Android 新技术趋势

xuda@bytedanc.com

字节跳动 Android 工程师

2019.7.9

ByteDance字节跳动

Catalog

- High-efficient development: Jetpack
- Functional programming: Kotlin
- Cross-platform framework: Flutter

Jetpack

Jetpack

https://developer.android.com/jetpack/

Jetpack

Jetpack is a collection of Android software components to make it easier for you to develop great Android apps.

Jetpack

These components help you

- ❏ Follow best practices
- ❏ Eliminate boilerplate code
- ❏ Build high quality, robust apps

Jetpack

Foundation

- AppCompat
- Android KTX
- Multidex
- Test

Jetpack

Architecture

- Data Binding
- Lifecycles
- LiveData
- Navigation
- Paging
- Room
- ViewModel
- WorkManager

Jetpack

Behavior

- Download manager
- Media & playback
- Notifications
- Permissions
- Preferences
- Sharing
- Slices

Jetpack

UI

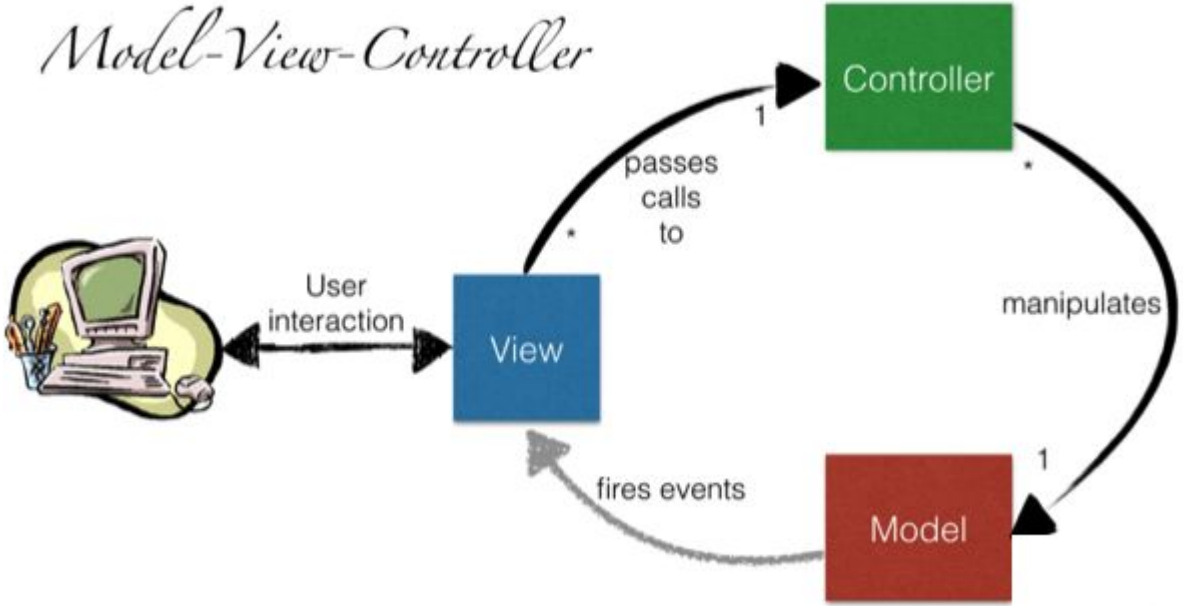
- Animation & transitions
- Auto
- Emoji
- Fragment
- Layout
- Palette
- TV
- Wear OS by Google

Architecture

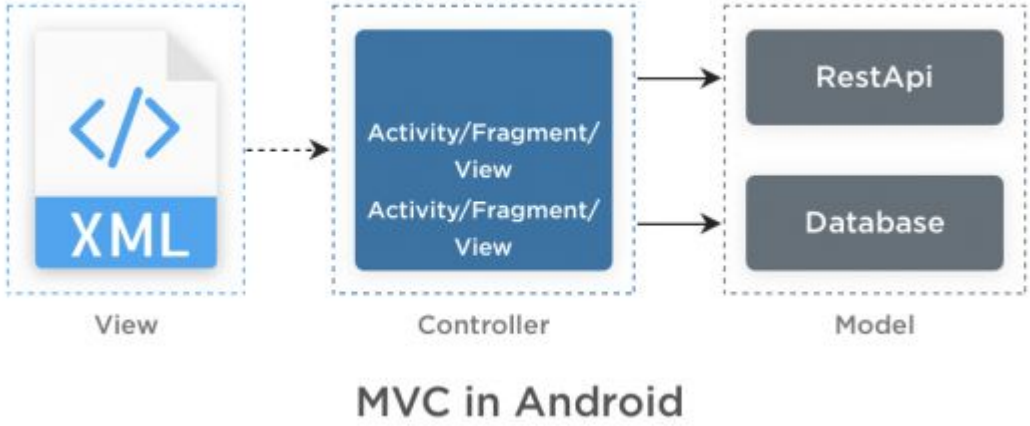
- MVC
- MVP
- MVVM

Samples:
https://github.com/googlesamples/android-architecture

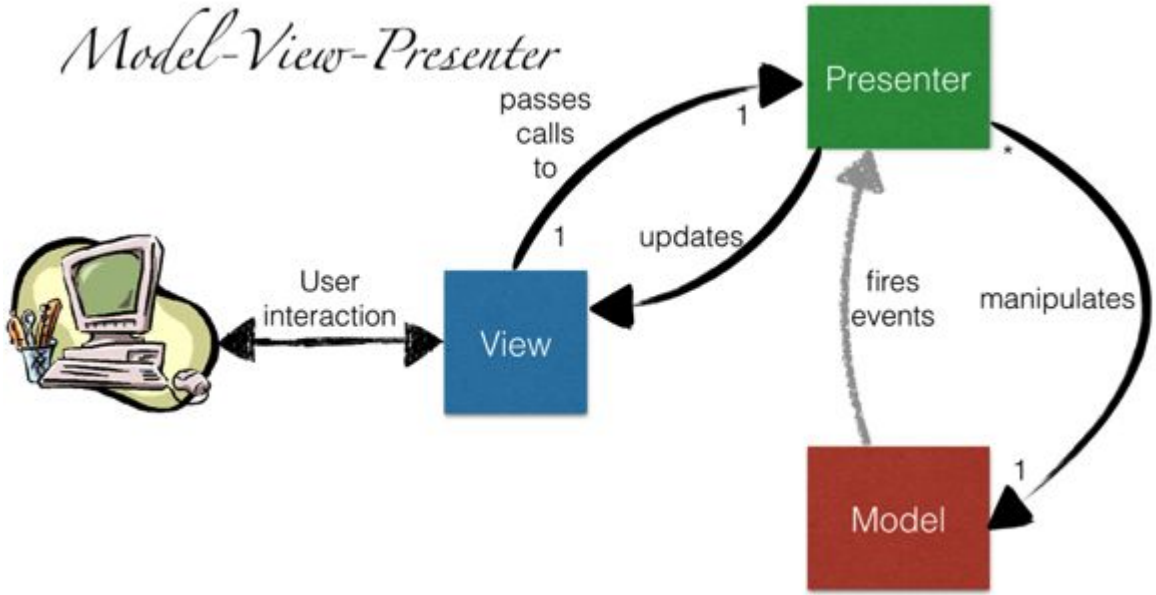
Architecture



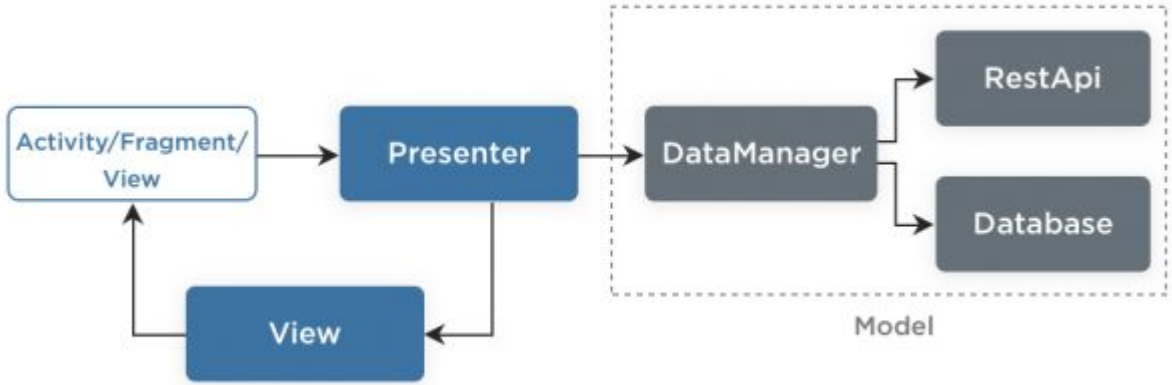
Architecture



Architecture

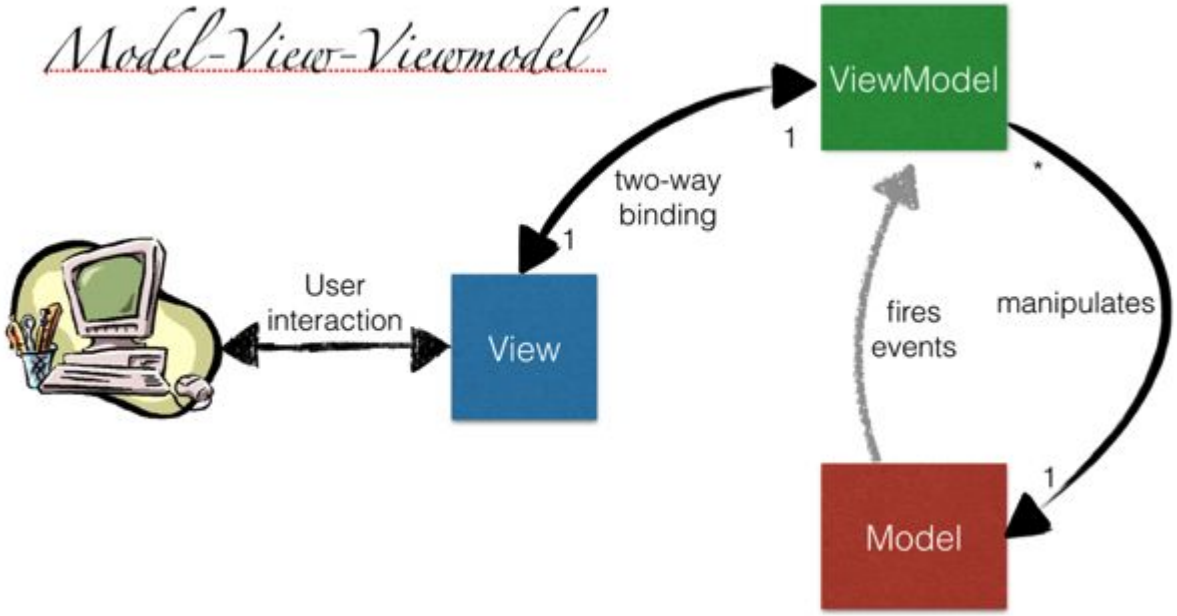


Architecture

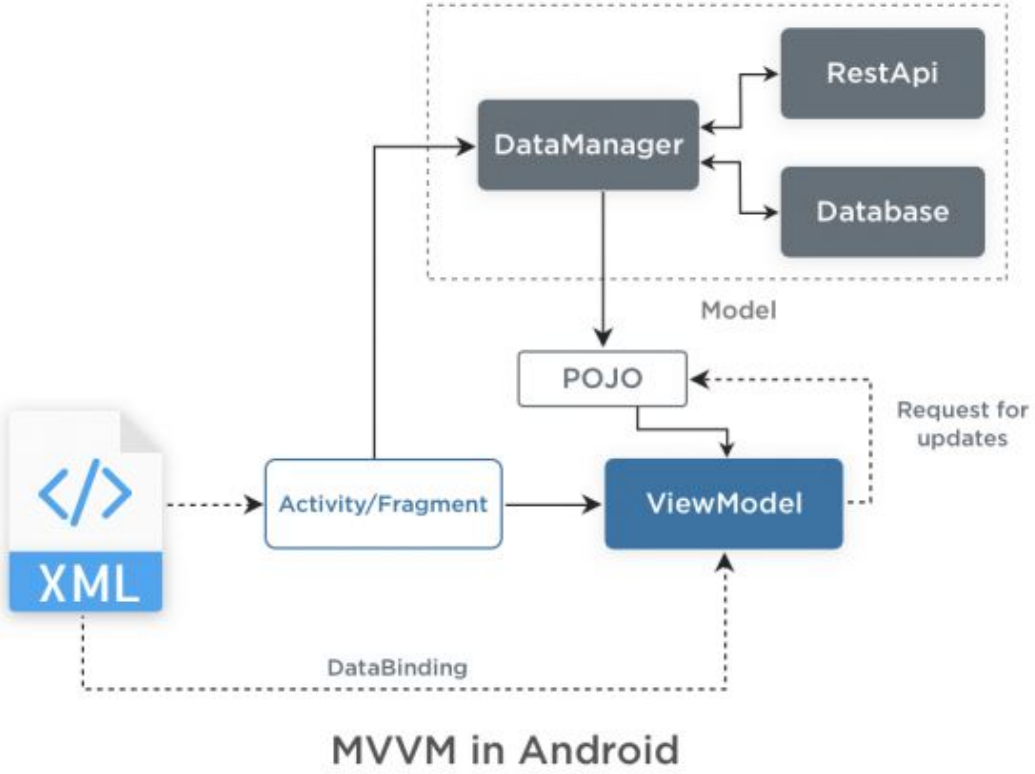


MVP in Android

Architecture

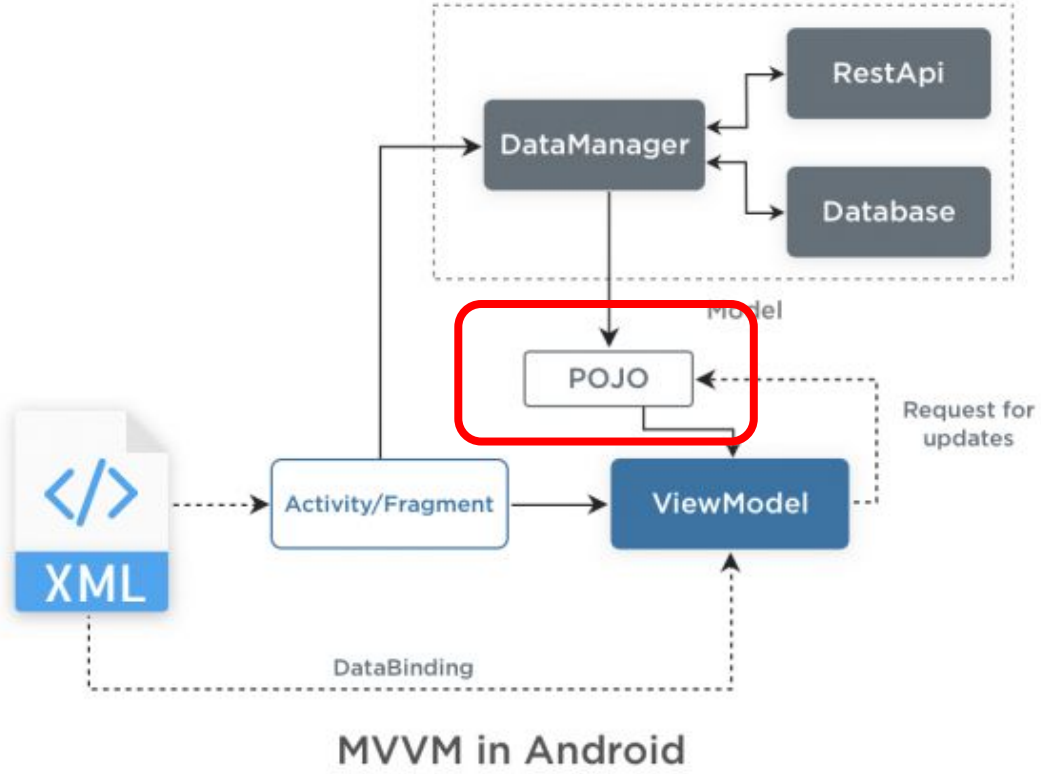


Architecture



MVVM in Android

LiveData



MVVM in Android

LiveData

Without LiveData

```
// ViewModel
List<User> userList;
List<UserLoadedListener > listeners = new ArrayList<>()

void registerListener( UserLoadedListener  lsn) {
    listeners.add(lsn)
}

void unregisterListener( UserLoadedListener  lsn) {
    listeners.remove(lsn)
}

void loadUsers() {
    userService.getUsers(data -> {
        userList = data;
        // notify UI
        for(UserLoadedListener  lsn : listeners) {
           lsn.onUserLoaded(userList);
        }
    })
}
```

```
// Activity
UserLoadedListener  listener = data -> {
    // update UI
}

// onCreate
userViewModel.registerListener(userListener);
userViewModel.loadUsers();
…
//onDestroy
userViewModel.unregisterListener(listener);
```

LiveData

With LiveData

```
// ViewModel
private MutableLiveData<List<User>> users;
public LiveData<List<User>> getUsers() {
    if (users == null) {
        users = new MutableLiveData<List<User>>();
        userRepository.loadUsers(data -> {
            userList.setValue(data);
        })
    }
    return users;
}
```

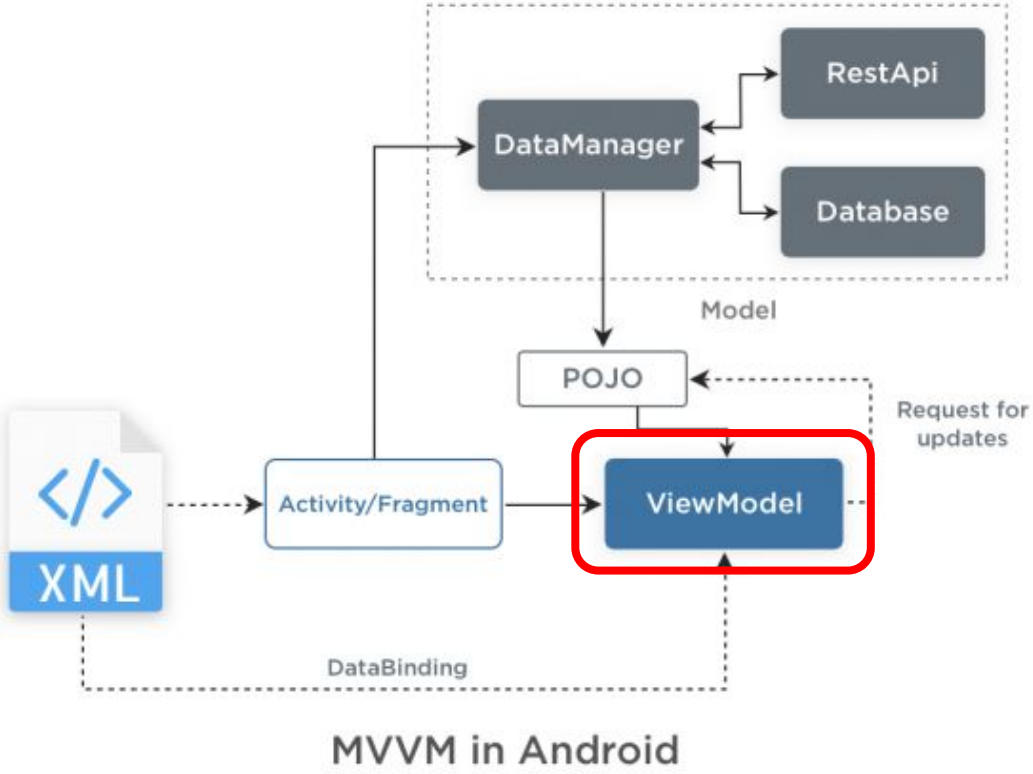
```
// Activity
getUsers().observe(this, users -> {
            // update UI
        });
```

LiveData

Transformations

```
// map
LiveData<User> userLiveData = ...;
LiveData<String> userName = Transformations.map(userLiveData, user -> {
    user.name + " " + user.lastName
});

// switchMap
private LiveData<User> getUser(String id) {
  ...;
}

LiveData<String> userId = ...;
LiveData<User> user = Transformations.switchMap(userId, id -> getUser(id) );
```

ViewModel



MVVM in Android

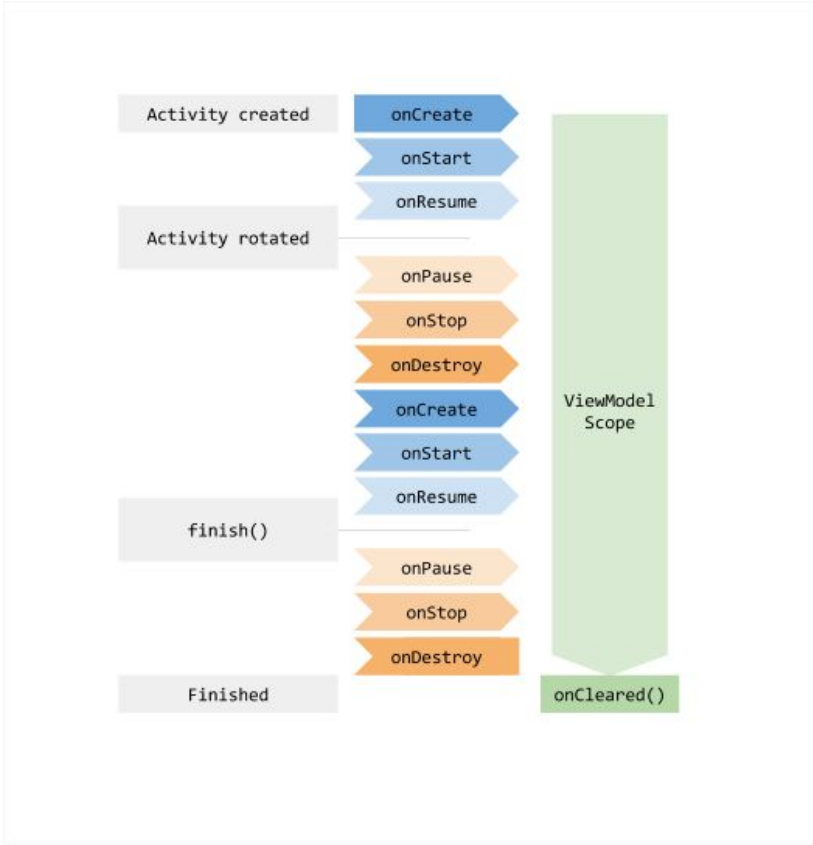
ViewModel

```
public class UserViewModel extends ViewModel {
    private MutableLiveData<List<User>> users;
    public LiveData<List<User>> getUsers() {
        if (users == null) {
            users = new MutableLiveData<List<User>>();
            userRepository.loadUsers(data -> {
                userList.setValue(data);
            })
        }
        return users;
    }
}
```

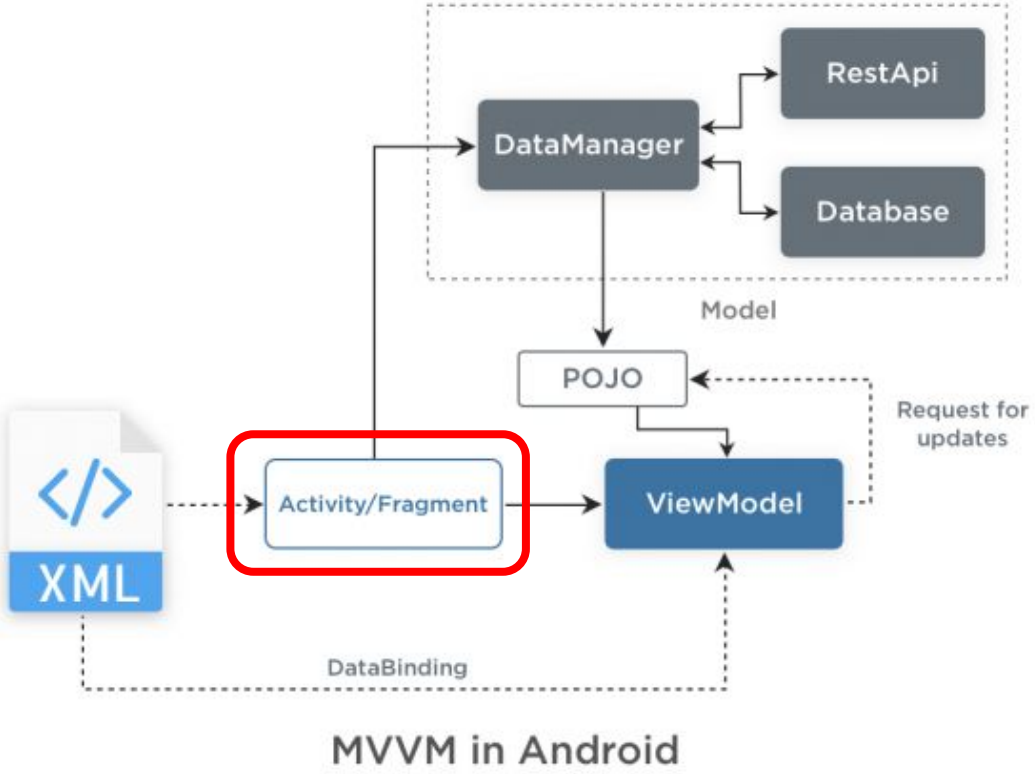
ViewModel

```
public class UserActivity extends AppCompatActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        UserViewModel model = ViewModelProviders.of(this).get(UserViewModel.class);
        model.getUsers().observe(this, users -> {
            // update UI
        });
    }
}
```

ViewModel



Lifecycles



MVVM in Android

Lifecycles

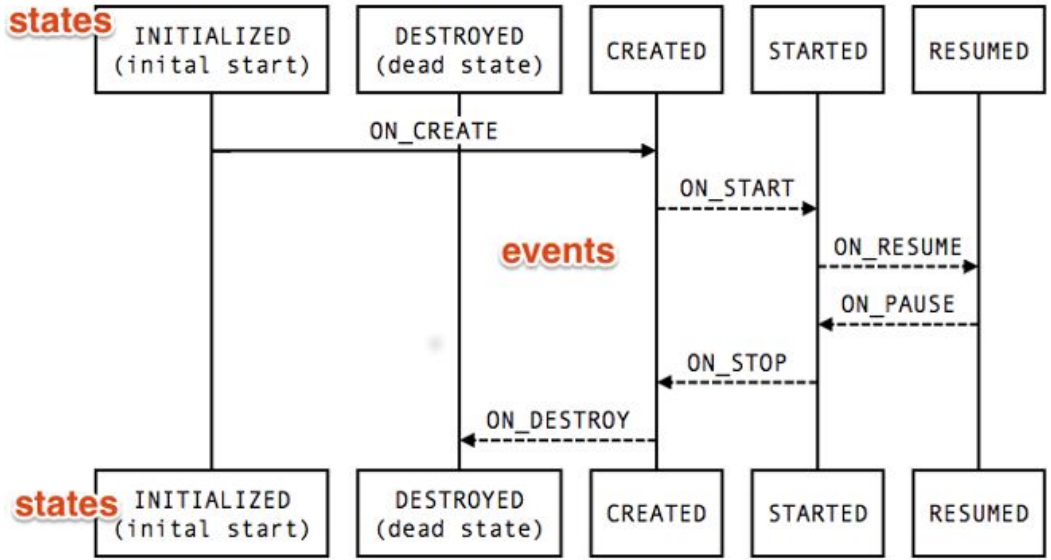
```
public class UserActivity extends AppCompatActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        UserViewModel model = ViewModelProviders.of(this).get(UserViewModel.class);
        model.getUsers().observe(this, users -> {
            // update UI
        });
    }
}
```

Lifecycles

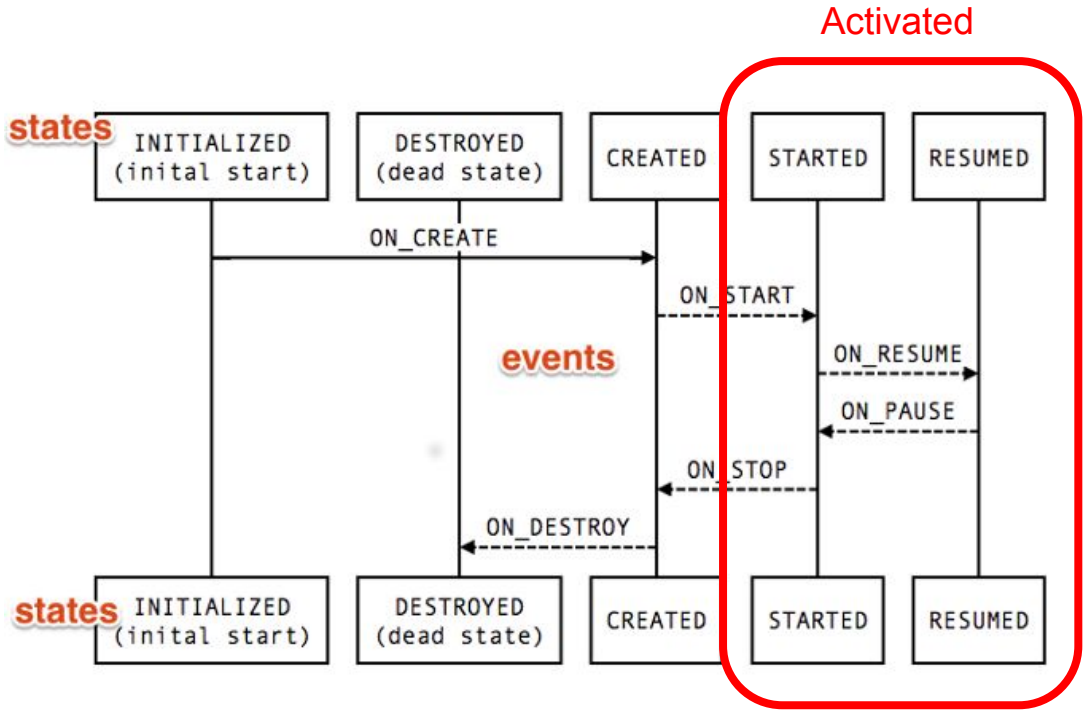
Fragments and Activities in Support Library 26.1.0 and later already implement the `LifecycleOwner` interface.

```
class AppCompatActivity extends Activity implements LifecycleOwner

class Fragment implements LifecycleOwner
```

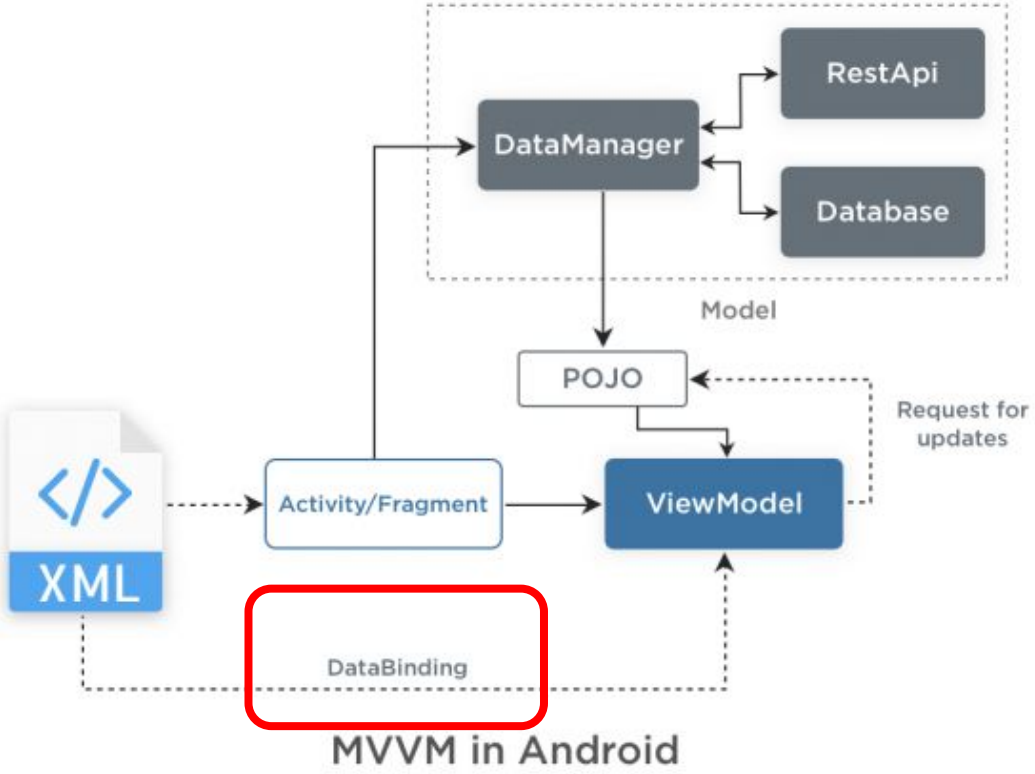
Lifecycles



Lifecycles



DataBinding



DataBinding

```java
public class User {
    public String firstname;
    public String lastname;
    public int age;
    public String gender;

    public User(String firstname, String lastname, int age, String gender){
        this.firstname = firstname;
        this.lastname = lastname;
        this.age = age;
        this.gender = gender;
    }
}
```

```xml
<RelativeLayout >
        <TextView
            android:id="@+id/firstnameLabel"
            android:text="@string/firstname" />

        <TextView
            android:id="@+id/firstnameTextView"
            android:text="@{user.firstname}" />

        <TextView
            android:id="@+id/lastnameLabel"
            android:text="@string/lastname" />

        <TextView
            android:id="@+id/lastnameTextView"
            android:text="@{user.lastname}" />
    </RelativeLayout>
</layout>
```

Data Binding

DataBinding

ViewModel

```
class UserModel extends ViewModel  {

    public final MutableLiveData<String> nickName = new MutableLiveData<>();

    public final MutableLiveData<Integer> age = new MutableLiveData<>();

    ...

}
```

DataBinding

XML

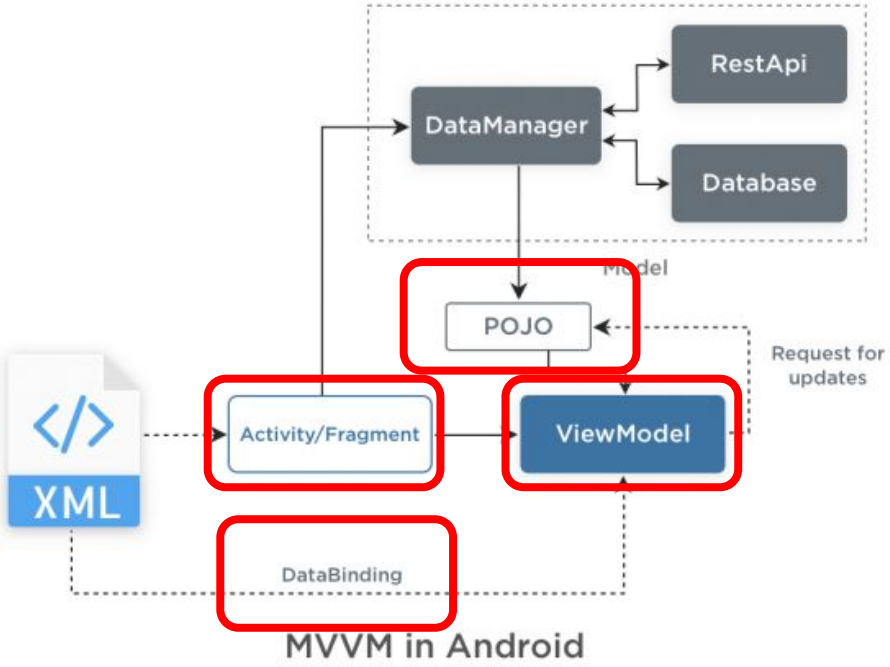
```
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable name="user" type="com.example.UserModel"/>
    </data>
    ...
        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{user.nickName}" />
    ...
</layout>
```

DataBinding

Activity

```
class UserActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        UserModel userModel = ViewModelProviders.of(getActivity()).get(UserModel.class)

        val binding: UserBinding = DataBindingUtil.setContentView(this, R.layout.user)

        binding.setLifecycleOwner(this)

        binding.viewmodel = userModel
    }

}
```

LiveData



- LiveData
- ViewModel
- Lifecycles
- Data Binding

Kotlin

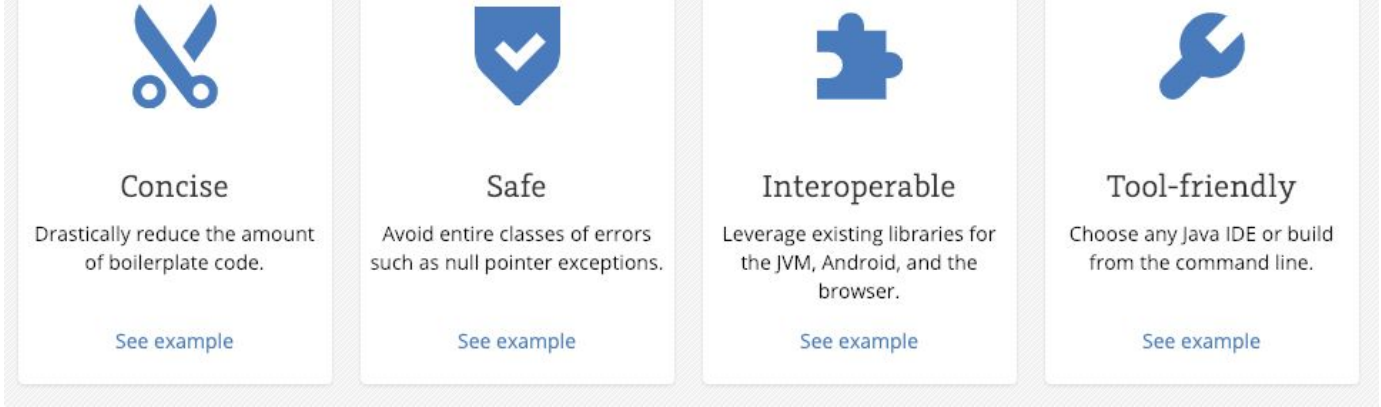
Kotlin

https://kotlinlang.org/

Kotlin

- A statically typed, cross-platform, general-purpose programming language with type inference.
- Mainly targets the JVM, but also compiles to JavaScript or native code (via LLVM).
- Sponsored by JetBrains, a software development company based in Prague, and is also backed by Google under the Kotlin Foundation.

Why Kotlin



Null Safety

```
// Java
void int getLastNameLength(User user) {
    if (user != null
        && user.info != null
        && user.info.basicInfo != null
        && user.info.basicInfo.name != null
        && user.info.basicInfo.name.lastName != null) {
        return user.info.basicInfo.name.lastName.length();
    } else {
        return -1;
    }
}
```

Null Safety

```java
// Java
void int getLastNameLength(User user) {
    if (user != null
        && user.info != null
        && user.info.basicInfo != null
        && user.info.basicInfo.name != null
        && user.info.basicInfo.name.lastName != null) {
        return user.info.basicInfo.name.lastName.length();
    } else {
        return -1;
    }
}
```

```kotlin
// Kotlin
fun getLastNameLength(val user: User?)
    = user?.info?.basicInfo?.name?.lastName?.length() ?: -1
```

Default Arguments

```
// Java
class View {
    public View(Context context) { … }

    public View(Context context, @Nullable AttributeSet attrs) {
        this(context, attrs, 0);
    }

    public View(Context context, @Nullable AttributeSet attrs,
                int defStyleAttr) {
        this(context, attrs, defStyleAttr, 0);
    }

    public View(Context context, @Nullable AttributeSet attrs,
                int defStyleAttr, int defStyleRes) {
        this(context);
        ...
    }
}
```

Default Arguments

```
// Java
class View {
    public View(Context context) { … }

    public View(Context context, @Nullable AttributeSet attrs) {
        this(context, attrs, 0);
    }

    public View(Context context, @Nullable AttributeSet attrs,
                int defStyleAttr) {
        this(context, attrs, defStyleAttr, 0);
    }

    public View(Context context, @Nullable AttributeSet attrs,
                int defStyleAttr, int defStyleRes) {
        this(context);
        ...
    }
}
```

```
// Kotlin
class View(context: Context, attrs: AttributeSet? = null, defStyleAttr: Int = 0,
    defStyleRes: Int = 0)
```

Lazy Initiation

```
// Java
private MediaPlayer mPlayer;

public synchronized void play(String url) {
    if (mPlayer == null) {
        mPlayer = createPlayer();
    }
    mPlayer.play(url);
}
```

Lazy Initiation

```
// Java
private volatile MediaPlayer mPlayer;

public synchronized void play(String url) {
    if (mPlayer == null) {
        mPlayer = createPlayer();
    }
    mPlayer.play(url);
}
```

```
// Kotlin
private val mPlayer: MediaPlayer by lazy {
    createPlayer()
}

fun play(val url: String) = mPlayer.play(url)
```

Data class

```
// Java
class User {
    public final String firstName;
    public final String lastName;
    public final int age;
    public final String avatarUrl;
    public final List<User> friends;

    public User(String firstName, String lastName,
                int age, List<User> friends) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
        this.avatarUrl = avatarUrl;
        this.friends = friends;
    }
}
```

Data class

```
// Java
class User {
    public final String firstName;
    public final String lastName;
    public final int age;
    public final String avatarUrl;
    public final List<User> friends;

    public User(String firstName, String lastName,
                int age, List<User> friends) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
        this.avatarUrl = avatarUrl;
        this.friends = friends;
    }
}
```

```
// Kotlin
data class User(val firstName: String, val lastName: String,
                val age: Int, val avatarUrl: String, val List<User> friends)
```

Data class

```
// Java
User mUser;
 …

void onAvatarUpdated(String newAvatarUrl) {
    List<User> friends = new ArrayList<>();
    for (User user : mUser.friends) {
        friends.add(user);
    }
    mUser = new User(mUser.firstName, mUser.lastName,
                            mUser.age, newAvatarUrl, friends)
}
```

Data class

```java
// Java
User mUser;
 …

void onAvatarUpdated(String newAvatarUrl) {
    List<User> friends = new ArrayList<>();
    for (User user : mUser.friends) {
        friends.add(user);
    }
    mUser = new User(mUser.firstName, mUser.lastName,
                     mUser.age, newAvatarUrl, friends)
}
```

```kotlin
// Kotlin
var mUser

newAvartarUrl -> {
    mUser = mUser.copy(avatarUrl = newAvatarUrl)
}
```

Extension Functions

```
// Java
public class SwapableArrayList<T> extends ArrayList<T> {

    public void swap(int index1, int index2) {
        T temp = get(index1);
        set(index1, get(index2));
        set(index2, temp);
    }

}
```

Extension Functions

```
// Java
public class SwapableArrayList<T> extends ArrayList<T> {


    public void swap(int index1, int index2) {
        T temp = get(index1);
        set(index1, get(index2));
        set(index2, temp);
    }

}
```

```
// Kotlin
fun MutableList<Int>.swap(index1: Int, index2: Int) {
    val tmp = this[index1]
    this[index1] = this[index2]
    this[index2] = tmp
}
```

Extension Properties

```
// Java
public class LastIndexArrayList<T> extends ArrayList<T> {

    public void getLastIndex() {
        return getSize() - 1;
    }

}
```

Extension Properties

```
// Java
public class LastIndexArrayList<T> extends ArrayList<T> {

    public void getLastIndex() {
        return getSize() - 1;
    }

}
```

```
// Kotlin
val <T> List<T>.lastIndex: Int
    get() = size - 1
```

Extension Properties

```
// Java
public class LastIndexArrayList<T> extends SwapableArrayList<T> {

    public void getLastIndex() {
        return getSize() - 1;
    }

}
```

```
// Kotlin
val <T> List<T>.lastIndex: Int
    get() = size - 1
```

When Expression

```
// Kotlin
when (x) {
    1 -> print("x == 1")
    2 -> print("x == 2")
    else -> { // Note the block
        print("x is neither 1 nor 2")
    }
}
```

When Expression

```kotlin
// Kotlin
when (x) {
    1 -> print("x == 1")
    2 -> print("x == 2")
    else -> { // Note the block
        print("x is neither 1 nor 2")
    }
}
```

```java
// Java
switch(x) {
    case 1:
        print("x == 1");
        break;
    case 2:
        print("x == 2");
        break;
    default:
        print("x is neither 1 nor 2");
        break;
}
```

When Expression

```
// Kotlin
when (x) {
    in 1..10 -> print("x is in the range")
    in validNumbers -> print("x is valid")
    !in 10..20 -> print("x is outside the range")
    else -> print("none of the above")
}
```

When Expression

// Kotlin

```
when (x) {
    in 1..10 -> print("x is in the range")
    in validNumbers -> print("x is valid")
    !in 10..20 -> print("x is outside the range")
    else -> print("none of the above")
}
```

// Java

Functional Programming

Kotlin functions are *first-class*, which means that they can be stored in variables and data structures, passed as arguments to and returned from other higher-order functions.

Functional Programming

```
// OOP
class Calculator {
    double plus(double a, double b) { return a + b; }

    double minus(double a, double b) { return a - b; }

    double multiply(double a, double b) { return a * b; }

    double divide(double a, double b) { return a / b; }

    ...
}
```

Functional Programming

```
// OOP
class Calculator {
    double plus(double a, double b) { return a + b; }

    double minus(double a, double b) { return a - b; }

    double multiply(double a, double b) { return a * b; }

    double divide(double a, double b) { return a / b; }

    ...
}
```

```
// FP
val plus: (Double, Double) -> Double = { a, b -> a + b }
val minus: (Double, Double) -> Double = { a, b -> a - b }
val multiply: (Double, Double) -> Double = { a, b -> a * b }
val divide: (Double, Double) -> Double = { a, b -> a / b }

fun operate(operator: (Double, Double) -> Double, val a: Double,
    val b: Double)  = operator(a, b)

val sum = operate(4, 6, plus)
```

Functional Programming

```
fun <T, R> Collection<T>.fold(
    initial: R,
    combine: (acc: R, nextElement: T) -> R
): R {
    var accumulator: R = initial
    for (element: T in this) {
        accumulator = combine(accumulator, element)
    }
    return accumulator
}
```

```
val items = listOf(1, 2, 3, 4, 5)

// sum
items.fold(0, {
    acc: Int, i: Int ->
    val result = acc + i
})

// multiply
val product = items.fold(1, Int::times)

// joined to string
val joinedToString = items.fold("Elements:", { acc, i -> acc + " " + i })
```

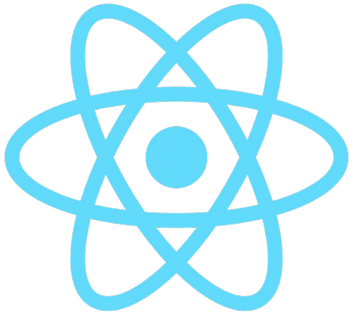
KTX

```
// Kotlin
sharedPreferences.edit()
    .putBoolean("key", value)
    .apply()


db.beginTransaction()
try {
    // insert data
    db.setTransactionSuccessful()
} finally {
    db.endTransaction()
}
```

```
// Kotlin + KTX
sharedPreferences.edit {
    putBoolean("key", value)
}


db.transaction {
    // insert data
}
```

Flutter

Cross-Platform

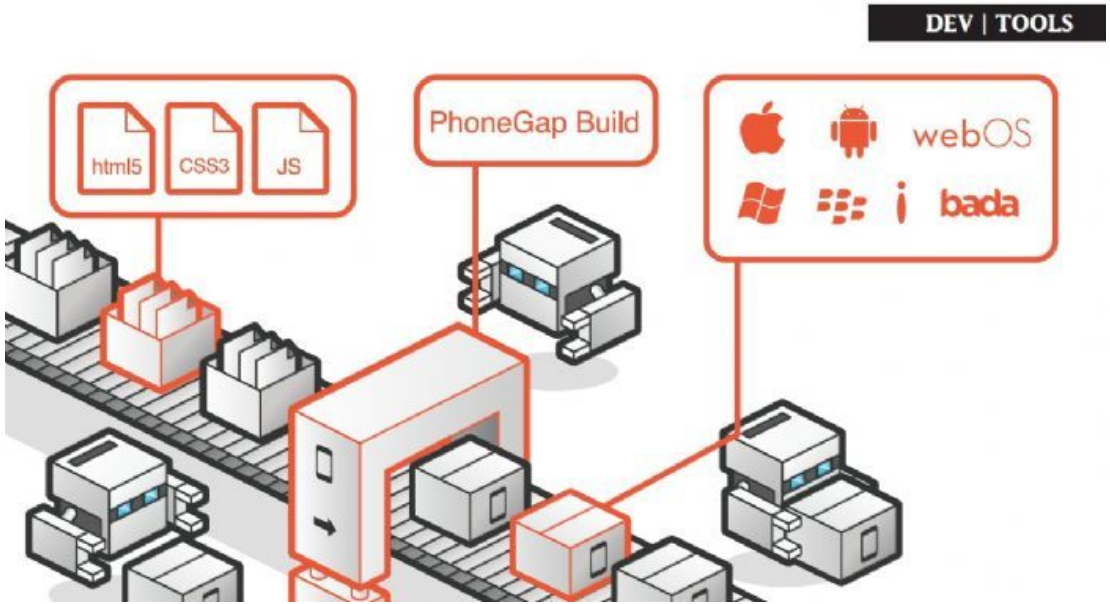


PhoneGap

Flutter

ByteDance字节跳动

PhoneGap



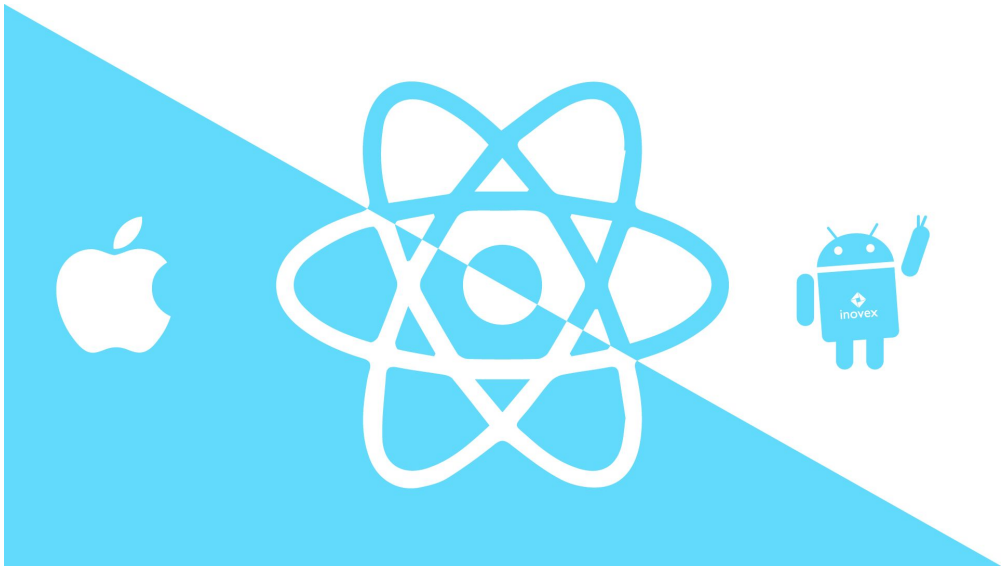
Flutter

ByteDance字节跳动

PhoneGap

- Poor Performance
- Lack Of UI Widgets
- No complete support to the features of an OS

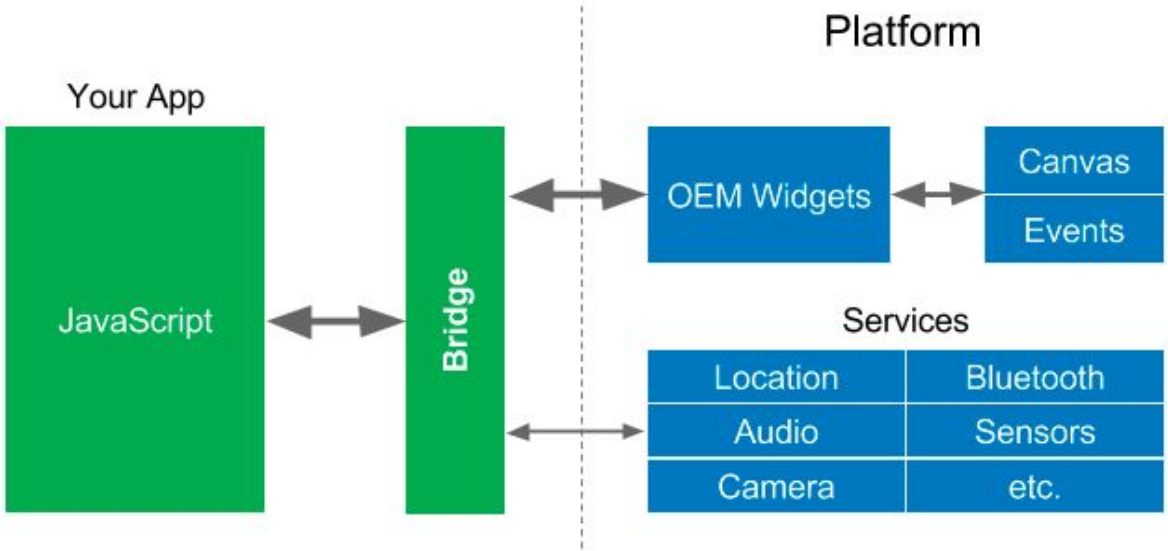
ReactNative



Flutter

ByteDance字节跳动

ReactNative

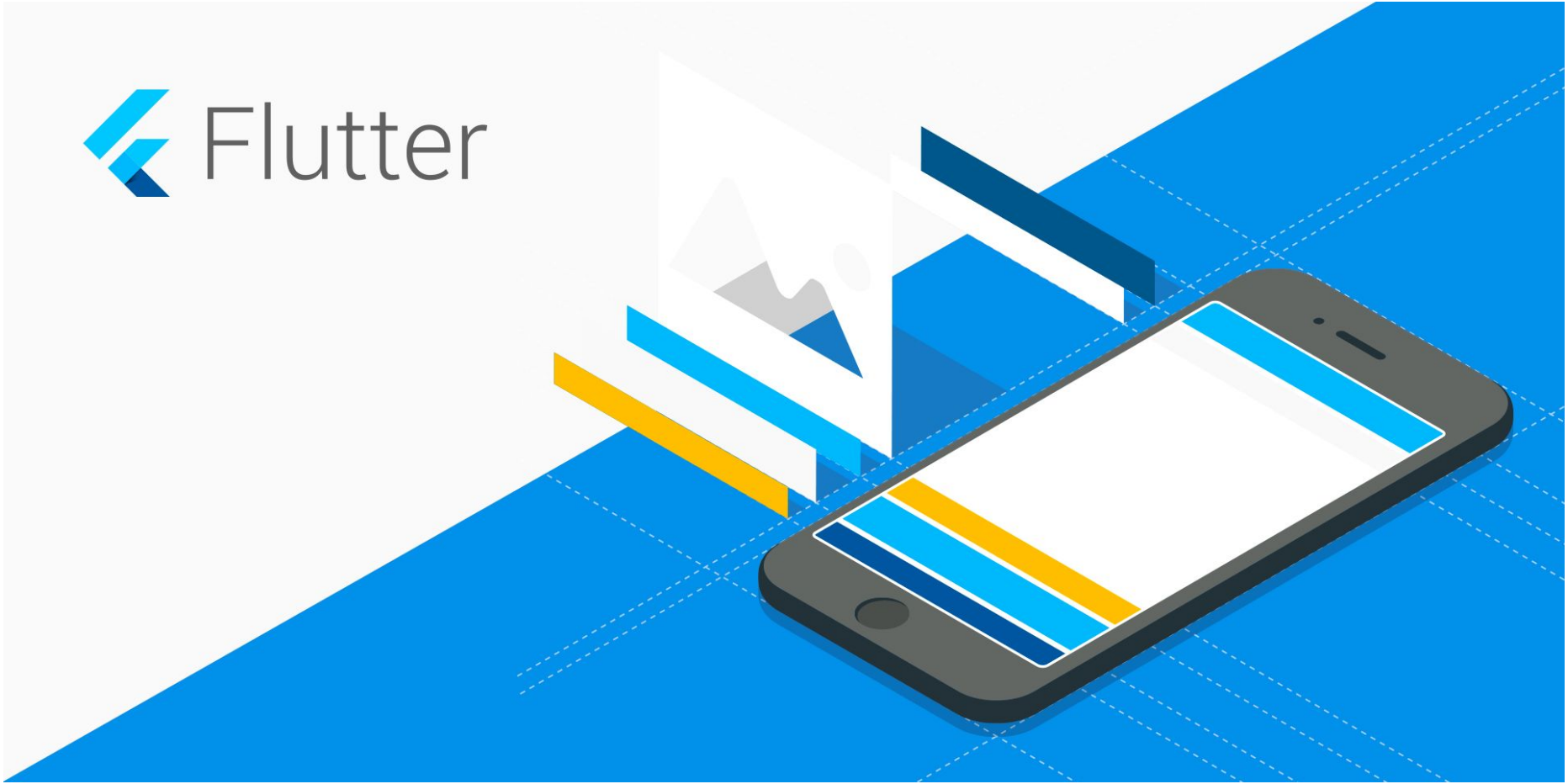


Flutter

ReactNative

- Poor Compatibility
- Lack Security Robustness
- Poor Memory Management

Flutter



Flutter

https://flutter.io/

ByteDance 字节跳动

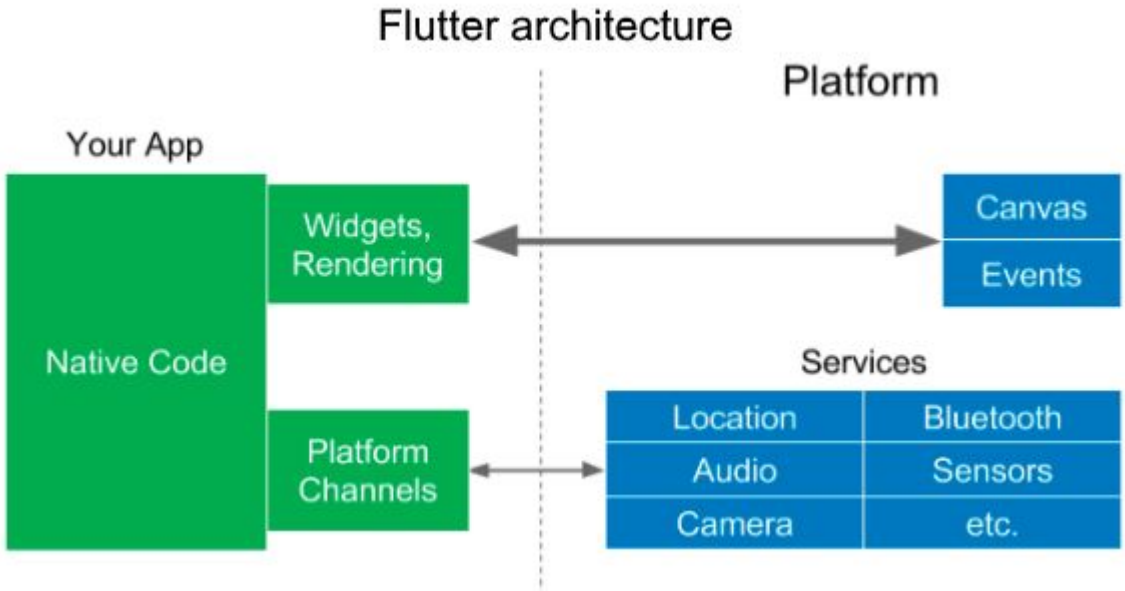
Flutter

Flutter allows you to build beautiful native apps on iOS and Android from a single codebase.

Dart

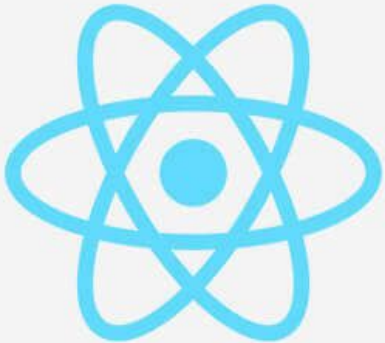
```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      decoration: BoxDecoration(color: Colors.white),
      child: Center(
        child: Text(
          'Hello World',
          textDirection: TextDirection.ltr,
          style: TextStyle(
            fontSize: 32.0,
            color: Colors.black87,
          ),
        ),
      ),
    );
  }
}
```

Flutter



Flutter

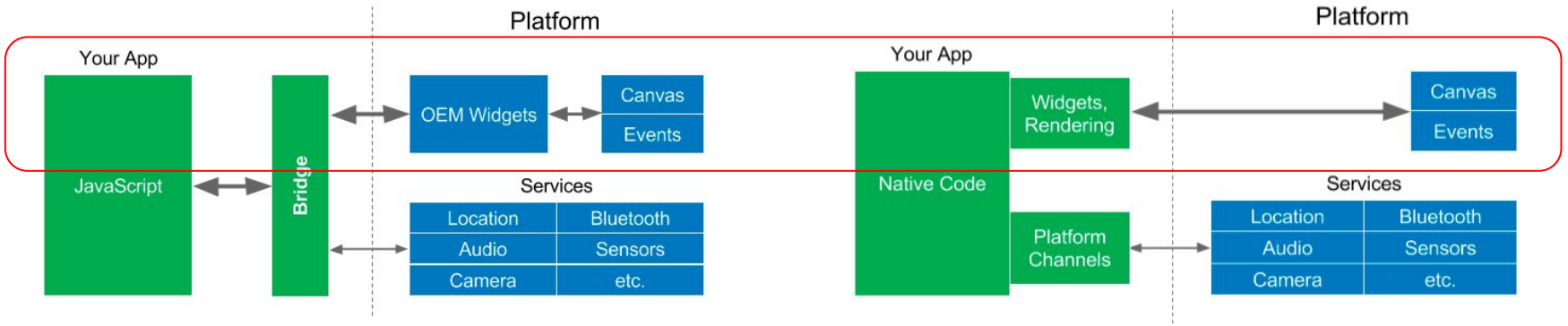
REACT NATIVE



VS

FLUTTER

Flutter

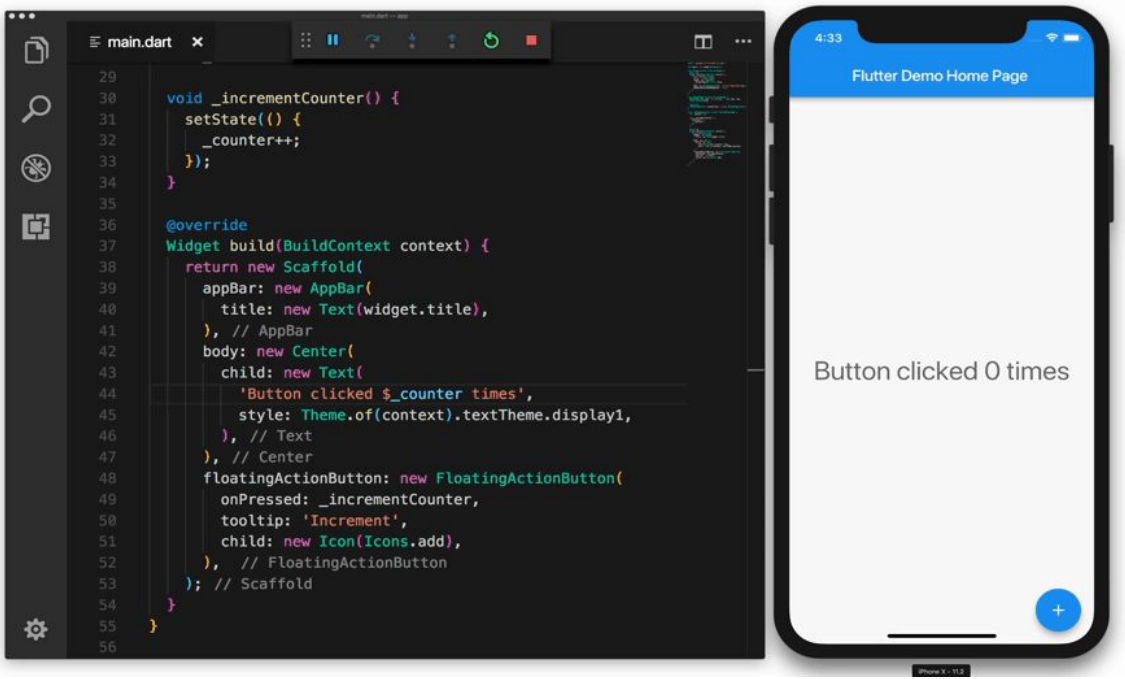


Flutter

Flutter

- Fast Development
- Expressive and Flexible UI
- Native Performance

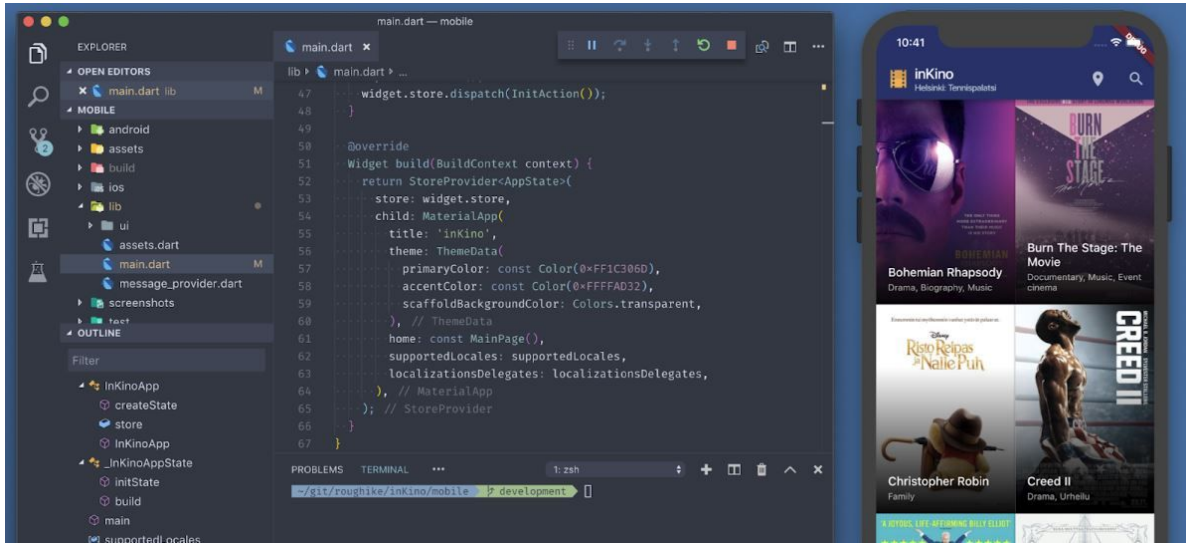
Hot Reload



Flutter

ByteDance 字节跳动

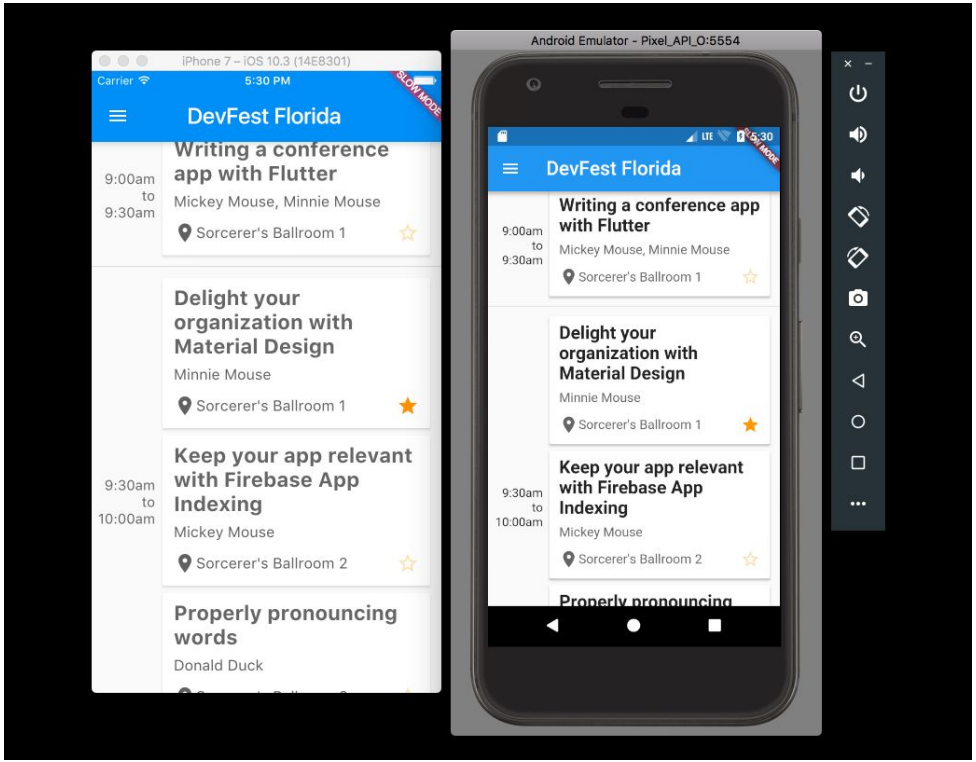
Flexible UI



Flutter

ByteDance 字节跳动

Cross-Platform



Flutter

ByteDance 字节跳动

Catalog

- High-efficient development: Jetpack

- Functional programming: Kotlin

- Cross-platform framework: Flutter

THANKS

.