

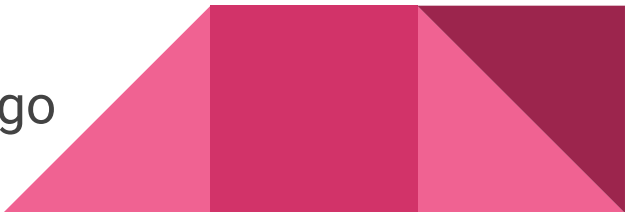
Offline-first em apps Xamarin

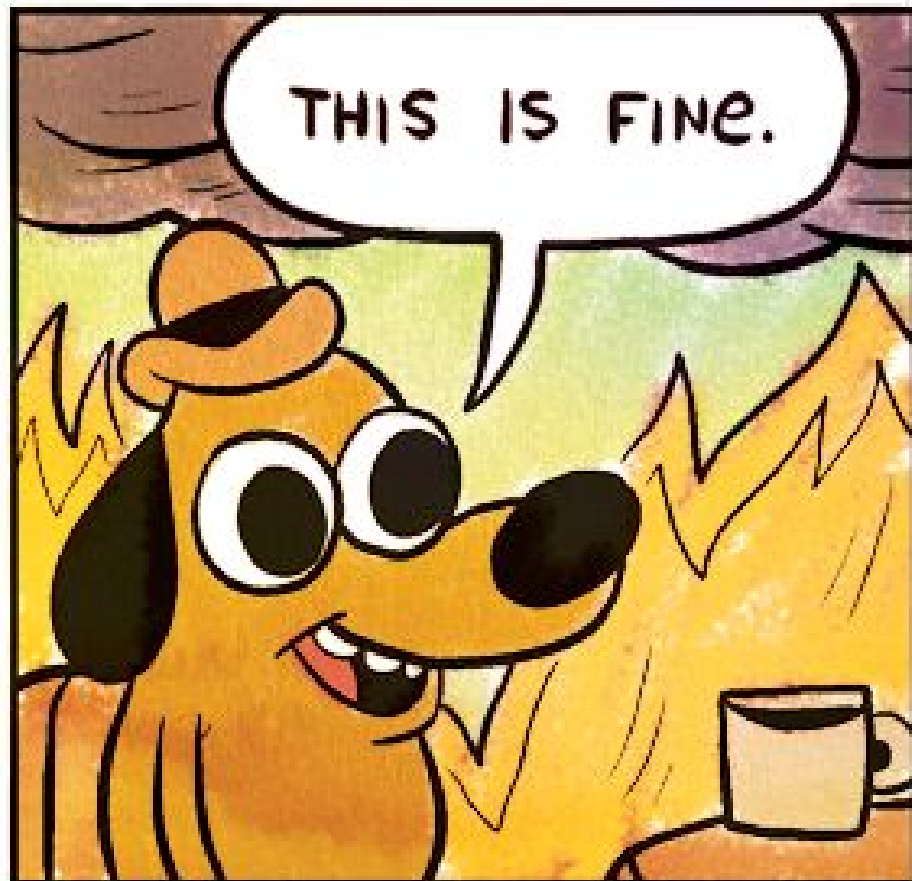
Agenda

- Acerca do aplicativo do case
- O que é modo offline?
- Decisões de arquitetura
- Detalhes de implementação
- Resumo
- Espaço para perguntas



O caos de quando eu entrei

- Time de suporte recebe reclamações constantes
 - Aplicativo atual contém inúmeros problemas
 - Problemas de sincronização com demais plataformas
 - Crashes acontecem sem motivo aparente
 - Modo offline não confiável
 - Base de código com manutenibilidade péssima
 - Não possui todas as funcionalidades que os clientes premium esperam
 - O time é totalmente novo e não conhece o código
- 

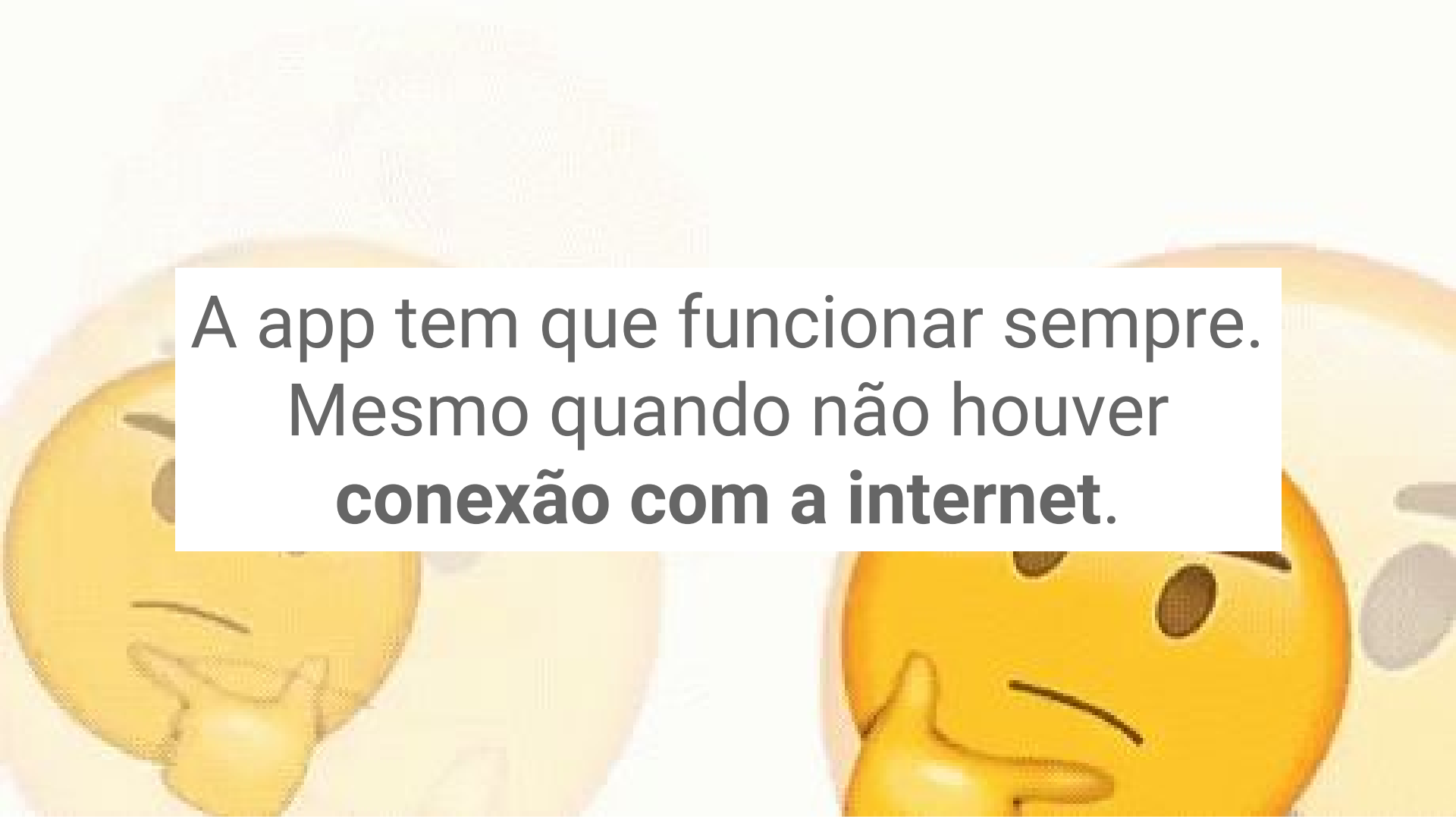


Vai ficar tudo bem, pode confiar

Solução

- O app **precisa** ser reescrito
- App inicial para iOS e Android
 - A ideia inicial é que o Core seja tão agnóstico de plataforma que apps macOS/UWP sejam possíveis
- Abordagem tradicional foi escolhida pois a UI terá uma tendência a ser complexa e o Xamarin.Forms não nos atenderia de maneira eficaz
- O requerimento principal é...



The background of the image features several large, semi-transparent yellow emoji faces. These faces have various expressions, including some with closed eyes and others with open eyes, and they are arranged in a way that they appear to be floating or overlapping. The overall color scheme is warm and bright, dominated by the yellow of the emojis and the white of the text box.

A app tem que funcionar sempre.
Mesmo quando não houver
conexão com a internet.



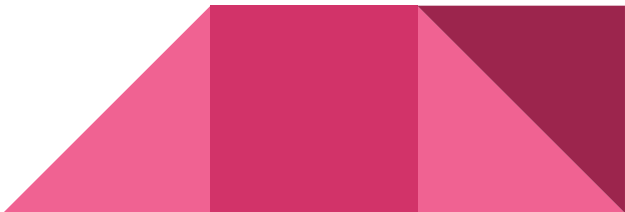
Offline first

O que é offline first?

- Conceito de mudar a maneira como o app lida com conectividade.
- O app assume que está **sempre offline e que ocasionalmente terá acesso a internet.**



Benefícios da abordagem

- Experiência fluída independente de onde o usuário esteja.
 - App se comportará como esperado em dispositivos com alta conectividade.
 - O usuário já espera esse tipo de comportamento, mas não sabe.
- 

Antes de começar, você precisa definir

- O que é estar offline?
- Como você irá se recuperar ao ficar online?
- Como serão resolvidos conflitos entre dados locais e dados oriundos de outros clients?





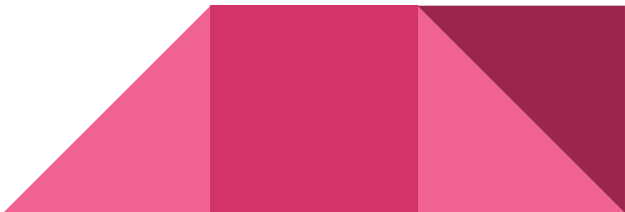
Arquitetura

Camadas

- Mapeamento da API
- Base de dados
- Camada que combina as fontes de dados
- ViewModels
- Camada de apresentação nativa



Como assim combinar as fontes de dados?

- ViewModels não sabem sobre o estado do app (online ou offline)
 - Elas não sabem da API
 - Não sabem da base de dados
 - Elas apenas consomem dados de uma fonte unificada que entende disso
- 

E como combinar os dados?

Usar callbacks



```
public interface ICallbackBasedDataSource
{
    void GetUser(Action<IUser> callback);
}
```


E como combinar os dados?

~~Usar callbacks~~

Usar tasks



```
public interface ITaskBasedDataSource
{
    Task<IUser> GetUser();
}
```

E como combinar os dados?

~~Usar callbacks~~

~~Usar tasks~~

Usar Observables



```
public interface IObservableBasedDataSource
{
    . . . IObservable<IUser> GetUser();
}
```

```
public class CommonSource : IObservableBasedDataSource
{
    private readonly IObservableBasedDataSource api;
    private readonly IObservableBasedDataSource database;

    public IObservable<IUser> GetUser()
    {
        var apiAndPersist = database.GetUser()
            .Do(persistToDatabase);

        return database.GetUser()
            .Concat(apiAndPersist);
    }

    private void persistToDatabase(IUser obj)
    {
        //TODO: Add persistency logic
    }
}
```



Sobre Observables

- Implementação

github.com/Reactive-Extensions/Rx.NET

- Representação

www.rxmarbles.com

- Docs:

reactivex.io



Implementação

Camada de rede

- Utilizamos HttpClient
- ~~Também utilizamos a lib ModernHttpClient, que usa libs nativas de cada plataforma para entregar ainda mais velocidade~~
- Não recomendamos `^(ツ)^/`



```
protected IObservable<T> CreateObservable<T>(Endpoint endpoint, IEnumerable<HttpHeader> headers, string body = "")
{
    var request = new Request(body, endpoint.Url, headers, endpoint.Method);
    return Observable.Create<T>(async observer =>
    {
        var response = await apiClient.Send(request).ConfigureAwait(false);
        if (response.IsSuccess)
        {
            var data = await Task.Run(() => serializer.Deserialize<T>(response.RawData)).ConfigureAwait(false);
            observer.OnNext(data);
            observer.OnCompleted();
        }
        else
        {
            observer.OnError(new ApiException(response.RawData));
        }
    });
}
```

Testes

- Testes de unidade garantem o funcionamento individual dos componentes
- Testes de integração executam contra a API de homologação para garantir que os todos os endpoints funcionam como esperado



=====

Tests.Unit

=====

Executing task: Tests.Unit

xUnit.net Console Runner (32-bit .NET 4.0.30319.42000)

Discovering: Toggl.PrimeRadiant.Tests

Discovering: Toggl.Multivac.Tests

Discovering: Toggl.Ultrawave.Tests

Discovering: Toggl.Foundation.Tests

Discovered: Toggl.PrimeRadiant.Tests

Discovered: Toggl.Foundation.Tests

Starting: Toggl.PrimeRadiant.Tests

Starting: Toggl.Foundation.Tests

Discovered: Toggl.Ultrawave.Tests

Starting: Toggl.Ultrawave.Tests

Discovered: Toggl.Multivac.Tests

Starting: Toggl.Multivac.Tests

Finished: Toggl.Multivac.Tests

Finished: Toggl.PrimeRadiant.Tests

Finished: Toggl.Foundation.Tests

Finished: Toggl.Ultrawave.Tests

=== TEST EXECUTION SUMMARY ===

| | | | | | |
|------------------------|------------|------------|------------|-------------|--------------|
| Toggl.Foundation.Tests | Total: 12, | Errors: 0, | Failed: 0, | Skipped: 0, | Time: 1.112s |
|------------------------|------------|------------|------------|-------------|--------------|

| | | | | | |
|----------------------|------------|------------|------------|-------------|--------------|
| Toggl.Multivac.Tests | Total: 43, | Errors: 0, | Failed: 0, | Skipped: 0, | Time: 0.759s |
|----------------------|------------|------------|------------|-------------|--------------|

| | | | | | |
|--------------------------|------------|------------|------------|-------------|--------------|
| Toggl.PrimeRadiant.Tests | Total: 13, | Errors: 0, | Failed: 0, | Skipped: 0, | Time: 0.900s |
|--------------------------|------------|------------|------------|-------------|--------------|

| | | | | | |
|-----------------------|------------|------------|------------|-------------|--------------|
| Toggl.Ultrawave.Tests | Total: 19, | Errors: 0, | Failed: 0, | Skipped: 0, | Time: 1.224s |
|-----------------------|------------|------------|------------|-------------|--------------|

--

-

-

-

GRAND TOTAL: 87

0

0

0

3.995s (2.421s)

Finished executing task: Tests.Unit

=====

Tests.Integration

=====

Executing task: Tests.Integration

xUnit.net Console Runner (32-bit .NET 4.0.30319.42000)

Discovering: Toggl.Ultrawave.Tests.Integration

Discovered: Toggl.Ultrawave.Tests.Integration

Starting: Toggl.Ultrawave.Tests.Integration

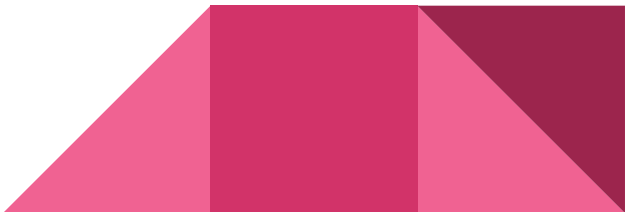
Finished: Toggl.Ultrawave.Tests.Integration

=== TEST EXECUTION SUMMARY ===

| | | | | | |
|-----------------------------------|------------|------------|------------|-------------|---------------|
| Toggl.Ultrawave.Tests.Integration | Total: 53, | Errors: 0, | Failed: 0, | Skipped: 0, | Time: 24.999s |
|-----------------------------------|------------|------------|------------|-------------|---------------|

Finished executing task: Tests.Integration


Camada de persistência

- Utilizamos Repository pattern
 - Modelos da base de dados possuem uma flag para indicar se eles estão em sync com o backend
 - Permite múltiplas implementações, sendo a atual escrita em Realm
- 

A close-up, profile shot of a young boy with short, wavy brown hair. He is looking upwards and to the left with a questioning or curious expression on his face. The background is dark and out of focus, suggesting an indoor setting with some structural elements. The lighting is soft, highlighting his features.

REALM ?

Por que Realm e não SQLite?

- Pensado para funcionar em plataformas mobile antes de mais nada
 - Suporte oficial para .net
 - Open Source, com wrappers em C#
 - API simples de se utilizar e que requer pouca configuração
- 

```
protected static IObservable<IUser> CreateObservable()
{
    return Observable.Create<IUser>(observer =>
    {
        try
        {
            var data = Realms.Realm.GetInstance().All<RealmUser>().Last();
            observer.OnNext(data);
            observer.OnCompleted();
        }
        catch (InvalidOperationException)
        {
            observer.OnError(new EntityNotFoundException());
        }
        catch (Exception ex)
        {
            observer.OnError(ex);
        }

        return Disposable.Empty;
    });
}
```


Camada combinada

- Recebe como dependencias a API e a base de dados
- Combina os dados conforme necessário
- Como todas as fontes são meramente observables, combina-las é simples!



EXPECTATIVA




Realidade

```
· public IObservable<IUser> Get()  
·     ⇒ storage.Single()  
·     .Concat<IUser>(  
·         userClient.Get()  
·         .Do(persist)  
·     );
```

Resumo

Pontos chave

- Tenha testes automatizados!
 - Faça com que as suas ViewModels, sempre que possível, tenham algo para exibir para o usuário
 - Trabalhe sempre offline e, quando der, fique online e sincronize
- 

Pontos chave

- Separe as suas dependências de forma que elas sejam substituídas facilmente
- Pense em todo o flow do seu negócio antes de começar a modelar para evitar surpresas



Links

Repositório

<https://github.com/toggl/mobileapp>

Estamos contratando

<https://jobs.toggl.com/mobile/>

Slides

<https://github.com/willsb/Talks>

Contribua!

<http://up-for-grabs.net/>



Perguntas?

Links

Repositório

<https://github.com/toggl/mobileapp>

Estamos contratando

<https://jobs.toggl.com/mobile/>

Slides

<https://github.com/willsb/Talks>



Thank you.