ANDROID JETPACK

MAKE BETTER APPS

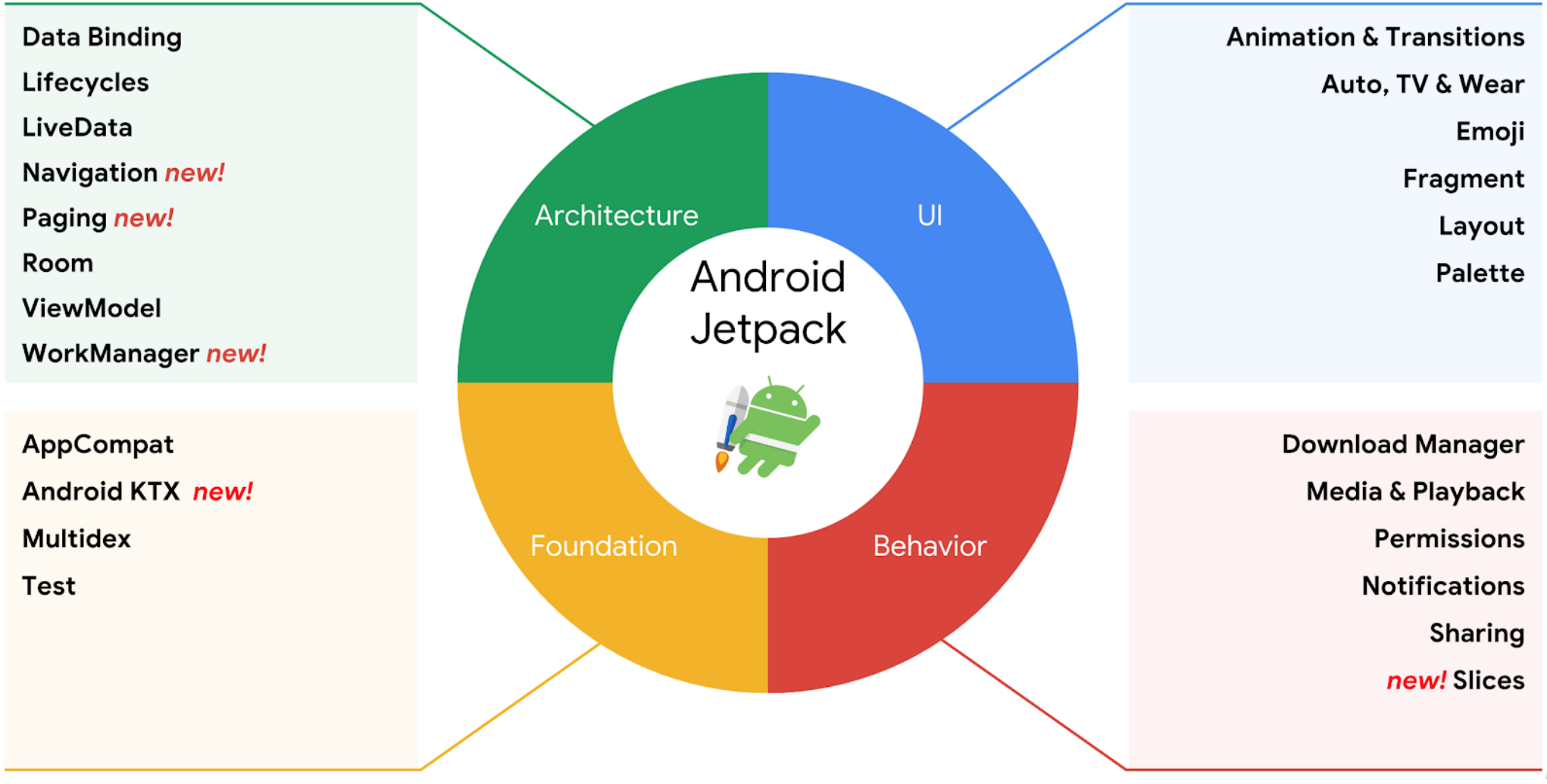
Michael Fazio - @faziodev

TODAY'S SESSION

- Jetpack Overview
- Adding Jetpack to Your App
- Foundation
- Architecture Components
- More Architecture Components
- Behavior
- UI
- Questions
- Lunch!


EXIT
DATA CLASSES
// Java
public class Person {
private final String first;
private final String last;
private int age;
public Person(String first, String last, int age) {
this.first = first;
this.last = last;
this.age = age;
}
public String getFirst() {
return first;
// Kotlin
data class Person(val first: String, val last: String, var age: Int)
2018
THAT CONFERENCE
Microsoft
twilio
THAT CONFERENCE
martinezMEDIA

SKYLINE


Data Binding
Lifecycles
LiveData
Navigation new!
Paging new!
Room
ViewModel
WorkManager new!
AppCompat
Android KTX new!
Multidex
Test
Architecture
UI
Android Jetpack
Foundation
Behavior
Animation & Transitions
Auto, TV & Wear
Emoji
Fragment
Layout
Palette
Download Manager
Media & Playback
Permissions
Notifications
Sharing
new! Slices

Why Kotlin?

Concise

Drastically reduce the amount of boilerplate code.

See example

Safe

Avoid entire classes of errors such as null pointer exceptions.

See example

Interoperable

Leverage existing libraries for the JVM, Android, and the browser.

See example

Tool-friendly

Choose any Java IDE or build from the command line.

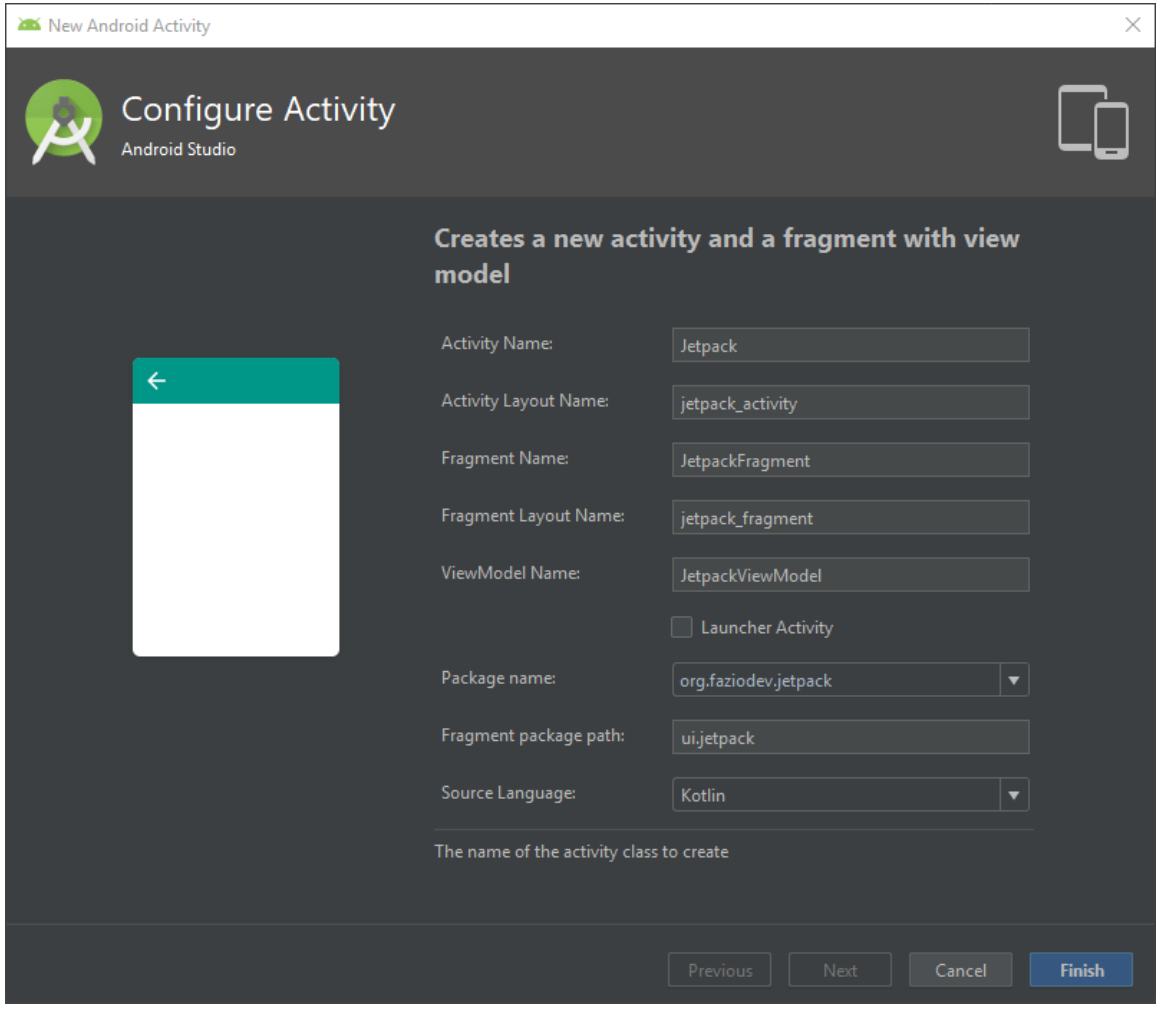
See example

play.kotlinlang.org

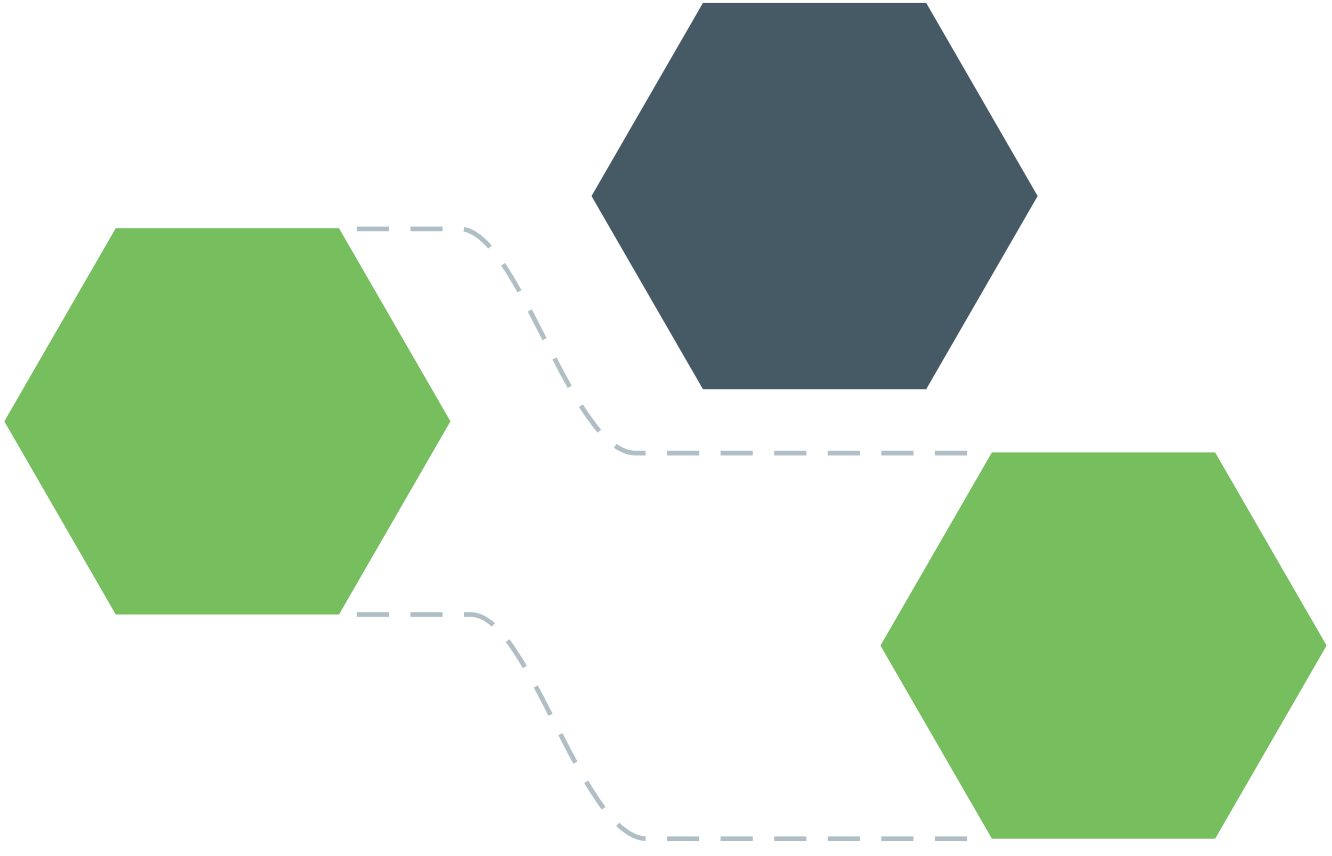
Kotlin Playground is an online sandbox to explore Kotlin programming language. Browse code samples directly in the browser

```
/*
 You can edit, run, and share this code.
 play.kotlinlang.org
*/


fun List<Player>.topTen(
    filterPredicate: (Player) -> Boolean,
    sortPredicate: (Player) -> Int,
    sortDescending: Boolean = false
): List<Player> = this.filter(filterPredicate).let { filteredPlayers ->
    (if (sortDescending) filteredPlayers.sortedByDescending(sortPredicate)
    else filteredPlayers.sortedBy(sortPredicate)).take(10)
```

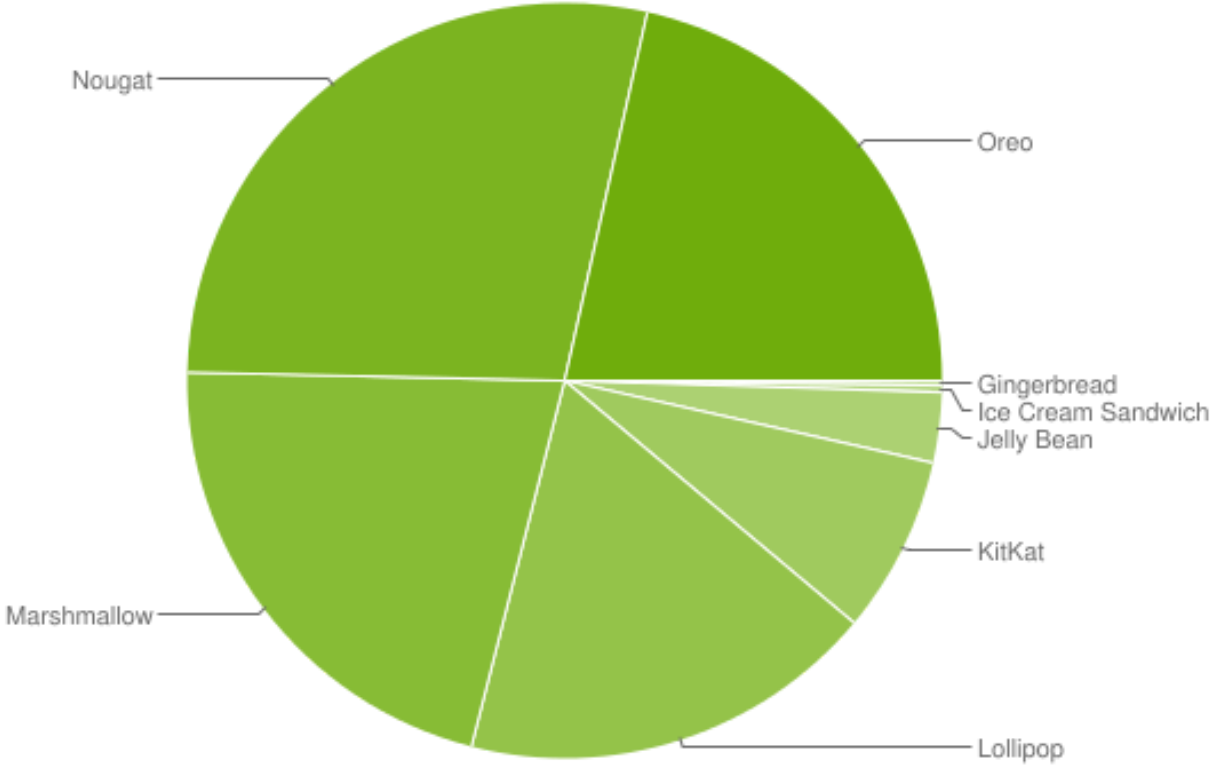
ADDING JETPACK



FOUNDATION



APPCOMPAT


Nougat
Oreo
Gingerbread
Ice Cream Sandwich
Jelly Bean
KitKat
Marshmallow
Lollipop

ANDROID KTX

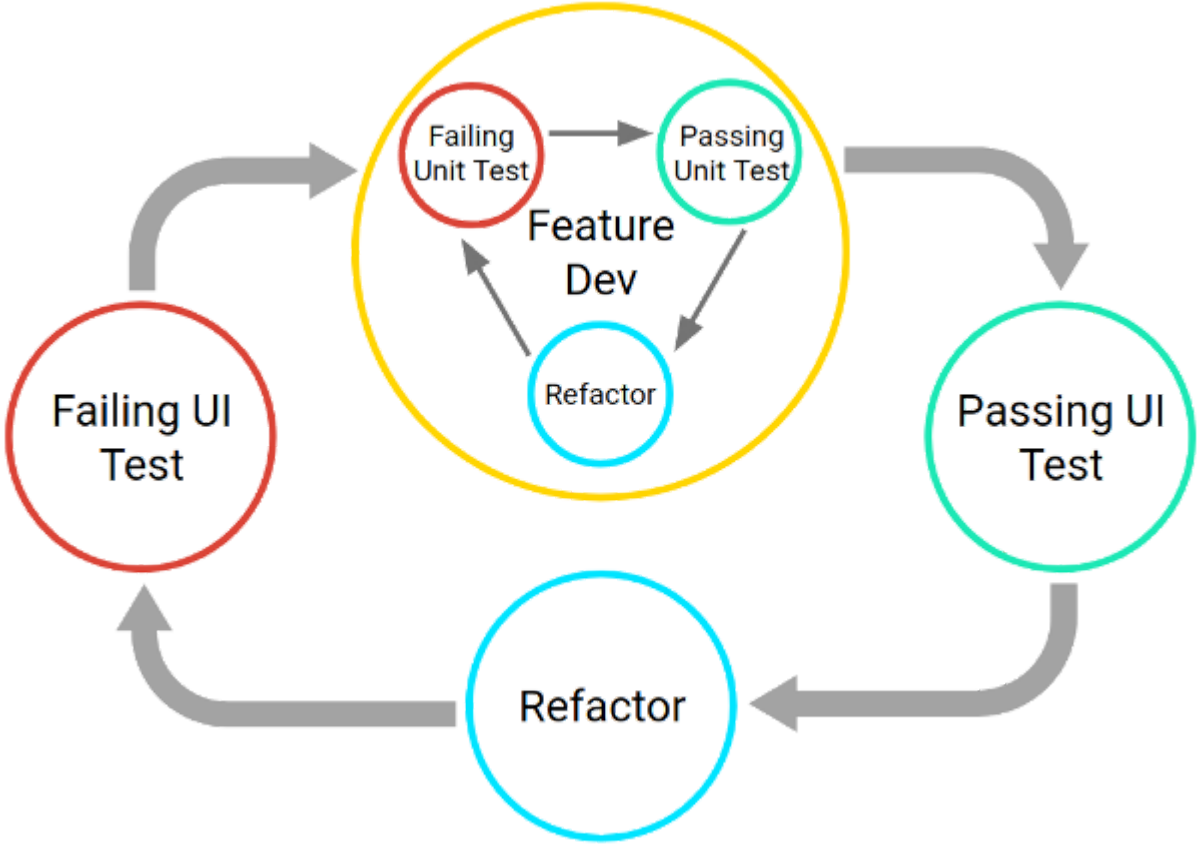
```
// Kotlin
view.viewTreeObserver.addOnPreDrawListener(
    object : ViewTreeObserver.OnPreDrawListener {
        override fun onPreDraw(): Boolean {
            viewTreeObserver.removeOnPreDrawListener(this)
            actionToBeTriggered()
            return true
        }
    }
)
```

```
// Kotlin + Android KTX
view.doOnPreDraw {
    actionToBeTriggered()
}
```

MULTIDEX

```
android {
    defaultConfig {
        ...
        minSdkVersion 21
        targetSdkVersion 28
        multiDexEnabled true
    }
    ...
}

// Required if your minSdkVersion is 20 or lower
dependencies {
  compile 'com.android.support:multidex:1.0.3'
}
```

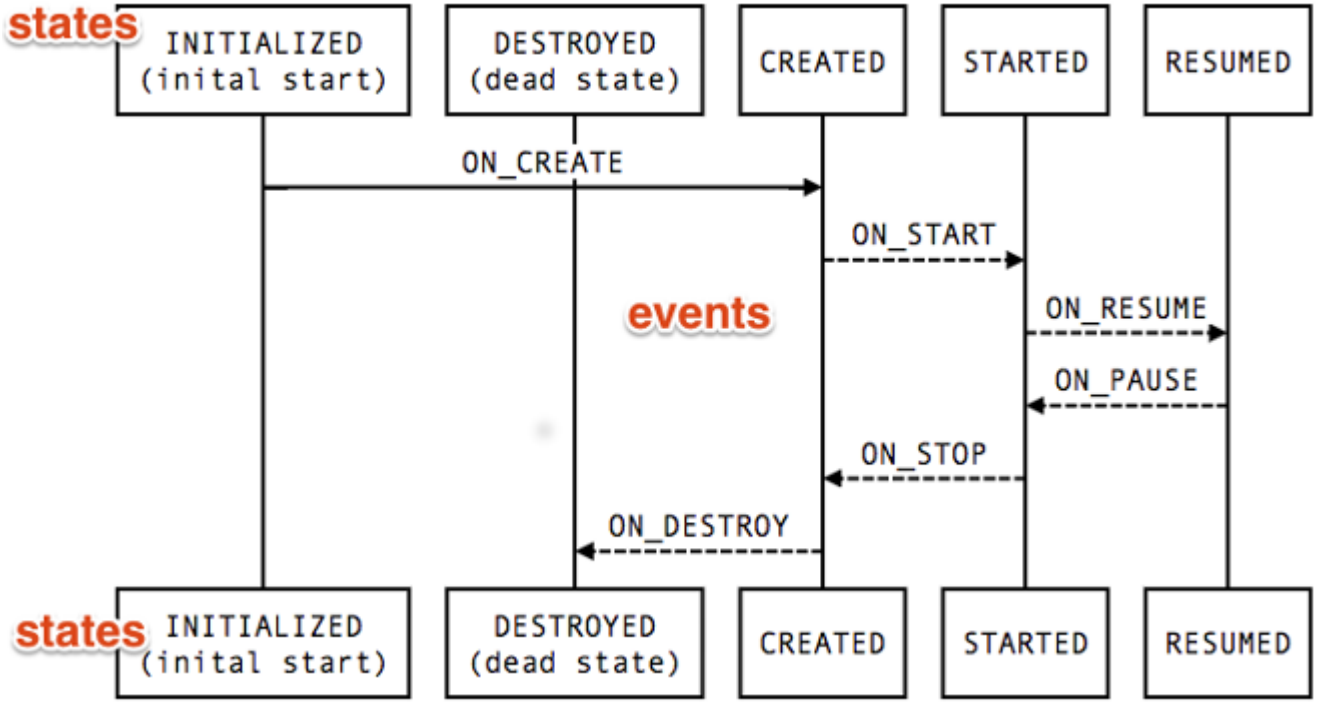
TESTING



https://developer.android.com/training/testing/

ARCHITECTURE COMPONENTS

LIFECYCLES


states
INITIALIZED
(inital start)
DESTROYED
(dead state)
CREATED
STARTED
RESUMED
ON_CREATE
ON_START
events
ON_RESUME
ON_PAUSE
ON_STOP
ON_DESTROY
states
INITIALIZED
(inital start)
DESTROYED
(dead state)
CREATED
STARTED
RESUMED

android.arch.lifecycle

```
if (lifecycle.currentState.isAtLeast(Lifecycle.State.STARTED)) {
    // do something
}
```

```
class MyObserver : LifecycleObserver {

    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
    fun connectListener() {
        ...
    }
    ...
}
```

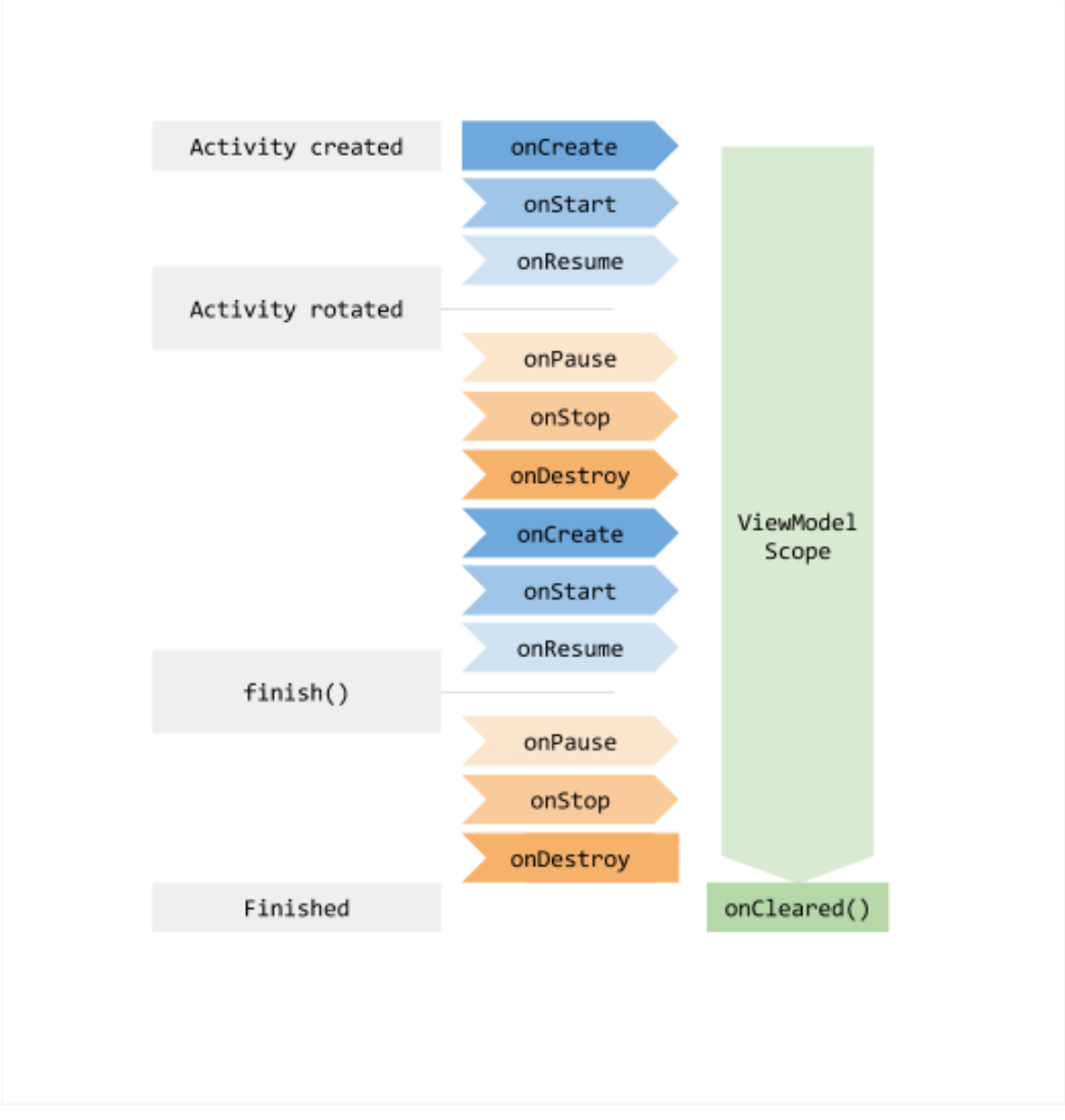
ViewModel

```
class MyViewModel : ViewModel() {
    private lateinit var users: MutableLiveData<List<User>>

    fun getUsers(): LiveData<List<User>> {
        if (!::users.isInitialized) {
            users = MutableLiveData()
            loadUsers()
        }
        return users
    }

    private fun loadUsers() {
        // Do an asynchronous operation to fetch users.
    }
}
```

USING A ViewModel

```
class MyActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        // Create a ViewModel the first time the system calls an activ
        // Re-created activities receive the same MyViewModel instance

        val model = ViewModelProviders.of(this).get(MyViewModel::class
        model.getUsers().observe(this, Observer<List<User>>{ users ->
            // update UI
        })
    }
}
```

LIFECYCLE OF A ViewModel



GETTING THE SAME ViewModel

```
class MasterFragment : Fragment() {
    private lateinit var model: SharedViewModel
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        model = activity?.run {
            ViewModelProviders.of(this).get(SharedViewModel::class.java)
        } ?: throw Exception("Invalid Activity")
    }
}

class DetailFragment : Fragment() {
    private lateinit var model: SharedViewModel
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        model = activity?.run {
            ViewModelProviders.of(this).get(SharedViewModel::class.java)
        } ?: throw Exception("Invalid Activity")
    }
}
```

LIVEDATA

```
public class CurrentWeekViewModel extends ViewModel {
    private MutableLiveData<String> currentWeekHeader;
    public MutableLiveData<String> getCurrentWeekHeader() {
        if(currentWeekHeader == null)
        currentWeekHeader = new MutableLiveData<String>();

        return currentWeekHeader;
    }
}
```

```
final CurrentWeekViewModel vm
    = ViewModelProviders.of(this).get(CurrentWeekViewModel::class);

final Observer<String> headerObserver = new Observer<String>() {
    @Override
    public void onChanged(@Nullable final String newValue) {
        headerTextView.setText(newValue);
    }
};

vm.getCurrentWeekHeader().observe(this, headerObserver);
```

LIVEDATA

ADVANTAGES

- Only updates components when active
- No more manual lifecycle handling
- Ensures your UI matches your data state
- No memory leaks
- No crashes due to stopped activities
- Always up-to-date data
- Proper configuration changes
- Sharing resources

LIVEDATA

```
class CurrentWeekViewModel : ViewModel() {
    val currentWeekHeader: MutableLiveData<String>
        = MutableLiveData<String>()
}
```

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?, state: Bundle?):
    val vm = ViewModelProviders
        .of(this)
        .get(CurrentWeekViewModel::class.java)

    val headerObserver = Observer<String> { newValue ->
        headerTextView.text = newValue
    }

    vm.currentWeekHeader.observe(this, headerObserver)
}
```

DATA BINDING

```java
// Java
final TextView headerTextView = findViewById(R.id.week_header_text);
headerTextView.setText(viewModel.getCurrentWeekHeader());
```

```kotlin
// Kotlin
findViewById<TextView>(R.id.week_header_text).apply {
    text = viewModel.currentWeekHeader
}
```

```xml
<TextView
    android:id="@+id/week_header_text"
    android:layout_height="wrap_content"
    android:layout_width="0dp"
    android:text="@{currentWeekViewModel.currentWeekHeader}"
    ...
/>
```

ENABLE DATA BINDING

```
android {
    ...
    dataBinding {
        enabled = true
    }
}
```

```
val binding = FragmentWeekBinding.inflate(inflater, container, false)
binding.setLifecycleOwner(this)
binding.currentWeekViewModel =
    ViewModelProviders.of(this).get(CurrentWeekViewModel::class.java)
```

DATA BINDING

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto">
    <data>
        <variable
            name="currentWeekViewModel"
            type="org.faziodev.timetracker.viewmodels.CurrentWeekViewM
            />
    </data>
    <ConstraintLayout... /> <!-- UI layout's root element -->
</layout>
```

DATA BINDING

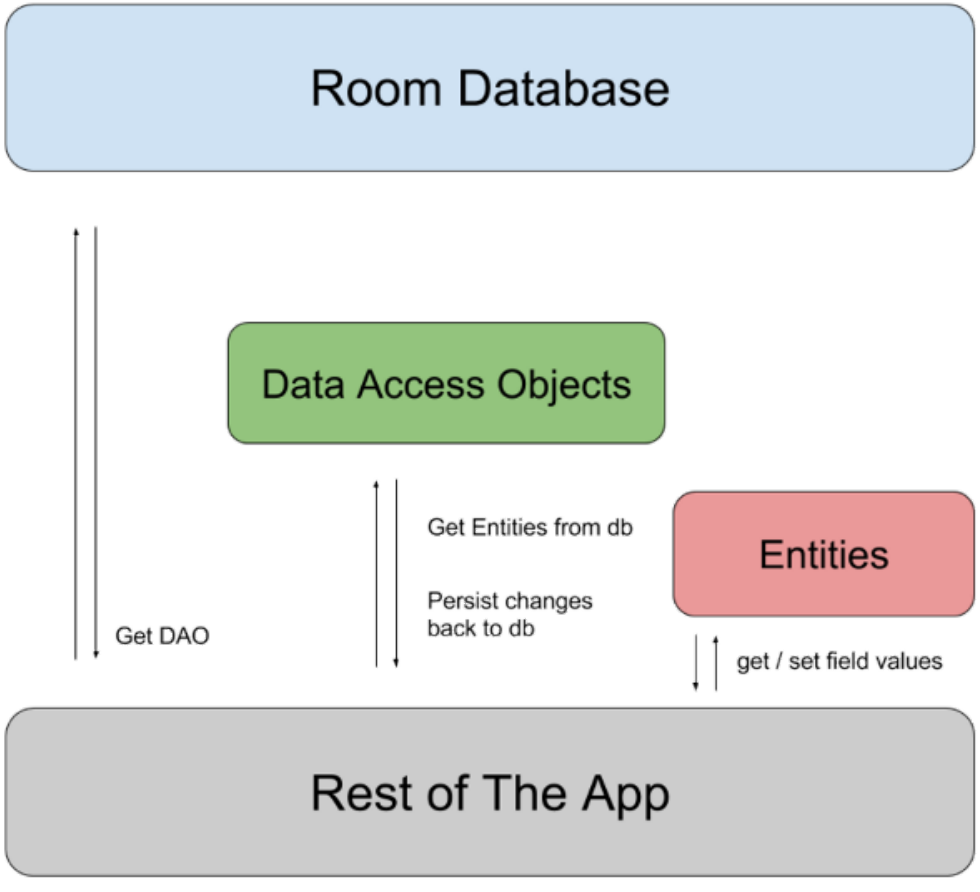
```
<TextView
    android:id="@+id/week_header_text"
    android:text="@{viewModel.currentWeekHeader}"
    ... />
```

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
    android:id="@+id/week_summary_fab"
    android:onClick="@{ () -> viewModel.onFabClicked() }"
    ... />
```

```
<TextView
    android:id="@+id/week_pace_text"
    android:text="@{@string/pace_this_week(viewModel.currentPaceHours)
    ... />
```

```
<CheckBox
    android:id="@+id/week_is_entered_into_system"
    android:checked="@={viewModel.enteredIntoSystem}"
    ... />
```

ROOM



https://developer.android.com/training/data-storage/room/

@Entity and @Dao

```
@Entity
data class User(
    @PrimaryKey(autoGenerate = true) var uid: Int,
    @ColumnInfo(name = "first_name") @NonNull var firstName: String,
    @ColumnInfo(name = "last_name") var lastName: String?
)
```

```
@Dao
interface UserDao {
    @Query("SELECT * FROM user")
    fun getAll(): List<User>

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
           "last_name LIKE :last LIMIT 1")
    fun findByName(first: String, last: String): User

    @Insert
    fun insertAll(vararg users: User)

    @Delete
    fun delete(user: User)
}
```

@Database and RoomDatabase

```
@Database(entities = arrayOf(User::class), version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao
}
```

```
val db = Room.databaseBuilder(
    applicationContext,
    AppDatabase::class.java, "database-name"
).build()
```

@DatabaseView

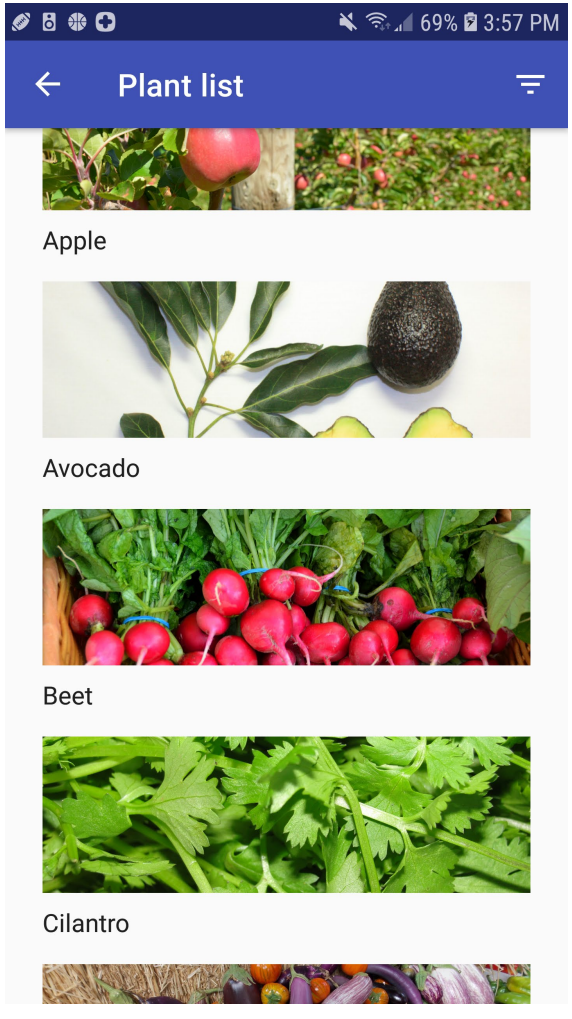
```
@DatabaseView("SELECT user.id, user.name, user.departmentId," +
        "department.name AS departmentName FROM user " +
        "INNER JOIN department ON user.departmentId = department.id")
data class UserDetail(
    var id: Long,
    var name: String?,
    var departmentId: Long,
    var departmentName: String?
)
```

```
@Database(entities = arrayOf(User::class),
          views = arrayOf(UserDetail::class), version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao
}
```

DATABASE MIGRATIONS

```
val MIGRATION_1_2 = object : Migration(1, 2) {
    override fun migrate(database: SupportSQLiteDatabase) {
        database.execSQL(
            "CREATE TABLE `Fruit` (`id` INTEGER, `name` TEXT, " +
                "PRIMARY KEY(`id`))")
    }
}

val MIGRATION_2_3 = object : Migration(2, 3) {
    override fun migrate(database: SupportSQLiteDatabase) {
        database.execSQL("ALTER TABLE Book ADD COLUMN pub_year INTEGER
    }
}

Room.databaseBuilder(applicationContext, MyDb::class.java, "database-n
        .addMigrations(MIGRATION_1_2, MIGRATION_2_3).build()
```

PAGING

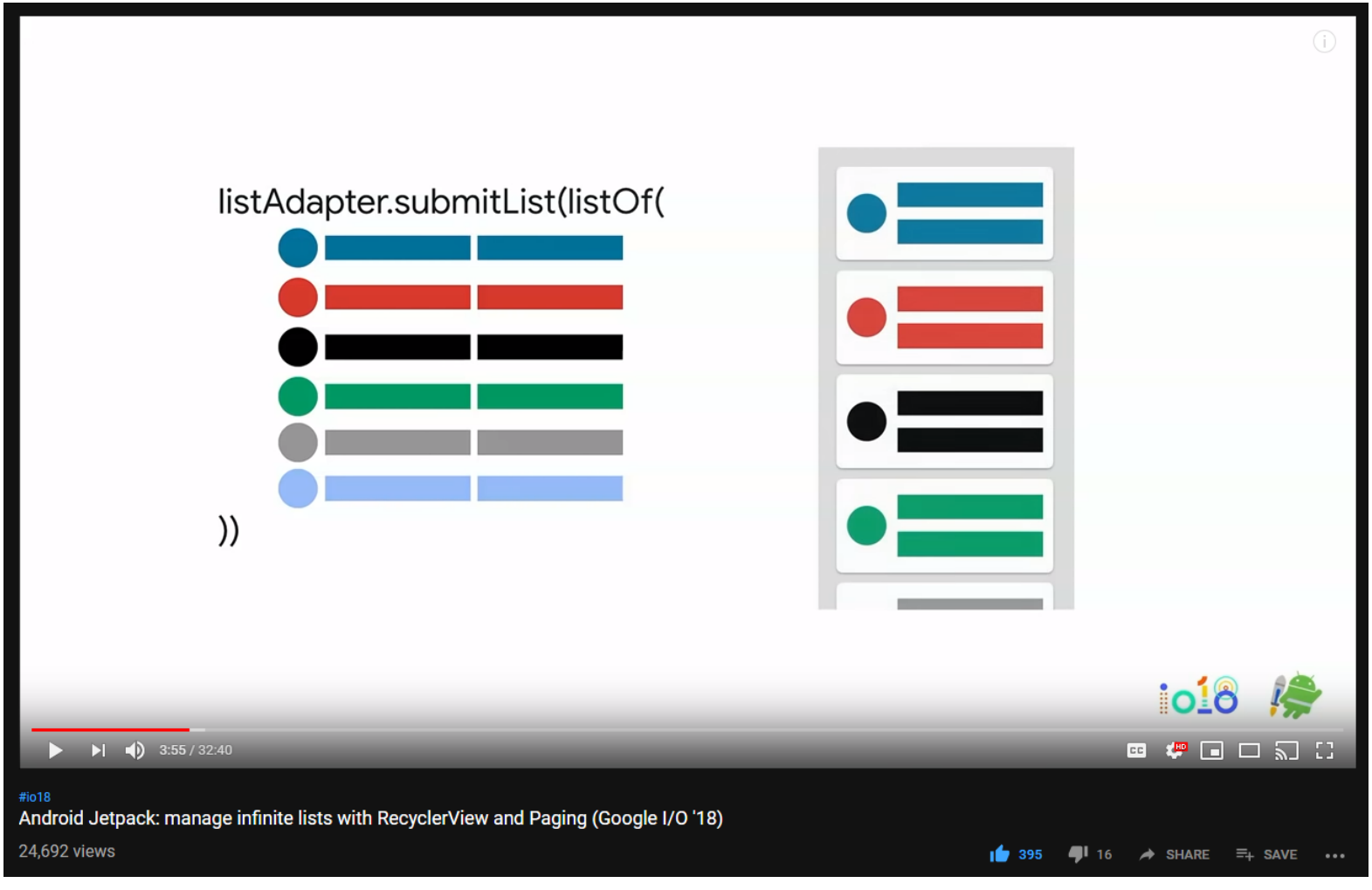


PagedList and PagedAdapter

```
// The Int type argument corresponds to a PositionalDataSource object.
val myConcertDataSource : DataSource.Factory<Int, Concert> =
        concertDao.concertsByDate()

val concertList = LivePagedListBuilder(
    myConcertDataSource, /* page size */ 20).build()
```

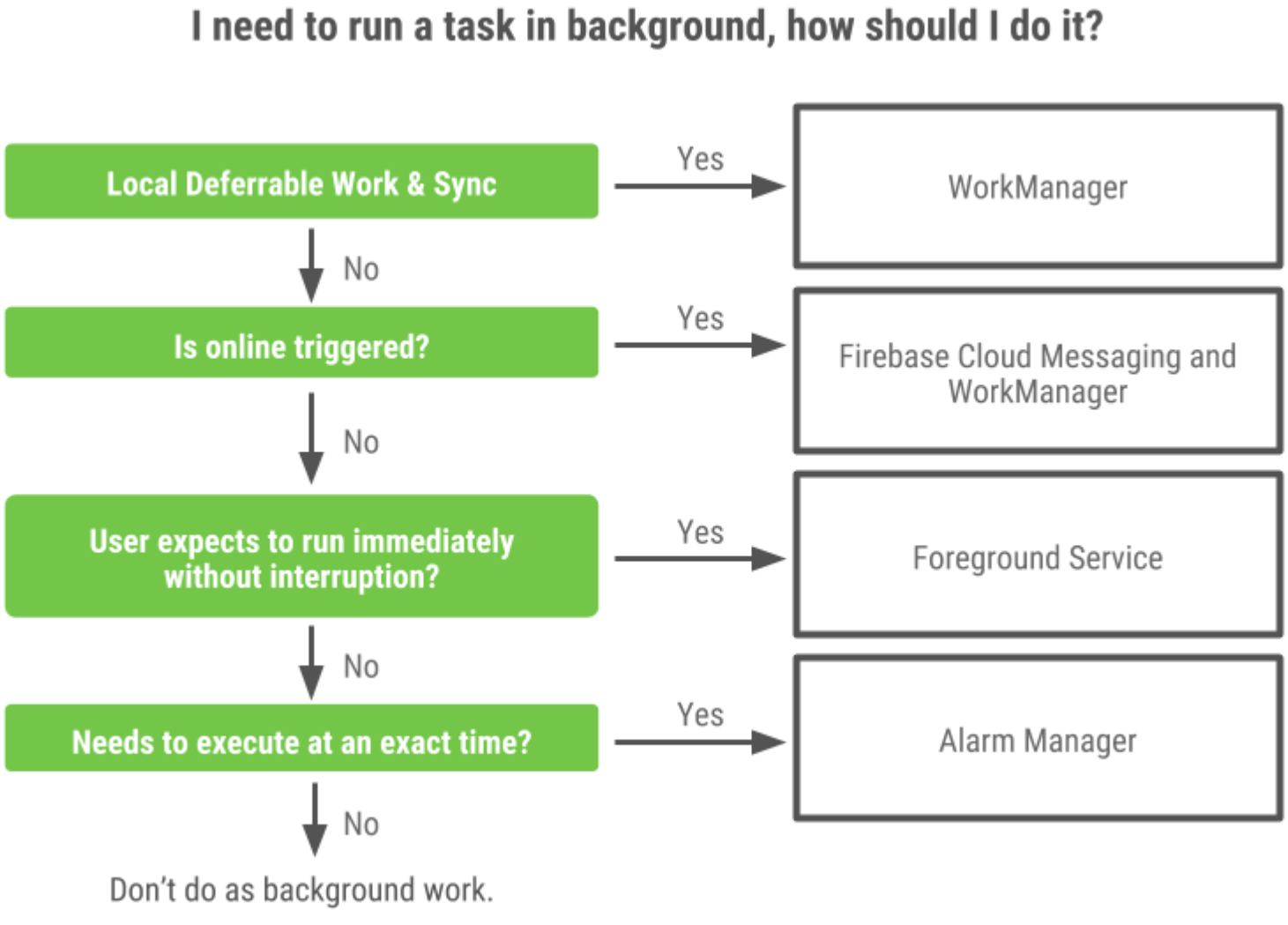
```
private val adapter = ConcertAdapter()
private lateinit var viewModel: ConcertViewModel

override fun onCreate(savedInstanceState: Bundle?) {
    viewModel
        = ViewModelProviders.of(this).get(ConcertViewModel::class.java
    viewModel.concertList.observe(
        this,
        Observer { adapter.submitList(it) }
    )
}
```

PAGING



https://l.faziodev.org/paging-youtube

WORKMANAGER



Worker and WorkRequest

```
class CompressWorker(context : Context, params : WorkerParameters)
    : Worker(context, params) {
    override fun doWork(): Result {
        doWork()
        // Indicate success or failure with your return value:
        // Result.retry() tells WorkManager to try again
        // Result.failure() says not to try again.
        return Result.success()
    }
}
```

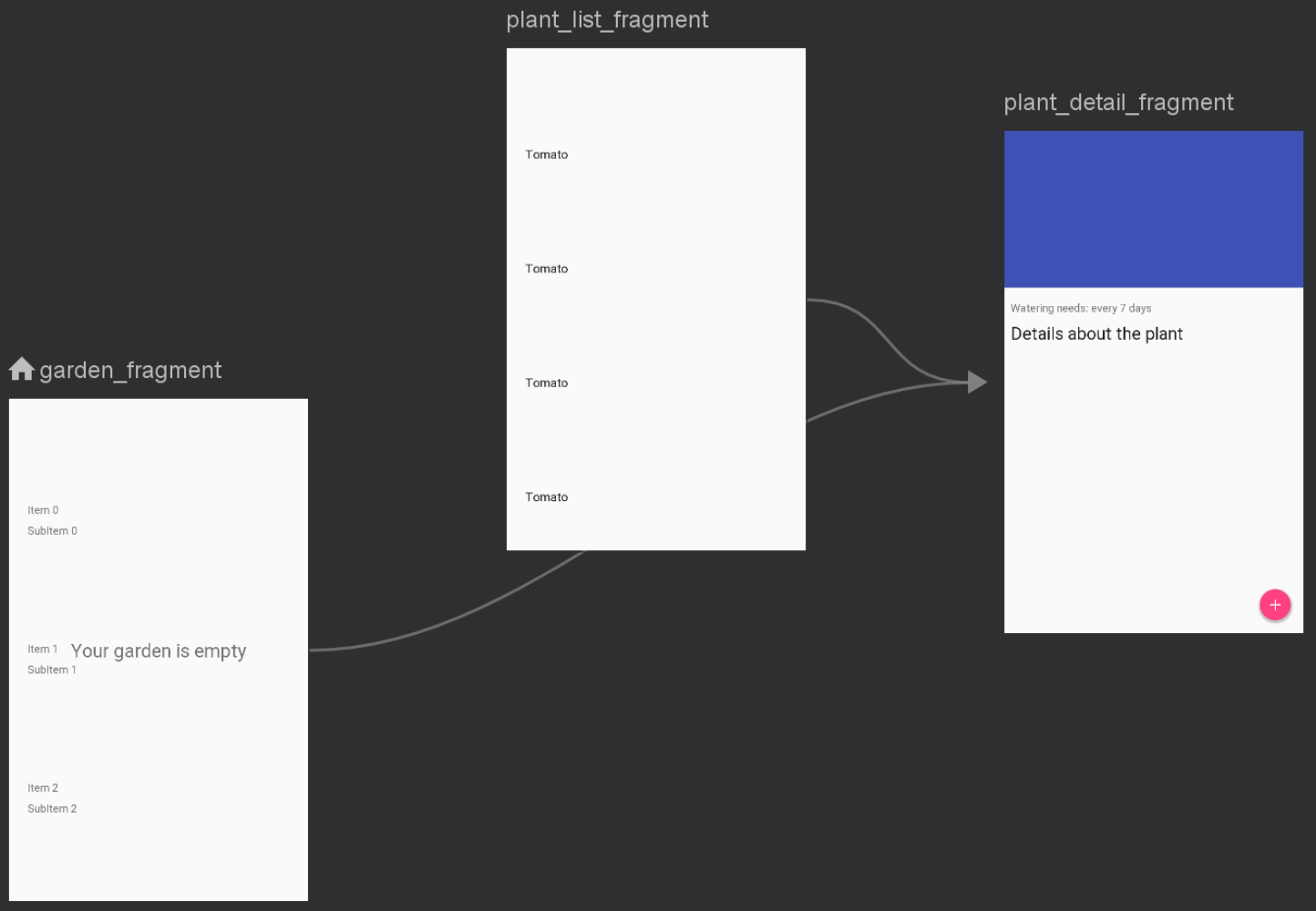
```
// Can also use PeriodicWorkRequest
val compressionWork = OneTimeWorkRequestBuilder<CompressWorker>().buil
WorkManager.getInstance().enqueue(compressionWork)
```

WorkManager and WorkInfo

```
// Can also use PeriodicWorkRequest
val compressionWork = OneTimeWorkRequestBuilder<CompressWorker>().buil
WorkManager.getInstance().enqueue(compressionWork)
```

```
WorkManager.getInstance().getWorkInfoByIdLiveData(compressionWork.id)
    .observe(lifecycleOwner, Observer { workInfo ->
        // Do something with the status
        if (workInfo != null && workInfo.state.isFinished) {
            // ...
        }
    })
```

NAVIGATION


plant_list_fragment
Tomato
Tomato
Tomato
Tomato
plant_detail_fragment
Watering needs: every 7 days
Details about the plant
garden_fragment
Item 0
SubItem 0
Item 1
Your garden is empty
SubItem 1
Item 2
SubItem 2

NAVIGATION

BENEFITS

- Handle fragment transactions
- Handle Up/Back actions correctly by default
- Providing standardized resources for animations and transactions
- Treating deep linking as a first-class operation
- Use nav drawers and bottom navs with minimal work
- Type safety when passing info during navigation
- Visual navigation in Android Studio

NAVIGATION

PRINCIPLES

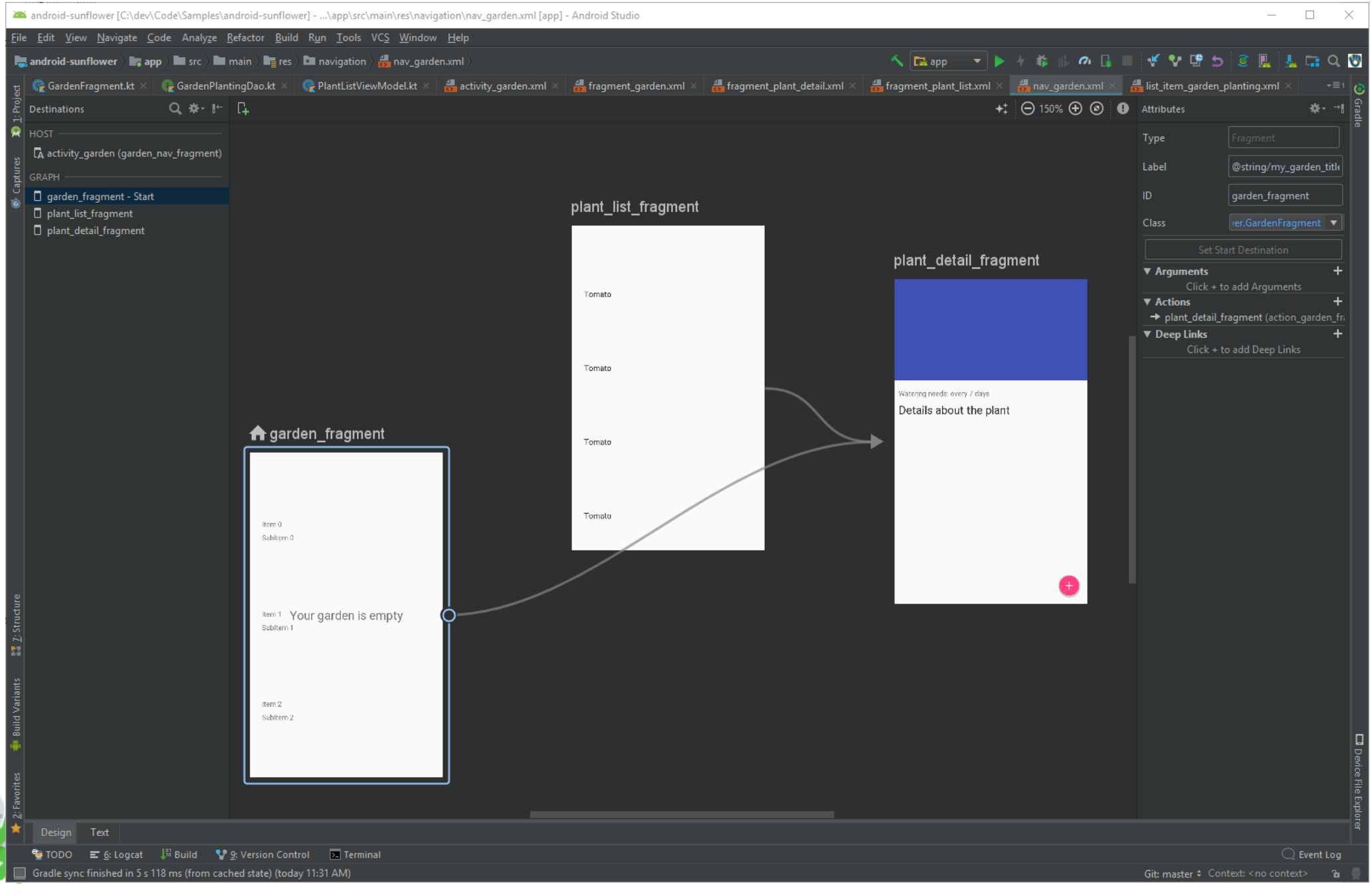
- Fixed start destination
- Nav state should be a stack of destinations
- Up button never exits the app
- Up and back are identical in your app
- Deep linking and navigating to a destination should have the same stack

nav_graph.xml

```
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/nav_graph"
    app:startDestination="@id/weekFragment">

    <fragment
        android:id="@+id/weekFragment"
        android:name="org.faziodev.timetracker.WeekFragment"
        android:label="fragment_week"
        tools:layout="@layout/fragment_week" />
    <fragment
        android:id="@+id/dashboardFragment"
        android:name="org.faziodev.timetracker.DashboardFragment"
        android:label="fragment_dashboard"
        tools:layout="@layout/fragment_dashboard" />
    <fragment
        android:id="@+id/tasksFragment"
```

NAVIGATION

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    val binding: ActivityMainBinding
        = DataBindingUtil.setContentView(this, R.layout.activity_main)

    this.navController
        = Navigation.findNavController(this, R.id.mainFragment)

    binding.bottomNav.setupWithNavController(this.navController)
}
```


android-sunflower [C:\dev\Code\Samples\android-sunflower] - ...\app\src\main\res\navigation\nav_garden.xml [app] - Android Studio
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
android-sunflower app src main res navigation nav_garden.xml
GardenFragment.kt GardenPlantingDao.kt PlantListViewModel.kt activity_garden.xml fragment_garden.xml fragment_plant_detail.xml fragment_plant_list.xml nav_garden.xml list_item_garden_planting.xml
Destinations
HOST
activity_garden (garden_nav_fragment)
GRAPH
garden_fragment - Start
plant_list_fragment
plant_detail_fragment
150%
plant_list_fragment
Tomato
Tomato
Tomato
Tomato
plant_detail_fragment
Watering needs: every 7 days
Details about the plant
garden_fragment
Item 0
SubItem 0
Item 1
Your garden is empty
SubItem 1
Item 2
SubItem 2
Attributes
Type Fragment
Label @string/my_garden_title
ID garden_fragment
Class er.GardenFragment
Set Start Destination
Arguments
Click + to add Arguments
Actions
plant_detail_fragment (action_garden_fr
Deep Links
Click + to add Deep Links
Design Text
TODO 6: Logcat Build 9: Version Control Terminal
Event Log
Gradle sync finished in 5 s 118 ms (from cached state) (today 11:31 AM)
Git: master Context: <no context>

BEHAVIOR

BEHAVIOR COMPONENTS

- Download Manager
- Media & Playback
- Notifications
- Permissions

PREFERENCES

```
<androidx.preference.PreferenceScreen
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <SwitchPreferenceCompat
        app:key="notifications"
        app:title="Enable message notifications"/>

    <Preference
        app:key="feedback"
        app:title="Send feedback"
        app:summary="Report technical issues or suggest new features"/

</androidx.preference.PreferenceScreen>
```

```
class MySettingsFragment : PreferenceFragmentCompat() {
    override fun onCreatePreferences(bundle: Bundle, rootKey: String)
        setPreferencesFromResource(R.xml.preferences, rootKey)
    }
}
```

SHARING

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
            android:id="@+id/menu_item_share"
            android:showAsAction="ifRoom"
            android:title="Share"
            android:actionProviderClass=
                "android.widget.ShareActionProvider" />
    ...
</menu>
```

SHARING

```
private var mShareActionProvider: ShareActionProvider? = null
...

override fun onCreateOptionsMenu(menu: Menu): Boolean {
    menuInflater.inflate(R.menu.share_menu, menu)

    menu.findItem(R.id.menu_item_share).also { menuItem ->
        // Fetch and store ShareActionProvider
        mShareActionProvider = menuItem.actionProvider as? ShareActionProvider
    }

    // Return true to display menu
    return true
}

// Call to update the share intent
private fun setShareIntent(shareIntent: Intent) {
    mShareActionProvider?.setShareIntent(shareIntent)
}
```

SLICES

Upcoming trip: Seattle
Jun 12-19 • 2 guests

Check In
12:00 PM, Jun 12

Check Out
11:00 AM, Jun 19

Get a ride
Car in 4 min

Home
12 miles | 15 min | $14.00

Work
22 miles | 35 min | $25.00

Weekly Mix
20 songs, 1hr 52 min

Autumn Fever
Bascilica

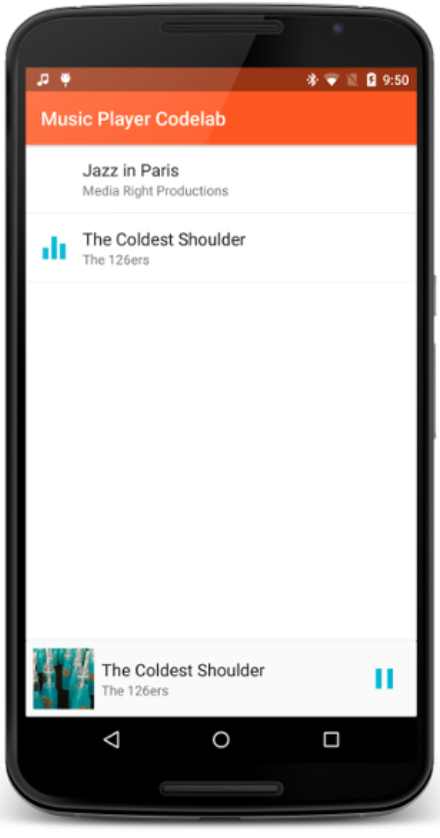
Urban
We Summit

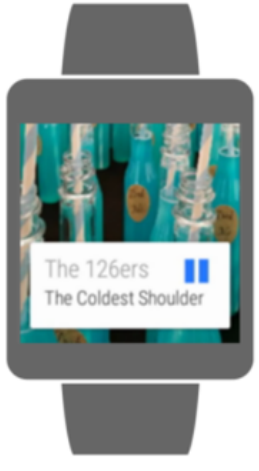
Flicker
Print Works

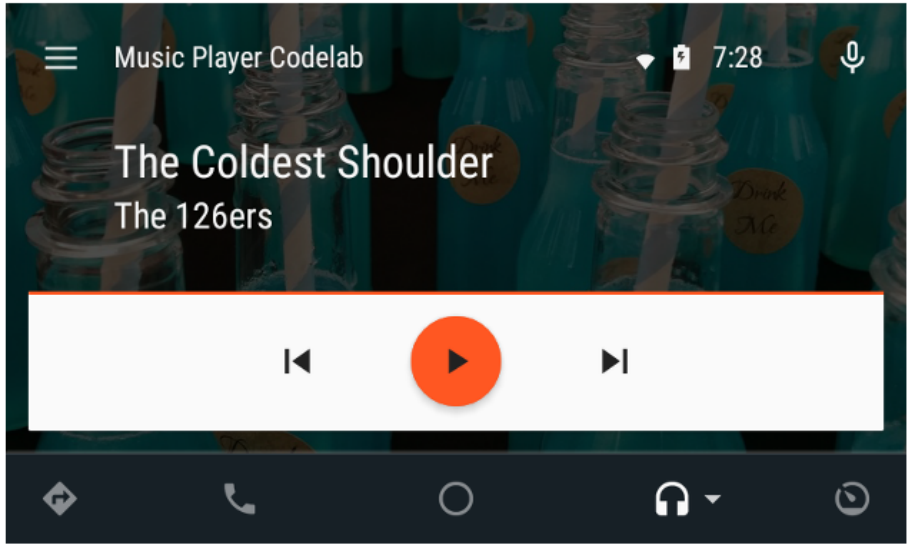
Brightness level
25%

UI

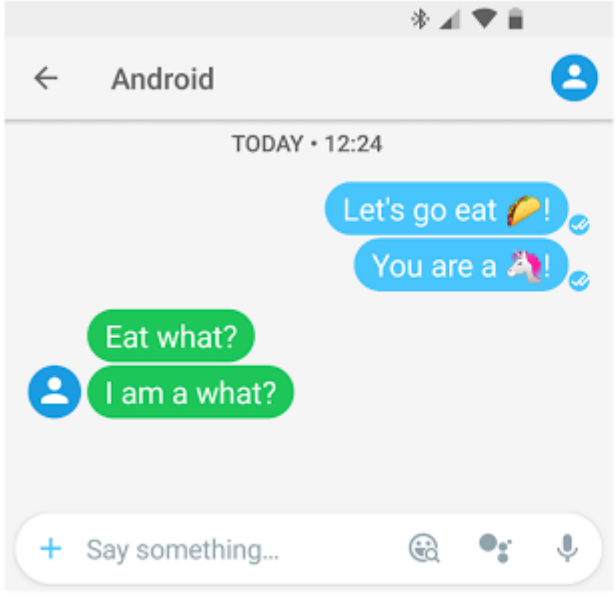
UI COMPONENTS

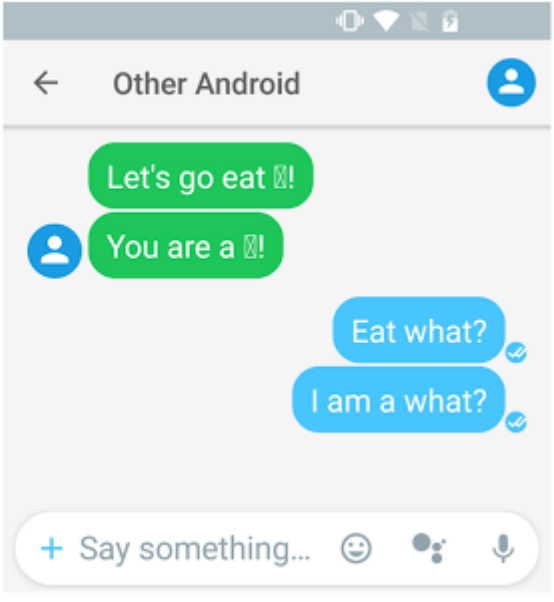




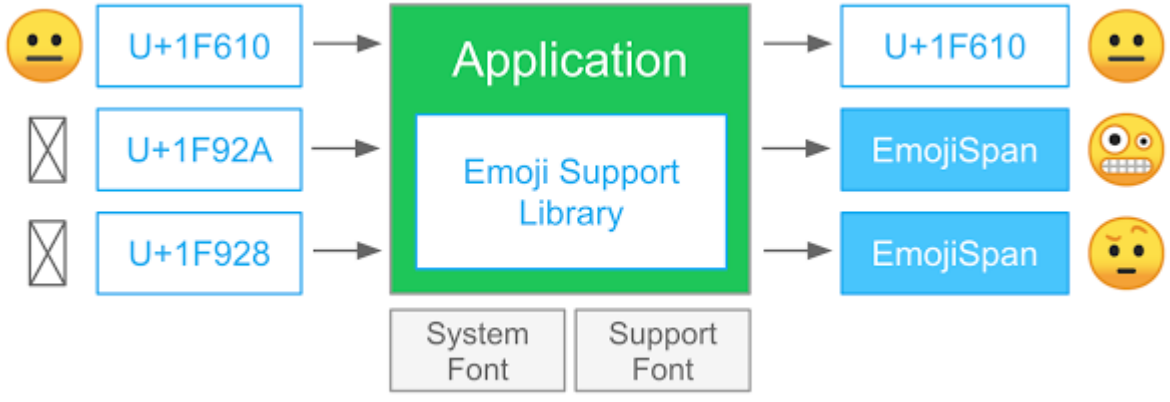


EMOJI

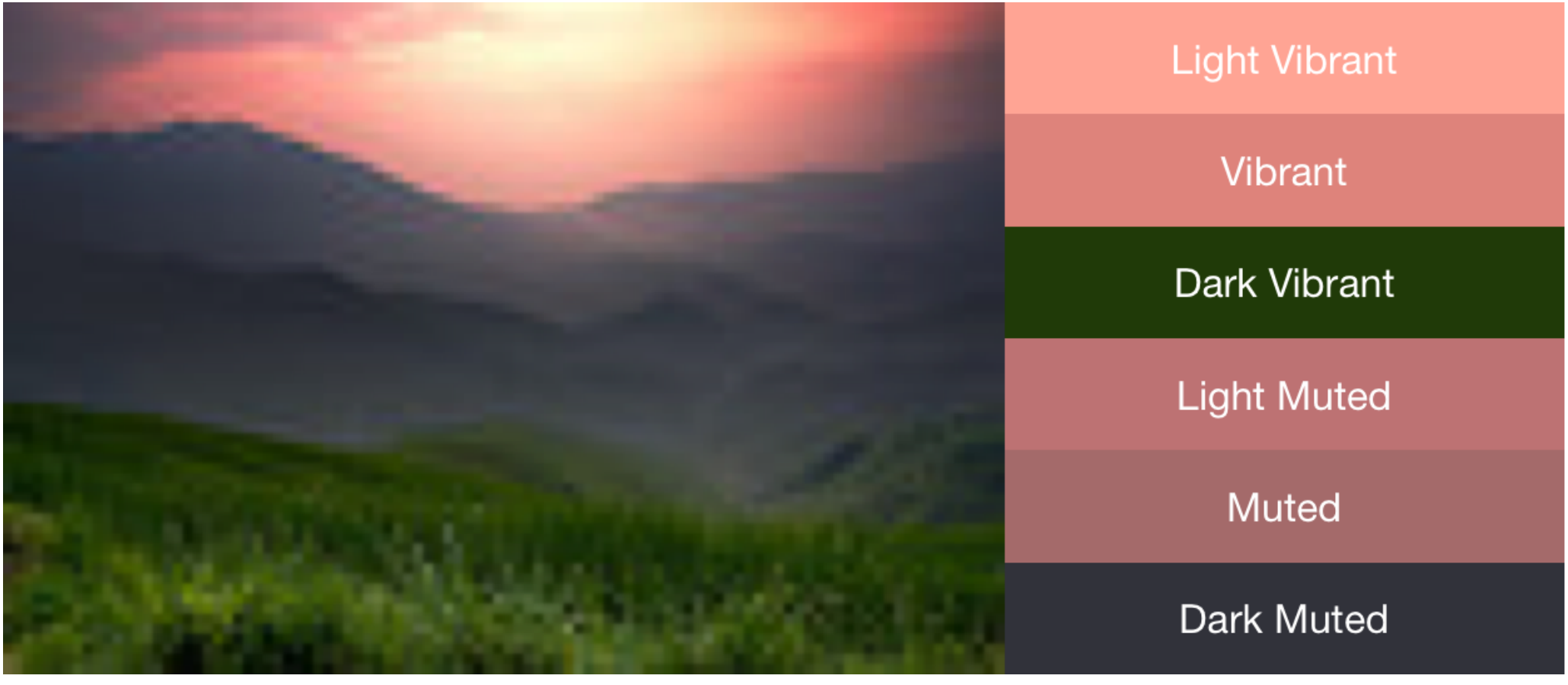

Android
TODAY • 12:24
Let's go eat 🌮!
You are a 🦄!
Eat what?
I am a what?
Say something...


Other Android
Let's go eat □!
You are a □!
Eat what?
I am a what?
Say something...

EMOJI


U+1F610
U+1F92A
U+1F928
Application
Emoji Support Library
System Font
Support Font
U+1F610
EmojiSpan
EmojiSpan

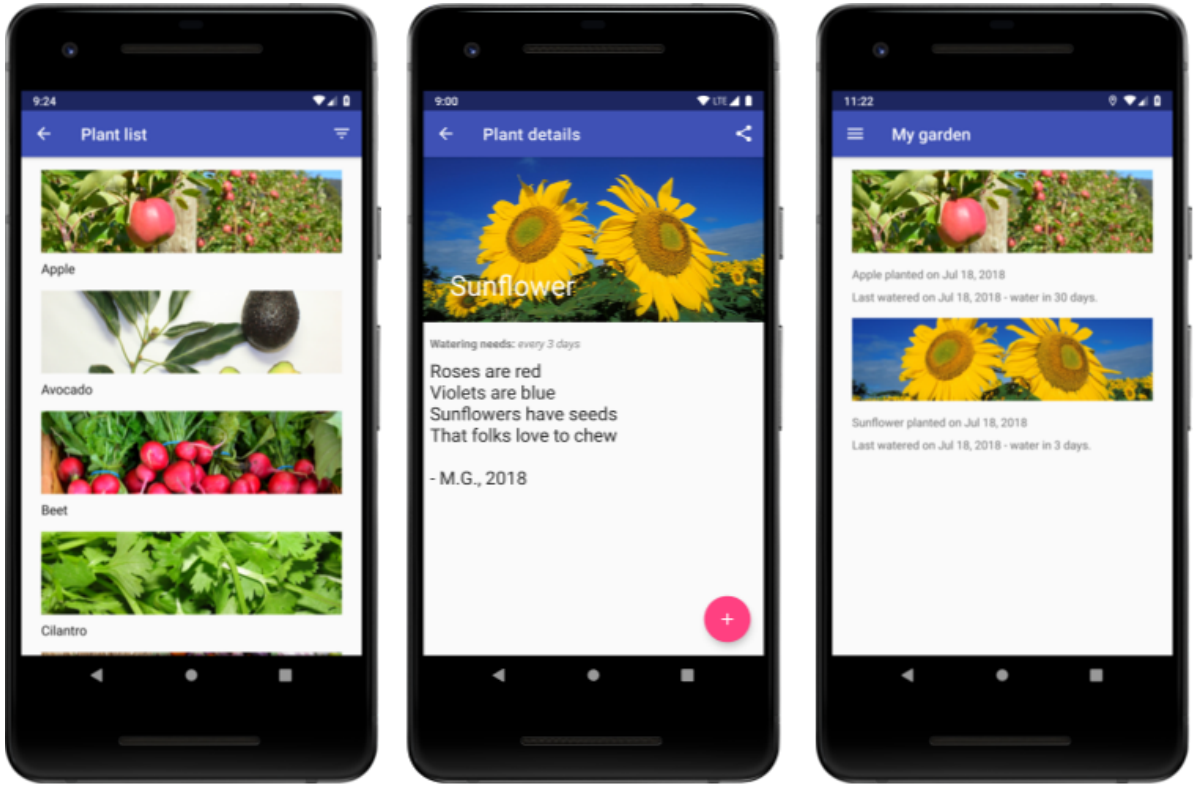
PALETTE


Light Vibrant
Vibrant
Dark Vibrant
Light Muted
Muted
Dark Muted

PALETTE

```
// Generate palette synchronously and return it
fun createPaletteSync(bitmap: Bitmap): Palette
    = Palette.from(bitmap).generate()

// Generate palette asynchronously and use it on a different
// thread using onGenerated()
fun createPaletteAsync(bitmap: Bitmap) {
    Palette.from(bitmap).generate { palette ->
        // Use generated instance
    }
}
```

ANDROID SUNFLOWER



https://github.com/googlesamples/android-sunflower

USEFUL LINKS

- Jetpack Home: https://g.co/jetpack
- Getting Started with Jetpack: https://goo.gl/bGnL7N
- Introducing Android Jetpack: https://youtu.be/LmkKFCfmnhQ
- Jetpack on YouTube: https://l.faziodev.org/jetpack-youtube
- Paging on YouTube: https://l.faziodev.org/paging-youtube
- Android Sunflower: https://l.faziodev.org/jetpack
- Room with a View: https://l.faziodev.org/android-rwav-kotlin

QUESTIONS?

THANKS!

Michael Fazio - @faziodev