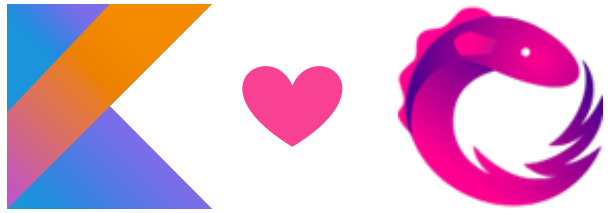


Maximize Android Mobility with **Kotlin** and **Reactive** Paradigm

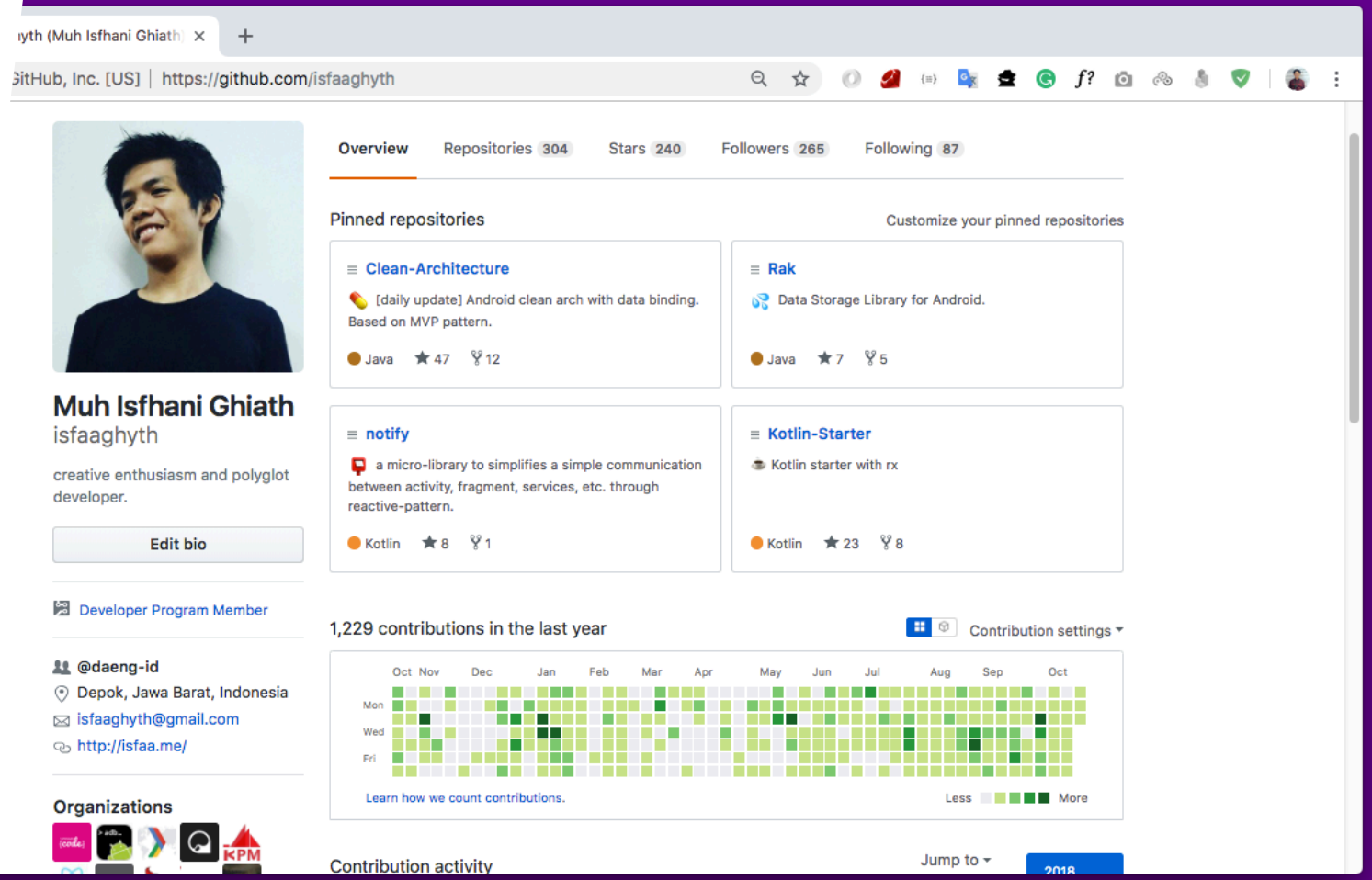
Muh **Isfha**ni Ghiath

@isfaaghyth / isfa@daeng.id

Chief Warrior, **Daeng.id**
Community Lead, **Developer Student Clubs**
Software Engineer Intern, **Tokopedia**
ROC (Representative), **Google Crowdsource**
Github Campus Expert (GCE), **Github**

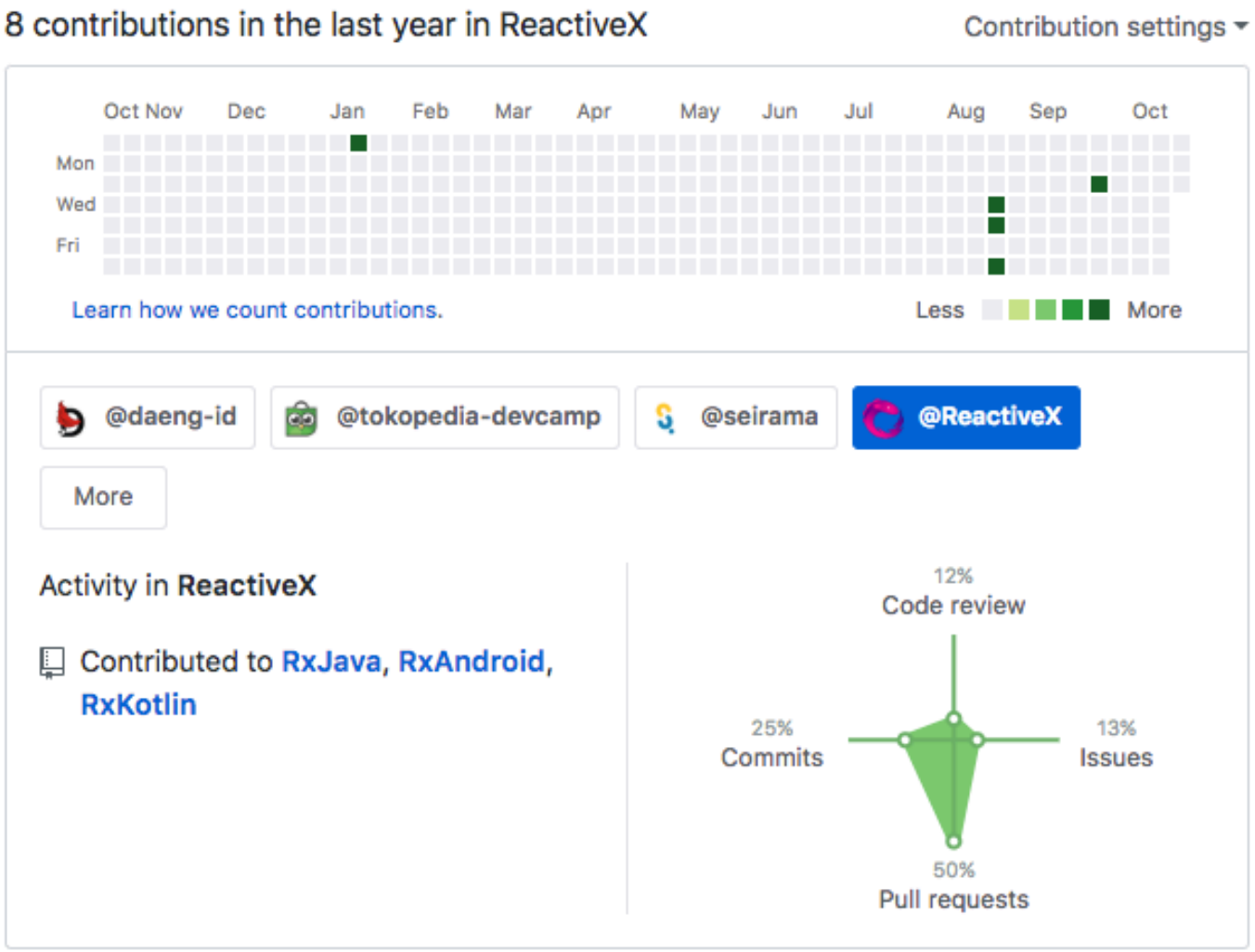
Find me at:

@isfaaghyth



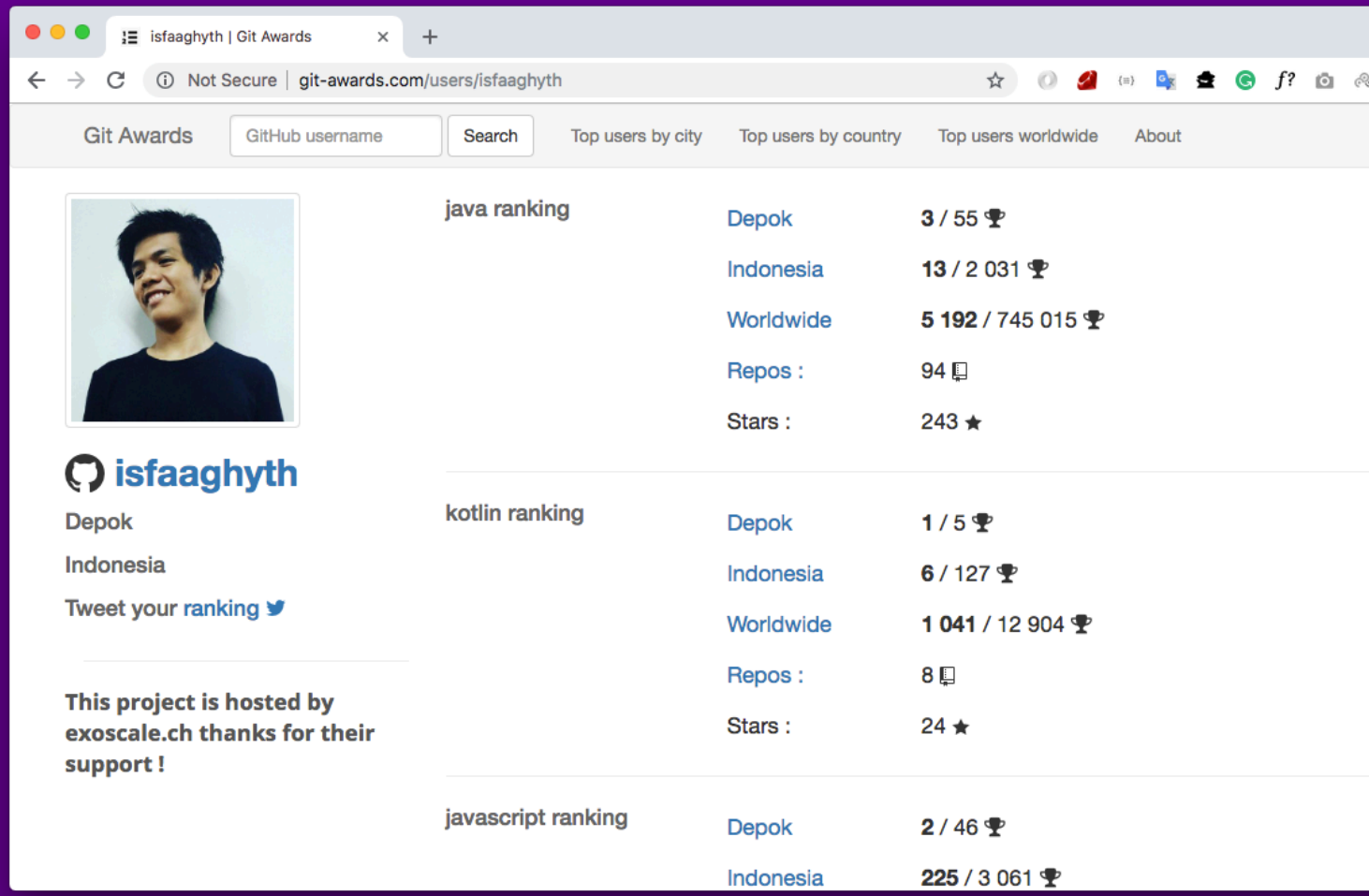
My contribution in Rx Project

RxJava, RxAndroid, RxKotlin.



Find me at:

@isfaaghyth

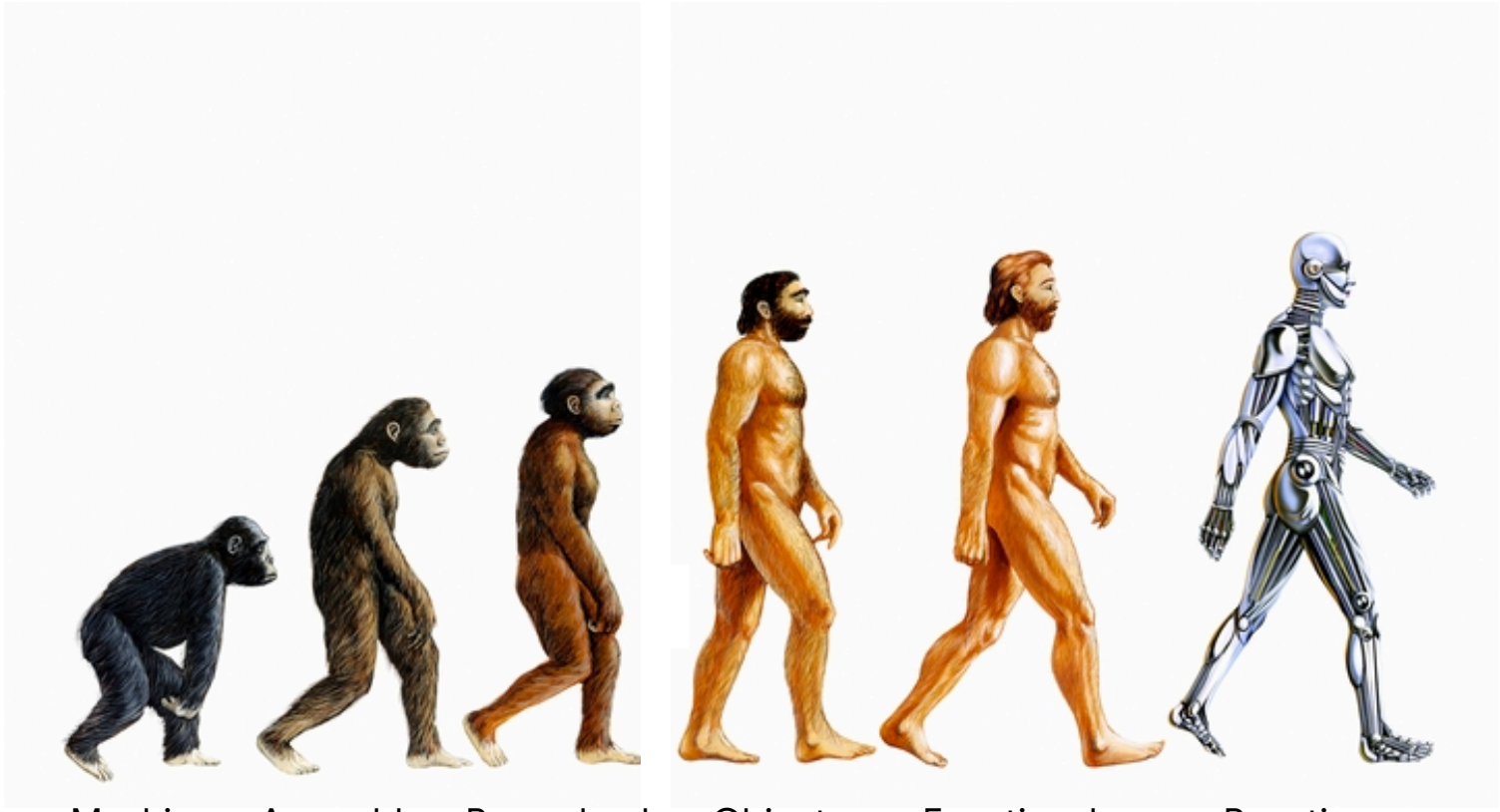


Reactive

/rēˈaktiv/

~~React Native~~

/rēˈak ney-tiv/



Machine Assembly Procedural Object Oriented Functional Reactive

Reactive Programming

In computing, reactive programming is a declarative **programming paradigm concerned** with **data streams** and the **propagation of change.**

ReactiveX

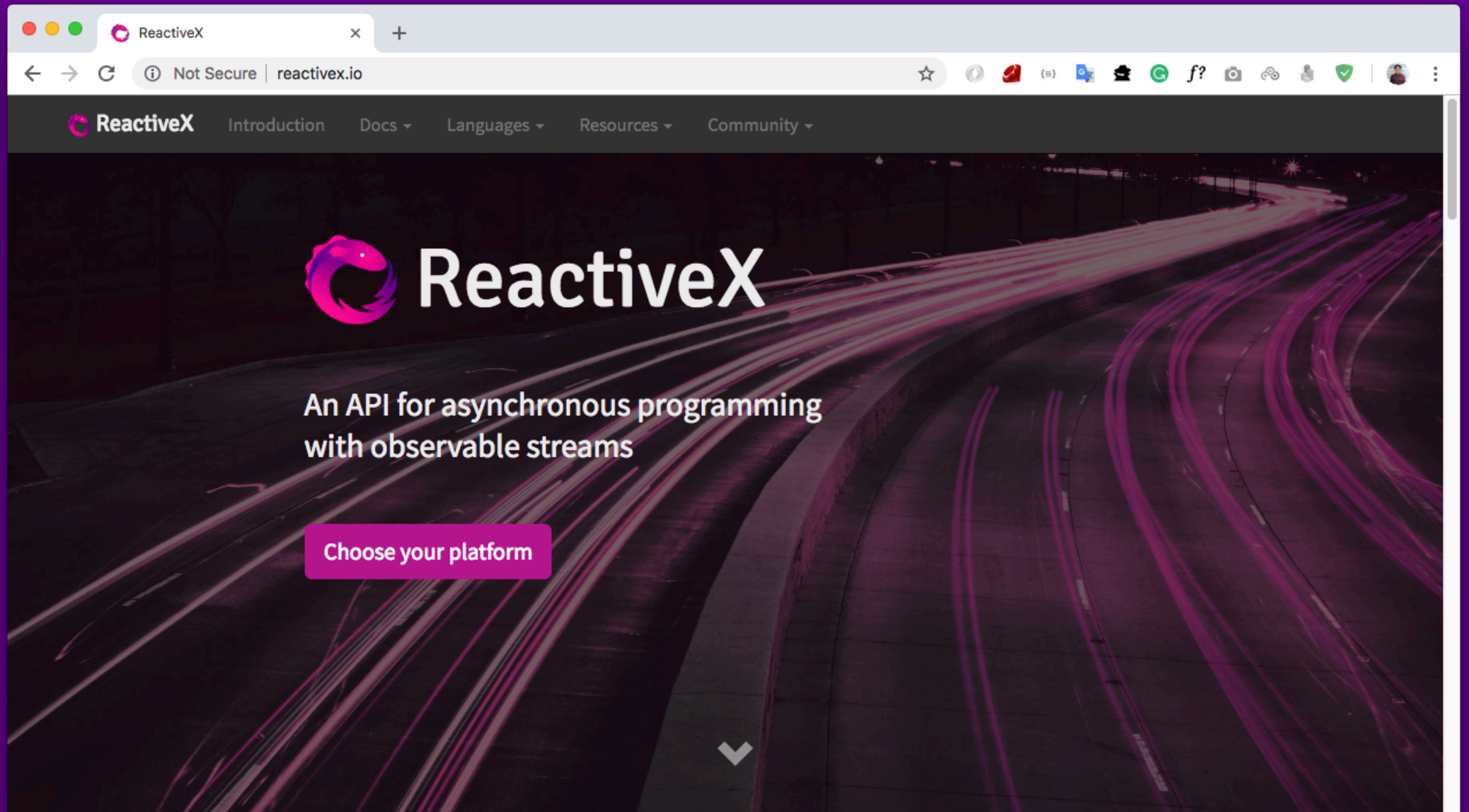
a library for composing asynchronous and event-based programs by using observables sequences.

Rx, which is a port of the Reactive-Extensions library from .NET, enables Android apps to be built in this `style`.

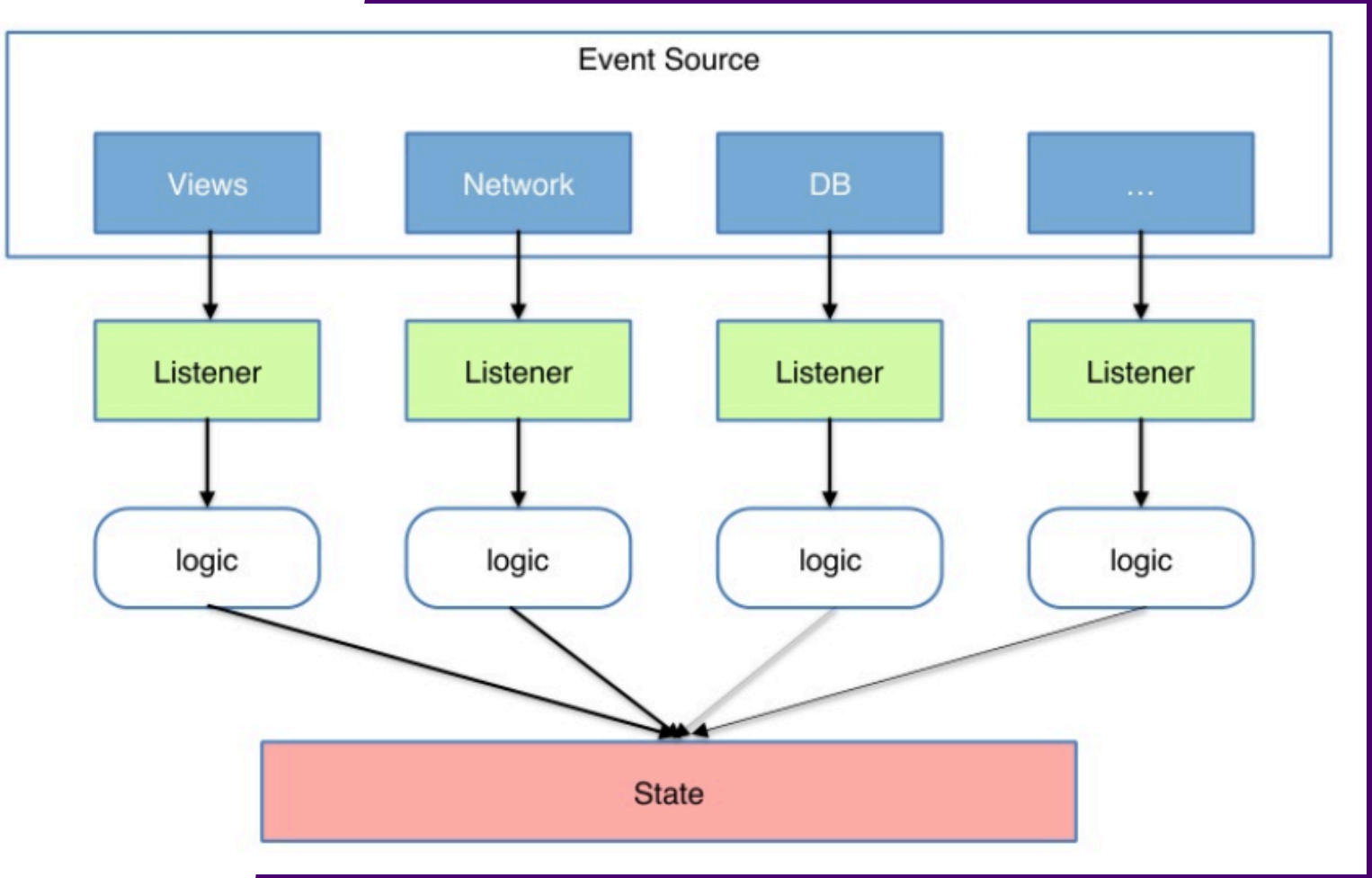
ReactiveX?

- composable data flow
- push concept
- combination of
  - observer pattern
  - iterator pattern
  - function programming

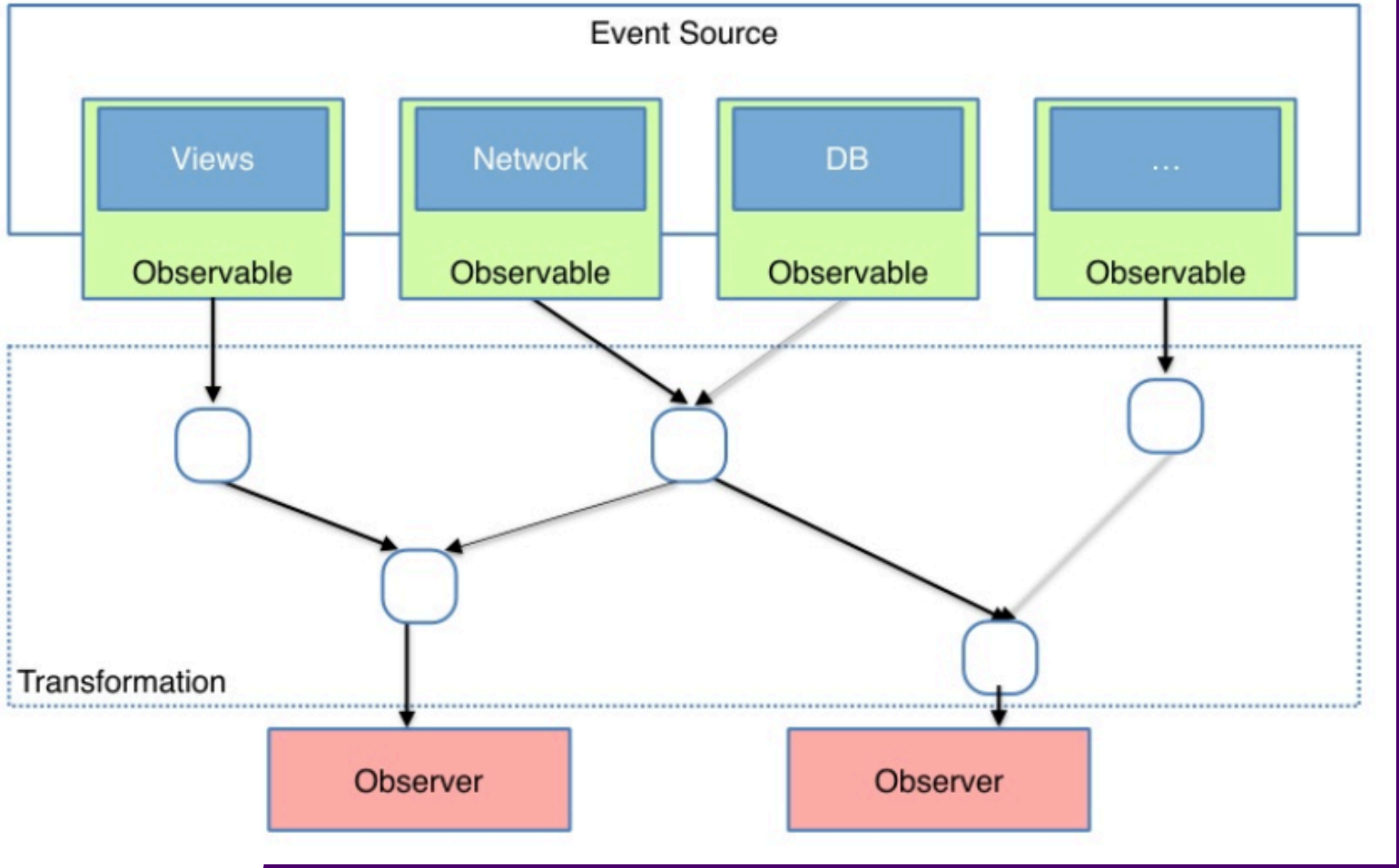
www.reactivex.io

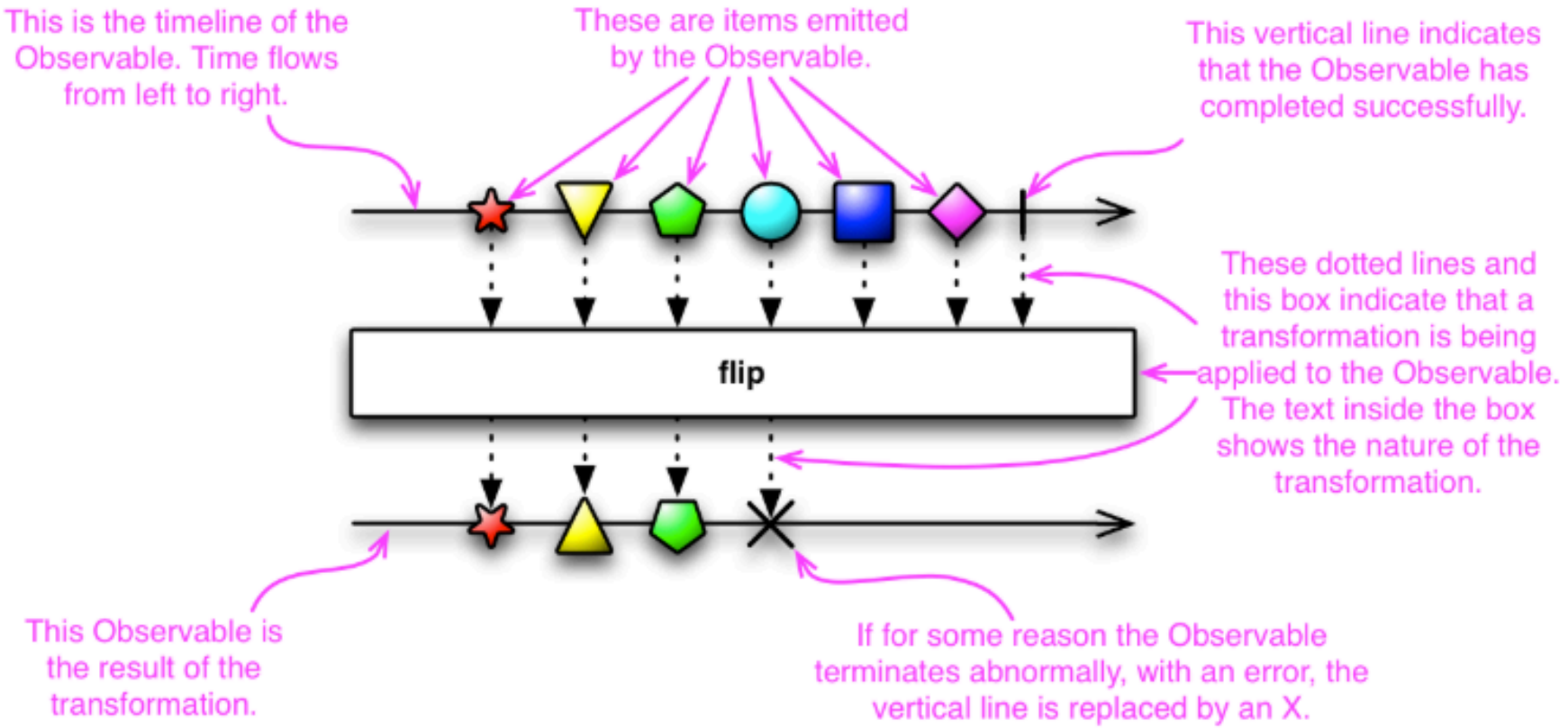


Typical Approach



Reactive Approach




This is the timeline of the Observable. Time flows from left to right.
These are items emitted by the Observable.
This vertical line indicates that the Observable has completed successfully.
These dotted lines and this box indicate that a transformation is being applied to the Observable. The text inside the box shows the nature of the transformation.
flip
This Observable is the result of the transformation.
If for some reason the Observable terminates abnormally, with an error, the vertical line is replaced by an X.

(**Java**) Simple String Transform with Rx

```java
@Test private void test() {
    Observable.just("Isfa", "Ganteng")
        .map(new Func1<String, String>() {
            @Override public String call(String s) {
                return s.toLowerCase();
            }
        })
        .reduce(new Func2<String, String, String>() {
            @Override public String call(String s1, String s2) {
                return s1 + "-" + s2;
            }
        })
        .subscribe(new Consumer<String>() {
            @Override public void call(String result) {
                Log.d("DSC", result);
            }
        });
}
```

(**Kt**) Simple String Transform with Rx and Functional

```
@Test fun test() {
    Observable.just("Isfa", "Ganteng")
        .map { s -> s.toLowerCase() }
        .reduce { (s1, s2) -> "$s1-$s2" }
        .subscribe { result -> Log.d("DSC", result) }
}
```

Data source:
“Isfa” dan “Ganteng”

Transformation

Subscription

Result:
“isfa-ganteng”

KeyPart

- Observable
- Observer or Subscriber
  - onNext(T)
  - onComplete(T)
  - onError(Throwable)
- Subject

What is ReactiveX good for?

Functional

Avoid intricate stateful programs, using clean input/output functions over observable streams.

Less is more

ReactiveX's operators often reduce what was once an elaborate challenge into a few lines of code.

Async error handling

Traditional try/catch is powerless for errors in asynchronous computations, but ReactiveX is equipped with proper mechanisms for handling errors.

Concurrency made easy

Observables and Schedulers in ReactiveX allow the programmer to abstract away low-level threading, synchronization, and concurrency issues.

What is ReactiveX good for?

- Making code simple and readable
- Async Processing
  - (no AsyncTask, AsyncTaskLoader, ...)
- Async Composition
  - RxJava offers simple chaining of async operations
  - Eliminates callback hell

GET /dscindonesia

```
{
   lead: [
        {
           name: “M.Isfha”,
           detail: 1278
        },
        {…}
   ]
}
```

GET /detail?id=

```
{
   img: “http://gavatar.com/isfha.png”,
   chapter: “STT Nurul Fikri”,
   region: “Jakarta”
}
```

Chaining Multiple Request with `flatMap`

```
interface Services {

    @GET("/dscindonesia")
    fun getDSCList(): Observable<DSCLead>

    @GET("/detail")
    fun getDetail(
        @Query("id") id: String
    ): Observable<DSCLeadDetail>

}
```

Observable

Observable

Chaining Multiple Request with `flatMap`

```
@Test fun chaining() {
    service().getDSCList()
            .flatMap { res1 -> service().getDetail(res1.detailId) }
            .SchedulerOn(Schedulers.io())
            .ObserveOn(AndroidThreads.mainThread())
            .subscribe()


}
```

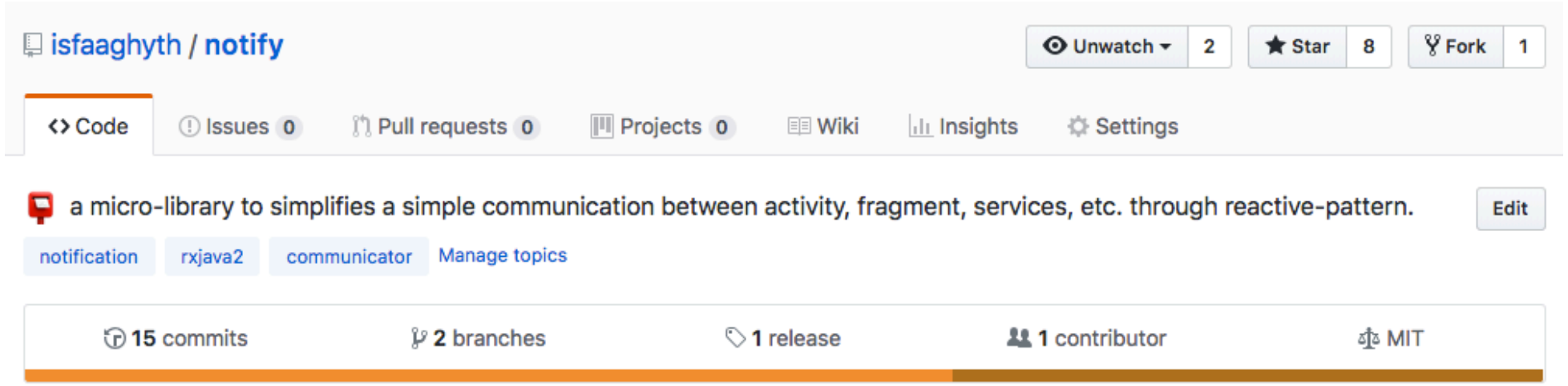
Operator

Schedulers

450+
operators

10+
Supported
Language

My First Library with Kt+Rx



Pubsub easily with Notify()

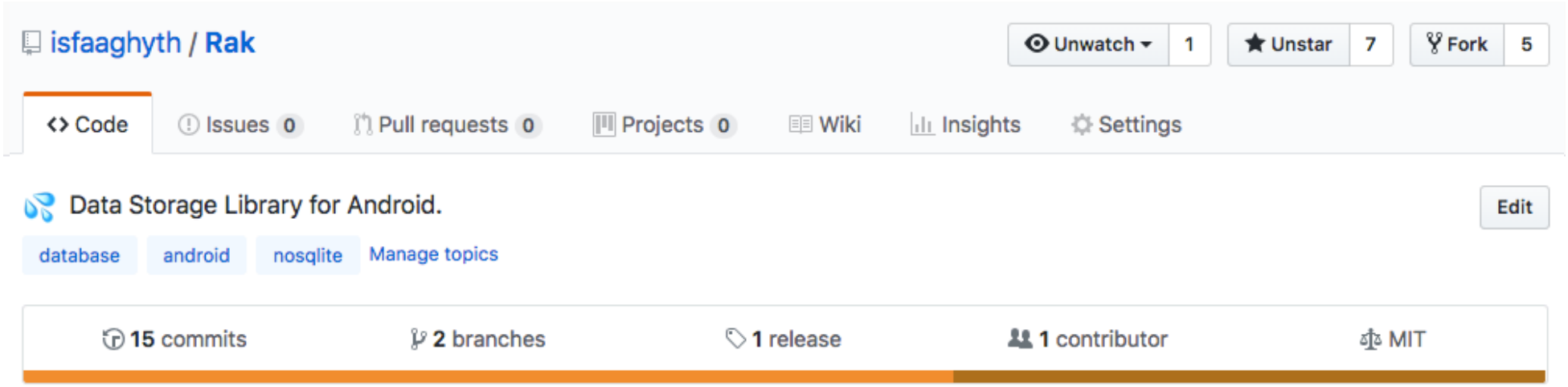
```
private val subject: Subject<Any> = PublishSubject.create()

fun <T> listen(tClass: Class<T>, subscriber: EventProvider,
               onNext: Consumer<T>): Disposable {
    return subject
            .subscribeOn(subscriber.io())
            .observeOn(subscriber.mainThread())
            .filter { o -> o.javaClass == tClass }
            .map { o -> o as T }
            .subscribe(onNext)
}
```

PublishSubject operator

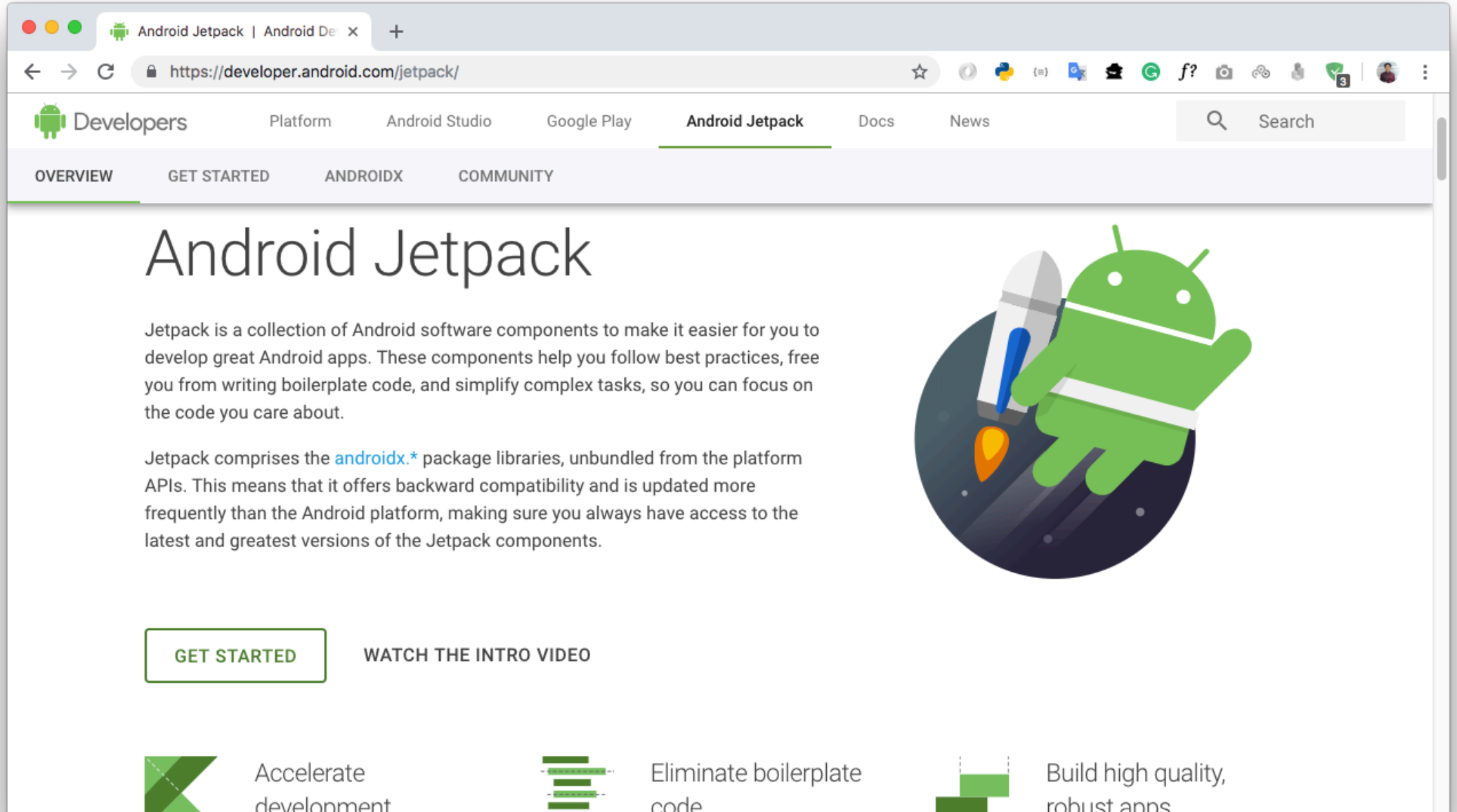
Schedulers

Subscriber

Keystore library with Rx



Android Architecture Components



Reactive Programming pada Android Menggunakan RxJava 2.0

Muh Isfhani Ghiath

Let’s develop your next app with **Rx**!

@isfaaghyth