ANDROID BLOGGING GUIDE

Exploring LaunchedEffect in Jetpack Compose

A deep dive into the powerful side effect handler for Jetpack Compose.

Mr Umbrella · Follow

Published in Level Up Coding · 3 min read · Apr 23

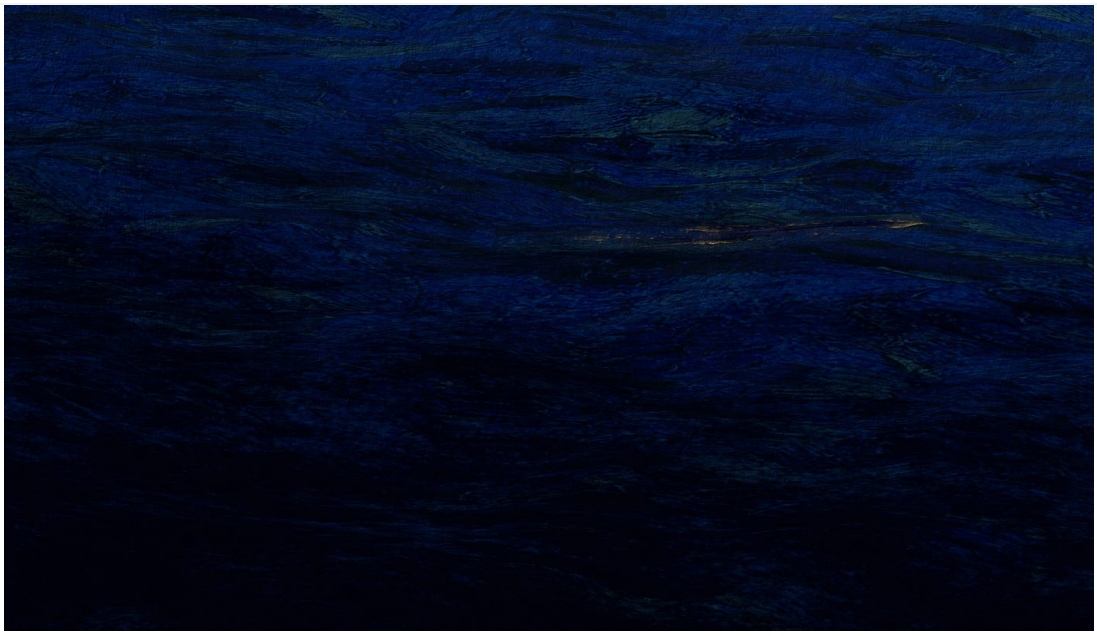


Photo by SIMON LEE on Unsplash

Introduction

LaunchedEffect is a powerful feature in Jetpack Compose that allows developers to perform side effects during the composition process. This feature is particularly useful for asynchronous operations, such as network calls or database queries, where the result needs to be displayed in the UI.

In this article, we will take a deep dive into LaunchedEffect and explore how it works, its benefits, and how to use it in your Jetpack Compose projects.

What is LaunchedEffect?

> LaunchedEffect is a side effect that allows you to launch a coroutine in response to a composition.

It is similar to the standard `launch` function in Kotlin coroutines, but it is designed specifically for use in Jetpack Compose.

LaunchedEffect is a suspending function that takes a lambda as its parameter. This lambda represents the coroutine that will be launched when the composition is recomposed. The coroutine can be used to perform asynchronous operations, such as network calls or database queries, and update the UI when the result is available.

How does LaunchedEffect work?

When a composition is recomposed, Jetpack Compose calls the lambda passed to LaunchedEffect. This lambda should contain the code that performs the asynchronous operation. Once the operation is complete, the coroutine should update the UI using the `MutableState` or `MutableStateFlow` objects.

One of the key benefits of LaunchedEffect is that it automatically cancels the coroutine when the composition is no longer active. This helps to prevent memory leaks and ensures that the app remains responsive.

Benefits of LaunchedEffect

LaunchedEffect has several benefits over other methods of performing side effects in Jetpack Compose:

1. **Automatic cancellation**: LaunchedEffect automatically cancels the coroutine when the composition is no longer active. This helps to prevent memory leaks and ensures that the app remains responsive.
2. **Simplified code**: LaunchedEffect allows you to perform asynchronous operations and update the UI in a single lambda. This simplifies the code and makes it easier to read and maintain.
3. **Improved performance**: LaunchedEffect is designed to work seamlessly with Jetpack Compose, ensuring that the app remains responsive even during long-running operations.

How to use LaunchedEffect

To use LaunchedEffect in your Jetpack Compose project, you need to follow these steps:

Import the `LaunchedEffect` function from the `androidx.compose.runtime` package:

```
import androidx.compose.runtime.LaunchedEffect
```

Use LaunchedEffect in your composition, passing in the coroutine to be launched:

```
LaunchedEffect(someState) {
  // Perform asynchronous operation
}
```

Update the UI using `MutableState` or `MutableStateFlow` objects:

```
val resultState = remember { mutableStateOf<Result?>(null) }

LaunchedEffect(someState) {
  val result = performAsynchronousOperation()
  resultState.value = result
}

if (resultState.value != null) {
  // Display the result in the UI
}
```

In this example, `someState` is a variable that is used to trigger the recomposition of the composition. When `someState` changes, the LaunchedEffect is triggered, and the coroutine is launched. Once the asynchronous operation is complete, the result is stored in the `resultState` variable, which is then used to update the UI.

Conclusion

LaunchedEffect is a powerful feature in Jetpack Compose that allows developers to perform asynchronous operations and update the UI in a single lambda. It simplifies the code and ensures that the app remains responsive, even during long-running operations.

If you are developing a Jetpack Compose app that requires asynchronous operations, LaunchedEffect is the recommended approach.

Thanks a lot for reading my article. *If you enjoyed this story, please* ***click the*** 👏 ***button and share*** *it to help others!* Follow me on Medium *for more awesome Android tips*. You can also find me on LinkedIn. Have a nice day! 😄

Continue Reading Android Stuff

Implement TabLayout with ViewPager in Android Jetpack Compose

Implement Android TabLayout in Jetpack Compose

Efficiently Display Data with Jetpack Compose: A Beginner's Guide to LazyColumn and LazyRow

Building Beautiful App Interfaces with Scaffold in Android

Building Interactive User Interfaces with Jetpack Compose Chip Component

Android | Jetpack Compose | Launchedeffect | Side Effects

Written by Mr Umbrella

424 Followers · Writer for Level Up Coding

Writer, enthusiastic developer

Follow

More from Mr Umbrella and Level Up Coding

Mr Umbrella in Level Up Coding

The Battle of State Management: Remember vs. StateFlow in Jetpa...

Which state management solution is right for your Android app?

· 3 min read · Apr 17

36

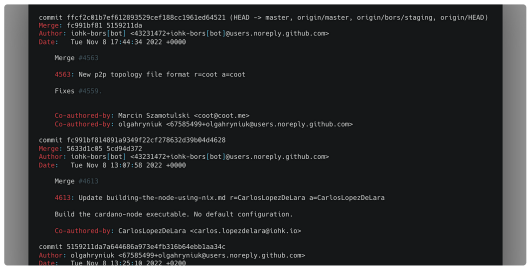
Sanjay Priyadarshi in Level Up Coding

I Spent 30 Days Studying A Programmer Who Built a $230...

Steal This Programmer Blueprint

· 13 min read · Aug 7

781 3



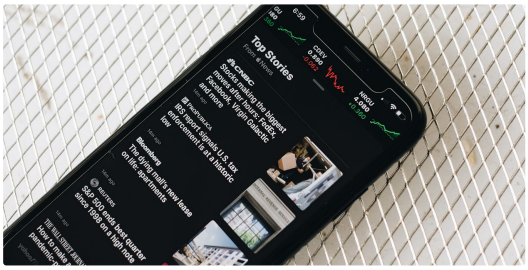
Jacob Bennett in Level Up Coding

Use Git like a senior engineer

Git is a powerful tool that feels great to use when you know how to use it.

· 4 min read · Nov 15, 2022

8.9K 100



Mr Umbrella in Level Up Coding

Efficiently Display Data with Jetpack Compose: A Beginner's...

Learn how to efficiently display large amounts of data in your app with Jetpack Compose's...

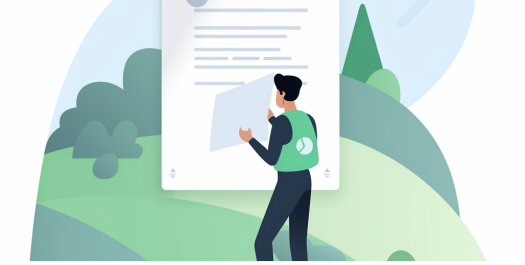
· 10 min read · Apr 12

16

See all from Mr Umbrella

See all from Level Up Coding

Recommended from Medium



Tippu Fisal Sheriff

Mastering the DRY Principle in Android Development with Kotlin

Write Efficient Code —Don't Repeat Yourself (DRY)

3 min read · Jul 24

60

Rafael Meneghelo

How to Request Permissions in Jetpack Compose: A Step-by-Ste...

Step-by-step guide to requesting permissions in Jetpack Compose.

3 min read · Aug 8

Lists

Now in AI: Handpicked by Better Programming

262 stories · 85 saves

Staff Picks

399 stories · 219 saves

Robin Wu

Passing events from Composable functions to MVI ViewModels

When creating interactive Jetpack compose UI backed by ViewModels, we usually pass i...

3 min read · Jul 31

Sujatha Mudadla

Interview questions on Android Jetpack Compose.

What is Android Jetpack Compose, and why was it introduced?

7 min read · Jun 8

Sevban Bayır

Exciting Future of Kotlin: K2 Compiler

The development of the Kotlin compiler started many years ago, and many things...

7 min read · Jun 19

Dheeraj Singh Bhadoria

Jetpack Compose Android App with MVVM Architecture and...

API Integration in Jetpack Compose Android App with MVVM and Retrofit

3 min read · May 23

1

See more recommendations