<https://www.youtube.com/watch?v=uRwQKikomtE>

Receber dados da Internet

## [1. Antes de começar](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet#0)

A maioria dos apps Android no mercado se conecta à Internet para executar algumas operações de rede. Por exemplo, acessar e-mails, mensagens ou informações semelhantes de um servidor de back-end. O Gmail, o YouTube e o Google Fotos são alguns exemplos de apps que se conectam à Internet para exibir os dados do usuário.

Aqui você usará bibliotecas de código aberto desenvolvidas para criar a camada de rede e receber dados de um servidor de back-end. Isso simplifica muito a busca dos dados e também ajuda o app a cumprir as práticas recomendadas do Android, como a execução de operações em uma linha de execução de segundo plano. Você também atualizará a interface do usuário do app se a Internet estiver lenta ou indisponível. Isso manterá o usuário informado sobre problemas de conectividade de rede.

## ****O que você já precisa saber****

* Como criar e usar fragmentos.
* Como usar os Componentes da arquitetura do Android ViewModel e LiveData.
* Como adicionar dependências em um arquivo do Gradle.

## ****O que você aprenderá****

* O que é um serviço da Web [REST](https://en.wikipedia.org/wiki/Representational_state_transfer) (link em inglês).
* Como usar a biblioteca [Retrofit](https://square.github.io/retrofit/" \t "_blank) (link em inglês) para se conectar a um serviço da Web REST na Internet e receber uma resposta.
* Como usar a biblioteca [Moshi](https://github.com/square/moshi" \t "_blank) (link em inglês) para analisar a resposta JSON em um objeto de dados.

## ****O que você aprenderá****

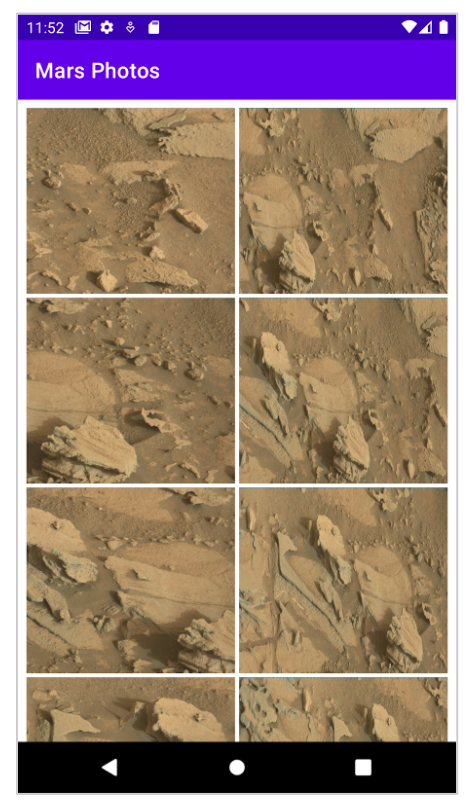
* Como modificar um app inicial para fazer uma solicitação de API de serviço da Web e lidar com a resposta.
* Como implementar uma camada de rede para o app usando a biblioteca Retrofit.
* Como analisar a resposta JSON do serviço da Web nos objetos LiveData do app com a biblioteca Moshi.
* Como usar a compatibilidade da biblioteca Retrofit para corrotinas para simplificar o código.

## O que é necessário

* Um computador com o Android Studio instalado.
* O código inicial do app MarsPhotos.

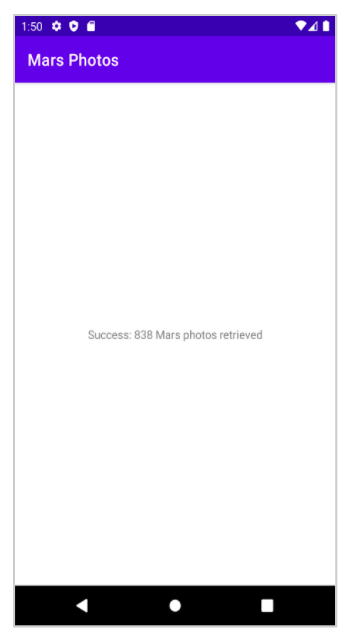
## [2. Visão geral do app](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet#1)

Nesse projeto, você trabalha com o app inicial **MarsPhotos**, que mostra imagens da superfície de Marte. Este app se conecta a um serviço da Web para recuperar e exibir as fotos de Marte. As imagens são fotos reais de Marte capturadas de rovers da NASA em Marte. Veja a seguir a captura de tela do app final, que contém uma grade de imagens de propriedades em miniatura, criadas usando um RecyclerView.



**Observação**: a captura de tela acima é a captura de tela do app final que você criará ao terminar o tutorial que está no final do próximo codelab. A captura de tela é mostrada neste codelab para dar uma ideia melhor do funcionamento geral do app.

A versão do app que você compilar neste codelab não terá muita beleza visual: ele se concentra na parte da camada de rede do app para se conectar à Internet e fazer o download dos dados brutos de propriedade usando um serviço da Web. Para garantir que os dados sejam recuperados e analisados corretamente, basta exibir o número de fotos recebidas do servidor de back-end em uma visualização de texto:



## [3. Conheça o app inicial MarsPhotos](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet#2)

### **Fazer o download do código inicial**

Este codelab oferece um código inicial para você se aprofundar nos recursos ensinados. O código inicial pode conter tanto um código familiar quanto desconhecido de codelabs anteriores. Você aprenderá mais sobre códigos desconhecidos nos próximos codelabs.

Se você estiver usando o código inicial do GitHub, o nome da pasta é android-basics-kotlin-mars-photos-app. Selecione essa pasta ao abrir o projeto no Android Studio.

**URL do código inicial**:

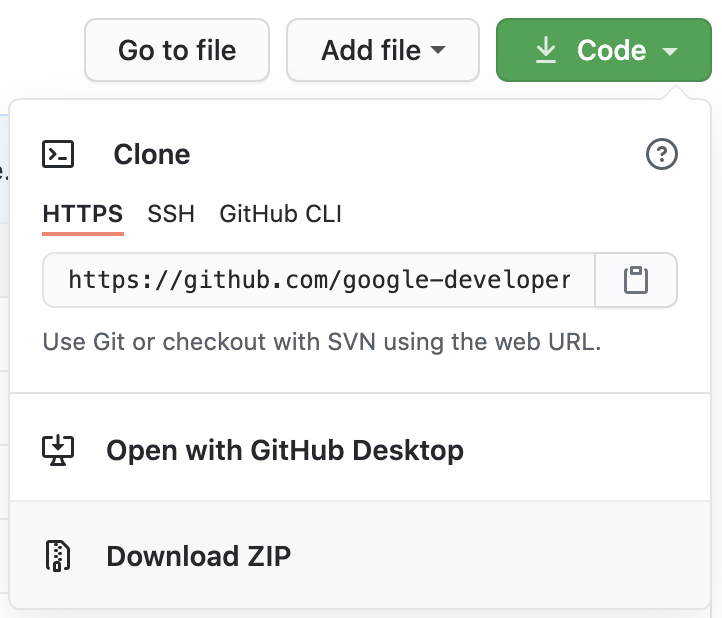
<https://github.com/google-developer-training/android-basics-kotlin-mars-photos-app/tree/starter> (link em inglês)

**Nome da ramificação**: starter

Para encontrar o código deste codelab e abri-lo no Android Studio, faça o seguinte.

## Buscar o código

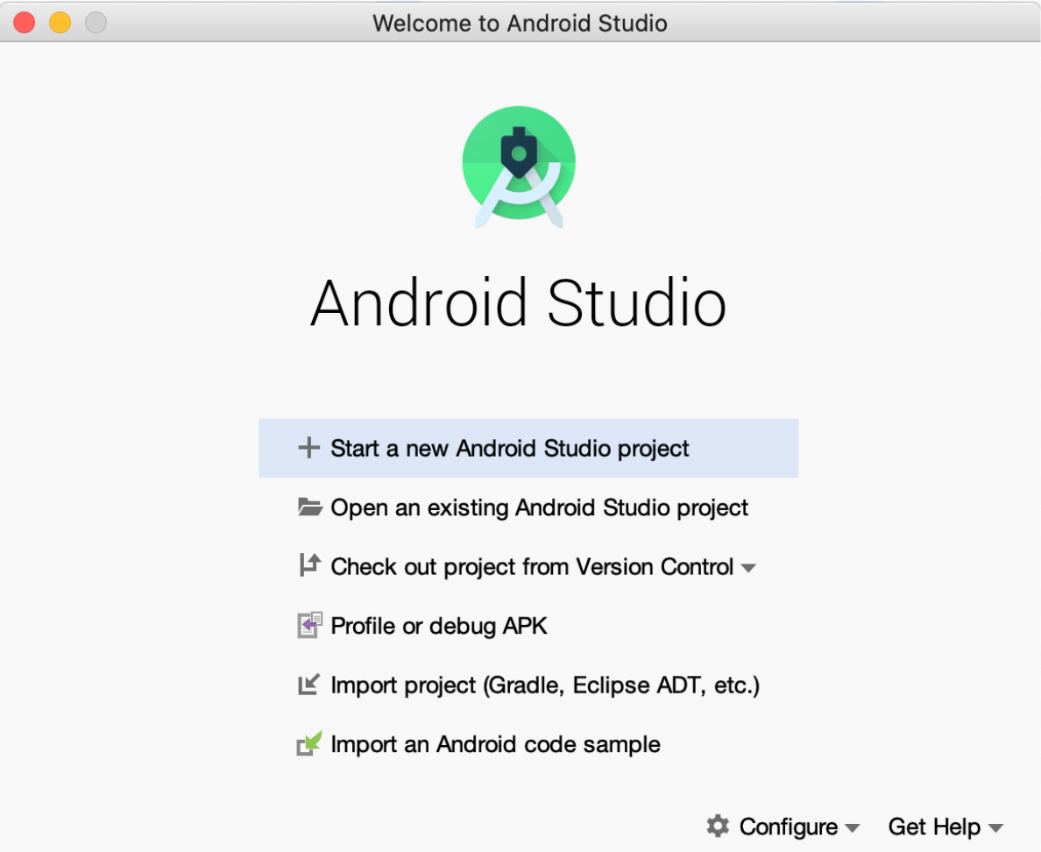
1. Clique no URL fornecido. Isso abrirá a página do GitHub referente ao projeto em um navegador.
2. Na página do GitHub do projeto, clique no botão **Code**, que exibirá uma caixa de diálogo.



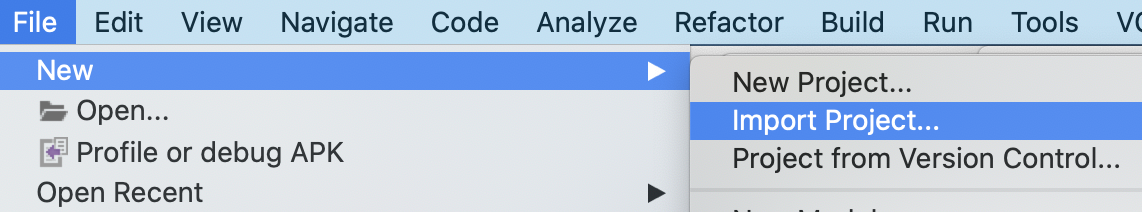
1. Na caixa de diálogo, clique no botão **Download ZIP** para salvar o projeto no seu computador. Aguarde a conclusão do download.
2. Localize o arquivo no computador, que provavelmente está na pasta **Downloads**.
3. Clique duas vezes no arquivo ZIP para descompactá-lo. Isso criará uma nova pasta com os arquivos do projeto.

## Abrir o projeto no Android Studio

1. Inicie o Android Studio.
2. Na janela **Welcome to Android Studio**, clique em **Open an existing Android Studio project**.



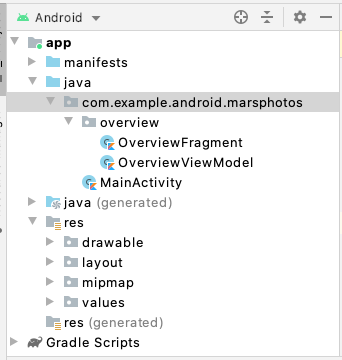
Observação: caso o Android Studio já esteja aberto, selecione a opção **File** > **New** > **Import Project**.



1. Na caixa de diálogo **Import Project**, vá até a pasta do projeto descompactada, que provavelmente está na pasta **Downloads**.
2. Clique duas vezes nessa pasta do projeto.
3. Aguarde o Android Studio abrir o projeto.
4. Clique no botão **Run** j7ptomO2PEQNe8jFt4nKCOw_Oc_Aucgf4l_La8fGLCMLy0t9RN9SkmBFGOFjkEzlX4ce2w2NWq4J30sDaxEe4MaSNuJPpMgHxnsRYoBtIV3-GUpYYcIvRJ2HrqR27XGuTS4F7lKCzg para criar e executar o app. Confira se ele funciona da forma esperada.
5. Procure os arquivos do projeto na janela de ferramentas **Project** para ver como o app foi implementado.

## Executar o código inicial

1. No Android Studio, abra o projeto que transferiu por download. O nome da pasta do projeto é android-basics-kotlin-mars-photos-app. A estrutura da pasta do código inicial é semelhante a esta.
2. No painel **Android**, clique para expandir **app** > **java**. O app tem uma pasta de pacote com o nome overview. Essa é a camada da IU do app.



1. Execute o app. Ao compilar e executá-lo, você verá a seguinte tela com um texto de marcador no centro. Ao final deste codelab, você atualizará o texto do marcador com o número de fotos extraídas.



1. Veja os arquivos para entender o código inicial. Para arquivos de layout, você pode usar a opção **Split** no canto superior direito para visualizar o layout e o XML ao mesmo tempo.

## ****Tutorial do código inicial****

Nesta tarefa, você se familiarizará com a estrutura do projeto. Veja a seguir uma explicação sobre arquivos e pastas importantes no projeto.

OverviewFragment:

* Este é o fragmento exibido na MainActivity. O texto de marcador que você viu na etapa anterior é exibido nesse fragmento.
* No próximo codelab, esse fragmento exibirá os dados recebidos do servidor de back-end de fotos de Marte.
* Essa classe armazena uma referência ao objeto OverviewViewModel.
* O OverviewFragment tem uma função onCreateView() que infla o layout do fragment\_overview usando a vinculação de dados, configura o proprietário do ciclo de vida da vinculação para ser ele mesmo e define a variável viewModel no objeto de vinculação para ele.
* Como o proprietário do ciclo de vida foi atribuído, qualquer objeto LiveData usado na vinculação de dados será observado automaticamente para quaisquer mudanças, e a IU será atualizada corretamente.

OverviewViewModel:

* Este é o modelo de visualização correspondente para o OverviewFragment.
* Essa classe contém uma propriedade MutableLiveData com o nome \_status e a propriedade de apoio dela. Atualizar o valor dessa propriedade atualiza o texto do marcador exibido na tela.
* O método getMarsPhotos() atualiza a resposta do marcador. Mais adiante no codelab, você usará esse método para exibir os dados buscados no servidor. O objetivo deste codelab é atualizar o status do LiveData no ViewModel usando dados reais da Internet.

res/layout/fragment\_overview.xml:

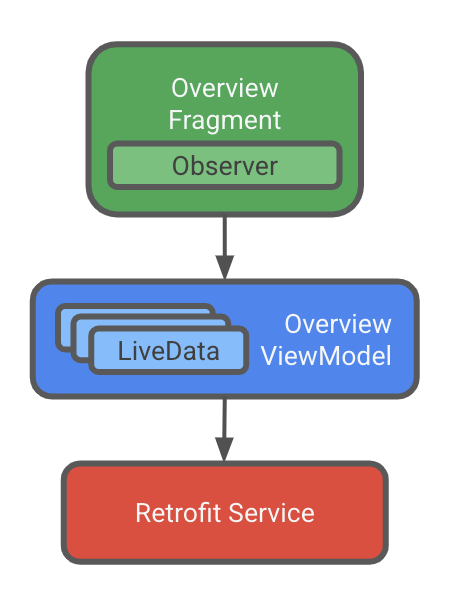
* Este layout está configurado para usar a vinculação de dados e consiste em uma única TextView.
* Ele declara uma variável OverviewViewModel e vincula o status do ViewModel à TextView.

MainActivity.kt: a única tarefa dessa atividade é carregar o layout da atividade, *activity\_main*.

*layout/activity\_main.xml:* é o layout da atividade principal com uma única FragmentContainerView apontando para o fragment\_overview. O fragmento de visão geral será instanciado quando o app for iniciado.

## [4. Visão geral do app](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet#3)

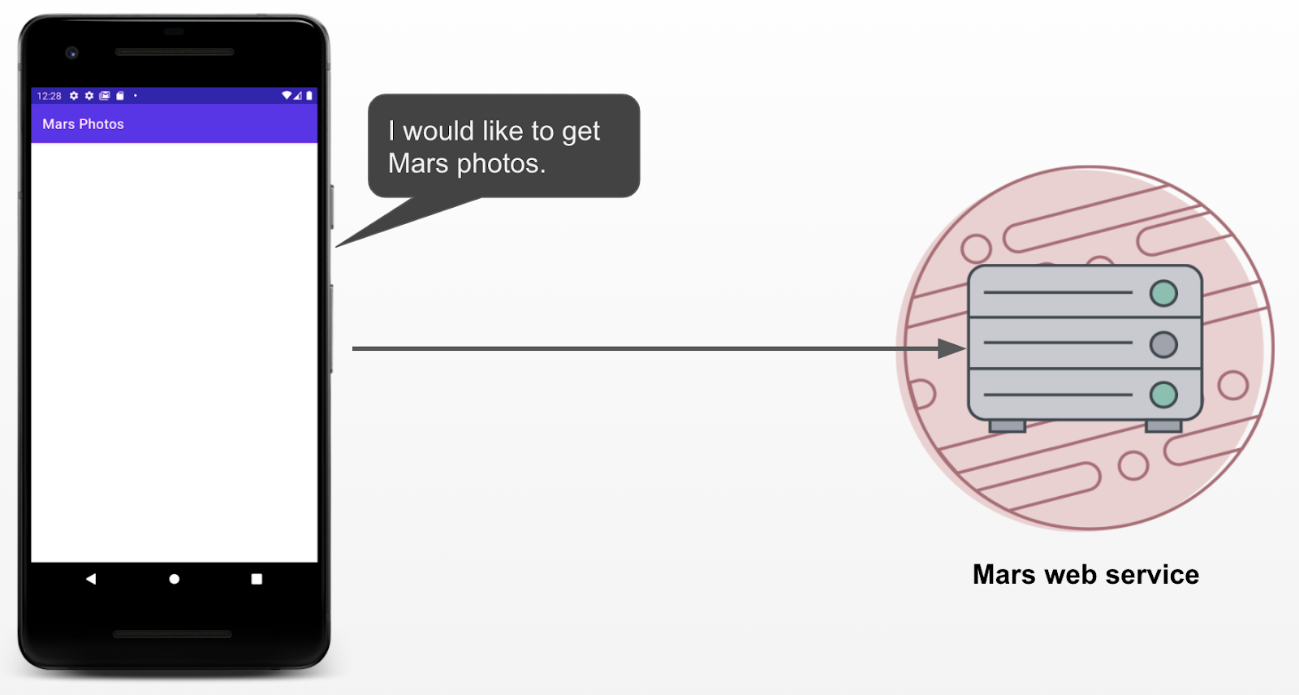
Neste codelab, você criará uma camada para o serviço de rede que se comunica com o servidor de back-end e busca os dados necessários. Você usará uma biblioteca de terceiro para implementar isso, a Retrofit. Você saberá mais sobre isso posteriormente. O ViewModel se comunica diretamente com essa camada de rede. O restante do app é transparente para essa implementação.

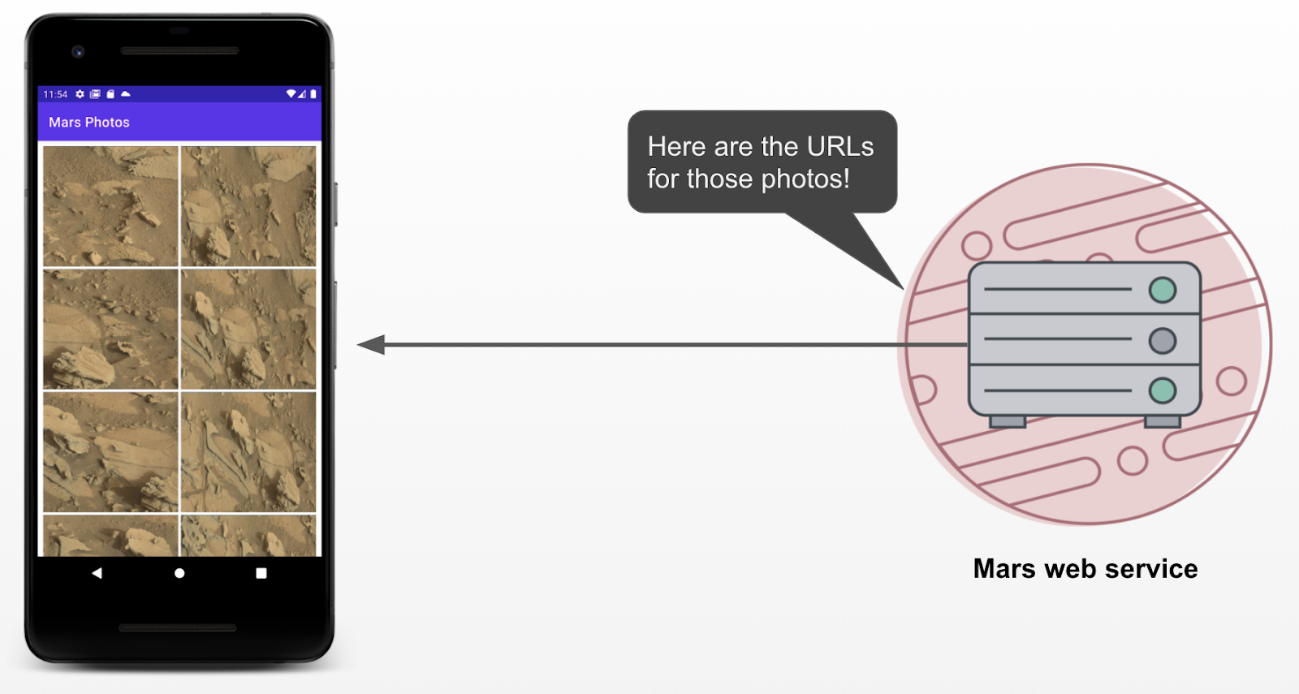


O OverviewViewModel é responsável por fazer a chamada de rede para acessar os dados de fotos de Marte. No ViewModel, você usa o LiveData com vinculação de dados compatível com o ciclo de vida para atualizar a IU do app quando os dados mudarem.

## [5. Serviços da Web e Retrofit](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet#4)

Os dados de fotos de Marte são armazenados em um servidor da Web. Para acessar esses dados no seu app, é necessário estabelecer uma conexão e se comunicar com o servidor na Internet.





Atualmente, a maioria dos servidores da Web executam serviços Web usando uma arquitetura da Web sem estado comum conhecida como [REST](https://en.wikipedia.org/wiki/Representational_state_transfer) (link em inglês), que significa**RE**presentional **S**tate **T**ransfer (Transferência de Estado Representacional, em inglês). Os serviços da Web que oferecem essa arquitetura são conhecidos como serviços RESTful.

As solicitações são feitas aos serviços RESTful da Web de maneira padronizada, usando [URI](https://en.wikipedia.org/wiki/Uniform_Resource_Identifier" \t "_blank)s (link em inglês). Um URI (Uniform Resource Identifier) identifica um recurso no servidor pelo nome, sem implicar a localização ou como acessá-lo. Por exemplo, no app desta lição, você recupera os URLs de imagem usando este URI de servidor (o servidor hospeda fotos e propriedades imobiliárias de Marte):

[android-kotlin-fun-mars-server.appspot.com](https://android-kotlin-fun-mars-server.appspot.com/?hl=pt-br) (link em inglês)

Um URL (Uniform Resource Locator) é um URI que especifica o meio de ação ou a representação do recurso. Ou seja, especifica o mecanismo de acesso primário e o local da rede.

**Exemplo**:

O URL a seguir acessa uma lista de todas as propriedades imobiliárias disponíveis em Marte.

[https://android-kotlin-fun-mars-server.appspot.com/realestate](https://android-kotlin-fun-mars-server.appspot.com/realestate?hl=pt-br)

O URL a seguir acessa uma lista de fotos de Marte:

[https://android-kotlin-fun-mars-server.appspot.com/photos](https://android-kotlin-fun-mars-server.appspot.com/photos?hl=pt-br)

Esses URLs se referem a um recurso identificado como [/realestate](https://android-kotlin-fun-mars-server.appspot.com/realestate?hl=pt-br) ou [/photos](https://android-kotlin-fun-mars-server.appspot.com/photos?hl=pt-br), que pode ser acessado pelo protocolo de transferência de hipertexto (http:) da rede (links em inglês). Você usará o endpoint [/photos](https://android-kotlin-fun-mars-server.appspot.com/photos?hl=pt-br) (link em inglês) neste codelab.

**Observação**: o URL da Web conhecido é, na verdade, um tipo de URI. Os URLs e os URIs são usados alternadamente neste curso, dependendo da API que estamos chamando.

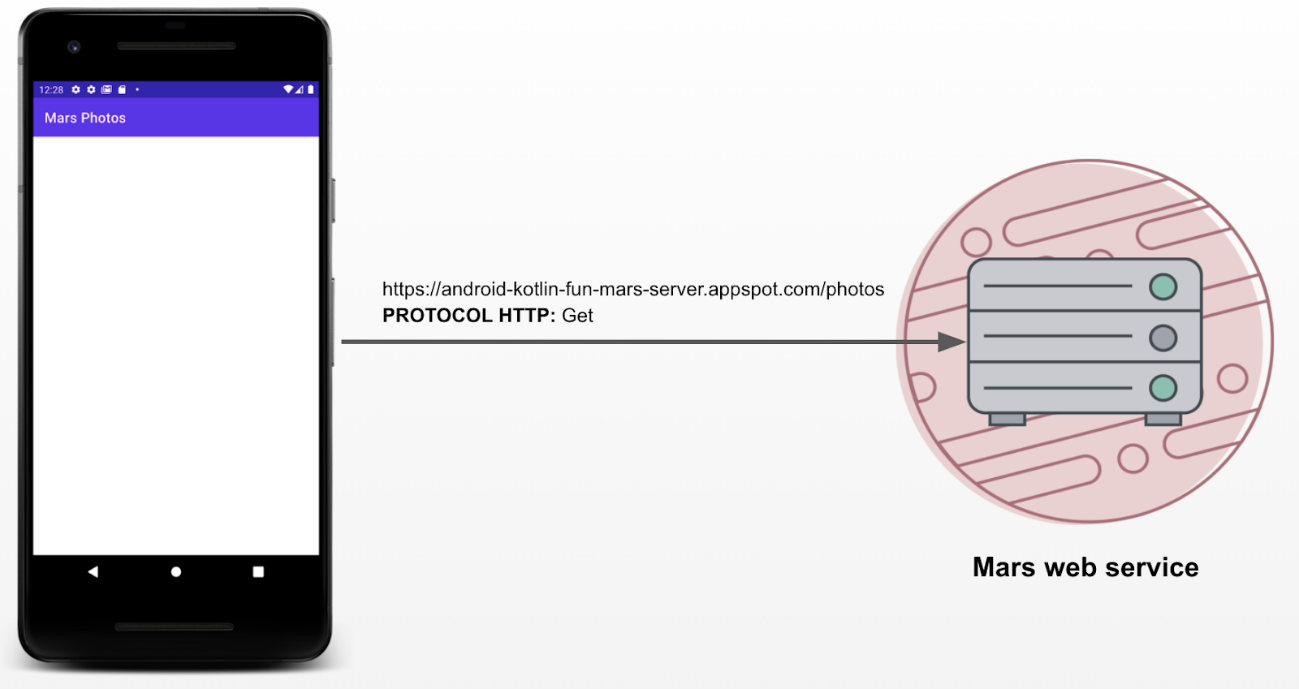
## ****Solicitação de serviço da Web****

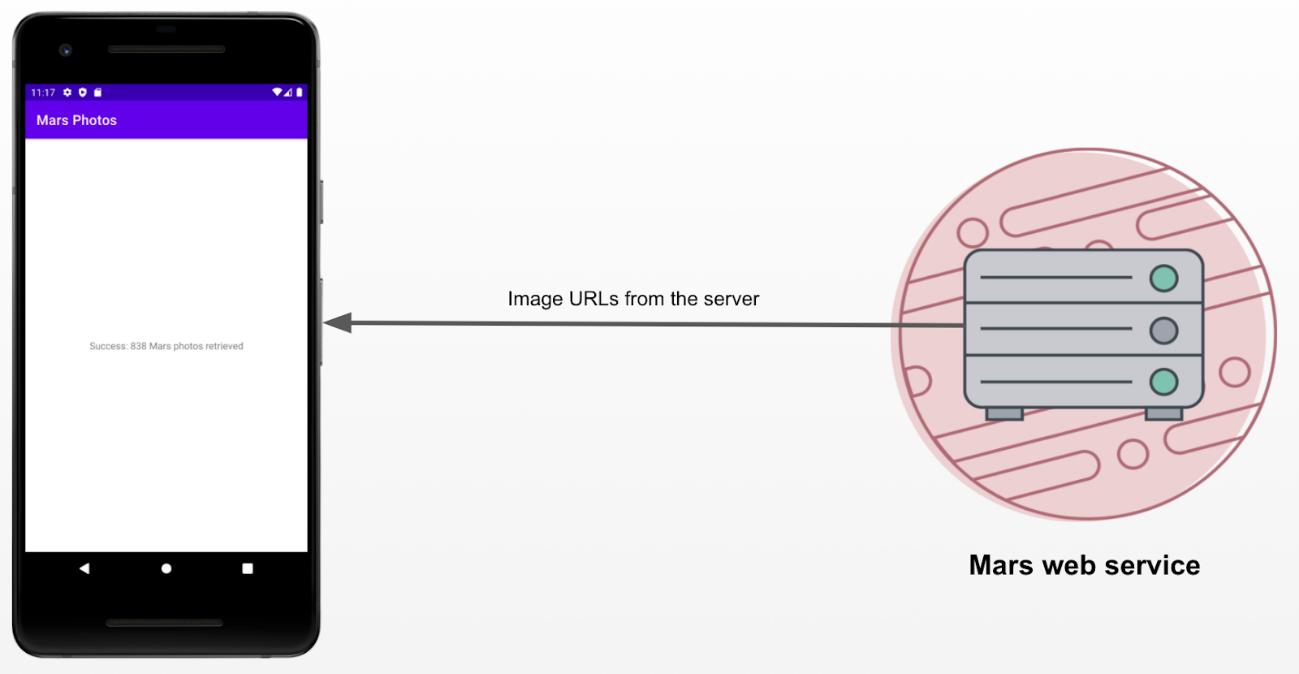
Cada solicitação de serviço da Web contém um URI e é transferida para o servidor usando o mesmo protocolo HTTP usado por navegadores da Web, como o Chrome. As solicitações HTTP contêm uma operação para informar ao servidor o que fazer.

Operações HTTP comuns incluem:

* GET para recuperar dados do servidor
* POST ou PUT para adicionar/criar/atualizar o servidor com novos dados
* DELETE para excluir dados do servidor

O app fará uma solicitação GET HTTP ao servidor para as informações de fotos de Marte e, em seguida, retornará uma resposta para o app, incluindo URLs de imagem.





A resposta de um serviço da Web geralmente oferece um dos formatos mais comuns da Web, como XML ou JSON, para representar dados estruturados em pares de chave-valor. Você aprenderá mais sobre JSON nas próximas tarefas.

Nesta tarefa, você estabelecerá uma conexão de rede e se comunicará com o servidor, e receberá uma resposta JSON. Você usará um servidor de back-end que já foi criado para você. Neste codelab, você usará a biblioteca Retrofit, uma biblioteca de terceiro para se comunicar com o servidor de back-end.

### **Bibliotecas externas**

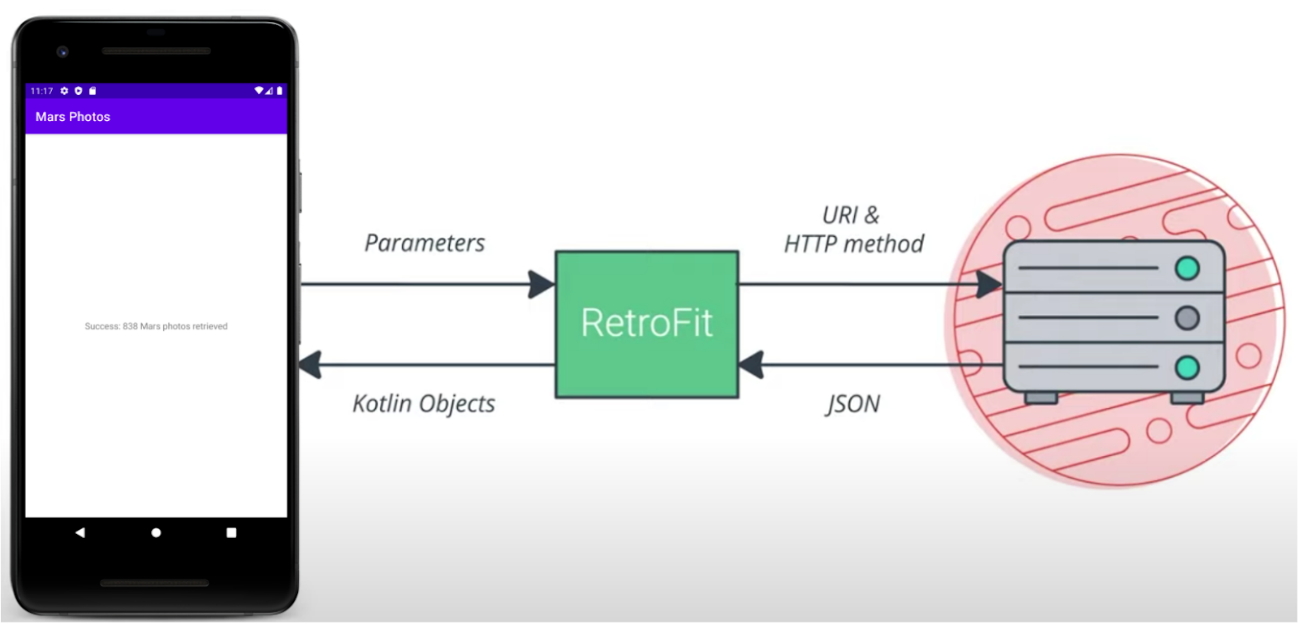
Bibliotecas externas ou bibliotecas de terceiros são como extensões para as principais APIs do Android. Elas são, principalmente, de código aberto, desenvolvidas pela comunidade e mantidas pelas contribuições em conjunto da enorme comunidade Android do mundo todo. Isso permite que desenvolvedores Android como você criem apps melhores.

**Aviso**: o uso de bibliotecas criadas e mantidas pela comunidade pode economizar muito tempo, mas é importante escolher essas bibliotecas com sabedoria, porque seu app é responsável pelo que o código faz nessas bibliotecas.

### **Biblioteca Retrofit**

A biblioteca Retrofit que você usará neste codelab para se comunicar com o serviço RESTful da Web de Marte é um bom exemplo de uma biblioteca com bastante apoio e mantida pela comunidade. Verifique a página do [GitHub](https://github.com/square/retrofit) (link em inglês) e confira os problemas em aberto (alguns são solicitações de recursos) e os problemas resolvidos. Se os desenvolvedores resolvem os problemas e respondem às solicitações de recursos frequentemente, isso significa que essa biblioteca tem boa manutenção e é uma ótima candidata para ser usada no app. A equipe também tem uma página de documentação da [Retrofit](https://square.github.io/retrofit/" \t "_blank) (link em inglês).

A biblioteca Retrofit se comunica com o back-end. Ela cria URIs para o serviço da Web com base nos parâmetros que transmitimos a ela. Falaremos mais sobre isso nas próximas seções.



## Adicionar dependências da Retrofit

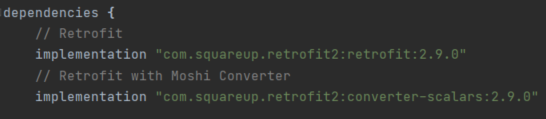
O Android Gradle permite adicionar bibliotecas externas ao projeto. Além da dependência da biblioteca, você também precisa incluir o repositório em que a biblioteca está hospedada. As bibliotecas do Google, como **ViewModel** e **LiveData**, da biblioteca do Jetpack, são hospedadas no repositório do Google. A maioria das bibliotecas da comunidade, como a Retrofit, é hospedada no JCenter.

1. Abra o arquivo de nível superior build.gradle(Project: MarsPhotos) do projeto. Observe os repositórios listados no bloco repositories. Você verá dois repositórios, google(), jcenter().

repositories {  
   google()  
   jcenter()  
}

1. Abra o arquivo do Gradle no nível do módulo, build.gradle (Module: MarsPhots.app).
2. Na seção dependencies, adicione estas linhas para as bibliotecas Retrofit:

// Retrofit  
implementation "com.squareup.retrofit2:retrofit:2.9.0"  
// Retrofit with Moshi Converter  
implementation "com.squareup.retrofit2:converter-scalars:2.9.0"



A primeira dependência é para a biblioteca Retrofit2, e a segunda dependência é para o conversor escalar da Retrofit. Esse conversor permite que a Retrofit retorne o resultado JSON como uma String. As duas bibliotecas trabalham juntas.

1. Clique em **Sync Now** para recriar o projeto com as novas dependências.



## Adicionar a compatibilidade com os recursos da linguagem Java 8

Muitas bibliotecas de terceiros, incluindo a Retrofit2, usam recursos da linguagem Java 8. O Plug-in do Android para Gradle fornece compatibilidade integrada para usar determinados recursos da linguagem Java 8.

1. Para usar os recursos integrados, é necessário incluir o seguinte código no arquivo build.gradle do seu módulo. Esta etapa já foi feita para você. Verifique se o código a seguir está presente no seu módulo build.gradle(Module: MarsPhotos.app).

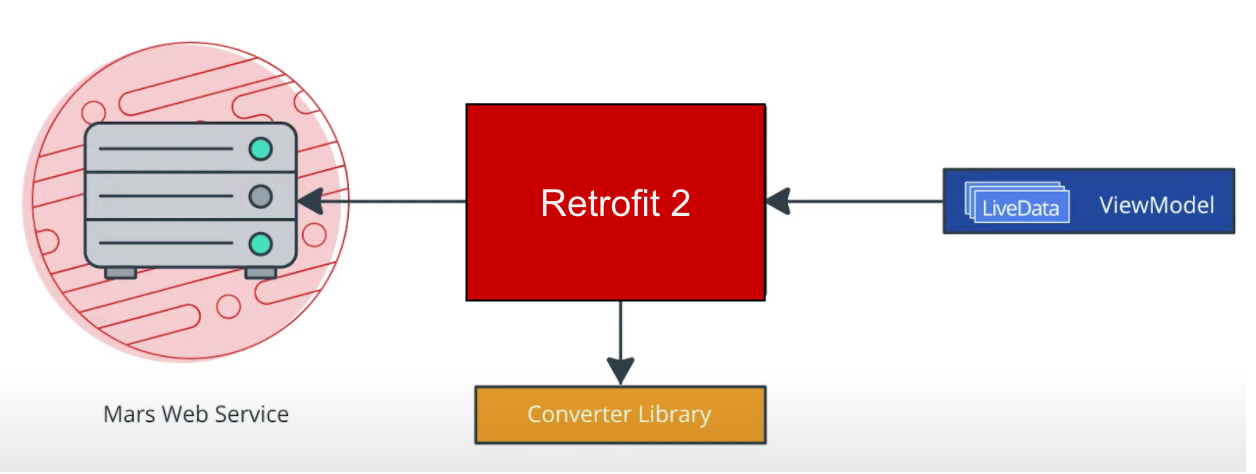
android {  
  ...  
  
  compileOptions {  
    sourceCompatibility JavaVersion.VERSION\_1\_8  
    targetCompatibility JavaVersion.VERSION\_1\_8  
  }  
  
  kotlinOptions {  
    jvmTarget = '1.8'  
  }  
}

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet)

## [6. Como se conectar à Internet](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet#5)

Você usará a biblioteca Retrofit para conversar com o serviço da Web de Marte e exibir a resposta JSON bruta como uma String. O marcador TextView exibirá a string de resposta JSON retornada ou uma mensagem indicando um erro de conexão.

A Retrofit cria uma API de rede para o app com base no conteúdo do serviço da Web. Ela busca dados do serviço da Web e os encaminha usando uma biblioteca de conversor separada, que sabe como decodificar os dados e retorná-los na forma de objetos, como String. A Retrofit inclui suporte integrado para formatos de dados conhecidos como XML e JSON. Por fim, a Retrofit cria o código para chamar e consumir esse serviço para você, incluindo detalhes críticos, como executar solicitações em linhas de execução de segundo plano.



Nesta tarefa, você adicionará uma camada de rede ao projeto **MarsPhotos** que o ViewModel usará para se comunicar com o serviço da Web. Você implementará a API de serviço Retrofit, nas etapas a seguir.

* Crie uma camada de rede, a classe MarsApiService.
* Crie um objeto Retrofit com o URL base e a fábrica de conversores.
* Crie uma interface que explique como a Retrofit se comunica com nosso servidor da Web.
* Crie um serviço Retrofit e exponha a instância ao serviço da API no restante do app.

Implemente as etapas acima:

1. Crie um novo pacote com o nome "network". No painel de projetos do Android, clique com o botão direito do mouse no pacote com.example.android.marsphotos. Selecione **New > Package**. No pop-up, adicione **network** ao final do nome do pacote sugerido.
2. Crie um novo arquivo Kotlin no novo pacote **network**. Nomeie o espaço como MarsApiService.
3. Abra o network/MarsApiService.kt. Adicione a seguinte constante para o URL base do serviço da Web.

private const val BASE\_URL =  
   "https://android-kotlin-fun-mars-server.appspot.com"

1. Logo abaixo dessa constante, adicione um builder da Retrofit para criar um objeto da Retrofit.

private val retrofit = Retrofit.Builder()

Importe retrofit2.Retrofit, quando solicitado.

1. A Retrofit precisa do URI base do serviço da Web e de uma fábrica de conversores para criar uma API de serviços da Web. O conversor informa à Retrofit o que fazer com os dados recebidos do serviço da Web. Nesse caso, você quer que a Retrofit busque uma resposta JSON do serviço da Web e retorne-a como uma String. A Retrofit tem um ScalarsConverter que aceita strings e outros tipos primitivos, portanto, você chama addConverterFactory() no builder com uma instância de ScalarsConverterFactory.

private val retrofit = Retrofit.Builder()  
   .addConverterFactory(ScalarsConverterFactory.create())

Importe retrofit2.converter.scalars.ScalarsConverterFactory quando solicitado.

1. Adicione o URI base do serviço da Web usando o método baseUrl(). Por fim, chame build() para criar o objeto Retrofit.

private val retrofit = Retrofit.Builder()  
   .addConverterFactory(ScalarsConverterFactory.create())  
   .baseUrl(BASE\_URL)  
   .build()

1. Abaixo da chamada para o builder da Retrofit, defina uma interface com o nome MarsApiService, que define como a Retrofit se comunicará com o servidor da Web usando solicitações HTTP.

interface MarsApiService {  
}

1. Na interface MarsApiService, adicione uma função com o nome getPhotos() para receber a string de resposta do serviço da Web.

interface MarsApiService {  
    fun getPhotos()  
}

1. Use a anotação @GET para informar à Retrofit que esta é uma solicitação GET e especifique o endpoint para esse método de serviço da Web. Nesse caso, o endpoint tem o nome photos. Conforme mencionado na tarefa anterior, você usará o endpoint [/photos](https://android-kotlin-fun-mars-server.appspot.com/photos?hl=pt-br) (link em inglês) neste codelab.

interface MarsApiService {  
    @GET("photos")  
    fun getPhotos()  
}

Importe retrofit2.http.GET quando solicitado.

1. Quando o método getPhotos() é invocado, a Retrofit anexa o endpoint photos ao URL base (que você definiu no builder da Retrofit) usado para iniciar a solicitação. Adicione um tipo de retorno da função à String.

interface MarsApiService {  
    @GET("photos")  
    fun getPhotos(): String  
}

### **Declarações de objetos**

Em Kotlin, as [declarações de objetos](https://kotlinlang.org/docs/reference/object-declarations.html#object-declarations) (link em inglês) são usadas para declarar objetos Singleton. O [padrão Singleton](https://en.wikipedia.org/wiki/Singleton_pattern) garante que uma instância de um objeto seja criada e tenha um ponto global de acesso a esse objeto. A inicialização da declaração do objeto é segura para linhas de execução e é feita no primeiro acesso.

Com o Kotlin, é fácil declarar Singletons. Veja abaixo um exemplo de declaração de objeto e como acessá-lo. A declaração de objeto sempre tem um nome depois da palavra-chave object.

**Exemplo**:

// Object declaration  
object DataProviderManager {  
    fun registerDataProvider(provider: DataProvider) {  
        // ...  
    }  
​  
    val allDataProviders: Collection<DataProvider>  
        get() = // ...  
}  
  
// To refer to the object, use its name directly.  
DataProviderManager.registerDataProvider(...)

A chamada para a função create() em um objeto da Retrofit é custosa e o app precisa de apenas uma instância do serviço da API da Retrofit. Assim, você expõe o serviço ao restante do app usando a declaração de objetos.

1. Fora da declaração da interface MarsApiService, defina um objeto público com o nome MarsApi para inicializar o serviço da Retrofit. Esse é o objeto Singleton público que pode ser acessado pelo restante do app.

object MarsApi {  
  
}

1. Na declaração do objeto MarsApi, adicione uma propriedade de inicialização lenta de objeto da Retrofit com o nome retrofitService e o tipo MarsApiService. Você faz essa inicialização lenta, para garantir que o objeto seja inicializado no primeiro uso. Você corrigirá o erro nas próximas etapas.

object MarsApi {  
    val retrofitService : MarsApiService by lazy {  
       }  
}

1. Inicialize a variável retrofitService usando o método retrofit.create() com a interface MarsApiService.

object MarsApi {  
    val retrofitService : MarsApiService by lazy {  
       retrofit.create(MarsApiService::class.java) }  
}

A configuração da Retrofit foi concluída. Toda vez que o app chamar MarsApi.retrofitService, o autor da chamada acessará o mesmo objeto Singleton da Retrofit que implementa o MarsApiService, criado no primeiro acesso. Na próxima tarefa, você usará o objeto Retrofit que implementou.

**Observação**: lembre-se de que a "instanciação lenta" ocorre quando a criação do objeto é atrasada até que seja necessária para evitar cálculos ou o uso desnecessário outros recursos de computação. O Kotlin tem [compatibilidade de primeira classe](https://kotlinlang.org/docs/reference/delegated-properties.html#lazy) (link em inglês) com a instanciação lenta.

## Chamar o serviço da Web no OverviewViewModel

Nesta etapa, você implementará o método getMarsPhotos() que chama o serviço da Retrofit e, em seguida, manipula a string JSON retornada.

### **ViewModelScope**

Um [ViewModelScope](https://developer.android.com/topic/libraries/architecture/coroutines?hl=pt-br" \l "viewmodelscope" \t "_blank) é o escopo integrado de corrotinas definido para cada ViewModel no app. Qualquer corrotina iniciada nesse escopo será cancelada automaticamente se o ViewModel for limpo.

Você usará o ViewModelScope para iniciar a corrotina e fazer a chamada de rede da Retrofit em segundo plano.

1. No MarsApiService, transforme getPhotos() em uma função de suspensão. Assim, você poderá chamar esse método em uma corrotina.

@GET("photos")  
suspend fun getPhotos(): String

1. Abra o overview/OverviewViewModel. Role para baixo até encontrar o método getMarsPhotos(). Exclua a linha que define a resposta de status para "Set the Mars API Response here!".. O método getMarsPhotos() estará vazio agora.

private fun getMarsPhotos() {  
  
}

1. No método getMarsPhotos(), inicie a corrotina usando viewModelScope.launch.

private fun getMarsPhotos() {  
    viewModelScope.launch {  
    }  
}

Importe androidx.lifecycle.*viewModelScope* e kotlinx.coroutines.launch quando solicitado.

1. No viewModelScope, use o objeto Singleton MarsApi para chamar o método getPhotos() da interface retrofitService. Salve a resposta retornada em uma val com o nome listResult.

viewModelScope.launch {  
    val listResult = MarsApi.retrofitService.getPhotos()  
}

Importe com.example.android.marsphotos.network.MarsApi quando solicitado.

1. Atribua o resultado que acabamos de receber do servidor de back-end ao arquivo \_status.value.

 val listResult = MarsApi.retrofitService.getProperties()  
 \_status.value = listResult

1. Execute o app. Ele fechará imediatamente. O app pode ou não exibir uma janela pop-up de erro.
2. Clique na guia **Logcat** no Android Studio e observe o erro no registro, que começa com uma linha como esta "------- beginning of crash".

--------- beginning of crash

22803-22865/com.example.android.marsphotos E/AndroidRuntime: FATAL EXCEPTION: OkHttp Dispatcher

Process: com.example.android.marsphotos, PID: 22803

java.lang.SecurityException: Permission denied (missing INTERNET permission?)

...

Essa mensagem de erro indica que o app pode não ter as permissões de INTERNET. Para resolver isso, adicione permissões de Internet ao app na próxima tarefa.

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet)

## [7. Adicionar permissão de Internet e processamento de exceção](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet#6)

## ****Permissões do Android****

O objetivo das permissões no Android é proteger a privacidade de um usuário do Android. Os apps Android precisam declarar ou solicitar permissões para acessar dados sensíveis do usuário, como contatos, registros de chamadas e alguns recursos do sistema, como a câmera ou a Internet.

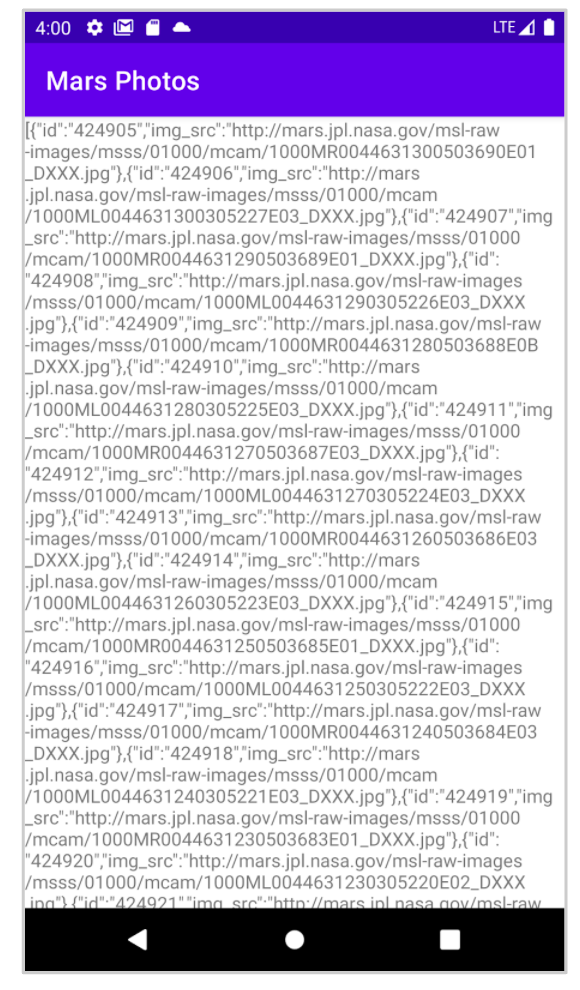
Para que o app acesse a Internet, ele precisa da permissão INTERNET. Conectar-se à Internet gera preocupações de segurança. Por isso, os apps não têm conectividade com a Internet por padrão. É necessário declarar explicitamente que o app precisa de acesso à Internet. Isso é considerado uma permissão normal. Para saber mais sobre os tipos de permissões do Android, consulte esta [documentação](https://developer.android.com/guide/topics/permissions/overview?hl=pt-br).

Nesta etapa, seu app declara as permissões necessárias incluindo tags <uses-permission> no arquivo AndroidManifest.

1. Abra o manifests/AndroidManifest.xml. Adicione esta linha imediatamente antes da tag <application>:

<uses-permission android:name="android.permission.INTERNET" />

1. Compile e execute o app novamente. Se a conexão com a Internet estiver funcionando, você verá o texto JSON com os dados relacionados às fotos de Marte. Você aprenderá mais sobre o formato JSON posteriormente no codelab.



1. Toque no botão **Voltar** do dispositivo ou emulador para fechar o app.
2. Coloque o dispositivo ou emulador no modo avião para simular um erro de conexão de rede. Abra o app novamente no menu Recentes ou reinicie-o no Android Studio.
3. Clique na guia **Logcat** no Android Studio e observe a exceção fatal no registro, que se parece com esta:

3302-3302/com.example.android.marsphotos E/AndroidRuntime: FATAL EXCEPTION: main

Process: com.example.android.marsphotos, PID: 3302

java.net.SocketTimeoutException: timeout

...

Essa mensagem de erro indica que o app tentou se conectar e o tempo expirou. Exceções como essa são muito comuns em situações de uso reais. Na próxima etapa, você aprenderá a lidar com essas exceções.

## ****Como processar exceções****

[Exceções](https://developer.android.com/reference/java/lang/Exception?hl=pt-br) são erros que podem ocorrer durante a execução (e não durante a compilação) e encerram o app abruptamente sem notificar o usuário. Isso pode resultar em uma experiência insatisfatória para o usuário. O processamento de exceções é um mecanismo que impede que o app seja encerrado abruptamente e seja fácil de usar.

As exceções podem ser causadas por algo simples, como uma divisão por zero ou um erro na rede. Essas exceções são semelhantes à NumberFormatException que você aprendeu em um codelab anterior.

Exemplos de possíveis problemas ao se conectar a um servidor:

* O URL ou URI usado pela API está incorreto.
* O servidor não está disponível e o app não conseguiu se conectar a ele.
* Problema de latência da rede.
* Conexão de Internet ruim ou inexistente no dispositivo.

Essas exceções não podem ser capturadas durante a compilação. Você pode usar um bloco try-catch para processar a exceção durante a execução. Para saber mais, consulte a [documentação](https://kotlinlang.org/docs/reference/exceptions.html) (link em inglês).

**Exemplo de sintaxe para o bloco try-catch**

try {  
    // some code that can cause an exception.  
}  
catch (e: SomeException) {  
    // handle the exception to avoid abrupt termination.  
}

No bloco try, você executa o código em que antecipa uma exceção, neste app, isso seria uma chamada de rede. No bloco catch, implemente o código que impede o encerramento abrupto do app. Se houver uma exceção, o bloco catch será executado para recuperar o app do erro em vez de encerrá-lo abruptamente.

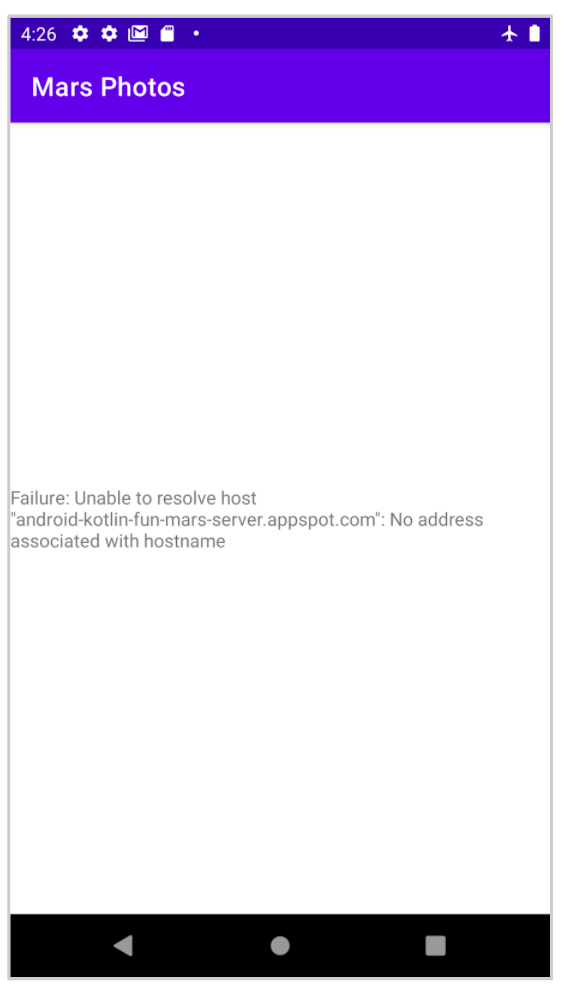
1. Abra o overview/OverviewViewModel.kt. Role para baixo até encontrar o método getMarsPhotos(). No bloco de inicialização, adicione um bloco try na chamada MarsApi para processar exceções. Adicione o bloco catch após o bloco try:

viewModelScope.launch {  
   try {  
       val listResult = MarsApi.retrofitService.getPhotos()  
       \_status.value = listResult  
   } catch (e: Exception) {  
  
   }  
}

1. No bloco catch {}, gerencie a resposta da falha. Exiba a mensagem de erro para o usuário definindo e.message como \_status.*value*.

catch (e: Exception) {  
   \_status.value = "Failure: ${e.message}"  
}

1. Execute o app novamente com o modo avião ativado. O app não será fechado abruptamente, mas exibirá uma mensagem de erro.



1. Desative o modo avião no smartphone ou emulador. Execute e teste o app. Verifique se tudo está funcionando bem e se você consegue ver a string JSON.

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet)

## [8. Analisar a resposta JSON com Moshi](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet#7)

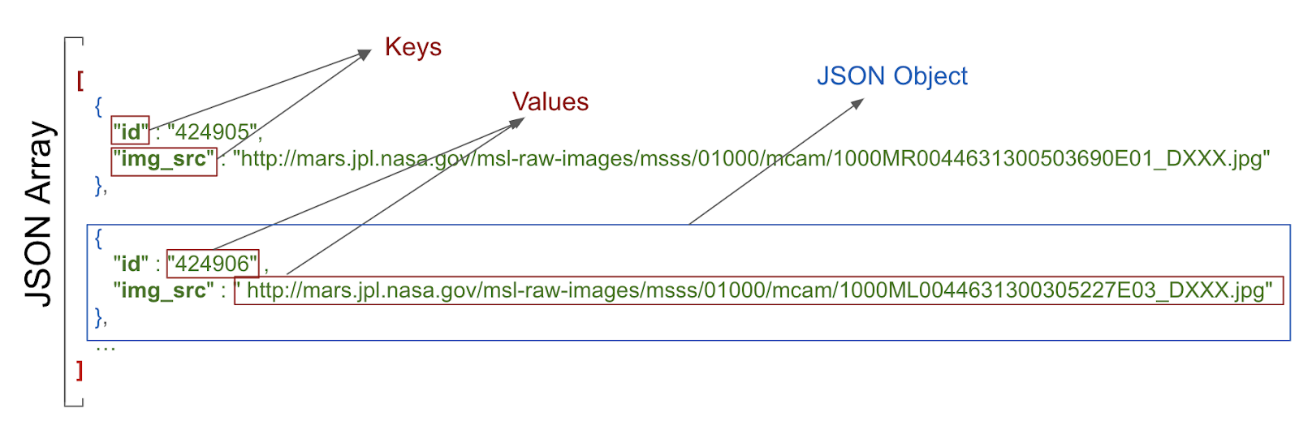
## ****JSON****

Os dados solicitados normalmente são formatados em um dos formatos de dados comuns, como XML ou JSON. Cada chamada retorna dados estruturados e seu app precisa saber qual é a estrutura para ler os dados da resposta.

Por exemplo, neste app, você recupera os dados do servidor: [https://](https://mars.udacity.com/realestate) [android-kotlin-fun-mars-server.appspot.com/photos](https://android-kotlin-fun-mars-server.appspot.com/photos?hl=pt-br) (link em inglês). Se você inserir esse URL no navegador,

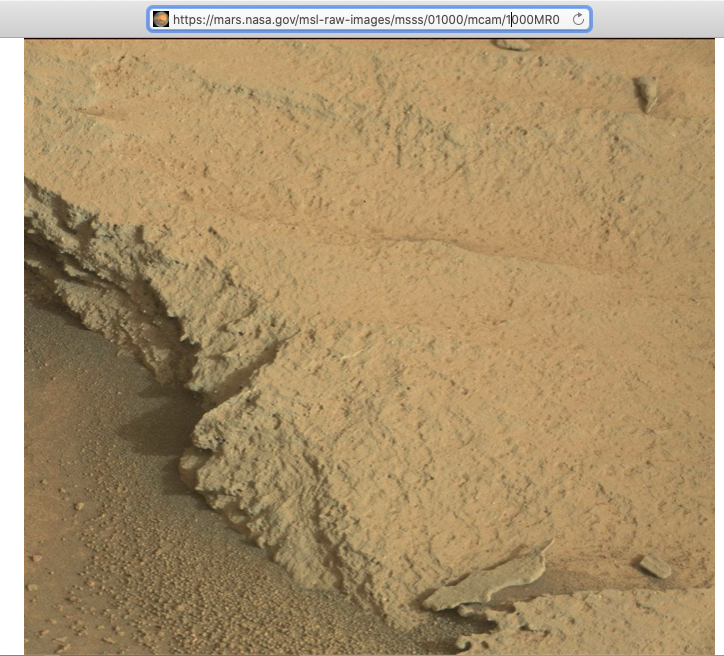
Você verá uma lista de IDs e URLs de imagem da superfície de Marte em um formato JSON.

### **Estrutura do exemplo de resposta JSON:**



* A resposta JSON é uma matriz, indicada por colchetes. A matriz contém objetos JSON.
* Os objetos JSON ficam entre chaves.
* Cada objeto JSON contém um conjunto de pares de nome-valor. O nome e o valor são separados por dois pontos.
* Os nomes ficam entre aspas.
* Os valores podem ser números, strings, booleanos, uma matriz, um objeto (objeto JSON) ou nulo.

Por exemplo, img\_src é um URL, que é uma string. Se você colar o URL em um navegador da Web, verá uma imagem da superfície de Marte.



Você receberá uma resposta JSON do serviço da Web de Marte, o que é um ótimo começo. Mas o que você realmente precisa são objetos Kotlin, não uma string JSON longa. Há uma biblioteca externa, [Moshi](https://github.com/square/moshi" \t "_blank) (link em inglês), que é um analisador JSON do Android que converte strings JSON em objetos Kotlin. A Retrofit tem um conversor que funciona com a Moshi, então ela é uma ótima biblioteca para o objetivo deste codelab.

Nesta tarefa, você usará a biblioteca Moshi com a Retrofit para analisar a resposta JSON do serviço da Web em objetos Kotlin úteis que representam fotos de Marte. Você mudará o app para que, em vez de exibir o JSON bruto, ele exiba o número de fotos de Marte retornadas.

## Adicionar dependências da biblioteca Moshi

1. Abra **build.gradle (Module: app)**.
2. Na seção de dependências, adicione o código abaixo para incluir a dependência da Moshi. Essa dependência adiciona a compatibilidade com a biblioteca JSON Moshi com compatibilidade com Kotlin.

// Moshi  
implementation 'com.squareup.moshi:moshi-kotlin:1.9.3'

1. Localize as linhas do conversor escalar para a Retrofit no bloco dependencies e mude estas dependências para usar converter-moshi:

**Substitua**

// Retrofit  
implementation "com.squareup.retrofit2:retrofit:2.9.0"  
// Retrofit with Moshi Converter  
implementation "com.squareup.retrofit2:converter-scalars:2.9.0"

**por**

// Retrofit with Moshi Converter  
implementation 'com.squareup.retrofit2:converter-moshi:2.9.0'

1. Clique em **Sync Now** para recriar o projeto com as novas dependências.

**Observação**: o projeto pode mostrar erros do compilador relacionados à dependência escalar da Retrofit. Você os corrigirá nas próximas etapas.

## ****Implementar a classe de dados de Fotos de Marte****

Uma entrada de exemplo da resposta JSON que você recebe do serviço da Web é semelhante a esta:

[{  
    "id":"424906",  
    "img\_src":"http://mars.jpl.nasa.gov/msl-raw-images/msss/01000/mcam/1000ML0044631300305227E03\_DXXX.jpg"  
},  
...]

No exemplo acima, cada entrada de foto de Marte tem estes pares de chave e valor JSON:

* id: o ID da propriedade, como uma string. Como está encapsulado entre " ", ele é do tipo String, não Integer.
* img\_src: o URL da imagem como uma string.

A Moshi analisa esses dados JSON e os converte em objetos Kotlin. Para fazer isso, a Moshi precisa ter uma classe de dados Kotlin para armazenar os resultados analisados, portanto, nesta etapa, você criará a classe de dados, MarsPhoto.

1. Clique com o botão direito do mouse no pacote **network** e selecione **New > Kotlin File/Class.**
2. No pop-up, selecione **Class** e digite MarsPhoto como o nome da classe. Isso criará um novo arquivo com o nome MarsPhoto.kt no pacote network.
3. Transforme MarsPhoto em uma classe de dados adicionando a palavra-chave data antes da definição da classe. Mude as chaves {} para parênteses (). Isso causa um erro, já que as classes de dados precisam ter pelo menos uma propriedade definida.

data class MarsPhoto(  
)

1. Adicione as seguintes propriedades à definição da classe MarsPhoto.

data class MarsPhoto(  
   val id: String, val img\_src: String  
)

Cada uma das variáveis na classe MarsPhoto corresponde a um nome de chave no objeto JSON. Para corresponder aos tipos da nossa resposta JSON específica, use objetos String para todos os valores.

Quando a Moshi analisa o JSON, ela faz a correspondência das chaves por nome e preenche os objetos de dados com os valores corretos.

### **Anotação @Json**

Algumas vezes, os nomes de chave em uma resposta JSON podem criar propriedades Kotlin confusas ou podem não corresponder ao estilo recomendado de programação. Por exemplo, no arquivo JSON, a chave img\_src usa um sublinhado, enquanto a convenção Kotlin para propriedades usa letras maiúsculas e minúsculas ("letras concatenadas").

Para usar nomes de variáveis na sua classe de dados que sejam diferentes dos nomes de chave da resposta JSON, use a anotação @Json. Neste exemplo, o nome da variável na classe de dados é imgSrcUrl. A variável pode ser mapeada para o atributo JSON img\_src usando @Json(name = "img\_src").

1. Substitua a linha da chave img\_src pela linha mostrada abaixo. Importe com.squareup.moshi.Json quando solicitado.

@Json(name = "img\_src") val imgSrcUrl: String

## Atualizar MarsApiService e OverviewViewModel

Nesta tarefa, você criará um objeto Moshi usando o Moshi Builder, semelhante ao builder da Retrofit.

Você substituirá a ScalarsConverterFactory pela KotlinJsonAdapterFactory para informar que a Retrofit pode usar a Moshi para converter a resposta JSON em objetos Kotlin. Você atualizará a API da rede e ViewModel para usar o objeto Moshi.

1. Abra o network/MarsApiService.kt. Há erros de referência não resolvidos para ScalarsConverterFactory. Isso ocorre devido à mudança da dependência da Retrofit feita em uma etapa anterior. Exclua a importação de ScalarConverterFactory. Você corrigirá o outro erro em breve.

**Remova**:

import retrofit2.converter.scalars.ScalarsConverterFactory

1. Na parte superior do arquivo, antes do builder da Retrofit, adicione o seguinte código para criar o objeto Moshi, semelhante ao objeto Retrofit.

private val moshi = Moshi.Builder()

Importe com.squareup.moshi.Moshi e com.squareup.moshi.kotlin.reflect.KotlinJsonAdapterFactory quando solicitado.

1. Para que as anotações da Moshi funcionem corretamente com o Kotlin, adicione a KotlinJsonAdapterFactory no builder da Moshi e chame build().

private val moshi = Moshi.Builder()  
   .add(KotlinJsonAdapterFactory())  
   .build()

1. Na declaração do objeto *retrofit*, mude o builder da Retrofit para usar a MoshiConverterFactory em vez da ScalarConverterFactory e transmita a instância moshi que você acabou de criar.

private val retrofit = Retrofit.Builder()  
   .addConverterFactory(MoshiConverterFactory.create(moshi))  
   .baseUrl(BASE\_URL)  
   .build()

Importe retrofit2.converter.moshi.MoshiConverterFactory quando solicitado.

1. Agora que você implementou a MoshiConverterFactory, peça à Retrofit para retornar uma lista de objetos MarsPhoto da matriz JSON em vez de retornar uma string JSON. Atualize a interface MarsApiService para que a Retrofit retorne uma lista de objetos MarsPhoto, em vez de retornar uma String.

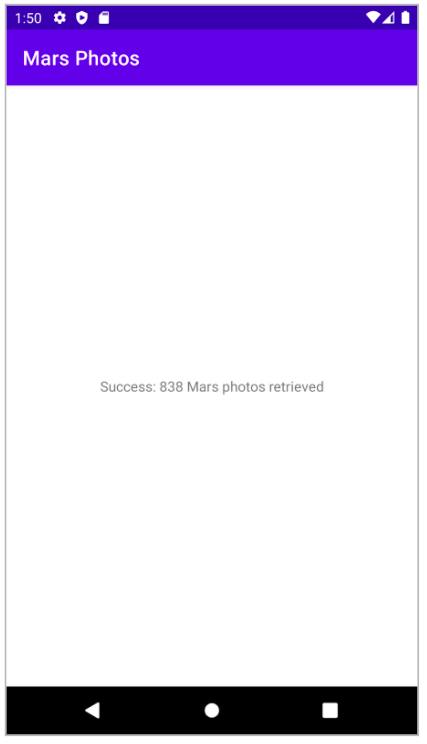
interface MarsApiService {  
   @GET("photo")  
   fun getPhotos(): List<MarsPhoto>  
}

1. Faça mudanças semelhantes no viewModel e abra OverviewViewModel.kt. Role para baixo até encontrar o método getMarsPhotos().
2. No método getMarsPhotos(), listResult é uma List<MarsPhoto> e não uma String. O tamanho dessa lista é o número de fotos que foram recebidas e analisadas. Para exibir o número de fotos extraídas, atualize \_status.*value* desta forma:

\_status.value = "Success: ${listResult.size} Mars photos retrieved"

Importe com.example.android.marsphotos.network.MarsPhoto quando solicitado.

1. Verifique se o modo avião está desativado no dispositivo ou emulador. Compile e execute o app. Desta vez, a mensagem mostrará o número de propriedades retornadas do serviço da Web, e não uma string JSON longa:



**Observação**: se a conexão de Internet não estiver funcionando, verifique se você desativou o modo avião no dispositivo ou emulador.

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet)

## [9. Código da solução](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet#8)

**Observação**: verifique se o nome do pacote no código-fonte a seguir corresponde ao nome do pacote no seu projeto.

build.gradle(Module : MarsPhotos.app)

Estas são as novas dependências que serão incluídas.

dependencies {

...

// Moshi

implementation 'com.squareup.moshi:moshi-kotlin:1.9.3'

// Retrofit with Moshi Converter

implementation 'com.squareup.retrofit2:converter-moshi:2.9.0'

...

}

Manifests/AndroidManifest.xml

Adicione a permissão de Internet, o código <uses-permission..>, do snippet de código abaixo.

<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.example.android.marsphotos">  
  
    <!-- In order for our app to access the Internet, we need to define this permission. -->  
    <uses-permission android:name="android.permission.INTERNET" />  
  
    <application  
        ...  
    </application>  
  
</manifest>

network/MarsPhoto.kt

package com.example.android.marsphotos.network  
  
import com.squareup.moshi.Json  
  
/\*\*  
\* This data class defines a Mars photo which includes an ID, and the image URL.  
\* The property names of this data class are used by Moshi to match the names of values in JSON.  
\*/  
data class MarsPhoto(  
   val id: String,  
   @Json(name = "img\_src") val imgSrcUrl: String  
)

network/MarsApiService.kt

package com.example.android.marsphotos.network  
  
import com.squareup.moshi.Moshi  
import com.squareup.moshi.kotlin.reflect.KotlinJsonAdapterFactory  
import retrofit2.Retrofit  
import retrofit2.converter.moshi.MoshiConverterFactory  
import retrofit2.http.GET  
  
private const val BASE\_URL =  
   "https://android-kotlin-fun-mars-server.appspot.com"  
  
/\*\*  
\* Build the Moshi object with Kotlin adapter factory that Retrofit will be using.  
\*/  
private val moshi = Moshi.Builder()  
   .add(KotlinJsonAdapterFactory())  
   .build()  
  
/\*\*  
\* The Retrofit object with the Moshi converter.  
\*/  
private val retrofit = Retrofit.Builder()  
   .addConverterFactory(MoshiConverterFactory.create(moshi))  
   .baseUrl(BASE\_URL)  
   .build()  
  
/\*\*  
\* A public interface that exposes the [getPhotos] method  
\*/  
interface MarsApiService {  
   /\*\*  
    \* Returns a [List] of [MarsPhoto] and this method can be called from a Coroutine.  
    \* The @GET annotation indicates that the "photos" endpoint will be requested with the GET  
    \* HTTP method  
    \*/  
   @GET("photos")  
   suspend fun getPhotos() : List<MarsPhoto>  
}  
  
/\*\*  
\* A public Api object that exposes the lazy-initialized Retrofit service  
\*/  
object MarsApi {  
   val retrofitService: MarsApiService by lazy { retrofit.create(MarsApiService::class.java) }  
}

Overview/OverviewViewModel.kt

package com.example.android.marsphotos.overview  
  
import androidx.lifecycle.LiveData  
import androidx.lifecycle.MutableLiveData  
import androidx.lifecycle.ViewModel  
import androidx.lifecycle.viewModelScope  
import com.example.android.marsphotos.network.MarsApi  
import kotlinx.coroutines.launch  
  
/\*\*  
\* The [ViewModel] that is attached to the [OverviewFragment].  
\*/  
class OverviewViewModel : ViewModel() {  
  
   // The internal MutableLiveData that stores the status of the most recent request  
   private val \_status = MutableLiveData<String>()  
  
   // The external immutable LiveData for the request status  
   val status: LiveData<String> = \_status  
   /\*\*  
    \* Call getMarsPhotos() on init so we can display status immediately.  
    \*/  
   init {  
       getMarsPhotos()  
   }  
  
   /\*\*  
    \* Gets Mars photos information from the Mars API Retrofit service and updates the  
    \* [MarsPhoto] [List] [LiveData].  
    \*/  
   private fun getMarsPhotos() {  
       viewModelScope.launch {  
           try {  
               val listResult = MarsApi.retrofitService.getPhotos()  
               \_status.value = "Success: ${listResult.size} Mars photos retrieved"  
           } catch (e: Exception) {  
               \_status.value = "Failure: ${e.message}"  
           }  
       }  
   }  
}

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet)

## [10. Resumo](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet#9)

## ****Serviços REST da Web****

* Um serviço da Web é uma funcionalidade baseada em software oferecida pela Internet que permite que o app faça solicitações e receba dados de volta.
* Os serviços comuns da Web usam uma arquitetura [REST](https://en.wikipedia.org/wiki/Representational_state_transfer) (link em inglês). Os serviços da Web que oferecem arquitetura REST são conhecidos como serviços RESTful. Serviços RESTful da Web são criados usando componentes e protocolos padrão da Web.
* Você faz uma solicitação a um serviço REST da Web de maneira padronizada usando URIs.
* Para usar um serviço da Web, um app precisa estabelecer uma conexão de rede e se comunicar com o serviço. Em seguida, o app precisa receber e analisar os dados de resposta em um formato que ele possa usar.
* A biblioteca [Retrofit](https://square.github.io/retrofit/" \t "_blank) (link em inglês) é uma biblioteca de cliente que permite que o app faça solicitações a um serviço REST da Web.
* Use os conversores para informar à Retrofit o que fazer com os dados que ela envia ao serviço da Web e os dados que ela recebe do serviço. Por exemplo, o conversor ScalarsConverter trata os dados do serviço da Web como uma String ou outro primitivo.
* Para permitir que seu app faça conexões com a Internet, adicione a permissão "android.permission.INTERNET" ao manifesto do Android.

## ****Análise JSON****

* A resposta de um serviço da Web é geralmente formatada em [JSON](https://www.json.org/) (link em inglês), um formato comum para representar dados estruturados.
* Um objeto JSON é uma coleção de pares de chave-valor.
* Uma coleção de objetos JSON é uma matriz JSON. Você recebe uma matriz JSON como resposta de um serviço da Web.
* As chaves em um par de chave-valor ficam entre aspas. Os valores podem ser números ou strings.
* A biblioteca [Moshi](https://github.com/square/moshi" \t "_blank) (link em inglês) é um analisador JSON do Android que converte uma string JSON em objetos Kotlin. A Retrofit tem um conversor que funciona com a Moshi.
* A Moshi combina as chaves em uma resposta JSON com as propriedades de um objeto de dados com o mesmo nome.
* Para usar um nome de propriedade diferente para uma chave, use a anotação @Json e o nome da chave JSON para anotar essa propriedade.

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet)

## [11. Saiba mais](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet#10)

Documentação para desenvolvedores Android:

* [Visão geral do ViewModel](https://developer.android.com/topic/libraries/architecture/viewmodel?hl=pt-br)
* [Visão geral do LiveData](https://developer.android.com/topic/libraries/architecture/livedata?hl=pt-br#kotlin)
* [MutableLiveData](https://developer.android.com/reference/android/arch/lifecycle/MutableLiveData?hl=pt-br)
* [ViewModelScope](https://developer.android.com/topic/libraries/architecture/coroutines?hl=pt-br#viewmodelscope)

Documentação do Kotlin:

* [Exceções: try, catch, finally, throw, nothing](https://kotlinlang.org/docs/reference/exceptions.html) (link em inglês)
* [Codelab de corrotinas](https://codelabs.developers.google.com/codelabs/kotlin-coroutines/?hl=pt-br)
* [Documentação oficial de corrotinas](https://kotlinlang.org/docs/reference/coroutines-overview.html) (link em inglês)
* [Contexto e agentes de corrotinas](https://kotlinlang.org/docs/reference/coroutines/coroutine-context-and-dispatchers.html) (link em inglês)

Outro:

* [Retrofit](https://square.github.io/retrofit/) (link em inglês)
* [Moshi](https://github.com/square/moshi) (link em inglês)
* [Moshi com Retrofit no Kotlin](https://proandroiddev.com/moshi-with-retrofit-in-kotlin-%EF%B8%8F-a69c2621708b) (link em inglês)

[**Voltar**](https://developer.android.com/codelabs/basic-android-kotlin-training-getting-data-internet?hl=pt-br&continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-kotlin-unit-4-pathway-2%3Fhl%3Dpt-br%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-training-getting-data-internet)