ESCAPING FROM THE FRAMEWORK

guided by Clean Archictecture

Ubiratan Soares
Agosto / 2017

ERA UMA VEZ UM DESENVOLVEDOR ...

QUE PRECISAVA
CRIAR O NOVO APP
QUE MUDARIA TUDO

ELE FOI DIRETO NA FONTE PARA SABER COMO FAZER

API Guides

Introduction
Platform Architecture
App Components
App Resources
App Manifest
User Interface
Animation and Graphics
Computation
Media and Camera
Location and Sensors
Connectivity
Text and Input
Data Storage
Storage Options
Data Backup
App Install Location

To read values, use `SharedPreferences` methods such as `getBoolean()` and `getString()`.

Here is an example that saves a preference for silent keypress mode in a calculator:

```
public class Calc extends Activity {
    public static final String PREFS_NAME = "MyPrefsFile";

    @Override
    protected void onCreate(Bundle state){
       super.onCreate(state);
       . . .

       // Restore preferences
       SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
       boolean silent = settings.getBoolean("silentMode", false);
       setSilent(silent);
    }

    @Override
    protected void onStop(){
       super.onStop();

      // We need an Editor object to make preference changes.
      // All objects are from android.context.Context
      SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
      SharedPreferences.Editor editor = settings.edit();
      editor.putBoolean("silentMode", mSilentMode);

      // Commit the edits!
      editor.commit();
    }
}
```

Using the Internal Storage

Use a Singleton Pattern

If your application makes constant use of the network, it's probably most efficient to set up a single instance of `RequestQueue` that will last the lifetime of your app. You can achieve this in various ways. The recommended approach is to implement a singleton class that encapsulates `RequestQueue` and other Volley functionality. Another approach is to subclass `Application` and set up the `RequestQueue` in `Application.onCreate()`. But this approach is discouraged; a static singleton can provide the same functionality in a more modular way.

A key concept is that the `RequestQueue` must be instantiated with the `Application` context, not an `Activity` context. This ensures that the `RequestQueue` will last for the lifetime of your app, instead of being recreated every time the activity is recreated (for example, when the user rotates the device).

Here is an example of a singleton class that provides `RequestQueue` and `ImageLoader` functionality:

```
public class MySingleton {
    private static MySingleton mInstance;
    private RequestQueue mRequestQueue;
    private ImageLoader mImageLoader;
    private static Context mCtx;

    private MySingleton(Context context) {
        mCtx = context;
        mRequestQueue = getRequestQueue();

        mImageLoader = new ImageLoader(mRequestQueue,
                new ImageLoader.ImageCache() {
            private final LruCache<String, Bitmap>
                    cache = new LruCache<String, Bitmap>(20);

            @Override
            public Bitmap getBitmap(String url) {
                return cache.get(url);
            }
```

Here is an example activity that implements the callback interfaces and adds them to the Google API Client:

```
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.common.api.GoogleApiClient.OnConnectionFailedListener;
import gms.drive.*;
import android.support.v4.app.FragmentActivity;

public class MyActivity extends FragmentActivity
        implements OnConnectionFailedListener {
    private GoogleApiClient mGoogleApiClient;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Create a GoogleApiClient instance
        mGoogleApiClient = new GoogleApiClient.Builder(this)
                .enableAutoManage(this /* FragmentActivity */,
                                  this /* OnConnectionFailedListener */)
                .addApi(Drive.API)
                .addScope(Drive.SCOPE_FILE)
                .build();

        // ...
    }
```

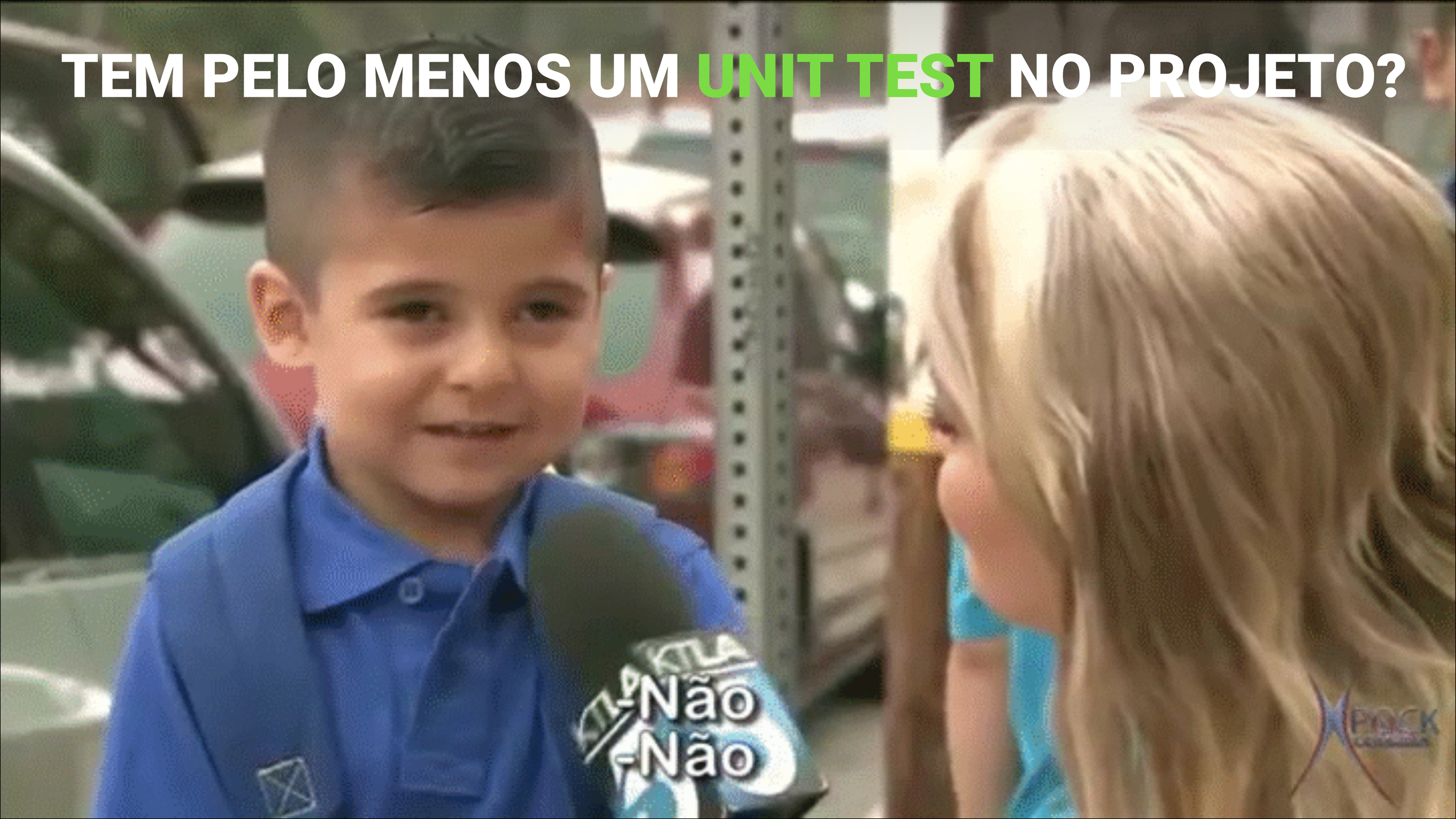
ALGUNS MESES
DEPOIS, O MUNDO NÃO
SERIA MAIS O MESMO


7:54
Google Play Store
Funny
Motion
Stickers
Face Camera — Snappy Photo
Fotoable,Inc.
14
INSTALL
Contains ads • In-app purchases
10
MILLION
Downloads
4.4
101,119
Photography
Similar
Face Camera = Motion Stickers + Artistic filters
+ Face Swap. By C403 Studio
READ MORE

PORÉM ...

SEM PAUSA PARA REFACTOR

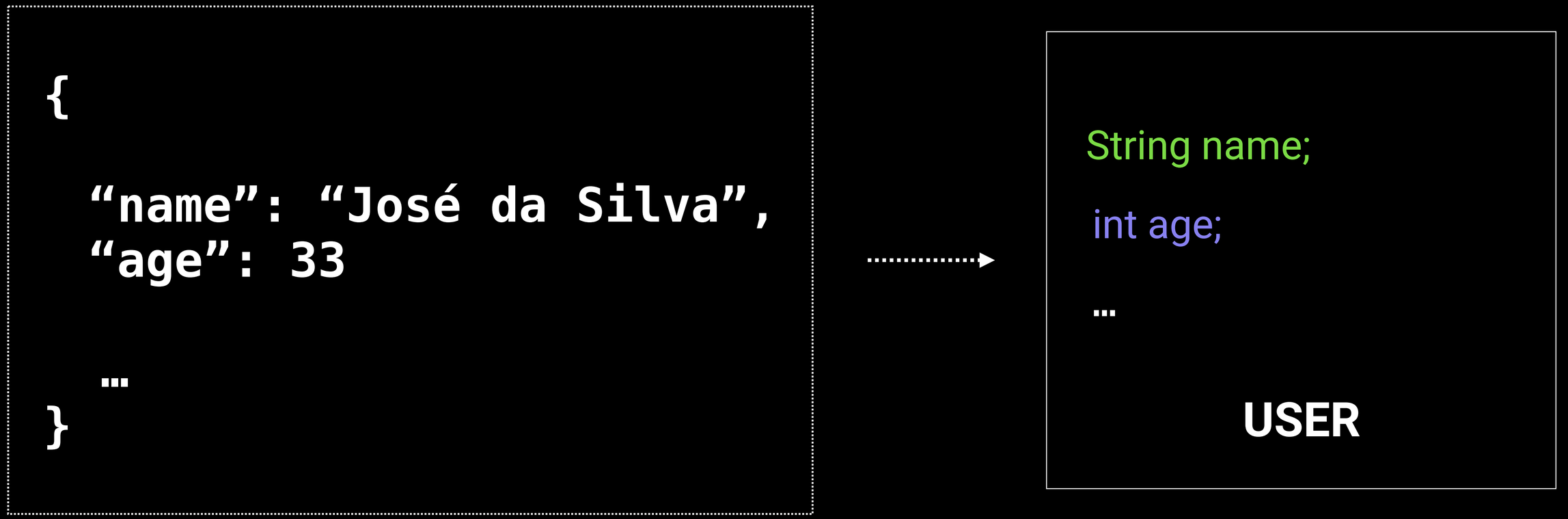
QUAIS OS PROBLEMAS NO MÉDIO PRAZO ???

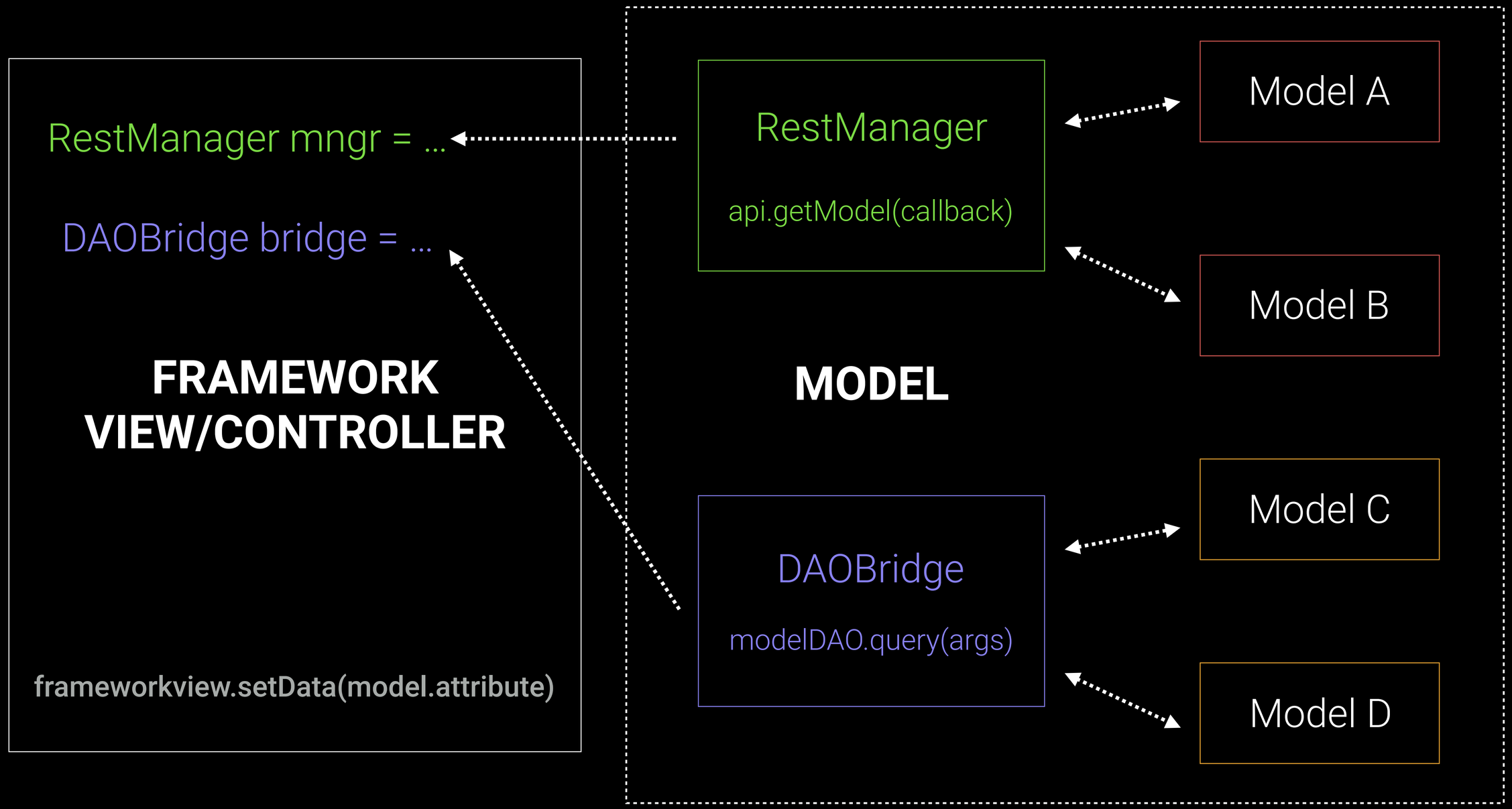

TEM PELO MENOS UM UNIT TEST NO PROJETO?
Não
Não


FAT MODELS / FAT VIEWS / FAT CONTROLLERS

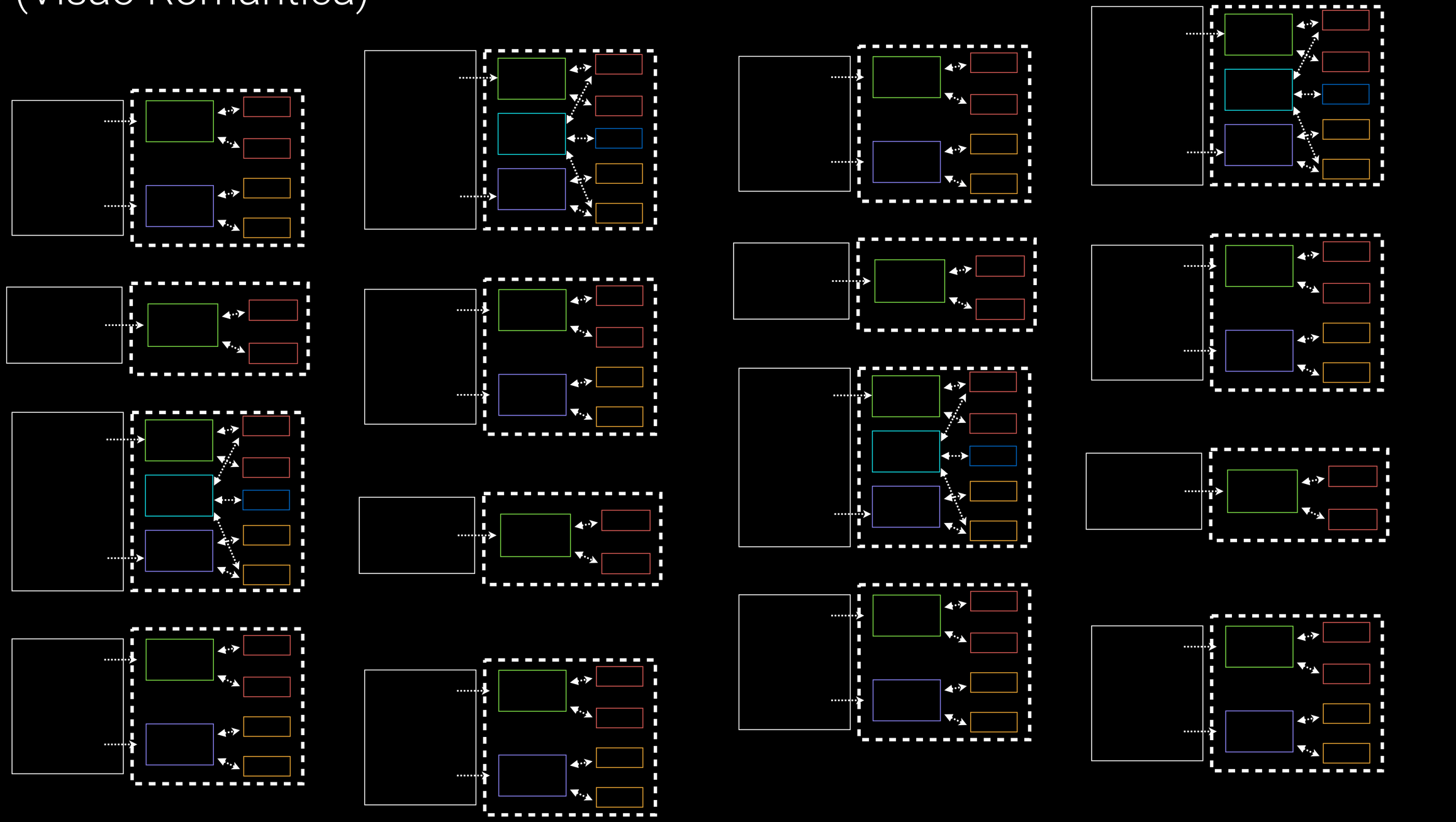
ACOPLAMENTO

UM SOFTWARE ENVIESADO PELOS DADOS ...

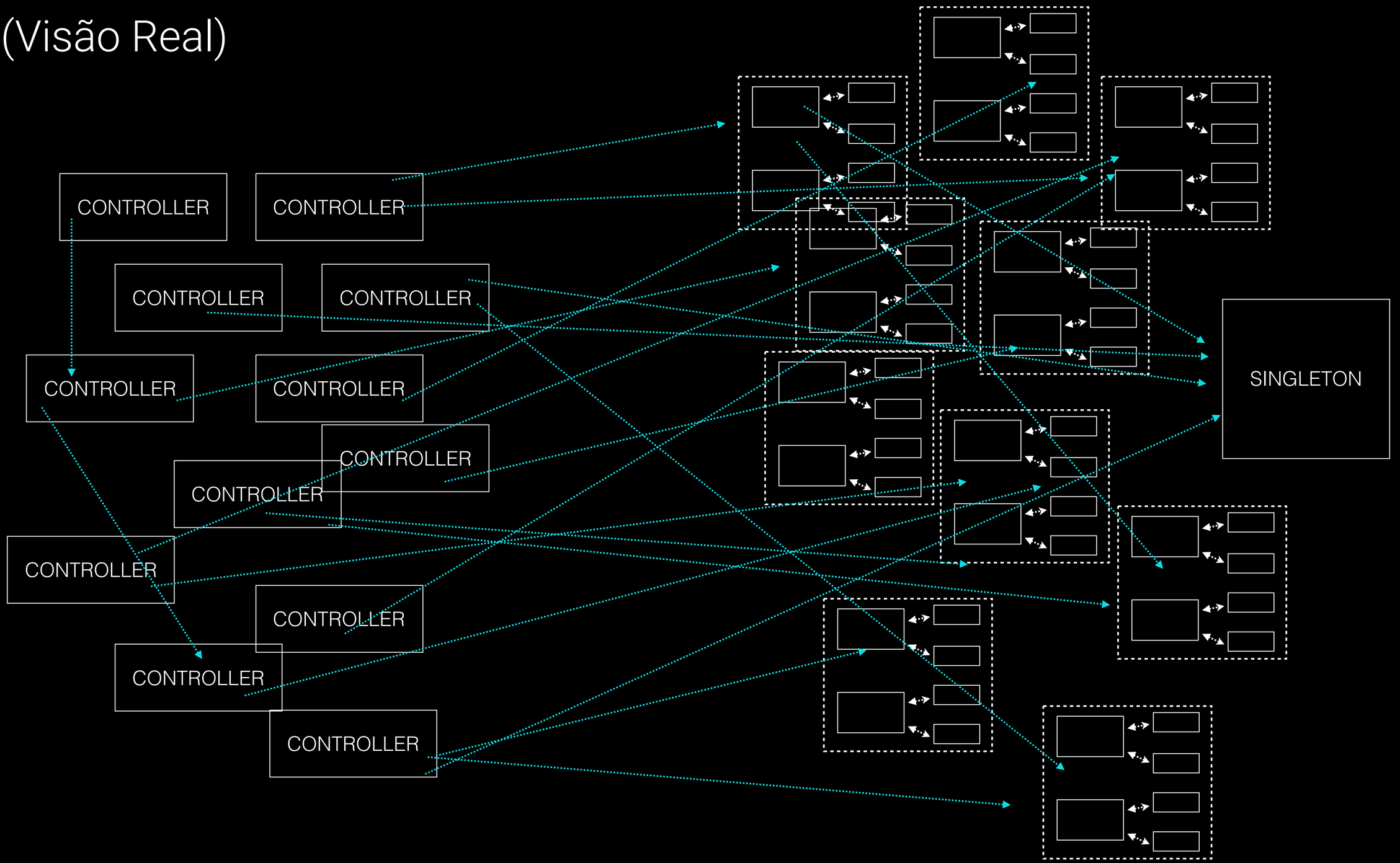

{
"name": "José da Silva",
"age": 33
…
}
String name;
int age;
…
USER


RestManager mngr = ...
DAOBridge bridge = ...
FRAMEWORK
VIEW/CONTROLLER
frameworkview.setData(model.attribute)
RestManager
api.getModel(callback)
DAOBridge
modelDAO.query(args)
MODEL
Model A
Model B
Model C
Model D

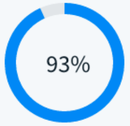
App (Visão Romântica)



App (Visão Real)

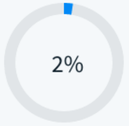


Device Statistics



Proximity On

App In Focus



Rooted

2.62 GB	273.00 MB
FREE SPACE	FREE RAM

Devices

samsung	54%
motorola	24%
LGE	11%
Other…	11%

Operating Systems

4	37%
6	34%
5	29%

Latest Session (an hour ago) **Download .txt** **View all sessions**

Fatal Exception: java.lang.IndexOutOfBoundsException
Inconsistency detected. Invalid item position 5(offset:5).state:12

</> Raw Text

android.support.v7.widget.RecyclerView$Recycler.getViewForPosition (RecyclerView.java:4659)
android.support.v7.widget.RecyclerView$Recycler.getViewForPosition (RecyclerView.java:4617)
android.support.v7.widget.LinearLayoutManager$LayoutState.next (LinearLayoutManager.java:1994)
android.support.v7.widget.LinearLayoutManager.layoutChunk (LinearLayoutManager.java:1390)
android.support.v7.widget.LinearLayoutManager.fill (LinearLayoutManager.java:1353)
android.support.v7.widget.LinearLayoutManager.scrollBy (LinearLayoutManager.java:1180)
android.support.v7.widget.LinearLayoutManager.scrollVerticallyBy (LinearLayoutManager.java:1031)
android.support.v7.widget.RecyclerView.scrollByInternal (RecyclerView.java:1529)
android.support.v7.widget.RecyclerView.onTouchEvent (RecyclerView.java:2486)
android.view.View.dispatchTouchEvent (View.java:10011)
android.view.ViewGroup.dispatchTransformedTouchEvent (ViewGroup.java:2833)
android.view.ViewGroup.dispatchTouchEvent (ViewGroup.java:2504)

WTF ?????


LIVE
NEWS
YOUR VOICE
YOUR VOTE
#GOPDebate
REPUBLICAN PRESIDENTIAL DEBATE
NEWS.com


AFV


ifbin.com

COMO REPRODUZIR ESSE ERRO?

COMO CORRIGIR ESSE ERRO?

COMO GARANTIR QUE NÃO OCORRERÁ NOVAMENTE?

COMO SABER SE A CORREÇÃO NÃO GEROU OUTROS ERROS?


SEUS
SONHOS

MALES DO ACOPLAMENTO

- Não torna a aplicação portável
- Vincula a aplicação a frameworks e ferramentas
- Torna difícil testar regras de negócio e comportamentos esperados
- Apodrece o código a médio prazo

COMO FAZER UM PROJETO DE SOFTWARE SUSTENTÁVEL ???

CONSTRUIR PARA PARA ESCALAR

O QUE É ARQUITETURA DE SOFTWARE ???

FLUX

REDUX

MVP

DDD

MVI

MVVM

VIPER

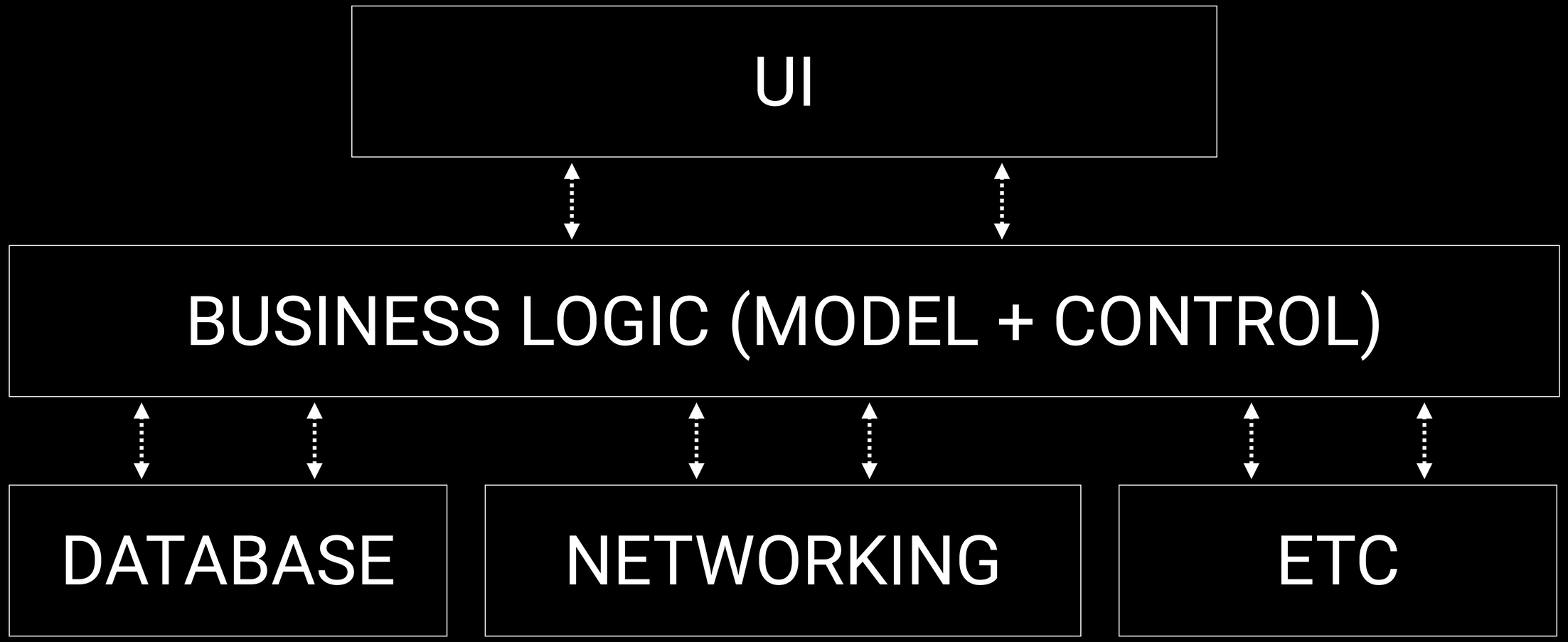
MVC

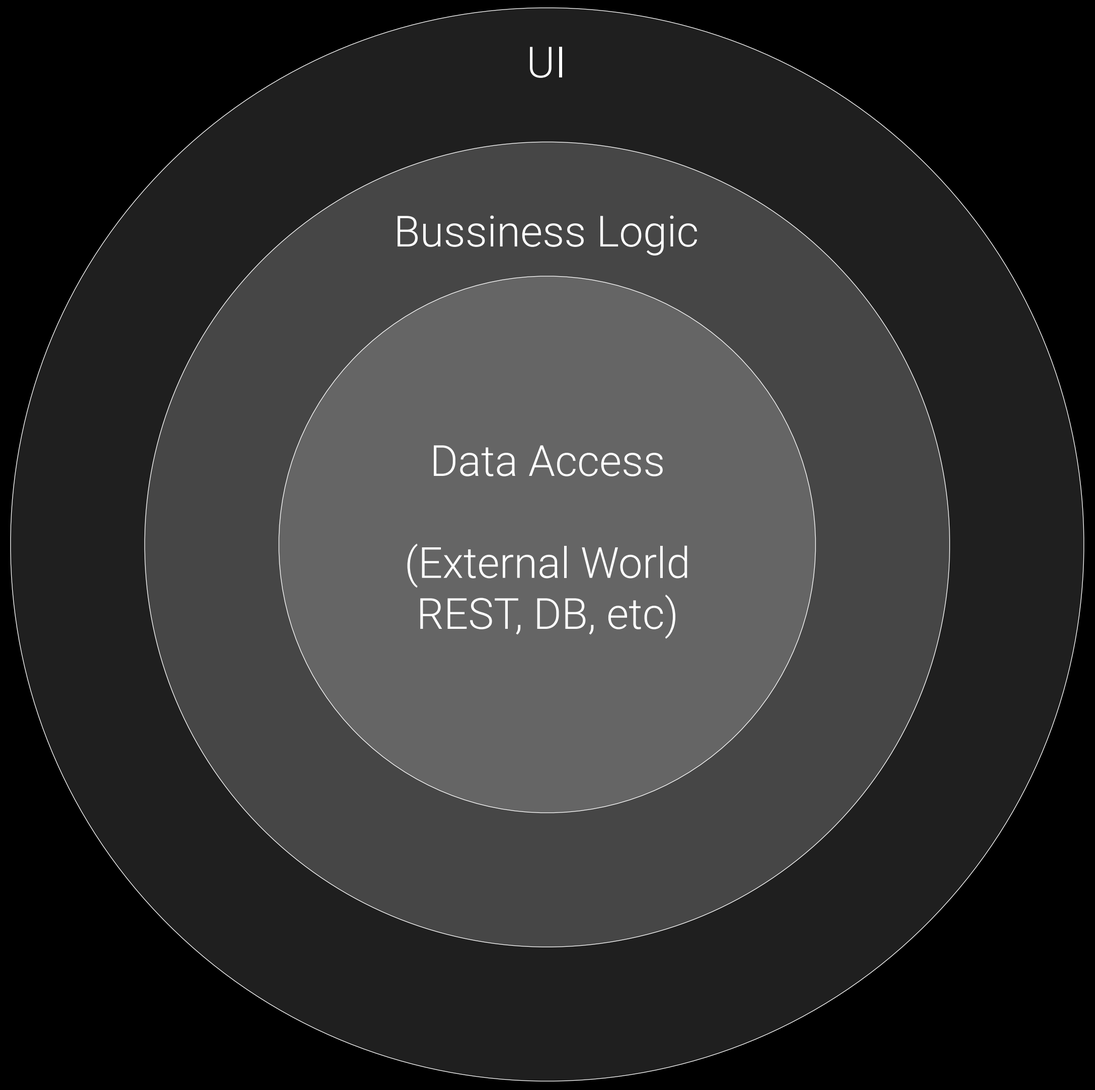
...

ARQUITETURA DE SOFTWARE

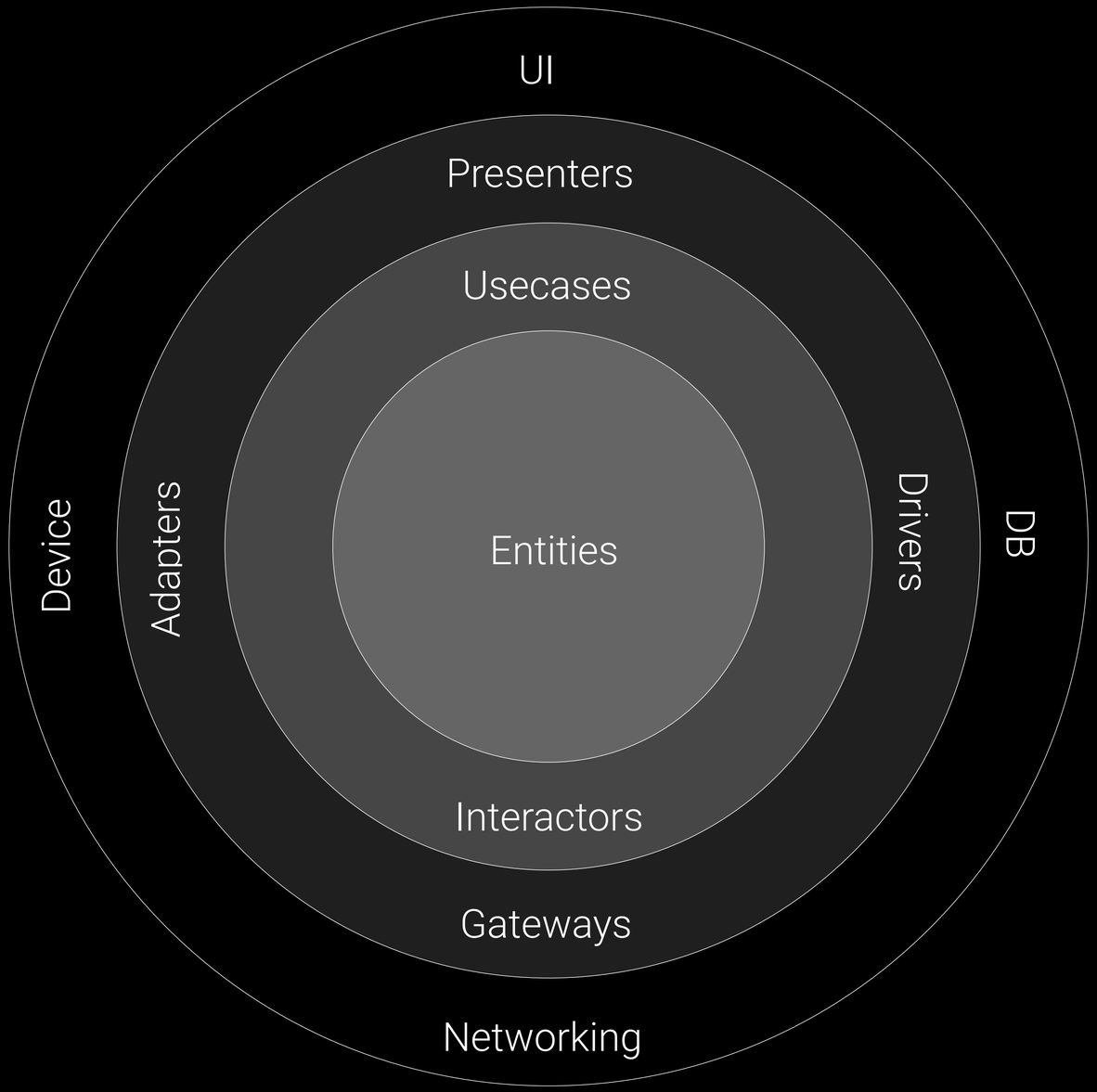
- Uma maneira de organizar as coisas
- Estilo de projeto, definido e defendido pelo time
- Estilos sobre as definições de componentes, camadas e relacionamentos
- Boa arquitetura : facilidade de manutenção e extensão, legibilidade, entendimento do que acontece, testabilidade, flexível
- Arquitetura ruim : difícil manutenção, rígida, frágil, difícil de se encontrar o que se precisa, sem testes, alta complexidade

DATA-CENTRIC ARCHITECTURE





DATA CENTRIC



DOMAIN CENTRIC (Clean)

https://vimeo.com/43612849

Clean Architecture

@unclebobmartin

8th Light

CLEAN CODERS

Code-casts for Software Professionals

5

NDC NORWEGIAN DEVELOPERS CONFERENCE

57:24

HD

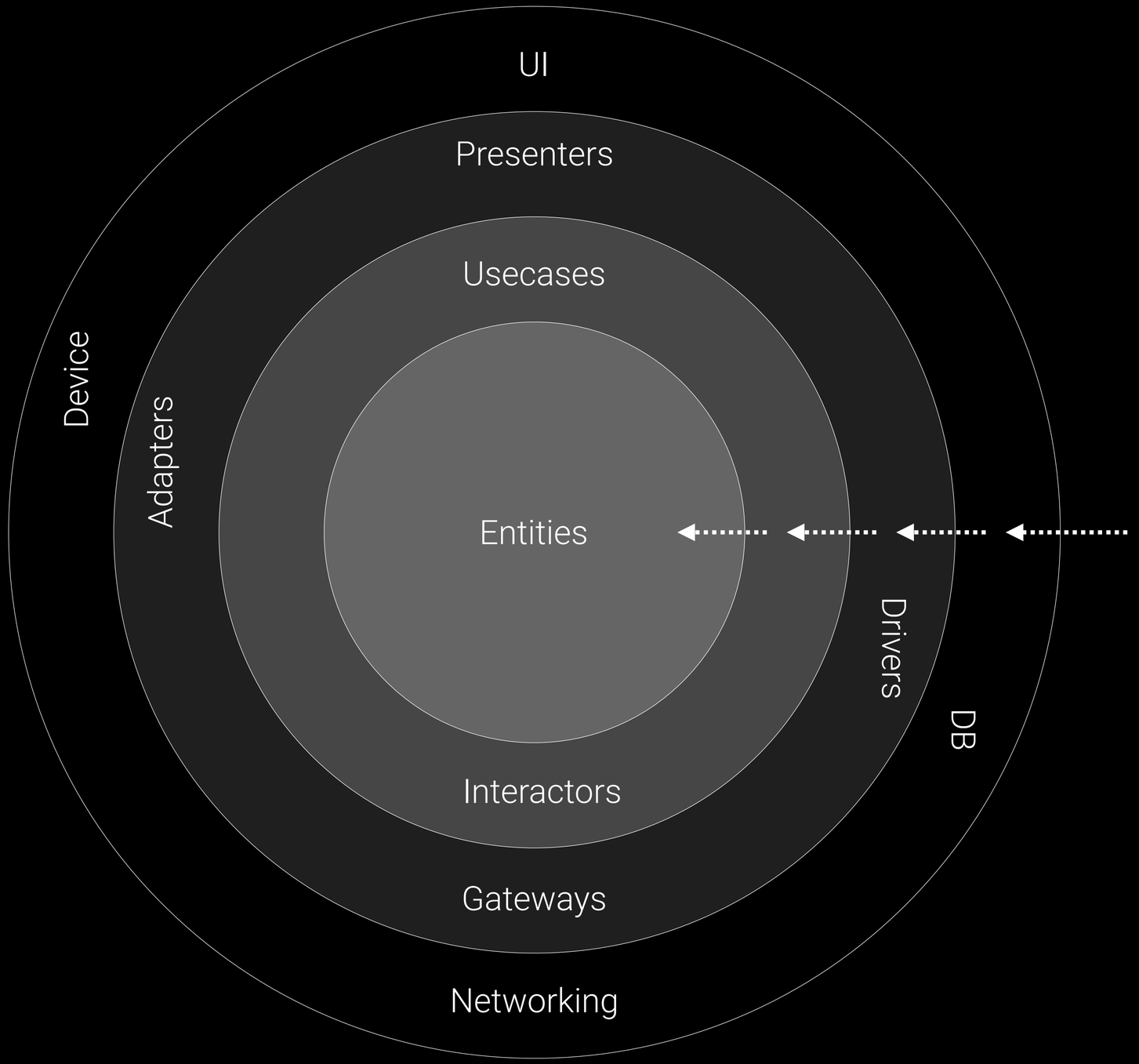
CLEAN ARCHICTECTURE

Motivações

- R.O.I em continuidade a longo prazo
- Favorece práticas como S.O.L.I.D e testabilidade
- Deixa explícitas as regras de negócio

Distinção entre essencial e detalhes

- *Database* é **detalhe**
- REST e networking são **detalhes**
- UI rendering e frameworks são **detalhes**
- Entidades (conceitos) são **essenciais**
- Casos de uso são **essenciais**
- Contratos de apresentação e de origem dos dados são **essenciais**



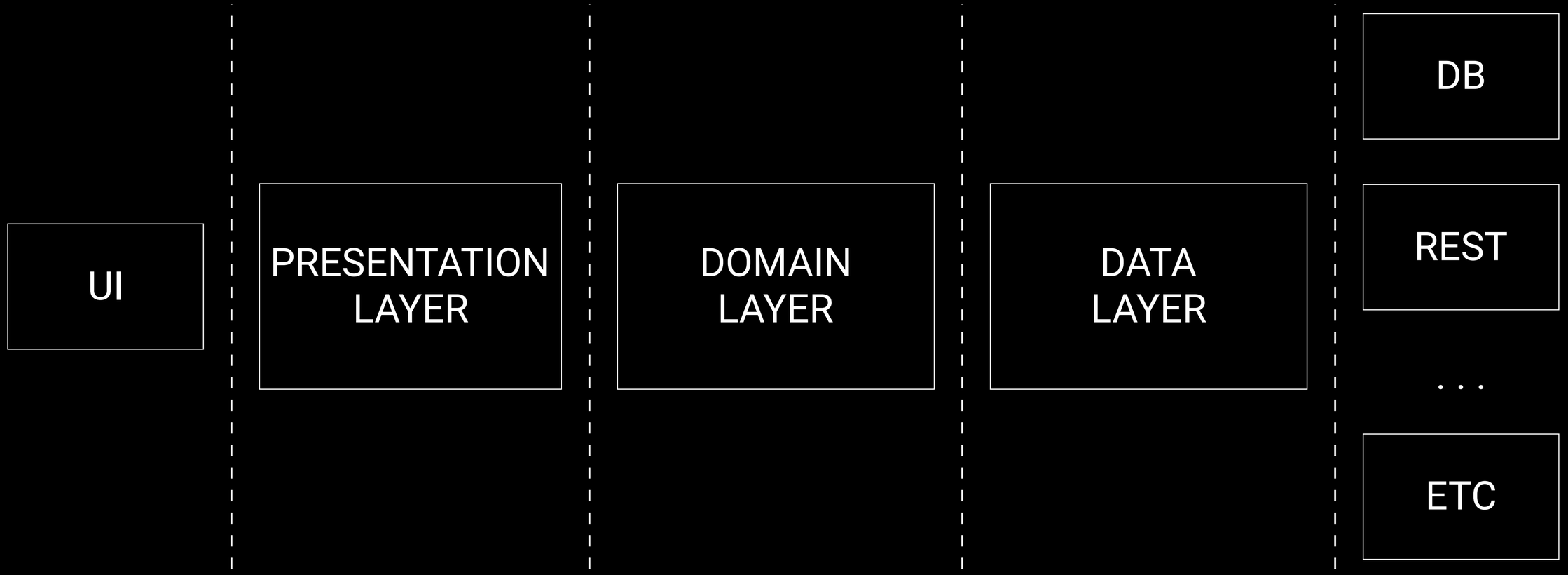
DEPENDENCY RULE

Camada mais externa em geral depende de um **contrato** com a camada mais interna

Camadas mais externas mais próximas às fronteiras da aplicação

Camadas ao centro contém as regras de negócio (o que a aplicação faz)

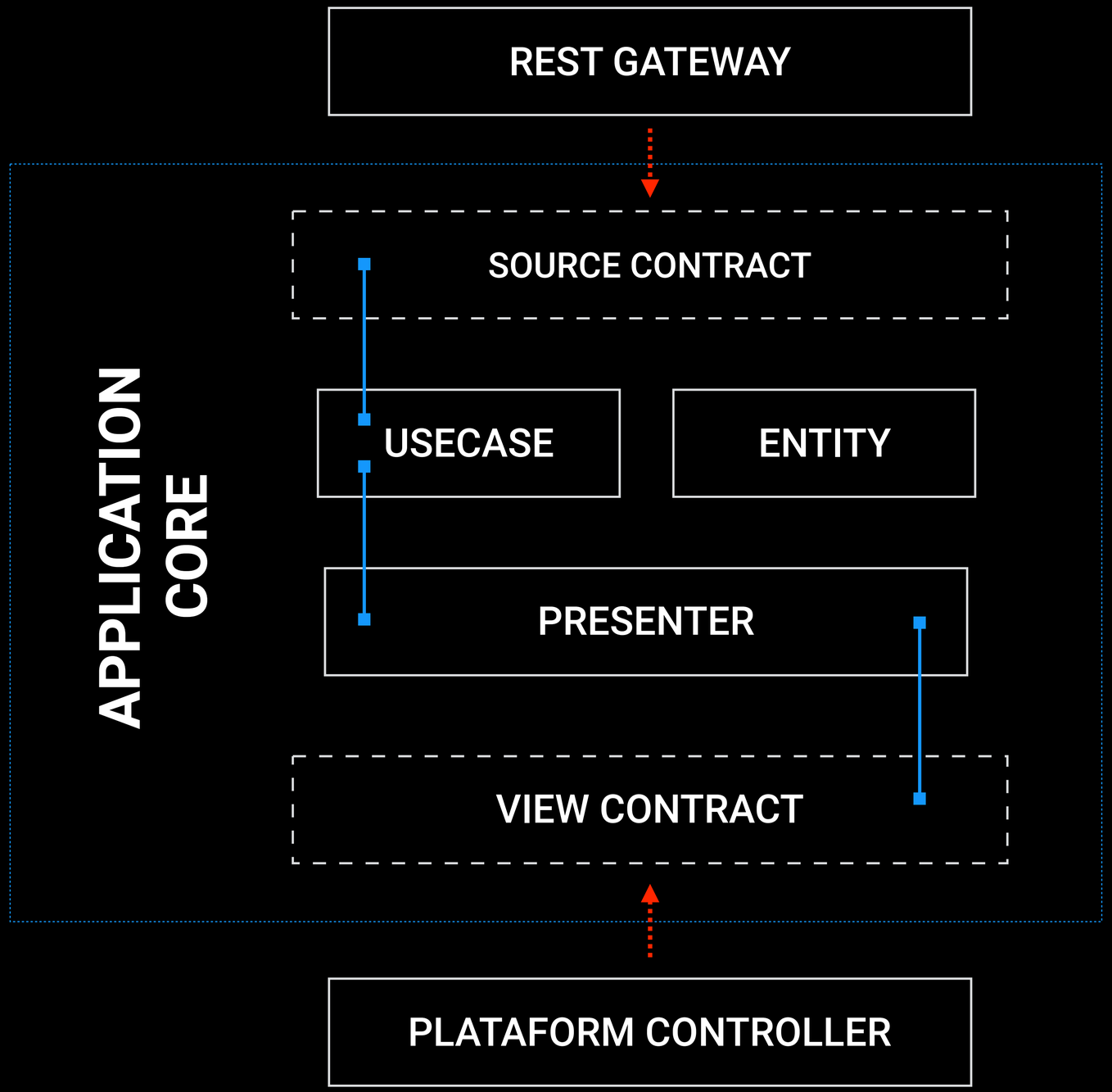
DOMAIN-CENTRIC ARCHITECTURE



TRÊS GRANDES PILARES

- O comportamento da sua aplicação deveria depender exclusivamente da linguagem, e não de frameworks e ferramentas
- Sua aplicação se relaciona com **comportamentos**, como por exemplo, interfaces para entregar e receber dados : frameworks implementam esses comportamentos
- O núcleo da aplicação deve ser 100% testável; isso inclui TODAS as regras de negócio

DEFINIR FRONTEIRAS



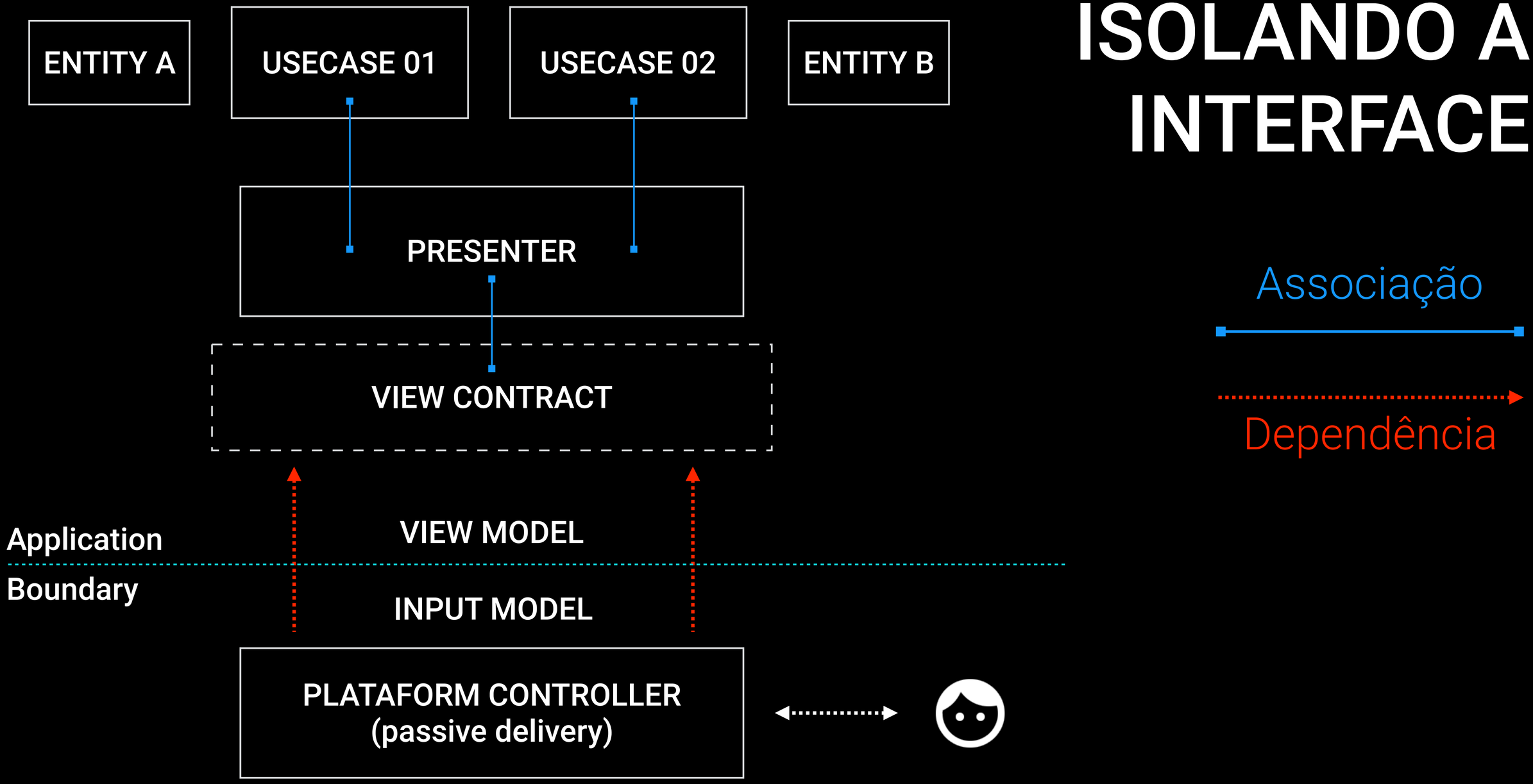
Associação

Dependência

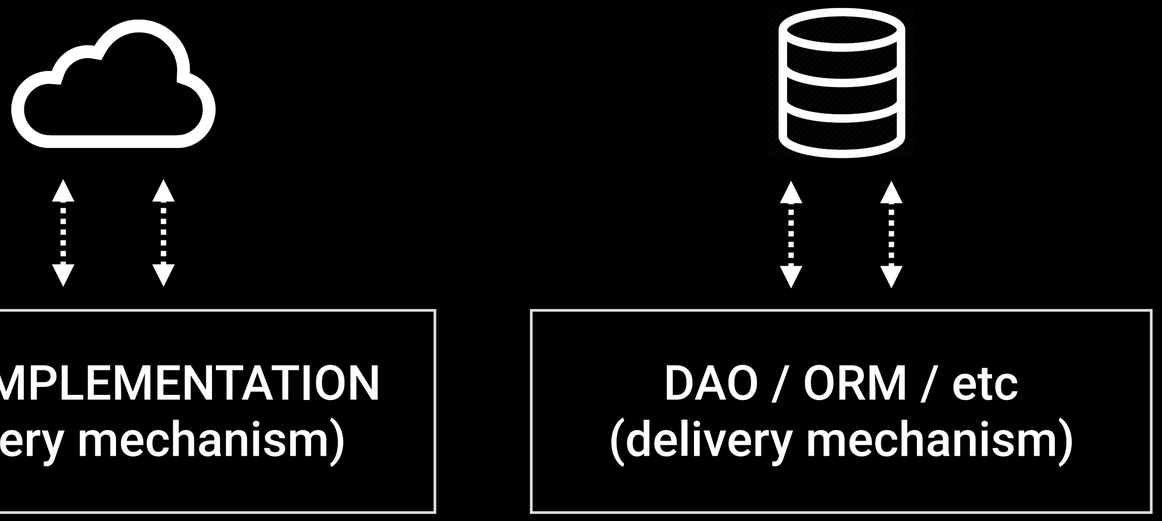
DOMÍNIO DA APLICAÇÃO

- Indica aquilo que a aplicação faz
- Casos de uso sobre entidades do domínio (modelos), que são as representações dos **conceitos para a aplicação** (não para a UI, nem para sistemas externos)
- Exemplos típicos :
  - **AddToBasket.with(Item)**
  - **RetrieveCustomerData.perform( )**
  - **PerformPurchase.with(Review)**
  - **etc**
- Casos de uso fazem muito sentido quando existem oportunidades explícitas de reúso

ISOLANDO A INTERFACE



ISOLANDO I/O



Associação

Dependência

SEPARE REPRESENTAÇÕES

Response Model

String description = “Blah”
String date = “2010-02-26T19:35:24Z”
int step = 2

Domain Model (Entity)

String description = “Blah”
LocalDateTime dateTime = (language representation)
TrackingStep currentStep = (enum)

ViewModel

String description = “Blah”
String formattedDate = “26/02/2010”
String currentStep = “Concluído”

CONECTANDO CAMADAS

Estratégia	Vantagens	Desvantagens
Síncrono	Método mais simples	Casos de uso não podem executar em paralelo; restrições de contexto
Callbacks	Geralmente suportados pela própria linguagem	Difícies de *debuggar* com concorrência e/ou múltiplas camadas; problemas com ciclo de vida de objetos
Barramento de eventos	Resolvem *Callback Hell*	Normalmente sem suporte a *threading*, mais difícil de *debuggar* e testar que *callbacks*
Reactive Programming	Assincronia e concorrência com API ao estilo síncrono; fáceis de testar	Difíceis de aprender, não necessariamente fáceis de *debuggar*

COMO TESTAR?

EXTERNAL WORLD ADAPTER

INTEGRATION TESTS
(DOUBLES)

SOURCE CONTRACT

UNIT TESTS
(Mocked Contract)

ENTITY

USECASE

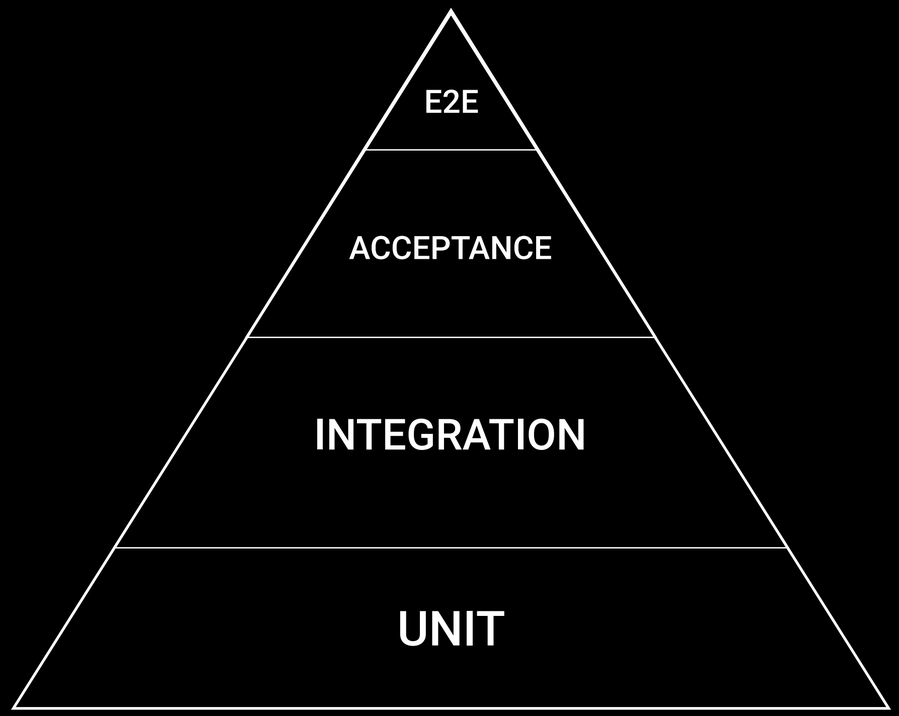
PRESENTER

UNIT TESTS
(Mocked Contract
+
Mocked Usecase)

VIEW CONTRACT

PLATAFORM CONTROLLER

FUNCTIONAL UI TESTS
INTEGRATION TESTS

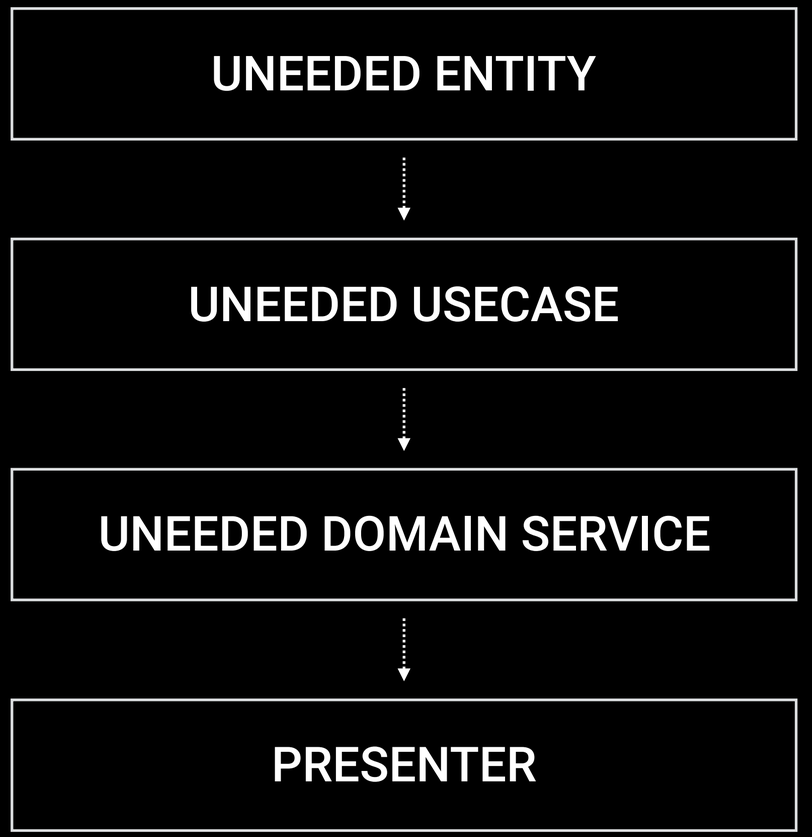


EVITE O SANDUÍCHE

- *Boillerplate* desnecessário (código e testes)
- Díficil de identificar no médio prazo
- Origens típicas :

  Adotar o mesmo modelo de *layers* em todo o lugar possível

  Tentar prever o futuro



ORGANIZE POR CONTEXTO

```
typical-packaging
    ├── activies
    ├── adapters
    ├── fragments
    ├── networking
    ├── ...
    ├── services
    ├── storage
    ├── util
    └── widgets
```

VS

```
clear-intentions-packaging
    ├── customer
    ├── checkout
    │   ├── basket
    │   ├── purchase
    │   └── review
    ├── orders
    ├── products
    ├── ...
    ├── push
    └── shared
```

ORGANIZAÇÃO FUNCIONAL

Benefícios

- Intenção explícita
- Localidade espacial
- Fácil Navegação
- Fácil identificar onde estão dependências de ferramentas

Potenciais problemas

- Pode quebrar convenções de frameworks
- Pode quebrar *scaffolding* / *file templates* / etc
- Testes deveriam seguir mesma organização, mas podem morar em diretórios diferentes (replicação manual)
- Normalmente o projeto nasce organizado por categorias : difícil migração

CONCLUINDO

DIFICULDADES TÍPICAS

- “É preciso escrever muito código para ter algo realmente funcionando”
- Flerte constante com *over-engineering*
- Decisões de quantas e quais camadas adotar e quais são as apropriadas dependem de contexto (time, projeto e negócio)
- Projetos em andamento (tipicamente) não nascem nessa estrutura e precisam ser evoluídos; identificar as fronteiras é fácil, identificar domínio e casos de uso pode ser mais difícil
- Não há ROI imediato, nem *Silver Bullets*

CONSIDERAÇÕES FINAIS

- *Clean Architecture* tem a ver com **idéias de organizar** as coisas e **não com fórmulas prontas**
- Defina sua arquitetura de maneira a **deixar claras as intenções** do seu código e da sua aplicação
- Sua aplicação é um grande roteador entre agentes de interesse, **separe o essencial** (o que ela é) **dos detalhes** (que podem ser substituídos)
- Comece pelo simples, extraindo comportamentos nas fronteiras do(s) framework(s) e fazendo inversão de controle o(s) mesmo(s)

UBIRATAN SOARES

Computer Scientist by ICMC/USP

Software Engineer, curious guy

Google Developer Expert for Android

Teacher, speaker, etc, etc

https://speakerdeck.com/ubiratansoares/escaping-from-the-framework

OBRIGADO

@ubiratanfsoares

ubiratansoares.github.io

https://br.linkedin.com/in/ubiratanfsoares