

Software Engineering Summit

Android Workshop



Instructor: Yashar Atajan
TA: Nick Capurso

About Yashar

Yashar Atajan
@yaxarat



Undergrad - Virginia Commonwealth University



Software Engineer working on Android SDKs.



About Nick

Nick Capurso

@nickcapurso



Undergrad - Texas Christian Univ (TCU)

Grad - George Washington Univ (GWU)

I also teach Android at GWU!



Capital One Wallet, Capital One Mobile, SDKs



Came in through the intern & TDP programs :)

Before we get started...

- We'll be using [this GitHub repo](#) as a base.
- The workshop assumes you completed the pre-reqs and have a working Android development environment.
 - Ability to run the app on the Android emulator or a physical device.
- Import the “Start” folder into Android Studio and run it on an emulator / device.

If you need help...

Short questions:

- Use the “Raise Hand” action in Zoom. Or feel free to speak out!
- You can also post short questions in the Zoom chat - we’ll be monitoring it!

Debugging help:

- Post your issue in the `#ses-may21help-xg` Slack channel with any relevant code / screenshots.
- During short breaks in the workshop, we will use Zoom breakout rooms for any in-depth debugging with screen share.
- Slack: [@yashar](#) (Instructor), [@nickcapurso](#) (TA)

Step-by-step instructions:

- Full text instructions for this workshop are available [on GitHub](#).



android Workshop

What're we building?

- A simple app that allows the user to login & view a list of recent transactions.

The image displays two screenshots of a mobile application interface. The left screenshot shows the login screen for "Summit Bank". It features a white background with a purple header bar. The title "Summit Bank" is centered at the top. Below it are two input fields: "Username" and "Password", each with a placeholder text inside. To the right of the "Password" field is a "Remember me" checkbox with the label "Remember me" next to it. At the bottom is a large purple rectangular button with the text "Login In" in white. The right screenshot shows a list of recent transactions for "Card 7890". The header reads "Hello, Summiteers" and "Your recent transactions for Card 7890". The list consists of five items, each with a small blue and pink card icon, the merchant name, and the amount: Starbucks (\$1.40), Macy's (\$35.00), Giant (\$64.37), Amazon (\$11.99), and Petsmart (\$29.12). Each transaction item has a thin gray border.

Merchant	Amount
Starbucks	\$1.40
Macy's	\$35.00
Giant	\$64.37
Amazon	\$11.99
Petsmart	\$29.12

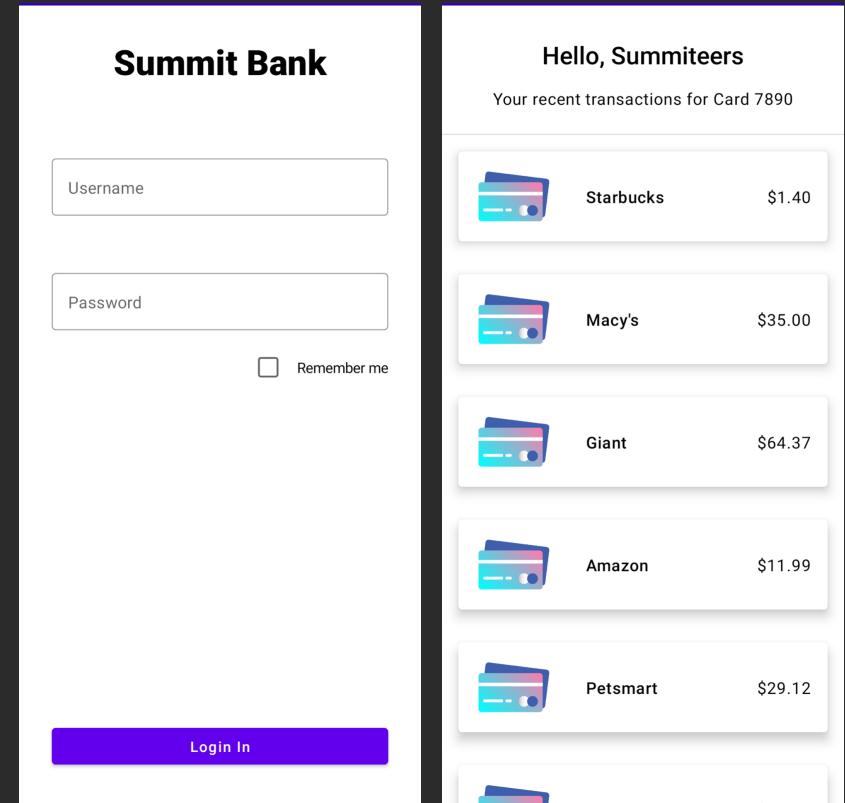
What're we building?

- Topics

- UI Design
- Launching new screens
- Networking (mocked)

- Extra

- Data Persistence
- Processing events on list items



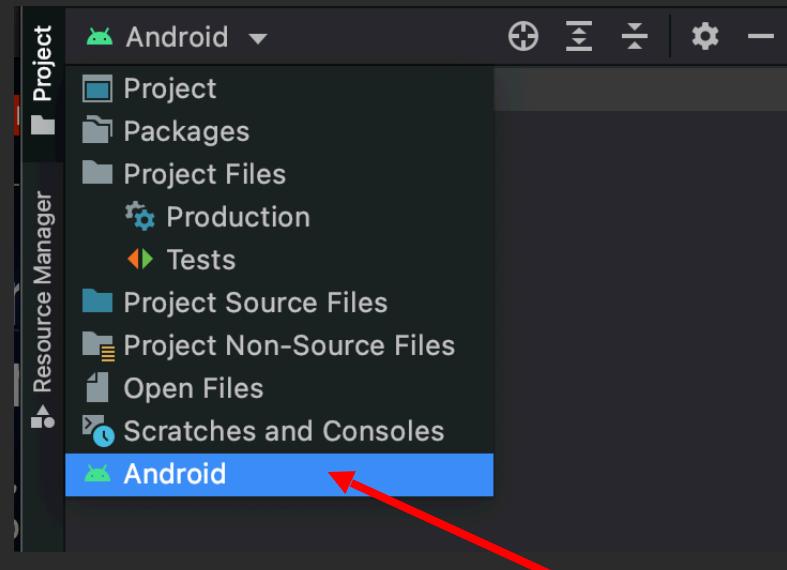
Android Studio

Some Android Studio configurations to make the workshop go more smoothly...

Android Studio

In the left-panel, switch to the “Android” view.

- I prefer this view of the project files as I think it is more concise and easy to navigate.



Click Android if not pre-set.

Android Studio

Open Settings / Preferences

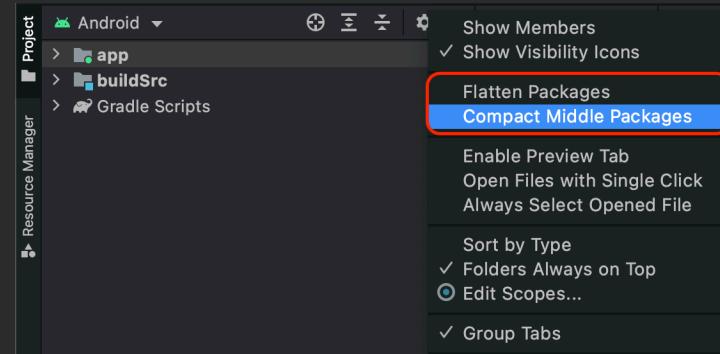
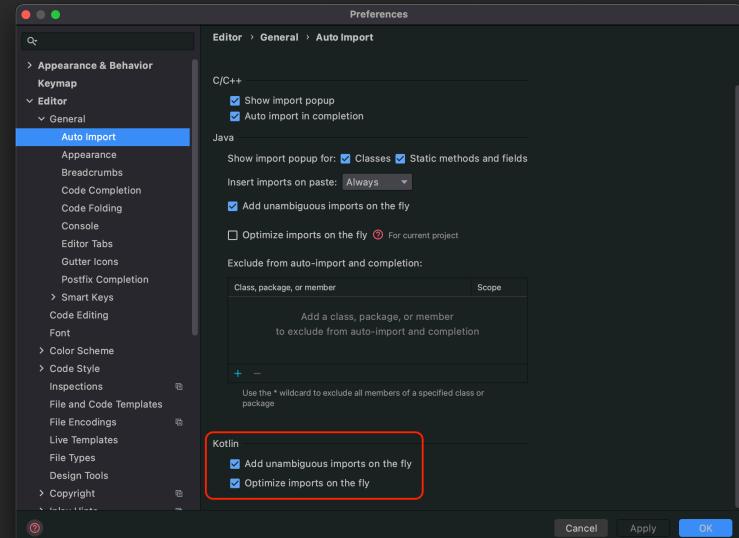
- (Windows) File → Settings
- (OS X) Android Studio → Preferences

Turn on auto-imports!

- Under: Editor → General → Auto Import
 - Enable “Add unambiguous imports...”
 - Enable “Optimize imports...”

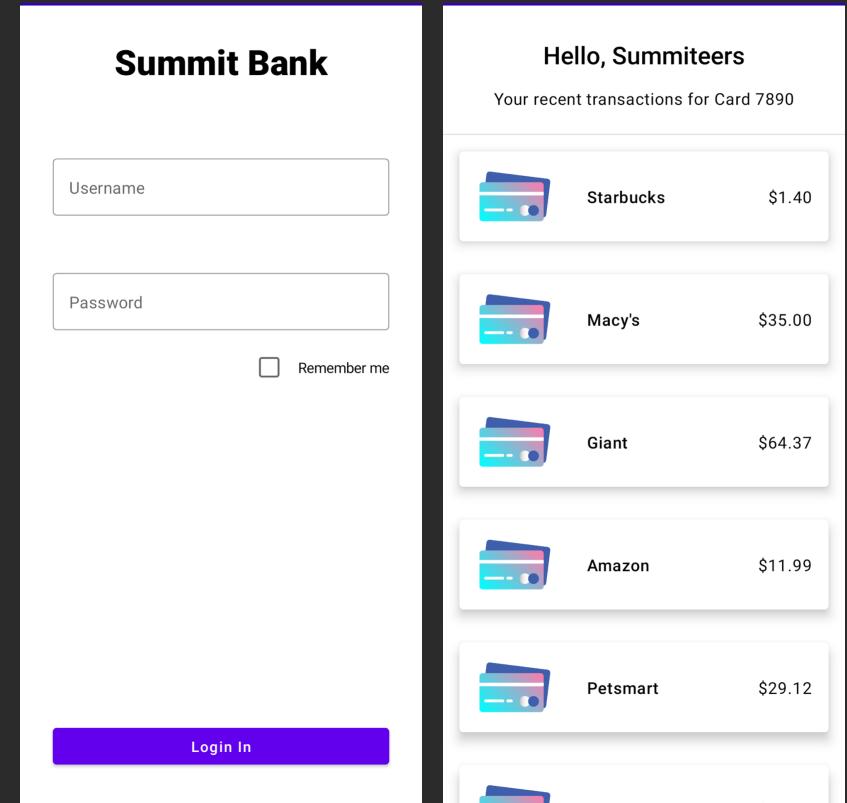
Do not flatten packages!

- Under three dotted settings, uncheck “Flatten / Compact middle packages”

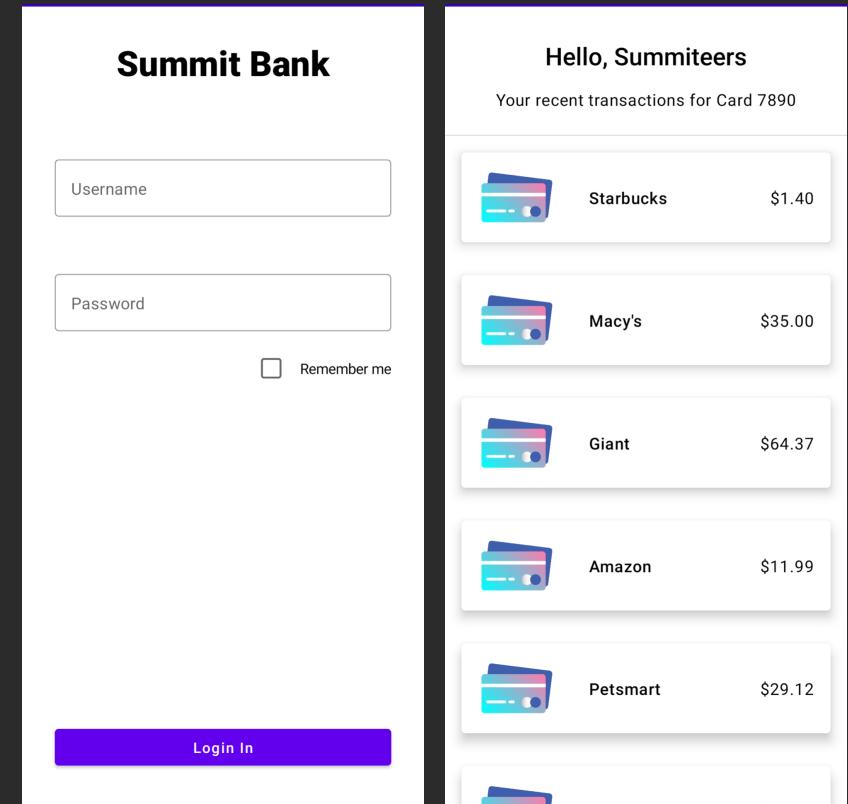


Fragments

- Apps are comprised of “Fragments” that live inside an “Activity”.
- You can think of these as the individual screens.



- Two main components:
 - UI design
 - What does the UI look like
 - UI behaviors
 - How does the UI react to user input

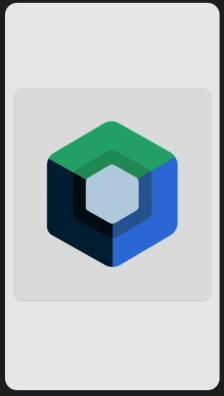


UI Design

UI Design

UIs are built with a Jetpack Compose framework.

```
@Composable
fun JetpackCompose() {
    Card {
        var expanded by remember { mutableStateOf(false) }
        Column(Modifier.clickable { expanded = !expanded }) {
            Image(painterResource(R.drawable.jetpack_compose))
            AnimatedVisibility(expanded) {
                Text(
                    text = "Jetpack Compose",
                    style = MaterialTheme.typography.h2,
                )
            }
        }
    }
}
```

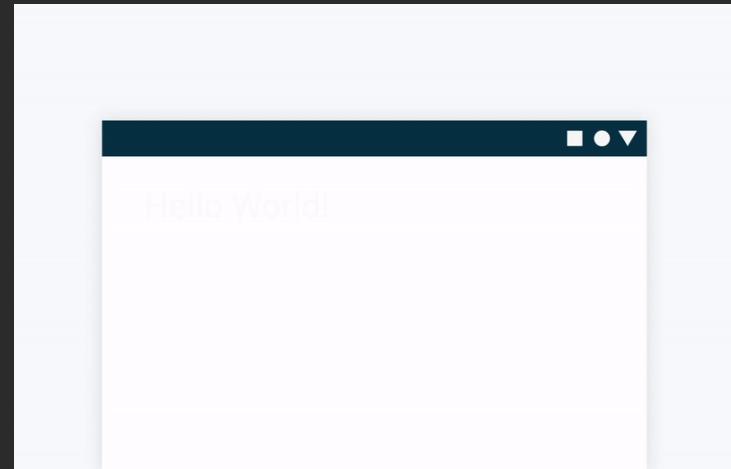


UI Design

Jetpack Compose is built around composable functions.

These functions let you define your app's UI programmatically.

Composable functions can only be called from within other composable functions.

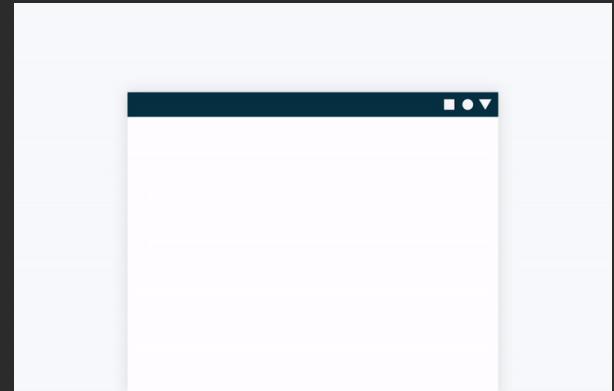


```
class MainActivity : ComponentActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContent {  
            Text("Hello world!")  
        }  
    }  
}
```

UI Design

UI elements are hierarchical,
with elements contained in
other elements.

You build a UI hierarchy by
nesting composable
functions.

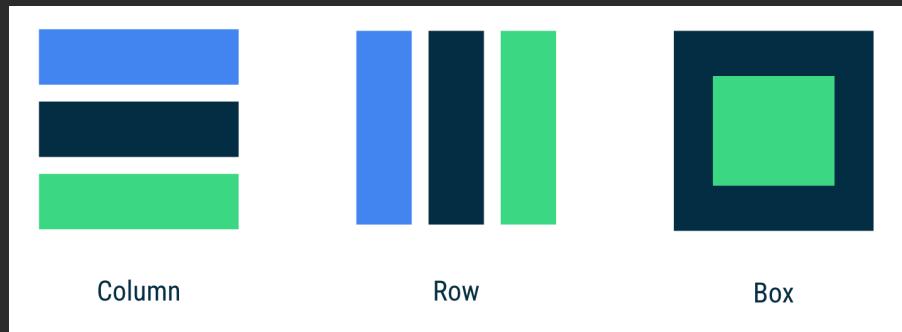


```
@Composable
fun NewsStory() {
    Column(
        modifier = Modifier.padding(16.dp)
    ) {
        Image(
            painter = painterResource(R.drawable.header),
            contentDescription = null
        )

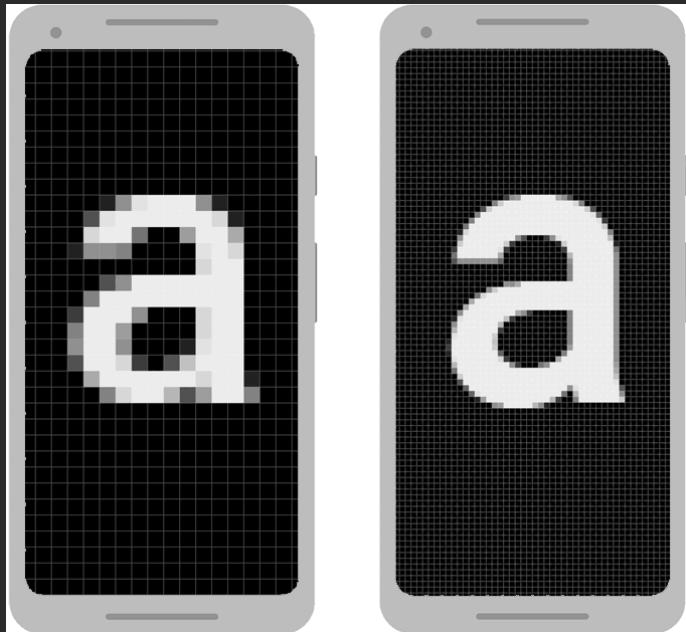
        Text("A day in Shark Fin Cove")
        Text("Davenport, California")
        Text("December 2018")
    }
}
```

UI Layouts

- **Column:** to place items vertically on the screen.
- **Row:** to place items horizontally on the screen.
- **Box:** to put one element on top of another.

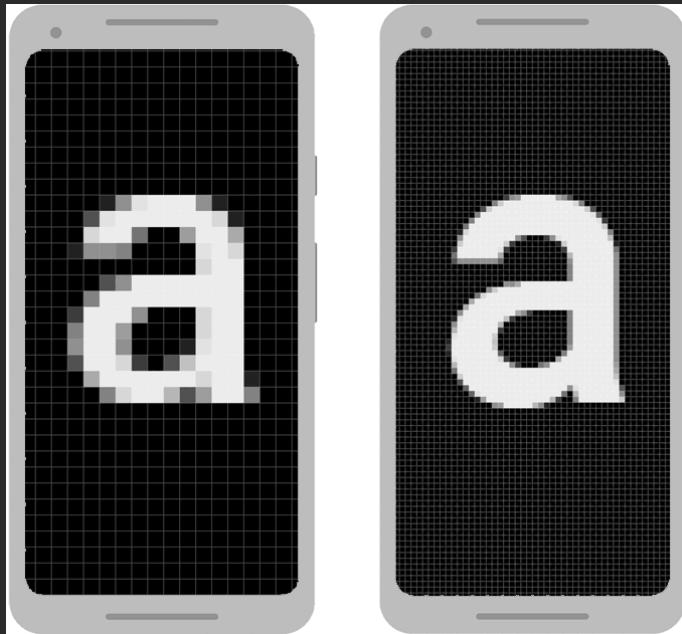


UI Layouts



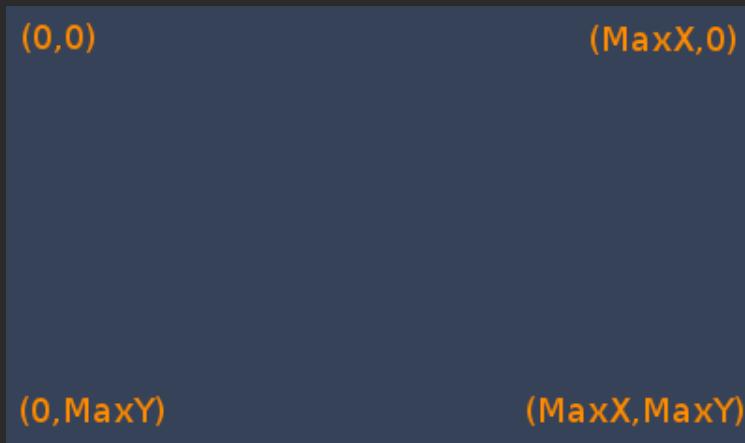
- On Android, we don't measure in pixels.
 - Two screens of the *same size* can have *different densities*.
- Android provides “dp” or *density-independent pixels* as a measure.

UI Layouts



- There's also “sp” or *scalable pixels*, which are used with text.
- This scales with device font size setting.

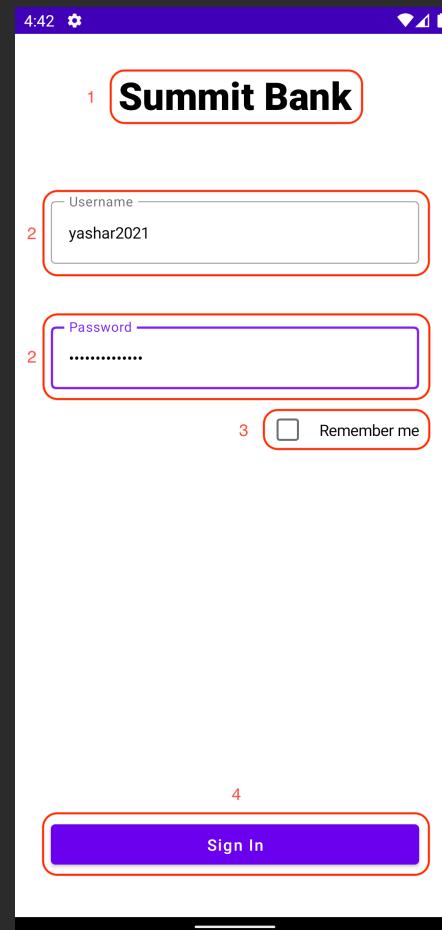
UI Layouts



- Android coordinate system starts from top left and extends to bottom right.
- Default position of all UI is at top left edge of (0,0)

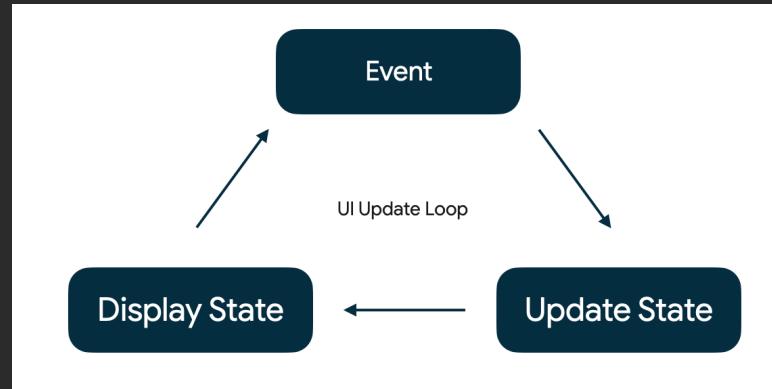
Login Screen UI

Let's build a simple login form, without the “Remember Me” checkbox.



State

- State is at the core of compose
- Textfield did not update since state was not updated
- State can be stored and updated using *mutableState*
- UI updating for an updated state is called “recomposition”

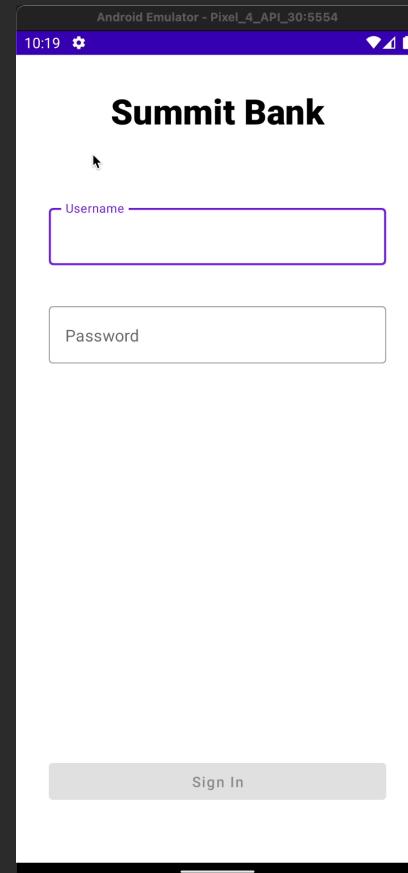


UI Behaviors

Login Screen Behaviors

Now we can actually write some code to add behaviors to the UI.

- Disable the “Sign In” button unless text was inputted.
- Show loading indicator while login



Login Screen Behaviors

Android apps use an “event-driven” programming style.

Rather than having a main() function and executing sequentially, we write code that responds to events.

- e.g. when the screen loads, when the user performs some input, etc.

Login Screen Behaviors

As an Android developer, you may want to react when:

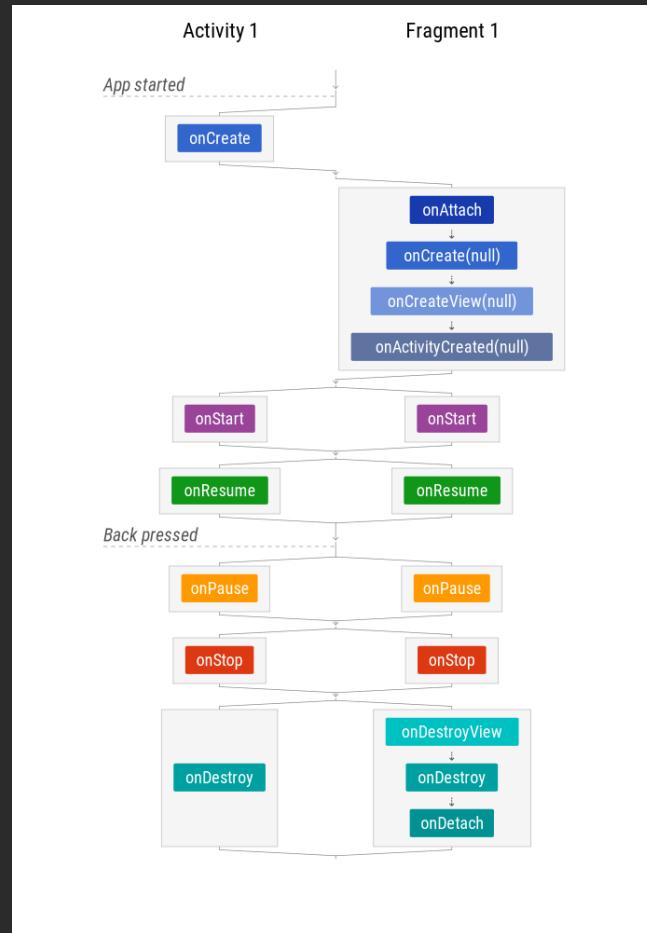
- When the screen is created
- When the user puts the app in the background
- When a popup is obscuring your screen
- etc.

Login Screen Behaviors

Android notifies your Activity and Fragment of these events via callbacks:

- onCreate
- onStart
- onStop
- etc.

This is called the “lifecycle”!



Login Screen Behaviors

The framework may decide to destroy or re-create a UI in response to certain user actions or device events that are completely out of your control. One such example is device screen rotation (a.k.a configuration change).

Networking

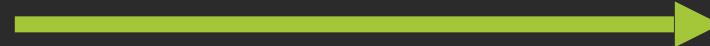
Networking

Usually, signing in involves making a network call.

- Server will validate credentials and return user data.



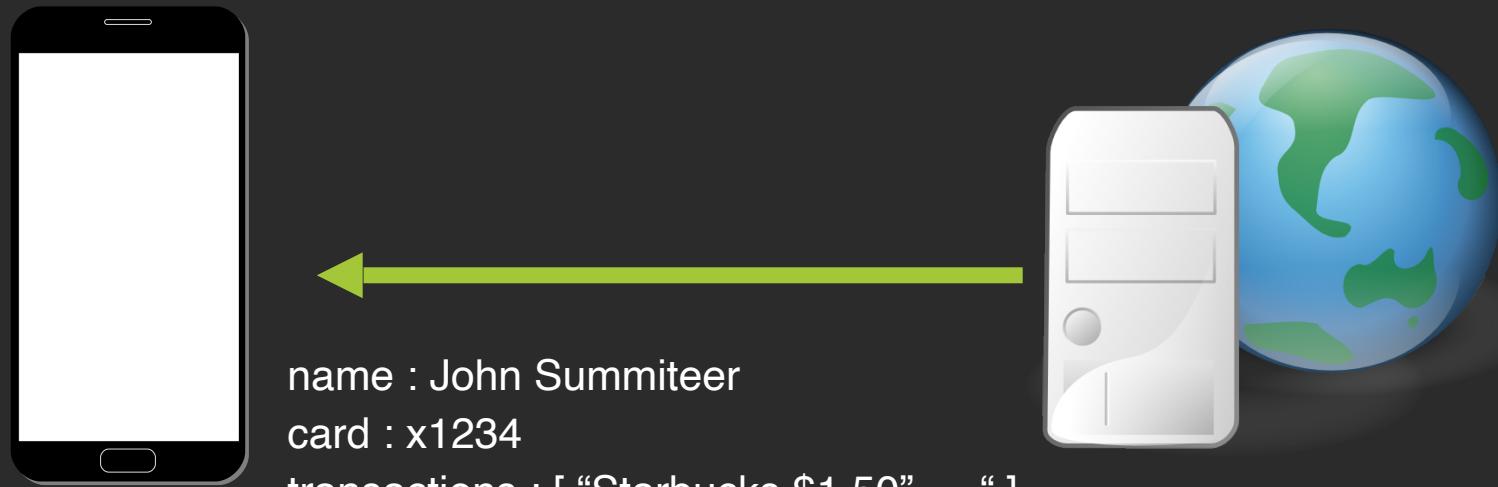
username : summituser01
password : abc123



Networking

Usually, signing in involves making a network call.

- Server will validate credentials and return user data.



Networking

Networking can be a complex subject in Android.

- Need to be on a background thread.
- Handling connection errors.
- Creating a network request.
- Parsing a network response.

Networking

The server sends back data using some standardized format.

JavaScript Object Notation (JSON) is fairly common.

- You'd also need to code to parse the data you need from the server response.

```
1  {
2    "name": "Summiteers",
3    "cardLastFour": "7890",
4    "transactions": [
5      {
6        "merchant": "Starbucks",
7        "amount": "$1.40"
8      },
9      {
10        "merchant": "Macy's",
11        "amount": "$35.00"
12      },
13      {
14        "merchant": "Giant",
15        "amount": "$64.37"
16      },
17      {
```

Networking

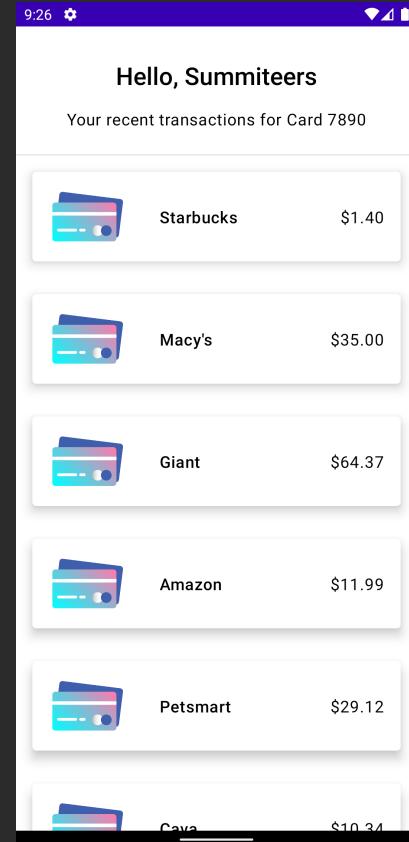
For the purpose of the workshop let's use a prebuilt “LoginService” class to retrieve user details after they click the “Sign In” button...

- User's name (“Summiteers”)
- User's card last 4 (“7890”)
- List of transactions (“Starbucks \$1.50, Amazon \$32.10, ...)

Second Fragment

We'll display the user's data on a separate Fragment called the "SummaryFragment".

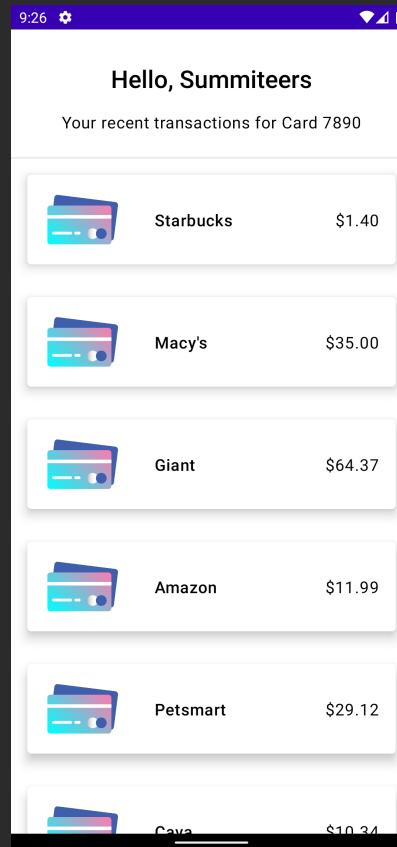
We need to create this new Fragment and launch it from our LoginFragment (passing it the data returned by the server).



Starting a second activity

On login success, we want to start the “SummaryFragment”

We also want to pass the user's details to that fragment to display.



Fragment Transaction

New fragments are launched using “FragmentManager”.

With fragmentManager, you:

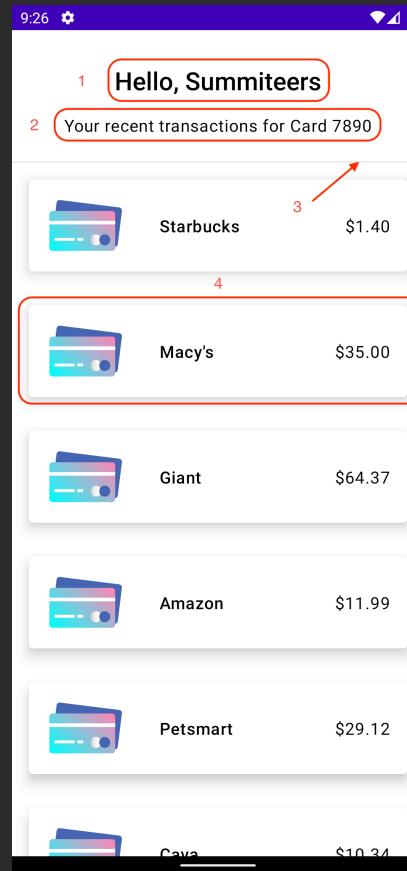
- Specify which fragment should be launched
- How the fragment should be launched
- Where the fragment should be launched

```
parentFragmentManager  
    .beginTransaction()  
    .replace(R.id.main_container, SummaryFragment())  
    .commit()
```

Second Fragment

First let's review what we need to create:

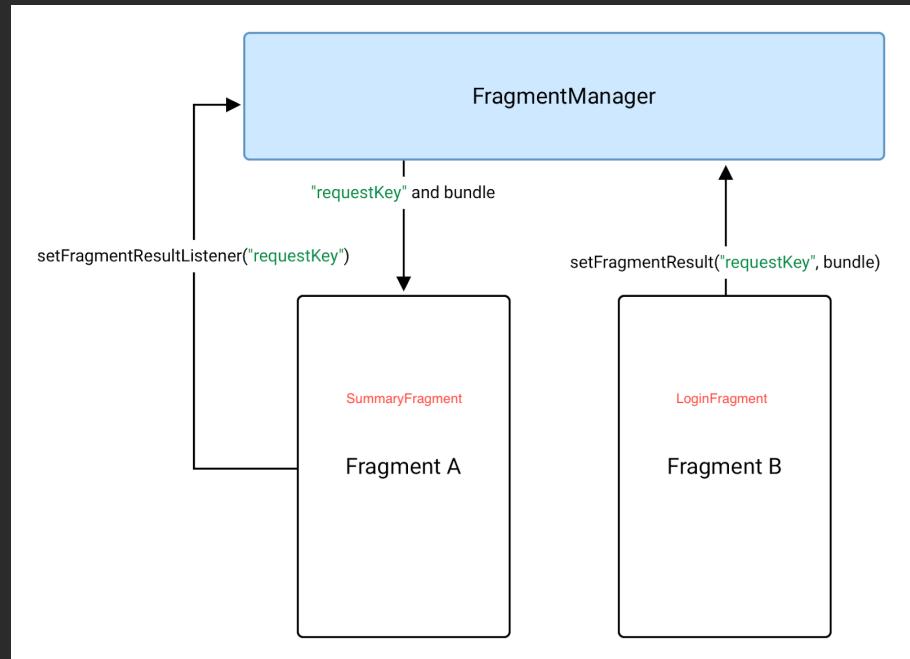
1. A simple text component that we have already created before
2. Same thing, a smaller text component!
3. A divider line component
4. A transaction card component with picture and texts representing that transaction



Fragment Result

You can pass information between fragments using FragmentResult.

It's a storage for fragments where multiple fragments can use a key to store specific data.

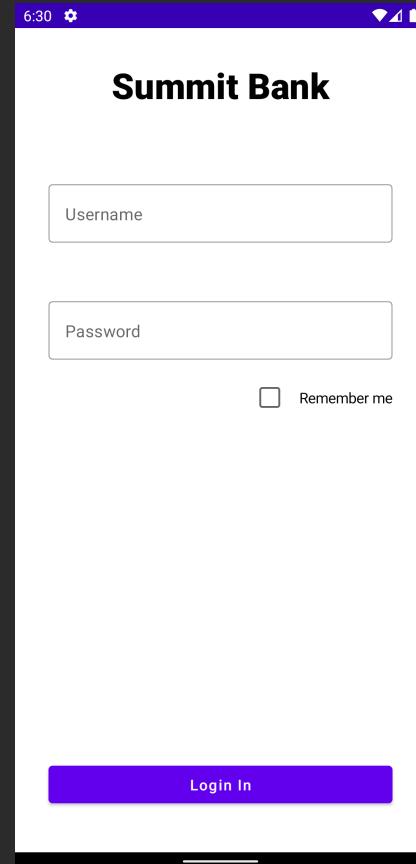


Extras

Extra features!

Let's add on a “Remember Me” feature to remember user credentials.

And also an ability to select each individual transactions.



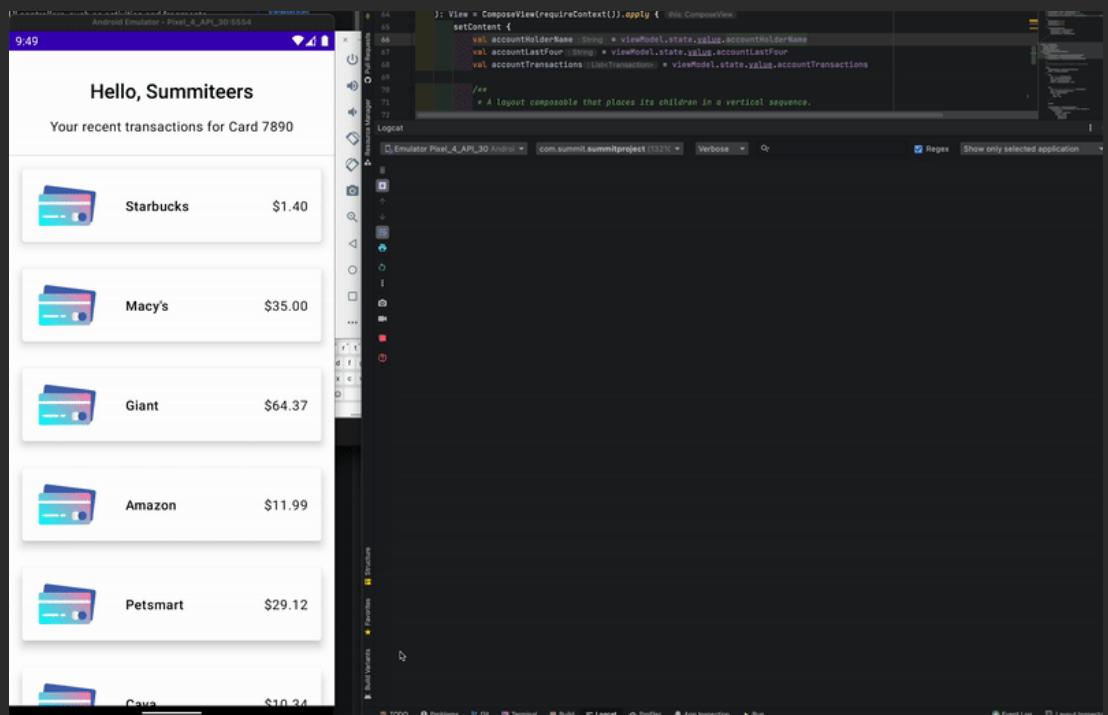
Simple data storage

You can use “SharedPreferences” to store simple key-value pairs to the app’s private storage.

```
/**  
 * The key under which the **username** is stored in sharedPreference.  
 */  
const val PREF_USERNAME = "username"  
  
/**  
 * The key under which the **remember_me** is stored in sharedPreference.  
 */  
const val PREF_REMEMBER_ME = "remember_me"
```

Reacting to list interactions

Let's also add a simple behavior when the user taps on a transaction.

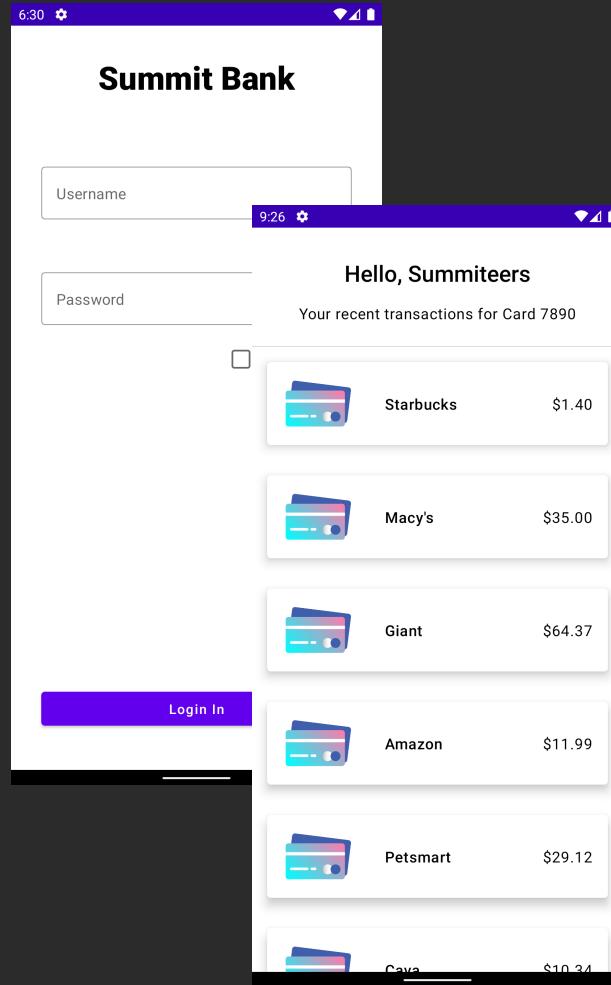


Wrapping Up

What did we build?

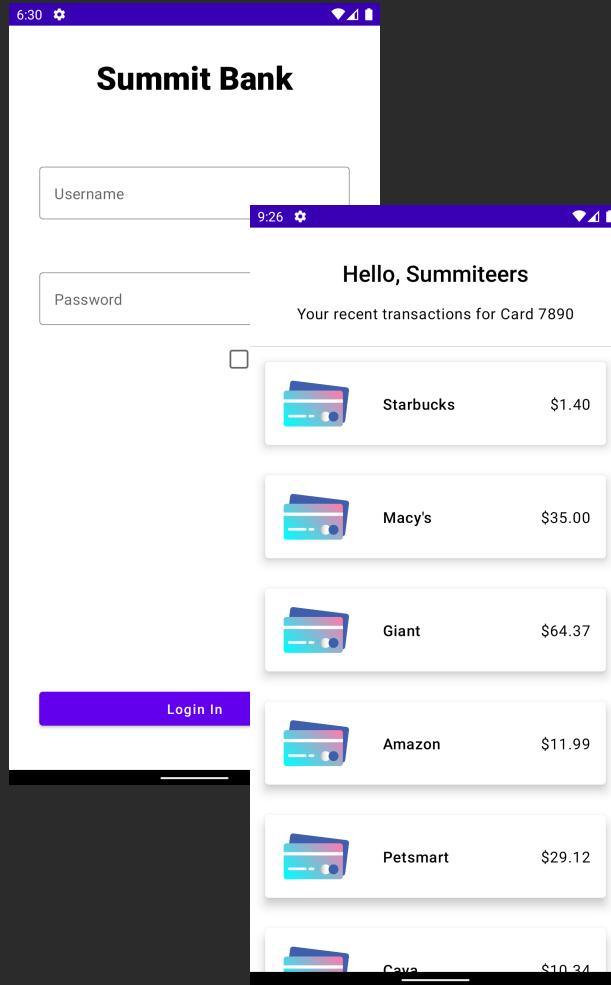
A simple app that allows the user to login & view a list of recent transactions:

- Two screens
- UI with Jetpack Compose
- Basic networking and data storage



What did we build?

If you want the completed project with comments, checkout the “Completed” or “Extra” folder.



Android at Capital One

- Over 200 Android (and iOS) devs work on the main Capital One Mobile app alone!
 - There are other apps built in-house too - like our CreditWise app.
 - A few devs have Flutter side-projects. Our conference app and website for the Android Summit conference is built in Flutter.



Want to continue learning Android?

- Start learning modern Android using Google Codelabs
 - [Android Kotlin Fundamentals](#)
- Check out this article in the C1 Tech Blog about helpful resources for learning Android
 - [The Most Helpful Resources From My First Year in Android](#)

Other Resources

“I want to learn more about compose”

- <https://developer.android.com/jetpack/compose>
- <https://developer.android.com/courses/pathways/compose>

Other Resources

“I want to learn more about networking”

- Basics:
<https://developer.android.com/training/basics/network-ops/connecting>
- Popular libraries for making network calls:
<http://square.github.io/okhttp/>
<https://square.github.io/retrofit/>
- Threading:
<https://developer.android.com/guide/components/processes-and-threads>
- JSON parsing:
https://www.tutorialspoint.com/android/android_json_parser.htm

Other Resources

“I want to interface with an API”

- Capital One’s Hackathon API (Nessie)
<http://api.reimaginebanking.com/>
- “Host Your Own” Fake API:
<https://www.mocky.io/>

Other Resources

“I want to use some hardware features”

- Camera:
<https://developer.android.com/guide/topics/media/camera>
- Sensors:
https://developer.android.com/guide/topics/sensors/sensors_overview
- Bluetooth:
<https://developer.android.com/guide/topics/connectivity/bluetooth>
- NFC:
<https://developer.android.com/guide/topics/connectivity/nfc/>

Other Resources

“I want a backend for user credentials and data”

- Firebase Authentication (free):
<https://firebase.google.com/docs/auth/>
- Firebase Realtime DB (also free):
<https://firebase.google.com/docs/database/>

Thanks for attending!

Find us on Slack and LinkedIn:

- Slack: @yashar
 - <https://www.linkedin.com/in/yaxarat/>
- Slack: @nickcapurso
 - <https://www.linkedin.com/in/nickcapurso>