về Jetpack Compose
- Cấu trúc project Jetpack Compose
- Các thành phần cơ bản của Jetpack Compose

Jetpack Compose là gì?

Jetpack Compose là một bộ công cụ hiện đại, được khuyên dùng cho Android để xây dựng giao diện người dùng gốc với ít mã hơn, các công cụ mạnh mẽ và API Kotlin trực quan. Công cụ này giúp đơn giản hoá và đẩy nhanh quá trình phát triển giao diện người dùng, bao gồm cả khả năng thích ứng với mọi kiểu dáng, từ điện thoại thông minh, thiết bị có thể gập lại, máy tính bảng cho đến TV và thiết bị đeo.

Compose cung cấp các thành phần giao diện người dùng được tạo sẵn. Tính năng này hoạt động với đồ hoạ, ảnh động và các phần tử hình ảnh khác.



Tại sao nên sử dụng Jetpack Compose

Ít code hơn

Việc viết ít mã hơn sẽ ảnh hưởng đến tất cả các giai đoạn phát triển: với tư cách là tác giả, bạn sẽ tập trung vào vấn đề hiện có, ít phải kiểm tra, gỡ lỗi và ít xảy ra lỗi hơn; với tư cách là người đánh giá hoặc người bảo trì, bạn có ít mã để đọc, hiểu, xem lại và bảo trì.

Compose cho phép bạn làm được nhiều việc hơn bằng ít mã hơn, so với việc sử dụng hệ thống Android View: các nút, danh sách hoặc ảnh động – bất kể bạn tạo gì, bây giờ, bạn không cần phải viết nhiều mã nữa

Tại sao nên sử dụng Jetpack Compose

Trực quan

Compose sử dụng API khai báo, nghĩa là tất cả những gì bạn cần làm là mô tả giao diện người dùng. Compose sẽ xử lý phần còn lại. API rất trực quan – dễ khám phá và sử dụng:

"*Lớp giao diện của chúng tôi trực quan và dễ đọc hơn rất nhiều. Chúng tôi có thể thực thi trong một tệp Kotlin duy nhất. Việc này trước đây thường phải dùng nhiều tệp XML để thực hiện việc xác định thuộc tính và chỉ định thông qua nhiều lớp phủ giao diện.*" (Twitter)

Tại sao nên sử dụng Jetpack Compose

Trực quan

Với Compose, bạn tạo các thành phần nhỏ, không có trạng thái không gắn với một hoạt động hoặc mảnh cụ thể. Điều đó giúp dễ dàng sử dụng lại và thử nghiệm chúng: “Chúng tôi đặt mục tiêu phân phối một bộ thành phần giao diện người dùng mới không có trạng thái, dễ sử dụng và duy trì cũng như trực quan để triển khai/mở rộng/tuỳ chỉnh. Compose thực sự là câu trả lời vững chắc cho chúng tôi trong vấn đề này.” (Twitter)

Tại sao nên sử dụng Jetpack Compose

Trực quan

Trong Compose, trạng thái hoàn toàn rõ ràng và được chuyển tới nội dung kết hợp. Bằng cách đó, có một nguồn đáng tin cậy duy nhất cho trạng thái đã được gói gọn và tách biệt. Sau đó, khi trạng thái ứng dụng thay đổi, giao diện người dùng của bạn sẽ tự động cập nhật. “Bạn sẽ bớt phải suy nghĩ khi đang suy luận về điều gì đó và giảm những hành vi nằm ngoài tầm kiểm soát hoặc không hiểu rõ của bạn” (Cuvva)

Tại sao nên sử dụng Jetpack Compose

Đẩy nhanh quy trình phát triển

Compose tương thích với tất cả mã hiện có của bạn: bạn có thể gọi mã Compose từ Views và Views từ Compose. Hầu hết các thư viện phổ biến như Di chuyển, ViewModel và coroutine Kotlin hoạt động với Compose, vì vậy bạn có thể bắt đầu áp dụng vào lúc và ở nơi bạn muốn. “Khả năng tương tác là lý do chúng tôi bắt đầu tích hợp Compose và điều chúng tôi nhận thấy là tính năng 'hoạt động luôn'. Chúng tôi nhận thấy rằng chúng tôi không phải suy nghĩ về những vấn đề như chế độ sáng và tối, cũng như toàn bộ trải nghiệm sẽ rất liền mạch”. (Cuvva)

Tại sao nên sử dụng Jetpack Compose

Đẩy nhanh quy trình phát triển

Khi dùng dịch vụ hỗ trợ đầy đủ của Android Studio, với các tính năng như xem trước trực tiếp, bạn sẽ lặp lại và vận chuyển mã nhanh hơn: “Các bản xem trước trong Android Studio luôn giúp tiết kiệm thời gian. Việc có thể tạo nhiều bản xem trước cũng giúp chúng tôi tiết kiệm thời gian. Thông thường, chúng tôi cần kiểm tra thành phần trên giao diện người dùng ở các trạng thái hoặc với các tuỳ chọn cài đặt khác nhau (như trạng thái lỗi hoặc với kích thước phông chữ khác, v.v.). Với khả năng tạo nhiều bản xem trước, chúng tôi có thể dễ dàng kiểm tra điều này. ”(Square)

Tại sao nên sử dụng Jetpack Compose

Mạnh mẽ

Compose cho phép bạn tạo các ứng dụng tuyệt đẹp có quyền truy cập trực tiếp vào API nền tảng Android và khả năng hỗ trợ tích hợp dành cho Material Design, Giao diện tối, ảnh động và nhiều ưu điểm khác: “Compose cũng đã giải quyết được nhiều vấn đề hơn ngoài giao diện người dùng khai báo -- API hỗ trợ tiếp cận, bố cục hỗ trợ tiếp cận, tất cả các loại nội dung đều được cải thiện. Việc biến ý tưởng thành sản phẩm cần ít bước hơn” (Square).

Tại sao nên sử dụng Jetpack Compose

Mạnh mẽ

Với Compose, việc mang lại chuyển động và sự sống cho ứng dụng của bạn thông qua ảnh động thật nhanh chóng và dễ dàng: “Trong Compose, rất dễ thêm ảnh động, nên có rất ít lý do để không tạo ảnh động những yếu tố như thay đổi màu/kích thước/độ cao” (Monzo), “bạn có thể tạo ảnh động mà không yêu cầu bất kỳ kỹ năng đặc biệt nào – không khác gì so với việc hiển thị một màn hình tĩnh” (Square).

Tại sao nên sử dụng Jetpack Compose

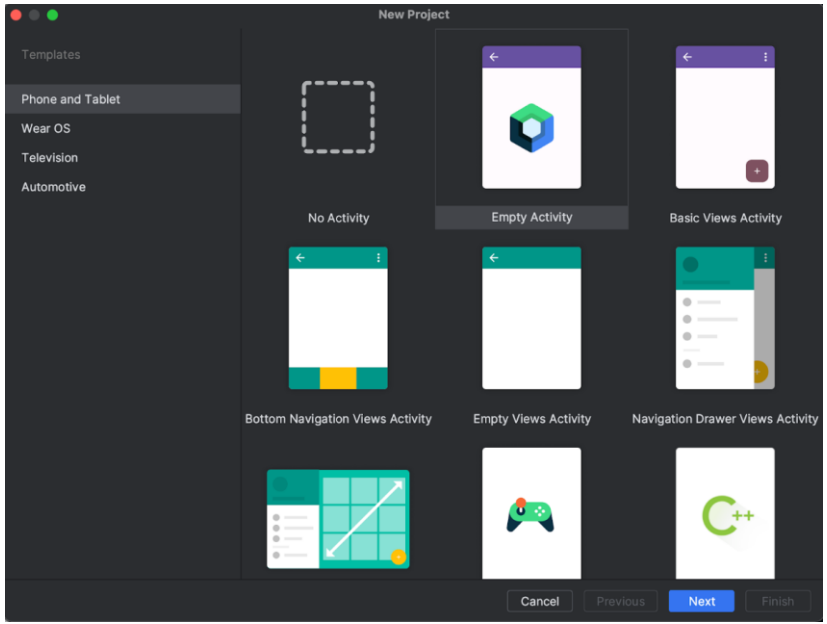
Mạnh mẽ

Dù bạn đang xây dựng bằng Material Design hay hệ thống thiết kế của riêng mình, thì bạn có thể linh hoạt triển khai thiết kế mà mình mong muốn: “Việc thiết kế Material Design tách biệt với nền tảng đã thực sự hữu ích đối với chúng tôi vì chúng tôi đang xây dựng hệ thống thiết kế, vốn thường yêu cầu nhiều công việc thiết kế từ Material.” (Square)

Tạo project đầu tiên

Để bắt đầu một dự án Compose mới, hãy mở Android Studio và nhấn vào nút **New Project**.

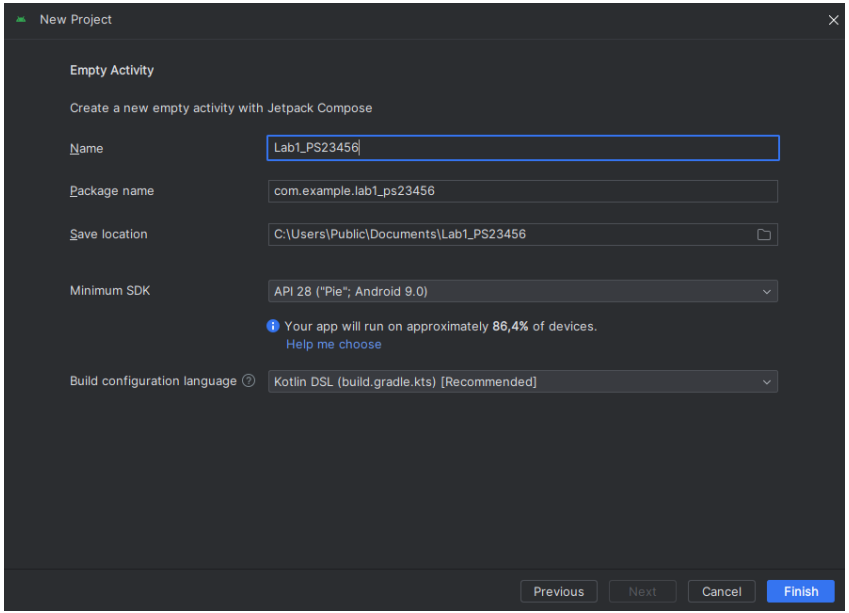
Chọn **Empty Activity** và nhấn **Next** để tiếp tục



Tạo project đầu tiên

Ở dialog New Project điền một số thông tin:

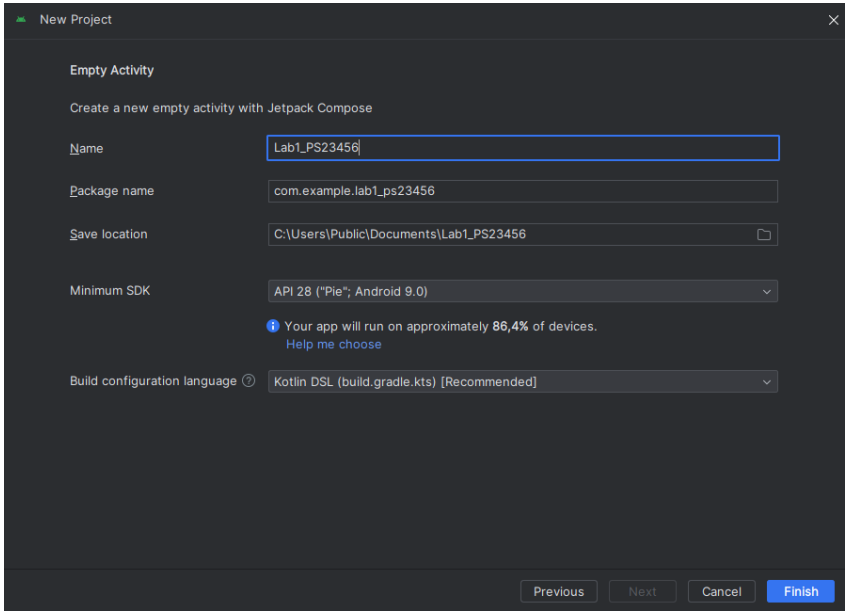
- Đặt tên project theo yêu cầu
- Đặt tên package (nếu cần)
- Chọn nơi lưu trữ project
- Chọn ngôn ngữ cho project là Kotlin
- Chọn minimum SDK



Tạo project đầu tiên

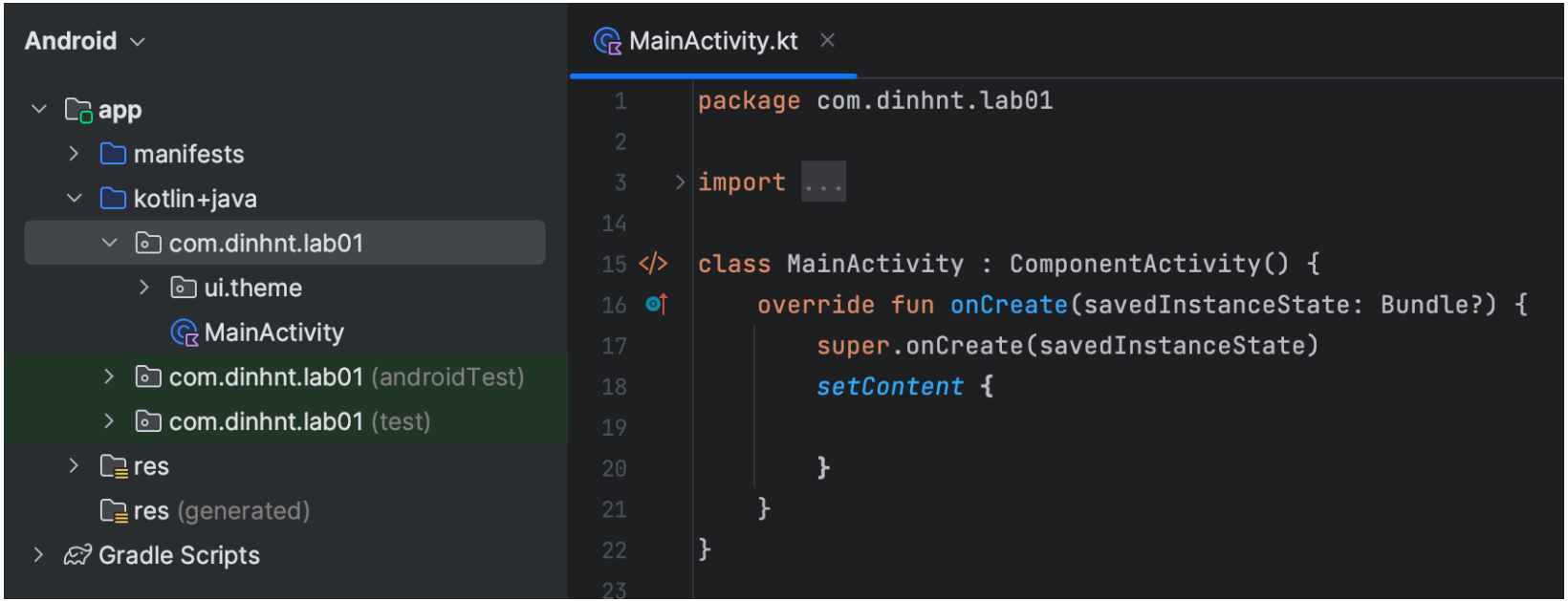
Ở dialog New Project điền một số thông tin:

- Đặt tên project theo yêu cầu
- Đặt tên package (nếu cần)
- Chọn nơi lưu trữ project
- Chọn ngôn ngữ cho project là Kotlin
- Chọn minimum SDK



Tạo project đầu tiên

Sau khi tạo xong project, ta được:



Tạo project đầu tiên

Trong Compose, Activity vẫn là điểm bắt đầu của ứng dụng Android. Trong dự án của chúng ta, MainActivity được khởi chạy khi người dùng mở ứng dụng (như được chỉ định trong tệp AndroidManifest.xml). Bạn sẽ sử dụng setContent để xác định bố cục, nhưng thay vì sử dụng tệp XML như thường làm trong hệ thống View truyền thống, hãy gọi các **composable function** trong đó.

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

        }
    }
}
```

Tạo project đầu tiên

Trong Compose, Activity vẫn là điểm bắt đầu của ứng dụng Android. Trong dự án của chúng ta, MainActivity được khởi chạy khi người dùng mở ứng dụng (như được chỉ định trong tệp AndroidManifest.xml). Bạn sẽ sử dụng setContent để xác định bố cục, nhưng thay vì sử dụng tệp XML như thường làm trong hệ thống View truyền thống, hãy gọi các **composable function** trong đó.

Tạo project đầu tiên

Tạo **composable function Greeting** có annotation là @Composable.

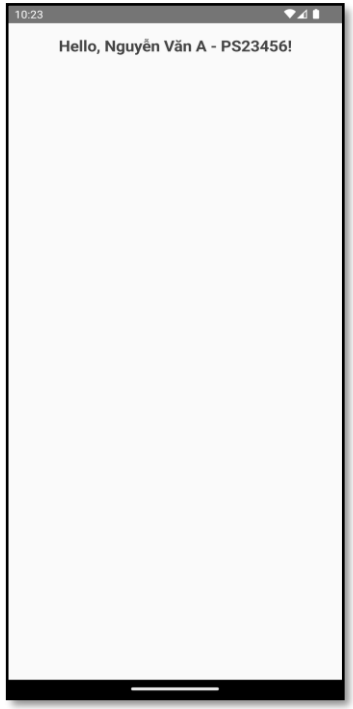
Thao tác này cho phép hàm của bạn gọi các hàm @Composable khác trong đó.

```
@Composable
fun Greeting(name: String) {
    Text(
        text = "Hello, $name!",
        modifier = Modifier
            .padding(0.dp, 16.dp)
            .fillMaxWidth(),
        color = Color.DarkGray,
        fontSize = 20.sp,
        fontWeight = FontWeight.Bold,
        textAlign = TextAlign.Center,
    )
}
```

Tạo project đầu tiên

Gọi hàm **Greeting** trong **setContent** trong onCreate của Activity

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Greeting("Nguyễn Văn A - PS23456")
        }
    }
}
```



Tạo project đầu tiên

Lưu ý: *Khi nhập các lớp liên quan đến* ***Jetpack Compose*** *trong dự án, hãy sử dụng các lớp đó trong các thư viện sau:*

*androidx.compose.** *cho các lớp trình biên dịch và thời gian chạy.*

*androidx.compose.ui.** *cho bộ công cụ và thư viện giao diện người dùng.*

Tạo project đầu tiên

Trong **Jetpack Compose** ta có thể xem trước giao diện của các **composable function** bằng annotation @Preview, annotation này sẽ cập nhật lại giao diện ngay lập tức khi có sự thay đổi.

Để sử dụng @Preview, bạn cần thêm annotation này trước hàm **composable function** mà bạn muốn xem trước như sau:

Tạo hàm **PreviewGreeting** và thêm annotation **@Preview**

```
@Preview(showBackground = true)
@Composable
fun PreviewGreeting() {
    Greeting("Nguyễn Văn A - PS23456")
}
```

Tạo project đầu tiên

Các thuộc tính của **@Preview**:

- **showBackground**: Kích hoạt nền mặc định cho bản xem trước, giúp nhận diện không gian thành phần.
- **widthDp** và **heightDp**: Đặt kích thước cho bản xem trước với chiều rộng và chiều cao cụ thể trong đơn vị 'dp’.
- **uiMode**: Thiết lập chế độ giao diện (ví dụ: ban đêm hoặc ban ngày) để kiểm tra giao diện trong các trường hợp sử dụng khác nhau.

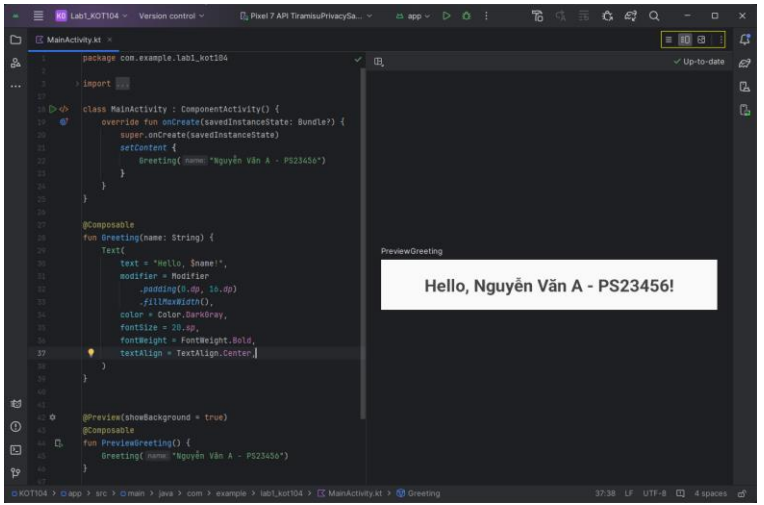
Tạo project đầu tiên

Các thuộc tính của **@Preview**:

- **fontScale**: Điều chỉnh tỷ lệ phông chữ, hữu ích cho việc kiểm tra UI với các cài đặt phông chữ khác nhau.
- **locale**: Xác định ngôn ngữ và khu vực, giúp kiểm tra UI trong các ngữ cảnh ngôn ngữ cụ thể.

Tạo project đầu tiên

Trong **Android Studio** chuyển sang chế độ **Split** ở góc trên bên phải của trình soạn thảo mã, bạn sẽ thấy các biểu tượng cho các chế độ xem khác nhau: **Code**, **Split**, và **Design**. Chọn "**Split**" để xem cả mã và bản xem trước cùng một lúc, hoặc chọn "**Design**" để chỉ xem bản xem trước.



Tạo project đầu tiên

Trong MainActivty.kt ta tiếp tục thực hiện các bước sau:

- Tạo **composable function GreetingCard**

```
@Composable
fun GreetingCard(msg: String) {
    var text by remember { mutableStateOf(msg) }

    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(24.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        MessageCard(msg = text)
        Button(onClick = { text = "Hi!" }) {
            Text("Say Hi!")
        }
    }
}
```

Tạo project đầu tiên

Sử dụng remember để khai báo một biến trạng thái text. Khi giá trị này thay đổi, UI sẽ tự động cập nhật. mutableStateOf được sử dụng để tạo một đối tượng trạng thái có thể thay đổi.

```
@Composable
fun GreetingCard(msg: String) {
    var text by remember { mutableStateOf(msg) }

    . . .
}
```

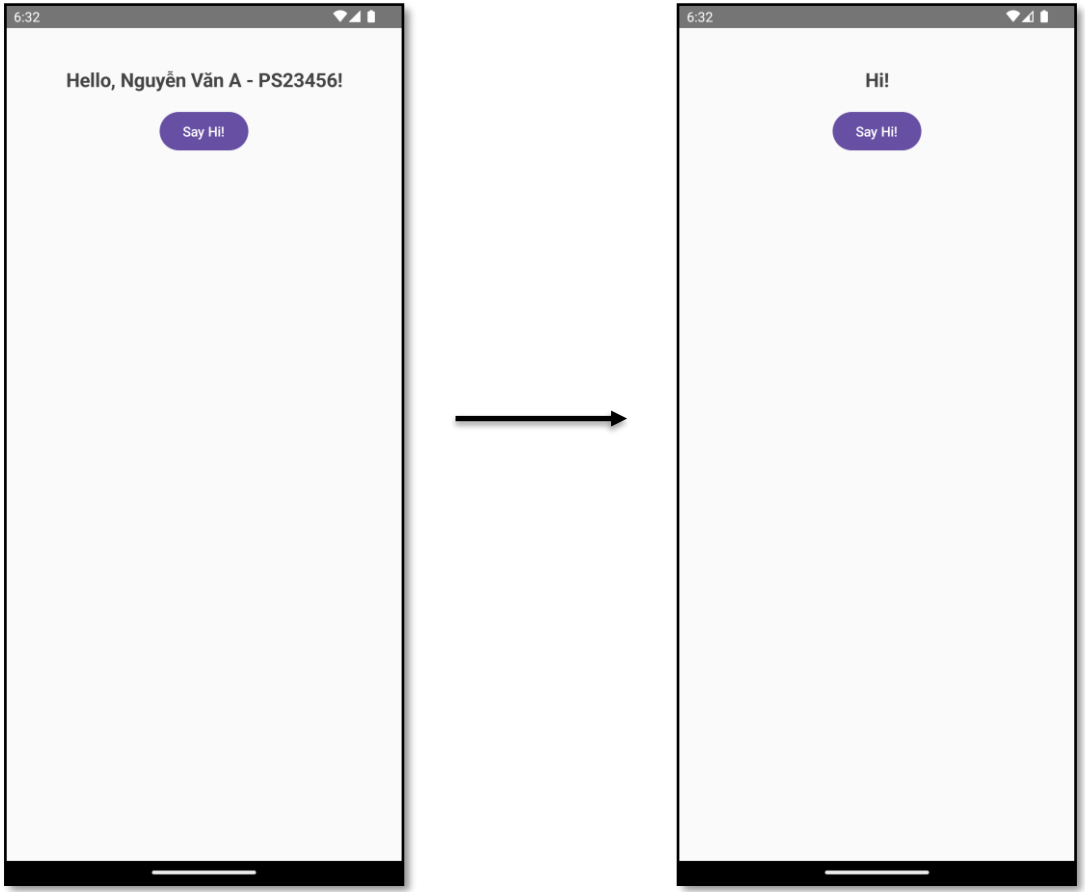
Tạo project đầu tiên

Đổi tên hàm **Greeting** thành **MessageCard** sau đó gọi **MessageCard** trong hàm **GreetingCard**.

```
@Composable
fun MessageCard(msg: String) {
    Text(
        text = msg,
        modifier = Modifier
            .padding(0.dp, 16.dp)
            .fillMaxWidth(),
        color = Color.DarkGray,
        fontSize = 20.sp,
        fontWeight = FontWeight.Bold,
        textAlign = TextAlign.Center,
    )
}
```

Tạo project đầu tiên

Ta được kết quả



Một ví dụ khác

Yêu cầu: Giá trị vẫn được giữ nguyên khi thực hiện thao tác xoay màn hình

Trong MainActivty.kt

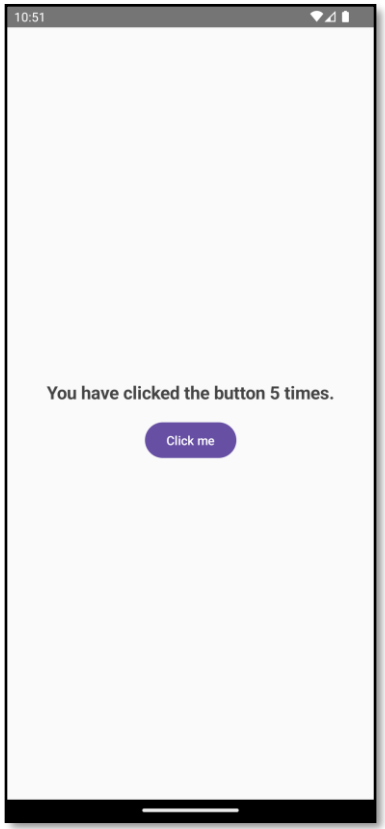
- Tạo 1 **composable function CounterCard**.

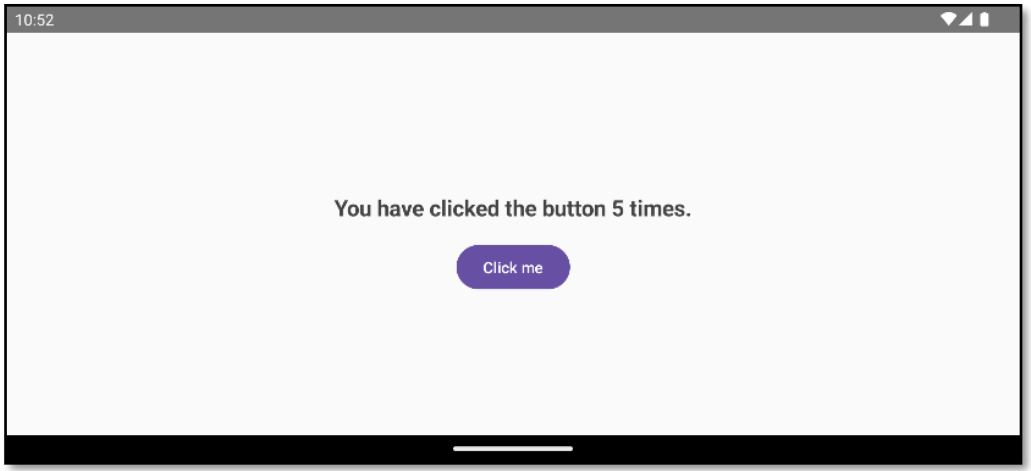
```
@Composable
fun CounterCard() {
    var count by rememberSaveable { mutableIntStateOf(0) }

    Column(
        modifier = ModifierfillMaxWidth().padding(24.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        MessageCard("You have clicked the button $count times.")
        Button(onClick = { count++ }) {
            Text("Click me")
        }
    }
}
```

Một ví dụ khác

- **rememberSaveable** được sử dụng để lưu giá trị của biến **count** qua các sự kiện cấu hình. Trong khi **remember** chỉ giữ giá trị trong suốt vòng đời của một **recomposition** và sẽ bị reset khi có thay đổi cấu hình (như xoay màn hình), **rememberSaveable** giữ giá trị qua các thay đổi đó.
- Chạy ứng dụng và quan xác kết quả khi xoay màn hình, đảm bảo giá trị của count không bị reset về 0.

Một ví dụ khác





Ở màn hình dọc hay ở màn hình ngang thì giá trị counter không thay đổi

Tài liệu tham khảo

- kotlinlang.org
- kotlinlang.org/docs
- play.kotlinlang.org/byExample

Thanks!