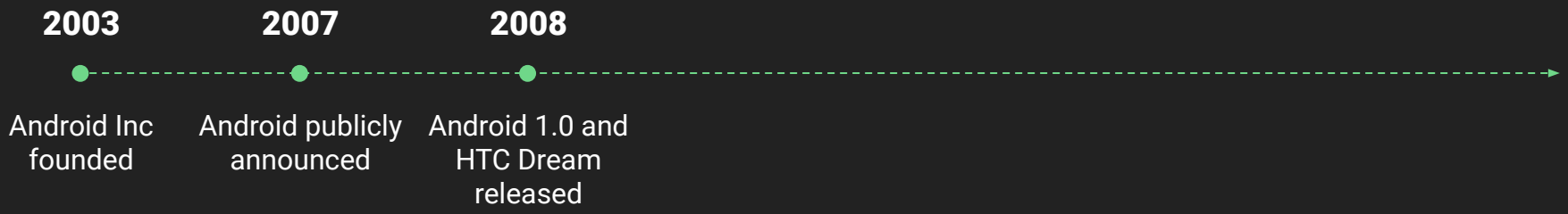
Engineering
Awesome
Conference.

Jetpack Compose

Where we came from



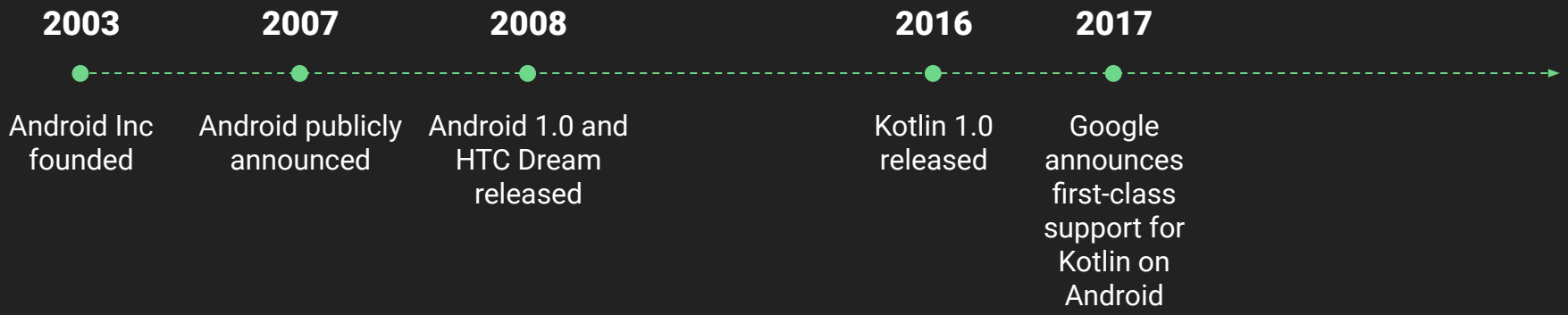
Where we came from

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Hello world"/>

</FrameLayout>
```

Where we came from

```
public class MainActivity extends AppCompatActivity {

  @Override
  public void onCreate(Bundle savedInstanceState){
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);
     TextView textView = (TextView) findViewById(R.id.textView);
      textView.setOnClickListener(new OnClickListener() {
            // Stuff
      });
  }
}
```

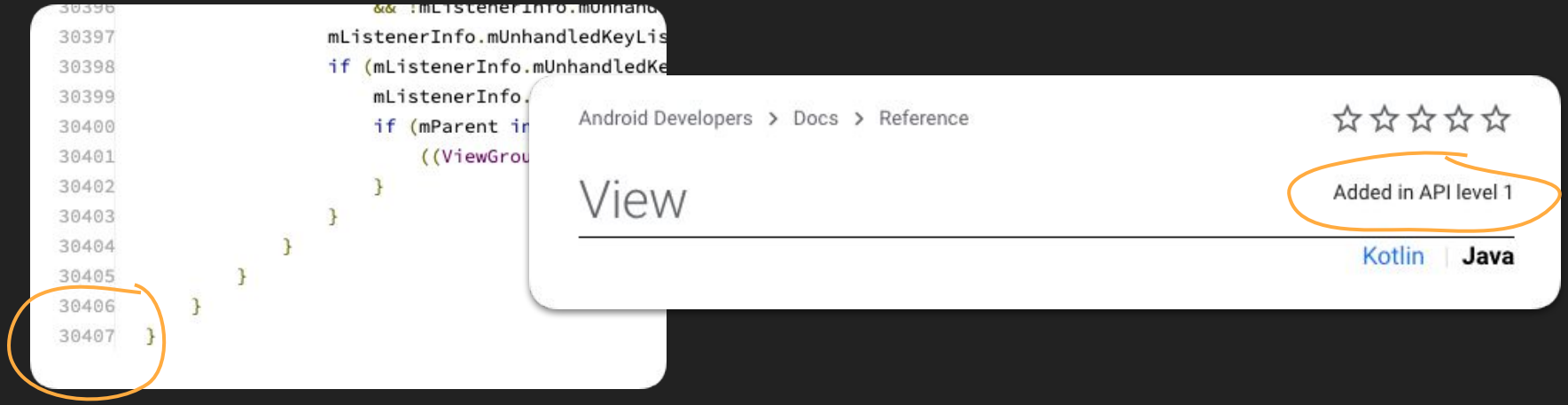
Where we came from



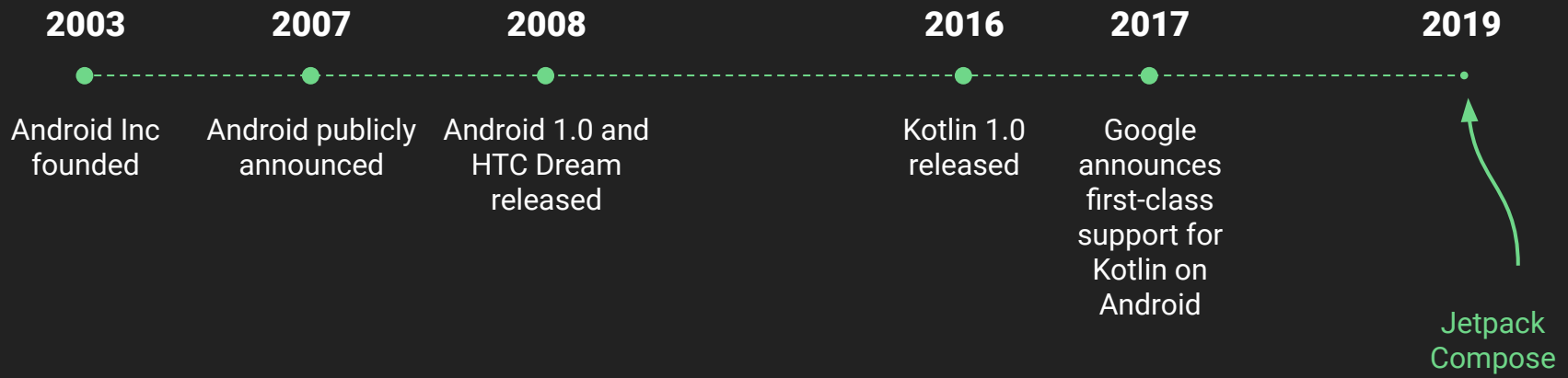
Where we came from

```
class MainActivity : AppCompatActivity() {
     override fun onCreate(savedInstanceState: Bundle?) {
          setContentView(R.layout.activity_main)

          textView.setOnClickListener { /* Stuff */ }
     }

}
```

Where we came from



Where we came from



Build better apps faster with Jetpack Compose

Jetpack Compose is Android's modern toolkit for building native UI. It simplifies and accelerates UI development on Android. Quickly bring your app to life with less code, powerful tools, and intuitive Kotlin APIs.

VIEW TUTORIAL

Less code

Do more with less code and avoid entire classes of bugs, so code is simple and easy to maintain.

Intuitive

Just describe your UI, and Compose takes care of the rest. As app state changes, your UI automatically updates.

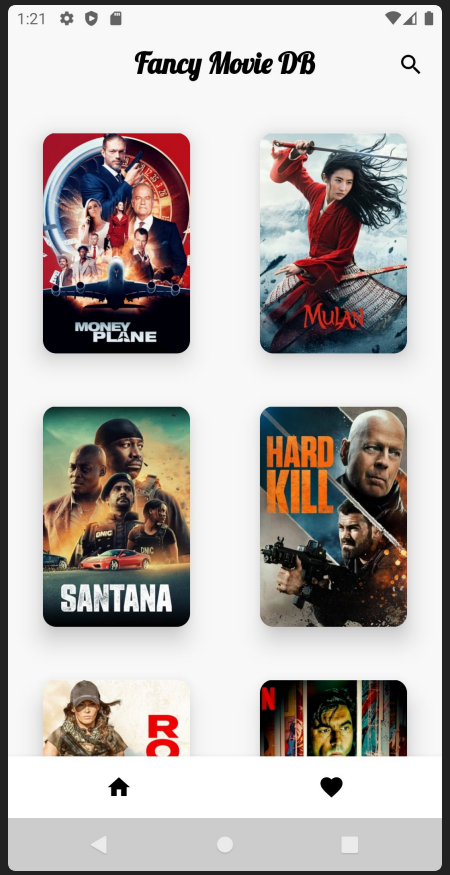
Accelerate Development

Compatible with all your existing code so you can adopt when and where you want. Iterate fast with live previews and full Android Studio support.

Powerful

Create beautiful apps with direct access to t
Android platform APIs and built-in support f
Material Design, Dark theme, animations, an
more.

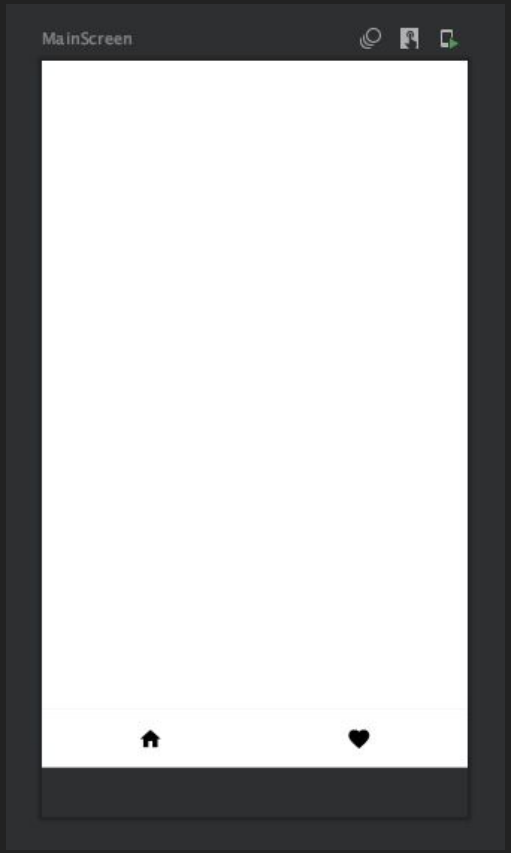
So let’s make something



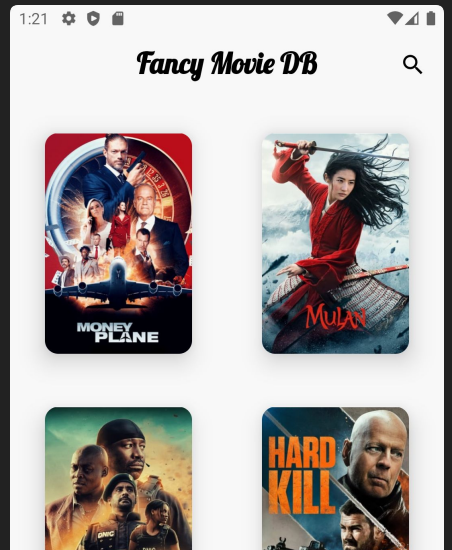
Composing

```
override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MovieDBTheme {
                MainScreen()
            }
        }
    }
}

@Preview
@Composable fun MainScreen() {
    Text("Hello world")
}
```

```
@Preview
@Composable fun MainScreen() {
      Scaffold(
          bodyContent = {
                  HomeScreen()
          },
          bottomBar = {
                BottomAppBar(
                    backgroundColor = Color.White,
                    elevation = 8.dp
                ) {
                    IconButton(onClick = {},) {
                        Icon(Icons.Filled.Home)
                    }
                    IconButton(onClick = {},) {
                        Icon(Icons.Filled.Favorite)
                    }
                }
      })
}
```

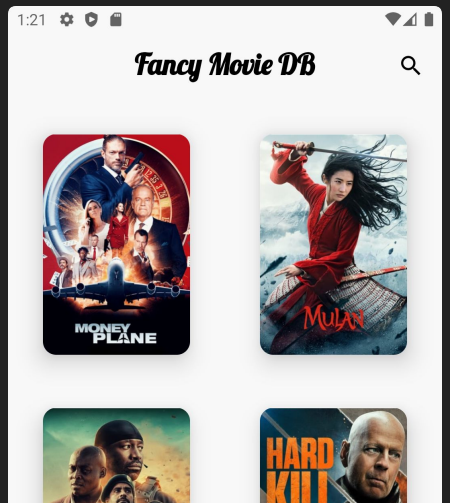


```
@Preview
@Composable fun HomeScreen() {
      Column(
              Modifier
                  .background(color = Color(0xFFf8f8f8))
                  .fillMaxHeight()
          ) {
              HomeTop()
              MovieList()
          }
}
```

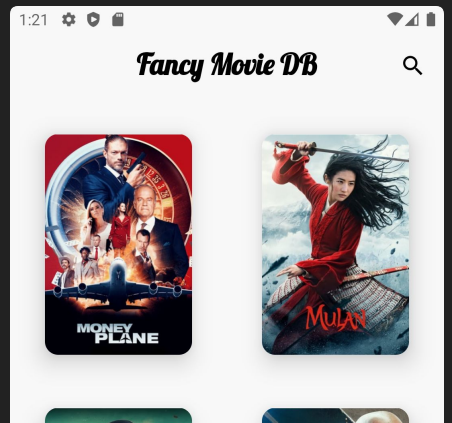


```
@Preview
@Composable fun HomeTop() {
      Row() {
          Spacer(Modifier.weight(1f))
          Text(
              "Fancy Movie DB",
              modifier = Modifier.weight(3f)...
          )
          IconButton(
              onClick = {},
              modifier = Modifier.weight(1f)
          ) {
              Icon(Icons.Filled.Search)
          }
      }
}
```

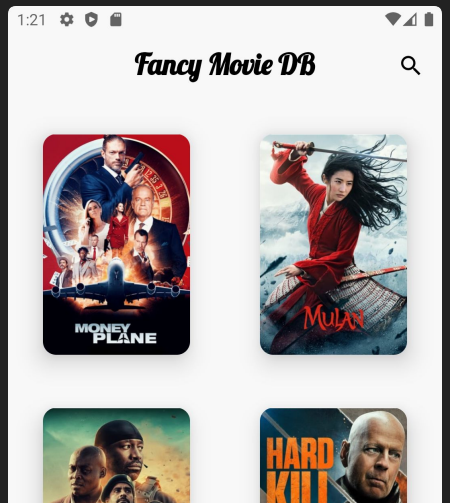
```
@Preview
@Composable fun MovieList() {
      val movies = emptyList<Movie>()
      val chunked = movies.chunked(2)
      LazyColumnFor(items = chunked) { subList ->
            Row() {
                  subList.forEach { movie ->
                      // TODO
                  }
            }
      }

}
```



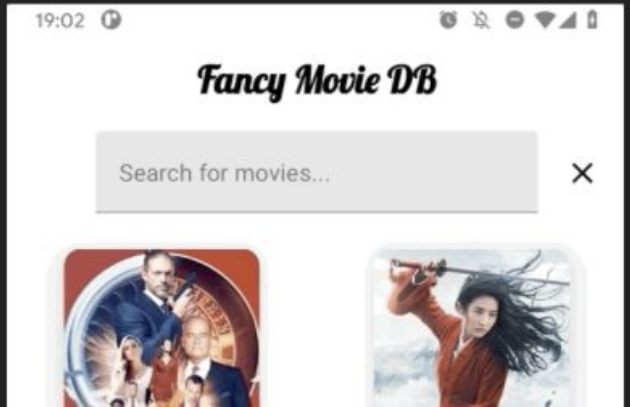
```
@Preview
@Composable fun MovieItem(movie: Movie) {
      Surface(
          elevation = 16.dp,
          shape = RoundedCornerShape(12.dp)
      ) {
          CoilImageWithCrossfade(
              data = movie.thumbUrl,
              contentScale = ContentScale.Crop,
          )
      }
}
```



```
@Preview
@Composable fun MovieList() {
      val movies = emptyList<Movie>()
      val chunked = movies.chunked(2)
      LazyColumnFor(items = chunked) { subList ->
            Row() {
                  subList.forEach { movie ->
                      MovieItem(movie)
                  }
            }
      }

}
```

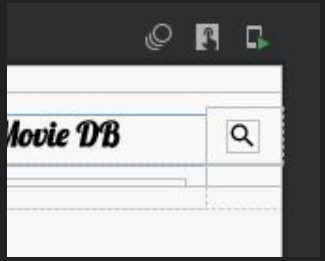


```
@Preview
@Composable fun HomeTop() {
    Row() {
        ...
    }
    Row() {
        Spacer()
        TextField(
            value = "",
            placeholder = {
                Text("Search for movies...")
            },
            onValueChange = {}
        )
        IconButton(onClick = {}) {
            Icon(Icons.Filled.Close)
        }
    }
}
```



State

```
@Preview
@Composable fun HomeTop() {
      val searchVisible = remember { mutableStateOf(false) }

      Row(...)
      Row(...)
}
```



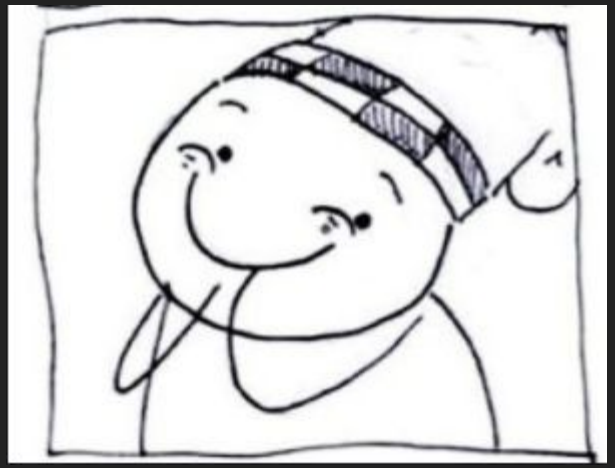
Animation

```
@Preview
@Composable fun HomeTop() {
      val searchVisible = remember { mutableStateOf(false) }

      val state = transition(
         definition = searchTransitionDefinition,
         initState = SearchState.Hidden,
         toState = if (searchVisible.value) SearchState.Visible else SearchState.Hidden
      )

      Row(...)
      Row(...)
}
```

```
enum class SearchState {
    Visible, Hidden
}

private val searchTransitionDefinition = transitionDefinition<SearchState> {
    state(SearchState.Visible) {
        this[searchButtonSizeState] = 0.dp
        this[searchButtonAlphaState] = 0f
        this[searchContentSizeState] = 56.dp
        this[searchContentAlphaState] = 1f
        this[topBarSizeState] = 64.dp
    }
    state(SearchState.Hidden) {
        this[searchButtonSizeState] = 48.dp
        this[searchButtonAlphaState] = 1f
        this[searchContentSizeState] = 0.dp
        this[searchContentAlphaState] = 0f
        this[topBarSizeState] = 92.dp
    }

    transition(fromState = SearchState.Hidden, toState = SearchState.Visible) {
         searchButtonSizeState using tween(durationMillis = 200)
        ...
    }
    transition(fromState = SearchState.Visible, toState = SearchState.Hidden) {
        ...
    }
}
```

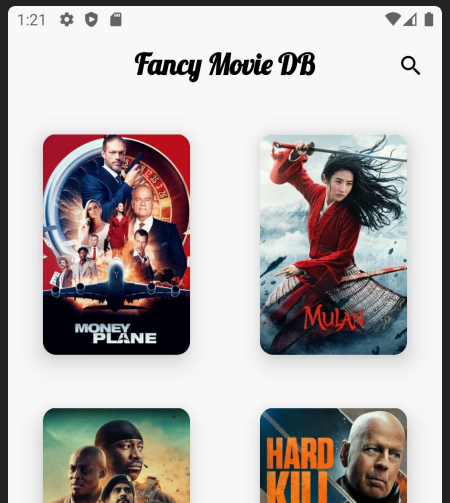
```
@Preview
@Composable fun HomeTop() {
      val searchVisible = remember { mutableStateOf(false) }

      val state = transition(
         definition = searchTransitionDefinition,
         initState = SearchState.Hidden,
         toState = if (searchVisible.value) SearchState.Visible else SearchState.Hidden
      )

      Row(...)
      Row(...)
}
```

```
@Preview
@Composable fun HomeTop() {
      val searchVisible = remember { mutableStateOf(false) }

      val state = transition(...)

      Row(...) {
            IconButton(
                  onClick = { searchVisible.value = !searchVisible.value },
                  modifier = Modifier
                      .preferredSize(state[searchButtonSizeState])
                      .drawOpacity(state[searchButtonAlphaState])
              ) {
                  Icon(Icons.Filled.Search)
              }

      }
      Row(
              modifier = Modifier.preferredHeight(state[searchContentSizeState])
          )
}
```




19:02
Fancy Movie DB

```
@Preview
@Composable fun MovieList() {
      val movies = emptyList<Movie>()
      val chunked = movies.chunked(2)
      LazyColumnFor(items = chunked) { subList ->
            Row() {
                  subList.forEach { movie ->
                      MovieItem(movie)
                  }
            }
      }

}
```



```
class HomeViewModel {
      val movies = mutableStateOf(emptyList<Movie>())
}
```

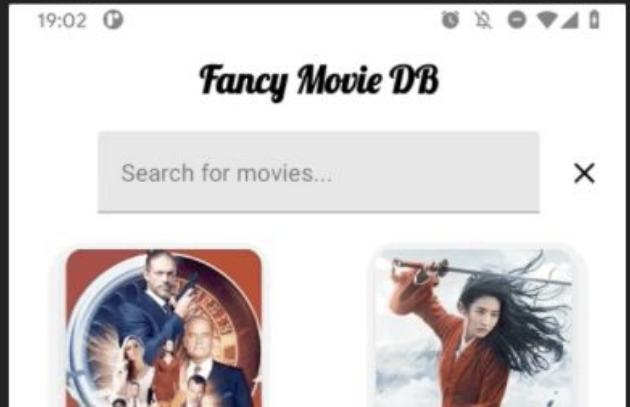
Ambients and Providers

```
val HomeAmbient = staticAmbientOf<HomeViewModel> {
      error("HomeViewModel not provided")
}

class HomeViewModel {
      val movies = mutableStateOf(emptyList<Movie>())
}


MainScreen() {
      Providers(HomeAmbient provides HomeViewModel()) {
            HomeScreen()
      }
}
```

```
@Preview
@Composable
fun HomeScreen() {
    val homeViewModel = HomeAmbient.current
    val movies = remember { homeViewModel.movies }

    Column(...) {
        HomeTop()
        MovieList(movies.value)
    }
}
```

Searching

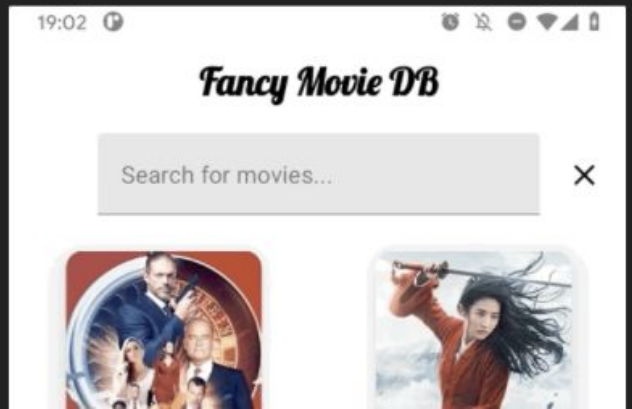
```
@Preview
@Composable fun HomeTop() {
      val homeViewModel = HomeViewModelAmbient.current
      ...
          TextField(
              value = "",
              placeholder = {
                  Text("Search for movies...")
              },
              onValueChange = {
                  homeViewModel.updateSearchText(it)
              }

          )
      ...
}
```



```
class HomeViewModel {
      val movies = mutableStateOf(emptyList<Movie>())

      private val searchMoviesUseCase = SearchMoviesUseCase()
      private val discoverMoviesUseCase = DiscoverMoviesUseCase()
      private val scope = CoroutineScope(SupervisorJob() + Dispatchers.Main.immediate)

      init { scope.launch { movies.value = discoverMoviesUseCase.discoverMovies() } }

      fun updateSearchText(textFieldValue: TextFieldValue) {
            if(textFieldValue.text.length > 2) {
                  scope.launch { movies.value = searchMoviesUseCase.search(textFieldValue.text)
            }
      }
}
```

Clearing search

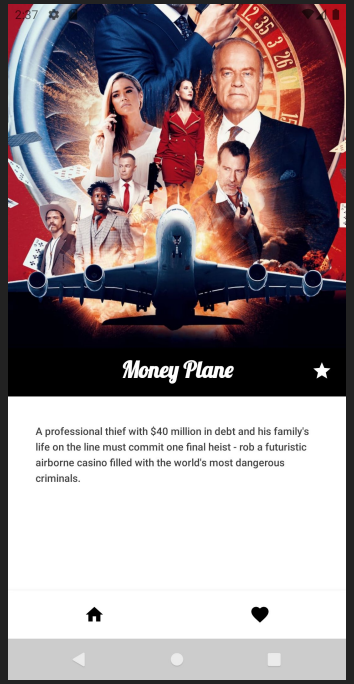
```
@Preview
@Composable fun HomeTop() {
      val homeViewModel = HomeViewModelAmbient.current
      ...
          IconButton(
                  onClick = {
                      homeViewModel.clearedSearch()
                      searchVisible.value = !searchVisible.value
                  },
              ) {
                  Icon(Icons.Filled.Close)
              }

      ...
}
```

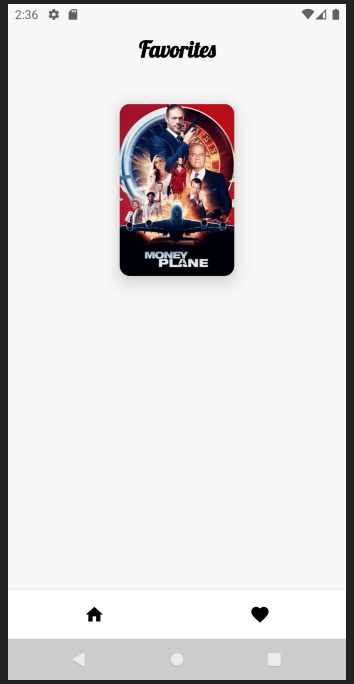


```
class HomeViewModel {
      val movies = mutableStateOf(emptyList<Movie>())

      …
      init { scope.launch { movies.value = discoverMoviesUseCase.discoverMovies() } }

      fun clearedSearch() {
          scope.launch { movies.value = discoverMoviesUseCase.discoverMovies() }
      }

      …
}
```

Engineering
Awesome
Conference.

Favorites





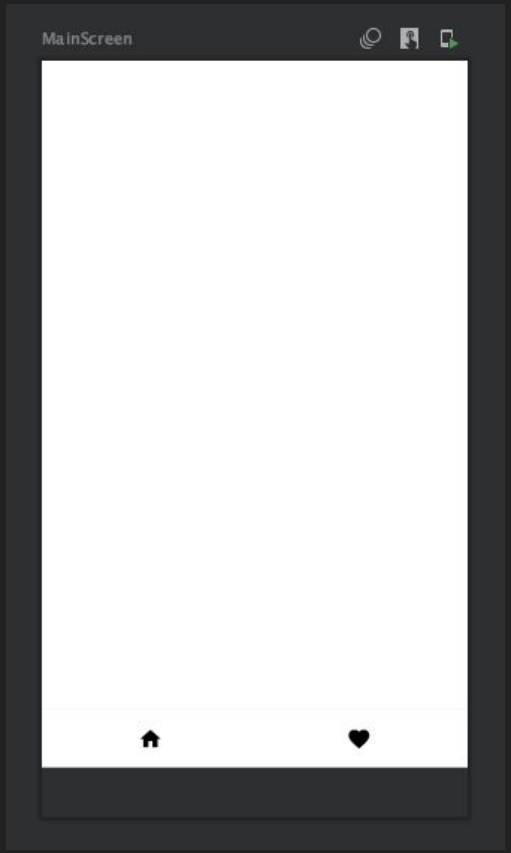
FavoritesViewModel

```
class FavoritesViewModel {
    private val _movies = MutableLiveData<List<Movie>>(emptyList())
    val movies: LiveData<List<Movie>> = _movies

    fun toggleMovie(movie: Movie) {
        val movies = movies.value ?: return
        if(movies.contains(movie)) {
            _movies.value = movies.filterNot { it == movie }
        } else {
            _movies.value = movies + movie
        }
    }
}
```

MovieScreen

```
@Composable
fun FavoriteButton(movie: Movie) {
    val favoritesViewModel = FavoritesViewModelAmbient.current
    val isFavorite = favoritesViewModel.movies.observeAsState().value?.contains(movie) == true
    val iconColor = if (isFavorite) Color.White else Color.DarkGray

    IconButton(
        onClick = {
            favoritesViewModel.toggleMovie(movie)
        },
        Icon(Icons.Filled.Star, tint = iconColor)
    }
}
```

```
@Preview
@Composable fun MainScreen() {
      Scaffold(
          bodyContent = {
                  HomeScreen()
          },
          bottomBar = {
                BottomAppBar(
                    backgroundColor = Color.White,
                    elevation = 8.dp
                ) {
                    IconButton(onClick = {},) {
                        Icon(Icons.Filled.Home)
                    }
                    IconButton(onClick = {},) {
                        Icon(Icons.Filled.Favorite)
                    }
                }
      })
}
```



Navigation

```
sealed class Screens {
    object Home: Screens()
    object Favorites: Screens()
    data class MovieScreen(val movie: Movie): Screens()
}
```

Navigation

```
val NavigationAmbient = staticAmbientOf { NavigationViewModel() }

class NavigationViewModel {
    val currentScreen = mutableStateOf<Screens>(Screens.Home)

    val canGoBack: Boolean
        get() = currentScreen.value != Screens.Home

    fun changeScreen(screen: Screens) {
        currentScreen.value = screen
    }

    fun pop() {
        currentScreen.value = Screens.Home
    }
}
```

Navigation

```
@Composable
fun MainScreen() {
    val navigationViewModel = NavigationAmbient.current
    val screens = navigationViewModel.currentScreen

    Scaffold(
        bodyContent = {
            Crossfade(current = screens.value) { screen ->
                when (screen) {
                    is Screens.Home -> HomeScreen()
                    is Screens.Favorites -> FavoritesScreen()
                    is Screens.MovieScreen -> MovieScreen(movie = screen.movie)
                }
            }
        },
        bottomBar = {
            ...
        }
    )
}
```

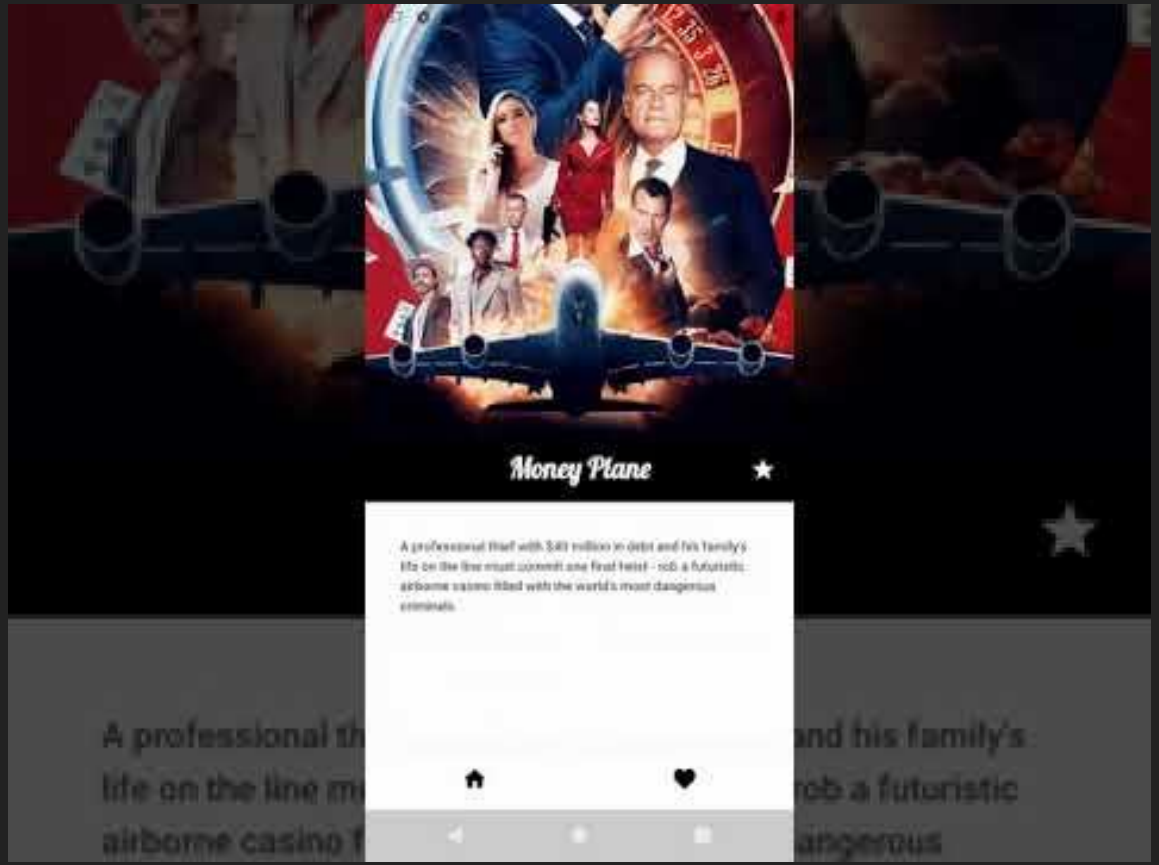
Navigation

```
@Preview
@Composable fun MovieItem(movie: Movie) {
      val navigationViewModel = NavigationAmbient.current

      Box(modifier = Modifier.clickable(onClick = {
            navigationViewModel.changeScreen(Screens.MovieScreen(movie))
      })) {
            Surface(...) {
                  CoilImageWithCrossfade(data = movie.thumbUrl)
            }
      }

}
```

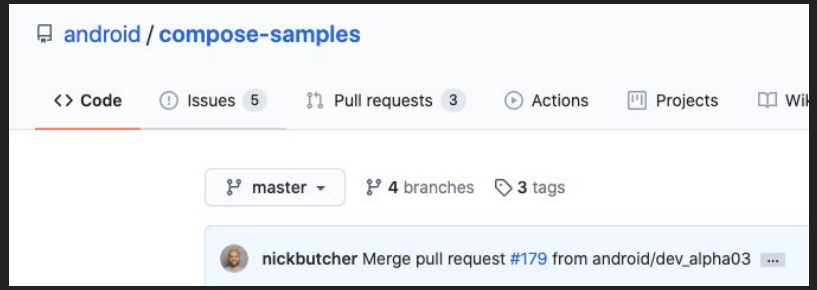
onBackPressed

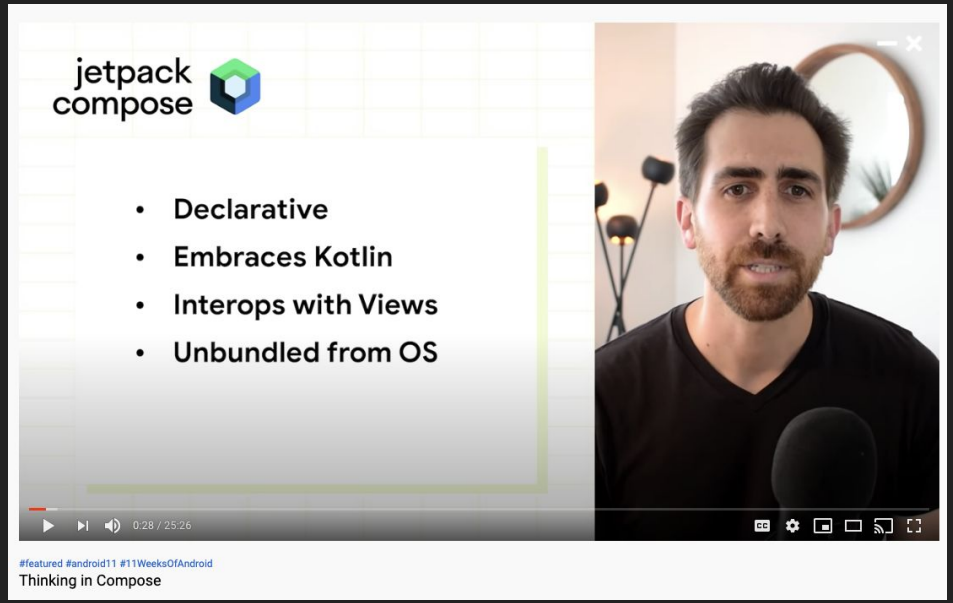
```
override fun onBackPressed() {
        if(navigationViewModel.canGoBack) {
            navigationViewModel.pop()
        } else {
            super.onBackPressed()
        }
    }
```

End result



Resources


android / compose-samples
Code
Issues 5
Pull requests 3
Actions
Projects
master
4 branches
3 tags
nickbutcher Merge pull request #179 from android/dev_alpha03


jetpack compose
Declarative
Embraces Kotlin
Interops with Views
Unbundled from OS
0:28 / 25:26
#featured #android11 #11WeeksOfAndroid
Thinking in Compose

Questions?