

MATERIAL EXTRA	2
NETWORKING - Comandos GET/POST	2
Servidores de Teste	2
Criação do Projeto	2
O Storyboard	3
O Código	3
As conexões	3
Propriedades	4
As funções principais	4
Os Selectors	6
Liberação de Acesso	7

## MATERIAL EXTRA

# NETWORKING - Comandos GET/POST

Os métodos HTTP mais usados para realizarmos requisições são:

- **GET** - O método GET é usado para fazer requisições no servidor, passando seus parâmetros de forma visível para o usuário.
- **POST** - O método POST envia dados para serem processados no servidor. Diferente do método GET, o POST é enviado de maneira mais segura, não especificando o conteúdo através da URL.

Neste material, iremos construir uma aplicação simples, mas que irá executar os métodos GET e POST.

## Servidores de Teste

Para conseguir demonstrar na prática o funcionamento dos métodos GET e POST, iremos utilizar dois servidores públicos de teste:

- <https://httpbin.org>
  - Para GET
  - Iremos recuperar o próprio endereço IP da requisição, portanto o endpoint que utilizaremos será:
    - <https://httpbin.org/ip>
  - Você pode testar o retorno utilizando o próprio navegador.
- <http://requestb.in>
  - Para POST
  - ATENÇÃO:
    - Para a operação de POST, precisamos criar um endpoint no servidor para publicar pelo aplicativo
    - Para isso executar os seguintes procedimentos:
      - Na página, selecionar <Create RequestBin>
      - Copie o endereço criado para a aplicação.
      - Exemplo:
        - <http://requestb.in/1a6d4if1>

## Criação do Projeto

Criar novo projeto - **NetWorking**

- O nosso projeto será dividido em duas partes:
  1. Criação de método GET para recuperar o nosso próprio IP
  2. Criação de requisição POST para publicar uma informação no servidor

## O Storyboard

A nossa View será muito simples, e conterá apenas dois elementos visuais:

- Um *Segmented Control*, para separar as operações de GET/POST
- Um *Label*, para exibir o retorno HTTP do servidor



## O Código

O código será todo construído na View Controller padrão do projeto, arquivo ViewController.swift. A partir dos próximos tópicos, basta digitar o código na sequência solicitada.

## As conexões

Na classe principal **ViewController.swift**, insira o seguinte código, para efetuar as conexões Outlet e Action:

```
// MARK: - Outlets
@IBOutlet weak var segNetwork: UISegmentedControl!
@IBOutlet weak var labelResultado: UILabel!

// MARK: - Actions
@IBAction func networkSelecionada(_ sender: UISegmentedControl) {

    switch segNetwork.selectedSegmentIndex {
    case 0:
        // GET
        enviarGET()
    case 1:
        // POST
        enviarPOST()
    default:
        print("Opção Inválida")
    }
}
```

OBS: Iremos codificar as funções **enviarGET()** e **enviarPOST()** mais pra frente.

## Propriedades

Também na classe principal **ViewController.swift**, precisaremos de 2 propriedades, para representar os endpoints para as operações de GET/POST:

```
// MARK: - Propriedades
let getEndPoint = "https://httpbin.org/ip"
let postEndPoint = "http://requestb.in/1lxv0mu1"
```

**ATENÇÃO:** A url de post deve ter sido gerada na página do servidor de POST - veja a seção **Servidores de Teste**.

## As funções principais

Codifique agora as funções principais do aplicativo, que irão respectivamente efetuar o GET do IP da própria requisição, e efetuar um POST em um servidor de teste.

Observe que utilizaremos a mesma sequência de comandos de um parse de JSON.

Inicialmente a função **enviarGET()**:

```
// MARK: - Funções de Apoio
func enviarGET() {

    // Validação do Endpoint, com guard
    guard let getUrl = URL(string: getEndPoint) else {
        print("Erro na URL de requisição")
        return
    }

    /* RELEMBRANDO A SEQUÊNCIA DE OPERAÇÕES DO PARSE
       1 - URL
       2 - SESSION
       3 - DATA
       4 - RESUME
    */

    // 1-URL
    let request = URLRequest(url: getUrl)

    // 2-SESSION
    let urlSession = URLSession.shared

    // 3-DATA
    let task = urlSession.dataTask(with: request, completionHandler: {

        data, response, error in

            // Retorno do protocolo HTTP
            // response = 200
            guard let realResponse = response as? HTTPURLResponse,
                  realResponse.statusCode == 200 else {
```

```

        print("Erro na resposta protocolo HTTP")
        return
    }

    // Tudo OK - Vamos fazer o parse do Json
    do {
        if let ipString = data {
            print("IP retornado: \(ipString)")

            // PARSE
            let jsonDictionary = try
JSONSerialization.jsonObject(with: ipString, options: .mutableContainers)
as! NSDictionary

            let origem = jsonDictionary["origin"] as! String

            // Atualizar a Interface
            // Desta vez, usando Selector
            self.performSelector(onMainThread:
#selector(ViewController.updateIPLabel(_:)), with: origem, waitUntilDone:
false)

        }
    } catch {
        print("Erro no parse")
    }

})

// 4-RESUME
task.resume()

}

```

OBS: O selector **updateIPLabel** será codificado mais adiante.

Agora a função **enviarPOST()**:

```

func enviarPOST() {

    // Validação do Endpoint, com guard
    guard let postURL = URL(string: postEndPoint) else {
        print("Erro na URL de requisição POST")
        return
    }

    // Seguindo os passos
    var request = URLRequest(url: postURL)
    let urlSession = URLSession.shared

    // No caso do POST, temos que definir alguns parâmetros
    let postParams: [String: Any] = ["hello": "Caelum_BR was here" as
Any]

    // Mais alguns parâmetros de configuração
    request.httpMethod = "POST"
    request.setValue("application/json; charset=utf-8",
forHTTPHeaderField: "Content-Type")
    do {
        request.httpBody = try JSONSerialization.data(withJSONObject:
postParams, options: JSONSerialization.WritingOptions())

```

```

    } catch {
        print("Erro na chamada POST")
    }

    // Pegar o retorno do servidor
    let task = URLSession.dataTask(with: request) {

        data, response, error in

        // Retorno do protocolo HTTP
        // response = 200
        guard let realResponse = response as? HTTPURLResponse,
        realResponse.statusCode == 200 else {
            print("Erro na resposta protocolo HTTP")
            return
        }

        // Parsear o JSON de resposta
        if let postString = String(data: data!, encoding:
String.Encoding.utf8) {

            // Atualizar a interface
            self.performSelector(onMainThread:
#selector(ViewController.updatePostLabel(_:)), with: postString,
waitUntilDone: false)
        }

        // Executar o comando
        task.resume()
    }

```

OBS: O selector **updatePostLabel** será codificado a seguir.

## Os Selectors

E por último iremos codificar os *selectors* das funções GET e POST, que têm simplesmente a finalidade de atualizar a tela com o resultado das requisições de rede:

```

// MARK: - Selectors

// Selector do GET
func updateIPLabel(_ text: String) {
    self.labelResultado.text = "GET - Seu IP é " + text
}

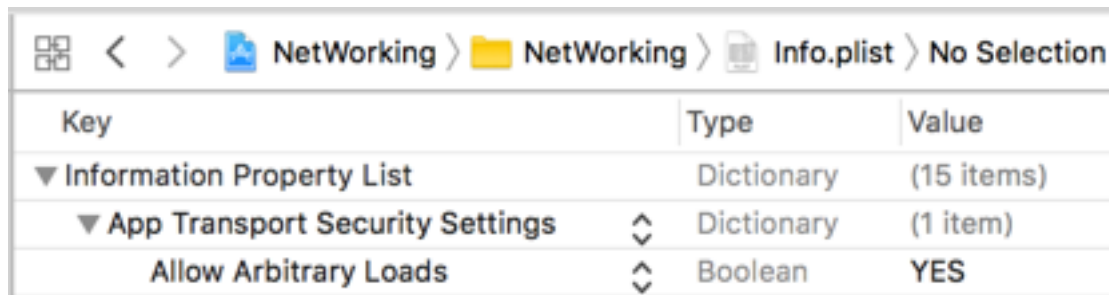
// SELECTOR do POST
func updatePostLabel(_ text: String) {
    self.labelResultado.text = "POST - " + text
}

```

# Liberação de Acesso

Como os servidores que estamos trabalhando não possuem conexão segura (são HTTP ao invés de HTTPS), será necessário forçar a liberação de acesso, para que o nosso aplicativo possa executar as conexões.

Portanto temos que editar o `info.plist` e adicionar a seguinte entrada:



The image shows a screenshot of a plist editor window. The breadcrumb path at the top is: `NetWorking > NetWorking > Info.plist`. The table below lists the contents of the plist file.

Key	Type	Value
▼ Information Property List	Dictionary	(15 items)
▼ App Transport Security Settings	Dictionary	(1 item)
Allow Arbitrary Loads	Boolean	YES