

# 1. Installer Postman

## introduction

Ces informations seront utiles aux développeurs qui souhaitent utiliser l'ensemble de fonctionnalités du site Web forismatic dans leurs propres logiciels.

Les appels de méthode API sont implémentés sous forme de requêtes HTTP à l'URL <http://api.forismatic.com/api/1.0/>. Les paramètres de requête sont transmis à l'aide de la méthode POST ou GET (URL-encoded). Le format de données de retour du serveur est défini par le paramètre de requête. Les formats de réponse suivants sont pris en charge:

- `xml`
- `json`
- `jsonp`
- `html`
- `texte`

## Méthode getQuote

Sélectionne un devis aléatoire à l'aide de la clé numérique transmise, si la clé n'est pas spécifiée, le serveur génère une clé aléatoire. La clé influence le choix de l'offre. Paramètres de requête:

- `method = getQuote` - nom de la méthode à invoquer
- `format = <format>` - l'un des formats de réponse pris en charge par le serveur
- `key = <integer>` - clé numérique, qui influence le choix de la citation, la longueur maximale est de 6 caractères
- `lang = <string>` - langue de réponse ("ru" ou "en")
- `jsonp = <string>` - nom de la fonction de rappel, utilisé pour le format jsonp uniquement ([exemple d'utilisation](#))

Exemple de requête:

```
POST:  
method = getQuote & key = 457653 & format = xml & lang = fr  
  
response:  
<forismatic>  
  <quote>  
    <quoteText> Brevity - l'âme de l'esprit </quoteText>  
    <quoteAuthor> </quoteAuthor>  
    <senderName> nom ou surnom de l'expéditeur du devis </senderName>  
    <senderLink> adresse e-mail ou site Web de l'expéditeur du devis </senderLink>  
  </quote>  
</forismatic>
```

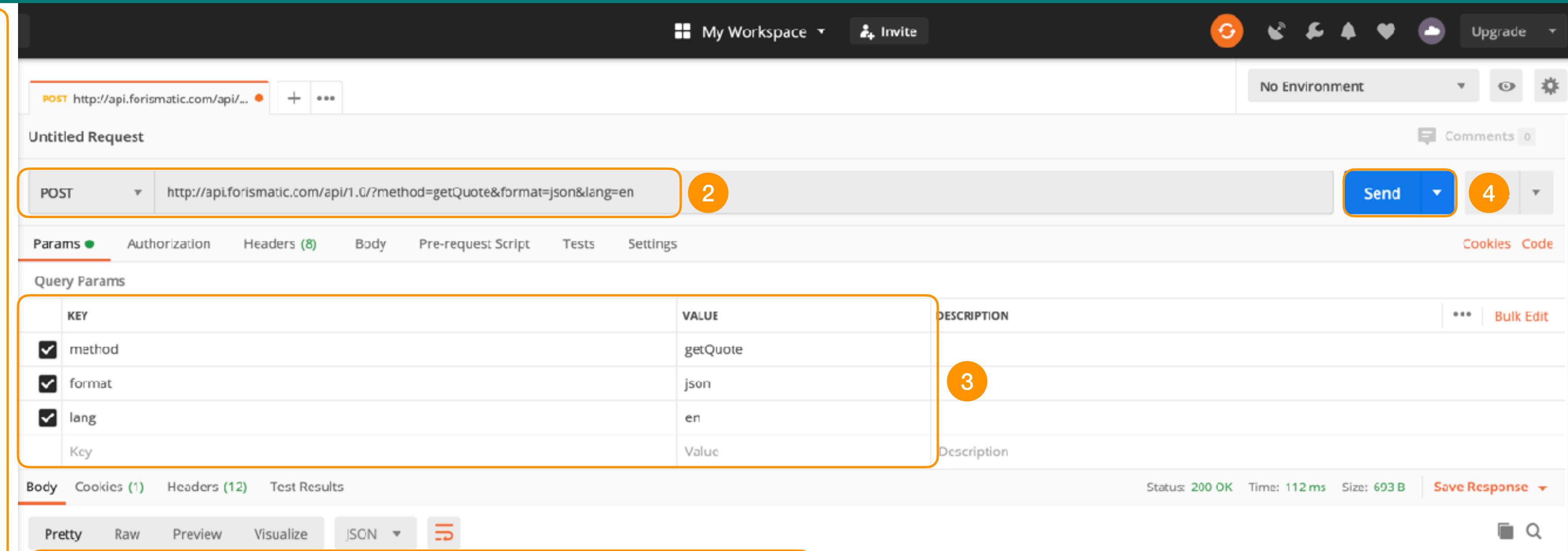
- [Programme utilitaire en ligne pour envoyer et tester les demandes d'API](#)
- [Bibliothèque ActionScript3 pour travailler avec l'API Forismatic](#)
- [Bibliothèque Forismatic Python par Andrey Basalyha](#)

1

1. Postman est un logiciel très pratique, car il permet de lancer des requêtes sur les URL de votre choix en indiquant tous les paramètres que vous voulez et sans écrire une seule ligne de code.

L'intérêt, c'est que cela vous permet de tester le comportement d'une API sans savoir à coder les appels réseau avant.

Ce prétravail vous fera gagner beaucoup de temps ensuite lorsque vous développerez dans l'application.



The screenshot shows the Postman interface with a POST request to `http://api.forismatic.com/api/1.0/?method=getQuote&format=json&lang=en`. The 'Query Params' table is highlighted with a red box and labeled '3'. The table contains three rows: 'method' with value 'getQuote', 'format' with value 'json', and 'lang' with value 'en'. The 'Body' tab shows the JSON response: 

```
1 {  
2   "quoteText": "Always be mindful of the kindness and not the faults of others.",  
3   "quoteAuthor": "Buddha",  
4   "senderName": "",  
5   "senderLink": "",  
6   "quoteLink": "http://forismatic.com/en/cfd91e85f8/"  
7 }
```

 The response body is highlighted with a red box and labeled '5'.

2. On fait copier / coller le lien

3. Dans le paramètre, on écrit les trois: méthode, format et langue

*Méthode* permet de récupérer la citation avec la valeur `getQuote`.

*Format* : le type de format que l'on souhaite pour la réponse. La documentation précise les différents formats disponibles: on choisit `json`. C'est un format de donnée très populaire dans le développement mobile.

*Langue* : on choisit l'anglais pour coder

4. On valide sur Send.

5. Les codes apparaissent automatiquement.

## 2. Découvrir URLSession

### 1 URLSession

#### 2 URLSessionConfiguration

Cookies ?  
TimeOut ?  
Header ?  
Cellulaire ?  
Wifi ?  
Cache ?

Lancer

Lancer

Lancer

URLSessionTask

GET

Htts: api.monsite.com/user

URLSessionTask

POST

Htts: api.twitter.com/tweet

URLSessionTask

GET

Htts: api.monsite.com/lune.gif

### 3

#### 1. Présentation d'URLSession

Quand on parle de URLSession, on parle en même temps de la classe et de la suite de classes correspondant. En effet il y a trois classe qui fonctionnent avec URLSession:

URLSession

URLSessionConfiguration

URLSessionTask

#### 2. Configuration d'URLSession

URLSession s'initialise avec URLSessionConfiguration. C'est la classe qui permet de configurer la session. Lors de la configuration, on essaie de répondre à ce genre de questions:

- Accepte-t-on les cookies ?
- Quelle est la politique de cache ?
- Quel est le timeout d'une requête ?
- Doit-on rajouter des éléments dans le reader ?
- Etc...

#### 3. Lancement d'une tâche

Une fois que URLSession est initialisé avec la bonne configuration, il peut lancer une tâche. Une tâche lance une requête réseau, reçoit la réponse, décide de l'interrompre. Cela est géré par la classe URLSessionTask. Elle est responsable de la gestion du cycle de vie de la requête.

### 3. Récupérer la citation avec requête POST

```
12 class QuoteService {
13
14     // A) CREATION DE LA REQUETE
15     private static let quoteURL = URL(string: "http://api.forismatic.com/api/1.0/")
16
17     // on cree une fonction statique qu'on va appeler getQuote et à l'intérieur de laquelle on va créer la requete
18     static func getQuote() {
19
20         // B) INITIALISATION
21         /// 1) On initialise une instance URLRequest en lui passant URL en paramètre
22         var request = URLRequest(url: quoteURL)
23
24         /// 2) Ensuite, on précise la méthode http choisie (en l'occurrence POST) avec la propriété httpMethod de URLRequest
25         request.httpMethod = "POST"
26
27         /// 3) On a besoin de passer des paramètres dans cette requête. Donc on va ajouter avec la propriété body de URLRequest
28         let body = "method=getQuote&format=json&lang=en"
29
30         /// 4) On écrit les paramètres sous la forme d'une chaîne de caractères.
31         /// on utilise la méthode data(using: String) qui permet de convertir une chaîne de caractères vers format Data.
32         /// on précise l'encodage en l'occurrence utf8
33         request.httpBody = body.data(using: .utf8)
34
35
36     // C) LANCEMENT DE LA TACHE
37
38     /// 1) Il faut une instance d URLSession. Pour cela, on va créer cette instance avec une configuration par défaut.
39     let session = URLSession(configuration: .default)
40
41     /// 2) Ensuite on va créer une tâche et plus précisément une instance de URLSessionDataTask
42     /// on crée une tâche qu'on stocke dans la constante task: let task
43     /// on utilise la méthode dataTask de URLSession.
44     /// elle prend en paramètre (with: request) -> c'est request de type URLRequest
45     /// elle admet deuxième paramètre fermeture {...}
46     /// dans cette fermeture qu'on va gérer la réponse de l'appel data, response, error
47     let task = session.dataTask(with: request) { (data, response, error) in
48
49         // D) GESTION DE LA REPONSE
50
51         /// 1) Lorsque l'on gère la réponse, on va vérifier qu'elle contient bien les données et les erreurs.
52         if let data = data, error == nil {
53
54             /// 2) Ensuite on va contrôler que le statusCode de la réponse est bien 200
55             /// on vérifie d'abord que la réponse est bien du type HTTPURLResponse et le code vaut 200
56             if let response = response as? HTTPURLResponse, response.statusCode == 200 {
57
58
59             // E) LECTURE DU JSON
60
61                 /// 1) JSONDecoder permet de convertir le format JSON vers un dictionnaire SWIFT
62                 /// elle a méthode decode de JSONDecoder qui renvoie un dictionnaire et prend en paramètre deux choses: String et Self
63                 /// String pour la clé et pour la valeur
64                 /// Self permet de faire référence au type
65                 /// les data reçus en réponse à l'appel réseau et d'éventuelles options qu'on laisse vides
66                 /// c'est quoi try? certains fonctions comme fonction decode, peuvent planter et dans ce cas, on essaie de lancer la fonction avec try?
67                 if let responseJSON = try? JSONDecoder().decode([String: String].self, from: data),
68
69                     /// 2) On met des print sur les variables text et author, pour afficher et voir les citations dans la console en appuyant sur le bouton New Quote dans le simulateur
70                     let text = responseJSON["quoteText"],
71                     let author = responseJSON["quoteAuthor"] {
72                         print(text)
73                         print(author)
74                     }
75
76                 }
77             }
78             /// 3) Maintenant la tâche est créée et on va lancer l'appel
79             task.resume()
80         }
81     }
82 }
```

The code is annotated with five numbered callouts:

- 1: Surrounds the creation of the URL and the static function declaration.
- 2: Surrounds the initialization of the URLRequest, specifying the method as POST.
- 3: Surrounds the preparation of parameters for the request.
- 4: Surrounds the creation of the URLSession and the start of the data task.
- 5: Surrounds the decoding of the JSON response and printing the results.

#### 4. Télécharger l'image avec une requête GET



1

```
12 class QuoteService {
13
14     //A) CREATION DE LA REQUETE
15     private static let quoteURL = URL(string: "http://api.forismatic.com/api/1.0/")
16
17     //On ajoute le lien picture
18     private static let pictureURL = URL(string: "https://source.unsplash.com/random/1000x1000")!
19
20     static func getQuote() {
21
22         // B) INITIALISATION
23         var request = URLRequest(url: quoteURL)
24         request.httpMethod = "POST"
25         let body = "method=getQuote&format=json&lang=en"
26         request.httpBody = body.data(using: .utf8)
27
28         //C) LANCEMENT DE LA TACHE
29         let session = URLSession(configuration: .default)
30         let task = session.dataTask(with: request) { (data, response, error) in
31
32             //D) GESTION DE LA REPONSE
33             if let data = data, error == nil {
34                 if let response = response as? HTTPURLResponse, response.statusCode == 200 {
35
36                     //E) LECTURE DU JSON
37                     if let responseJSON = try? JSONDecoder().decode([String: String].self, from: data),
38                         let text = responseJSON["quoteText"],
39                         let author = responseJSON["quoteAuthor"] {
40
41                         // Pour modifier, on appelle la fonction getImage une fois qu'on recu la citation
42                         getImage{ (data) in
43                             print(data)
44                             print(author)
45                             print(text)
46
47                         }
48
49                     }
50
51                 }
52             }
53             task.resume()
54
55         }
56
57         // Ensuite, on va lancer un appel dans une fonction getImage
58         private static func getImage(completionHandler: @escaping (Data?) -> Void) {
59
60             //On cree URLSession avec la configuration par defaut
61             let session = URLSession(configuration: .default)
62
63             // On cree une tache avec methode dataTask et URLSession. Ensuite on ajoute l'appel avec task.resume
64             let task = session.dataTask(with: pictureURL) { (data, response, error) in
65
66                 //Gestion de la reponse
67                 if let data = data, error == nil {
68                     if let response = response as? HTTPURLResponse, response.statusCode == 200 {
69                         //On va ajouter une fermeture en parametre de la func getImage, on va nommer completionHandler
70                         completionHandler(data)
71
72                     }
73
74                 }
75             }
76             task.resume()
77
78         }
79
80     }
81
82 }
```

2

3

## 5. Utiliser le résultat de la requête

```
//  
// Quote.swift          Creation du model Quote  
// ClassQuote  
//  
// Created by PARISATO on 11/06/2020.  
// Copyright © 2020 OpenClassrooms. All rights reserved.  
  
import Foundation  
  
struct Quote {  
    var text: String  
    var auteur: String  
    var imageData: Data  
}
```

1

Les callbacks sont très simples. Le contrôleur va appeler une fonction du modèle, et il va indiquer, en paramètre de la fonction, l'action à effectuer lorsque la réponse est reçue. Et pour cela, on va simplement utiliser les fermetures.

```
//1) On commence modifier la déclaration de func getQuote pour ajouter le parametre callback  
// un boolean sucess qui permet de savoir si l'appel a réussi ou non  
// un objet quote qui est l'objet qu'on a récupéré et construit avec les requetes  
// on utilise en parametre de fermeture un boolean et un objet quote optionnel car si la requete échoue, on n'a pas d'objet à renvoyer.  
static func getQuote(callback: @escaping (Bool, Quote?) -> Void) {
```

2

```
    var request = URLRequest(url: quoteURL)  
    request.httpMethod = "POST"  
    let body = "method=getQuote&format=json&lang=en"  
    request.httpBody = body.data(using: .utf8)  
  
    let session = URLSession(configuration: .default)  
    let task = session.dataTask(with: request) { (data, response, error) in  
        if let data = data, error == nil {  
            if let response = response as? HTTPURLResponse, response.statusCode == 200 {  
                if let responseJSON = try? JSONDecoder().decode([String: String].self, from: data),  
                    let text = responseJSON["quoteText"],  
                    let author = responseJSON["quoteAuthor"] {  
                    getImage { (data) in  
                        if let data = data {  
  
                            //2) Pour envoyer le callback, on va commencer par créer l'objet quote à partir des trois données reçues: author, text, data  
                            let quote = Quote(text: text, auteur: author, imageData: data)  
  
                            //3) On envoie le callback. On passe le boolean à true car on est dans le cas où la requete a réussi. On passe l'objet quote qu'on vient de créer.  
                            callback(true, quote)  
  
                            //4) On fait plein de vérification, on rajoute des else a tous ces if et on va envoyer un callback d'échec.  
                        } else {  
                            callback(false, nil)  
                        }  
                    }  
                } else {  
                    callback(false, nil)  
                }  
            } else {  
                callback(false, nil)  
            }  
        }  
        task.resume()  
    }
```

Envoie du callback

```
private static func getImage(completionHandler: @escaping (Data?) -> Void) {  
    let session = URLSession(configuration: .default)  
    let task = session.dataTask(with: pictureURL) { (data, response, error) in  
  
        //Si le téléchargement échoue, on ne peut pas renvoyer de données, donc on renvoie nil. Cette information utilisée ensuite dans la fonction changeQuote pour renvoyer les bons parametres dans le callback.  
        if let data = data, error == nil {  
            if let response = response as? HTTPURLResponse, response.statusCode == 200 {  
                completionHandler(data)  
            } else {  
                completionHandler(nil)  
            }  
        } else {  
            completionHandler(nil)  
        }  
        task.resume()  
    }
```

3

4

ClassQuote > iPhone SE (2nd generation) ClassQuote | Build ClassQuote: Succeeded | Today at 01:15 1

Réception des callback

ClassQuote ClassQuote ClassQuote Controller ViewController.swift No Selection

ClassQuote

ClassQuote

AppDelegate.swift

Info.plist

Model

QuoteService.swift

Quote.swift

Controller

ViewController.swift

View

Main.storyboard

Assets.xcassets

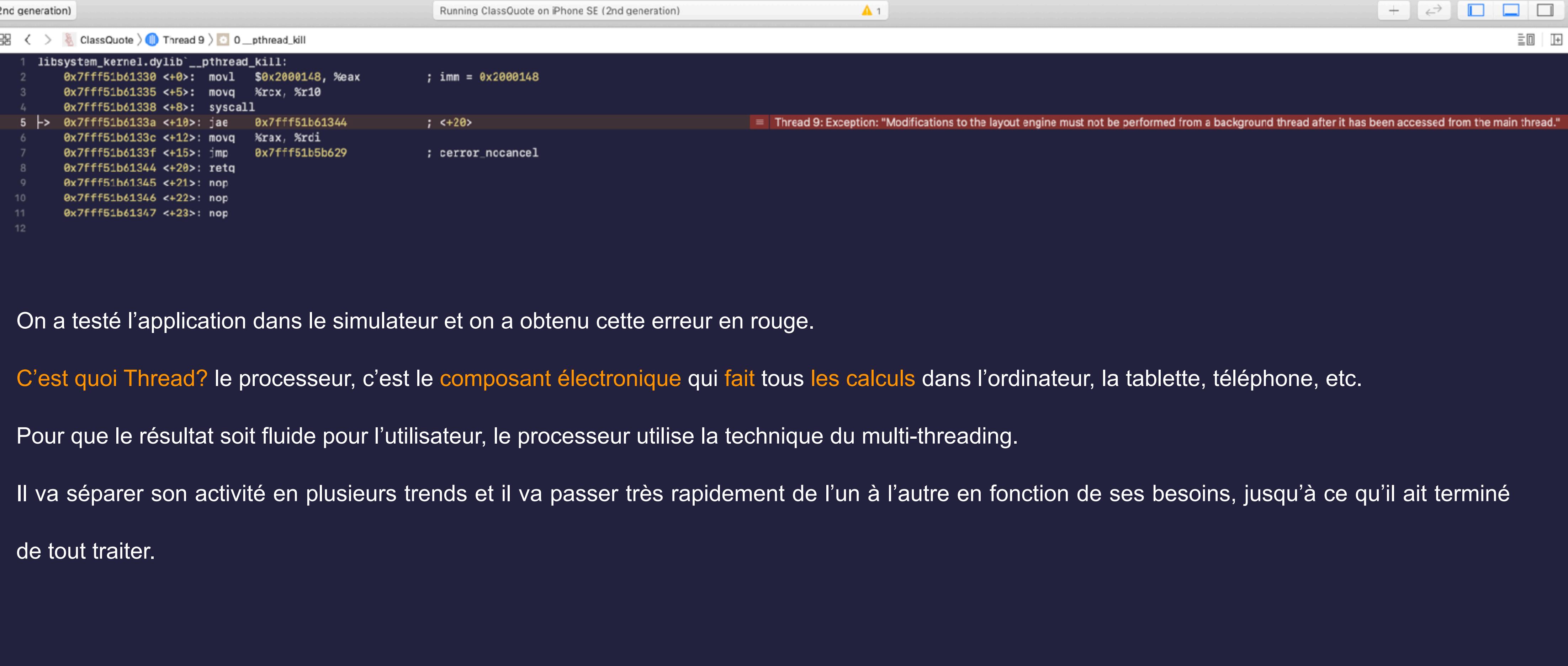
LaunchScreen.storyboard

Products

```
8
9 import UIKit
10
11 class ViewController: UIViewController {
12
13     @IBOutlet weak var imageView: UIImageView!
14     @IBOutlet weak var quoteLabel: UILabel!
15     @IBOutlet weak var authorLabel: UILabel!
16     @IBOutlet weak var newQuoteButton: UIButton!
17     @IBOutlet weak var activityIndicator: UIActivityIndicatorView!
18
19     override func viewDidLoad() {
20         super.viewDidLoad()
21         addShadowToQuoteLabel()
22     }
23
24     private func addShadowToQuoteLabel() {
25         quoteLabel.layer.shadowColor = UIColor.black.cgColor
26         quoteLabel.layer.shadowOpacity = 0.9
27         quoteLabel.layer.shadowOffset = CGSize(width: 1, height: 1)
28     }
29
30     @IBAction func tappedNewQuoteButton() {
31         QuoteService.getQuote { [weak self] (success, quote) in
32             if success, let quote = quote {
33                 //Afficher la citation
34                 self?.update(quote: quote)
35             } else {
36                 //Présenter un message d'erreur
37                 self?.presentAlert()
38             }
39         }
40     }
41
42     private func update(quote: Quote) {
43         quoteLabel.text = quote.text
44         authorLabel.text = quote.auteur
45         imageView.image = UIImage(data: quote.imageData)
46     }
47
48     private func presentAlert() {
49         let alertVC = UIAlertController(title: "Error", message: "The quote download failed.", preferredStyle: .alert)
50         alertVC.addAction(UIAlertAction(title: "OK", style: .cancel, handler: nil))
51         present(alertVC, animated: true, completion: nil)
52     }
53 }
54
```

5

## 6. Débloquer l'interface avec le multi-threading

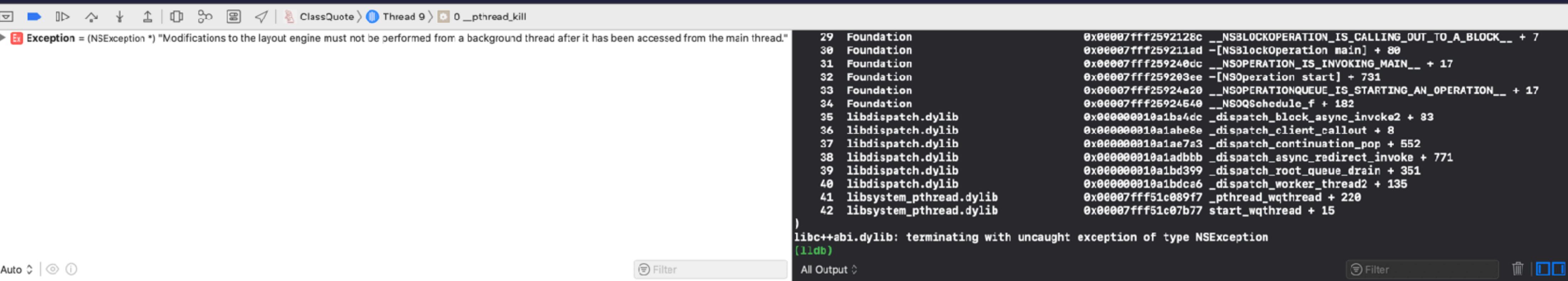


On a testé l'application dans le simulateur et on a obtenu cette erreur en rouge.

C'est quoi Thread? le processeur, c'est le composant électronique qui fait tous les calculs dans l'ordinateur, la tablette, téléphone, etc.

Pour que le résultat soit fluide pour l'utilisateur, le processeur utilise la technique du multi-threading.

Il va séparer son activité en plusieurs treads et il va passer très rapidement de l'un à l'autre en fonction de ses besoins, jusqu'à ce qu'il ait terminé de tout traiter.



▶ ClassQuote > iPhone SE (2nd generation)

Running ClassQuote on iPhone SE

< > A ClassQuote > ClassQuote > Model > QuoteService.swift > No Selection

```
10
11 class QuoteService {
12
13     private static let quoteURL = URL(string: "http://api.forismatic.com/api/1.0/")!
14     private static let pictureURL = URL(string: "https://source.unsplash.com/random/1000x1000")!
15
16     static func getQuote(callback: @escaping (Bool, Quote?) -> Void) {
17         var request = URLRequest(url: quoteURL)
18         request.httpMethod = "POST"
19
20         let body = "method=getQuote&format=json&lang=en"
21         request.httpBody = body.data(using: .utf8)
22
23         let session = URLSession(configuration: .default)
24         let task = session.dataTask(with: request) { (data, response, error) in
25             DispatchQueue.main.async {
26                 if let data = data, error == nil {
27                     if let response = response as? HTTPURLResponse, response.statusCode == 200 {
28                         if let responseJSON = try? JSONDecoder().decode([String: String].self, from: data),
29                             let text = responseJSON["quoteText"],
30                             let author = responseJSON["quoteAuthor"] {
31                             getImage { (data) in
32                                 if let data = data {
33
34                                     let quote = Quote(text: text, auteur: author, imageData: data)
35                                     callback(true, quote)
36                                 } else {
37                                     callback(false, nil)
38                                 }
39                             }
40                         } else {
41                             callback(false, nil)
42                         }
43                     } else {
44                         callback(false, nil)
45                     }
46                 } else {
47                     callback(false, nil)
48                 }
49             }
50         }
51         task.resume()
52     }
53
54     private static func getImage(completionHandler: @escaping (Data?) -> Void) {
55         let session = URLSession(configuration: .default)
56         let task = session.dataTask(with: pictureURL) { (data, response, error) in
57             DispatchQueue.main.async {
58                 if let data = data, error == nil {
59                     if let response = response as? HTTPURLResponse, response.statusCode == 200 {
60                         completionHandler(data)
61                     } else {
62                         completionHandler(nil)
63                     }
64                 } else {
65                     completionHandler(nil)
66                 }
67             }
68         }
69         task.resume()
70     }
71 }
```

Les queues sont des objets qui permettent au développeur de repartir des tâches sur plusieurs files différentes, notamment dans le but de ne pas bloquer l'application lorsqu'une tâche est trop longue.

Pour revenir dans la Main Queue, on écrit *DispatchQueue.main.async*

1. Pour gérer les queues avec Swift, on utilise une classe qui s'appelle `DispatchQueue`. Elle permet de revenir dans la Main Queue. Pour revenir dans la Main Queue, on écrit `DispatchQueue.main.async`. On va mettre tous les codes callback à l'intérieur de cet appel `DispatchQueue`.
  2. Dans la fonction `getImage`, on écrit la même chose avec `DispatchQueue` et on met les codes à l'intérieur de cet appel.

## 7. Gérer les requêtes concurrentes avec le singleton pattern

## 8. Appréhender la difficulté de tester un appel réseau

```
ClassQuote < > ClassQuoteTests > Test.swift > No Selection
1
2
3 import Foundation
4
5 func testDownloadWebData() {
6
7     // On crée l'expectation en lui donnant une simple description.
8     let expectation = XCTestExpectation(description: "Télécharge le site openclassrooms.com")
9
10    // On prépare une requête
11    let url = URL(string: "https://openclassrooms.com")!
12    let dataTask = URLSession(configuration: .default).dataTask(with: url) { (data, _, _) in
13
14        // On vérifie qu'il y a bien des données qui ont été chargées, c'est ici que le test a lieu.
15        XCTAssertNotNil(data)
16
17        // On déclare que l'expectation est terminée, on peut clore le test.
18        expectation.fulfill()
19    }
20
21    // On lance la requête.
22    dataTask.resume()
23
24    // On attend que l'expectation soit terminée, avec une durée maximum de 10 secondes.
25    wait(for: [expectation], timeout: 10.0)
26 }
27
```

Les **expectations** sont une mécanique présente dans le framework **XCTest** qui permet justement d'attendre un certain délai avant de terminer l'**exécution du test**.

### Voici les étapes de l'expectation :

1. On crée l'expectation
2. Ensuite, on va créer ici une requête
3. Ici, le test a effectivement lieu
4. Enfin, on va tout simplement dire que l'expectation est terminée
5. On lance la requête
6. On attend que l'expectation soit terminée avec une durée maximum de 10 secondes

Cette méthode est simple et a l'avantage de fonctionner tout de suite sans modification du code de l'application.

Mais elle a un défaut: chaque test va mettre en 2 et 1 à secondes selon la qualité du réseau à s'exécuter.

Le but des tests est de protéger le code et éviter l'apparition de bugs.

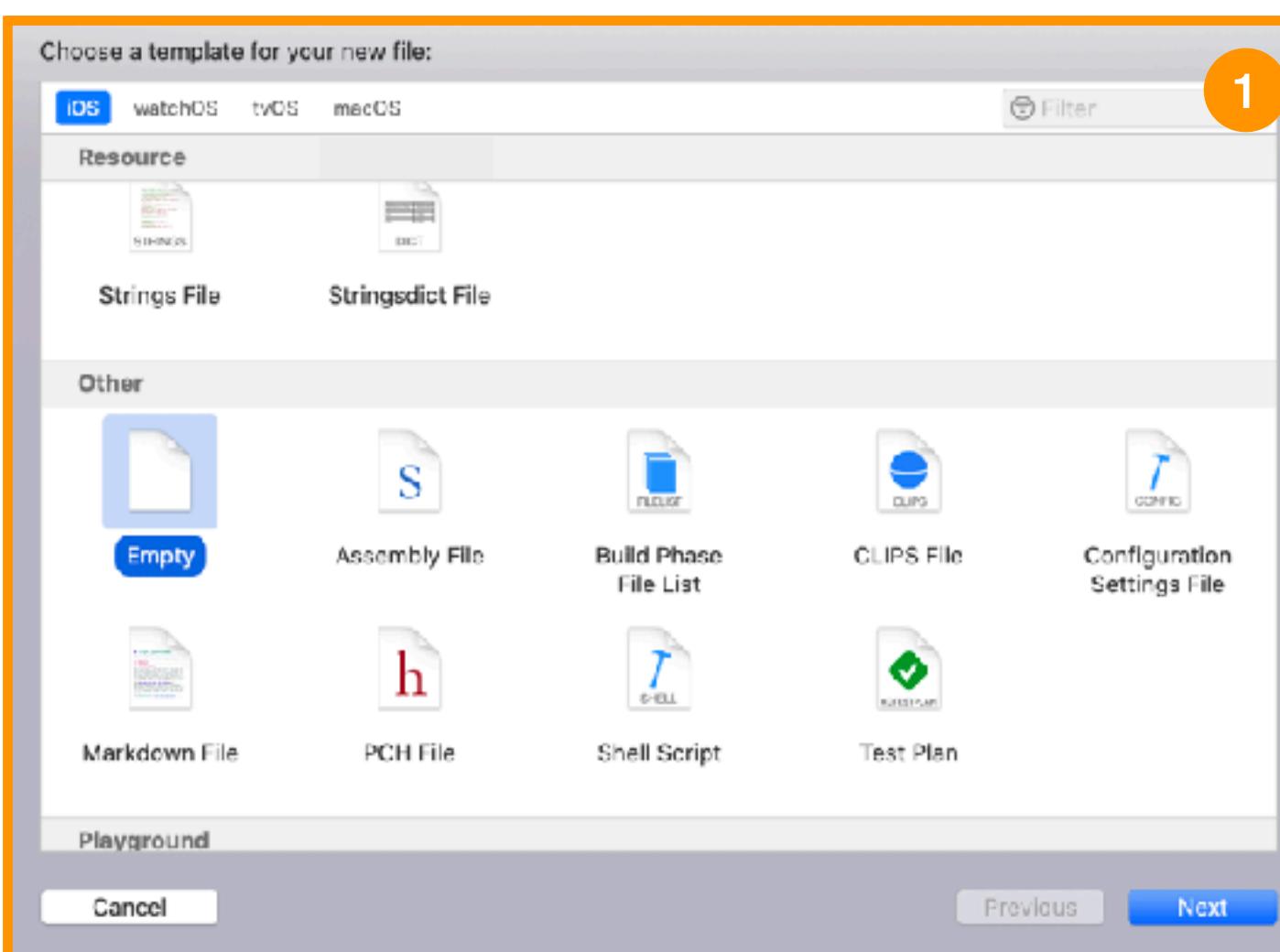
## 9. Créer le jeu de donnée de test

The screenshot shows the Xcode project structure. The left sidebar lists the project files: ClassQuote (Info.plist, Supporting Files, Model, View, Controller), ClassQuoteTests (ClassQuoteTests.swift, Info.plist, Quote.json), ClassQuoteUITests (ClassQuoteUITests.swift, Info.plist), and Products. The Quote.json file is selected in the center editor area. The code content is:

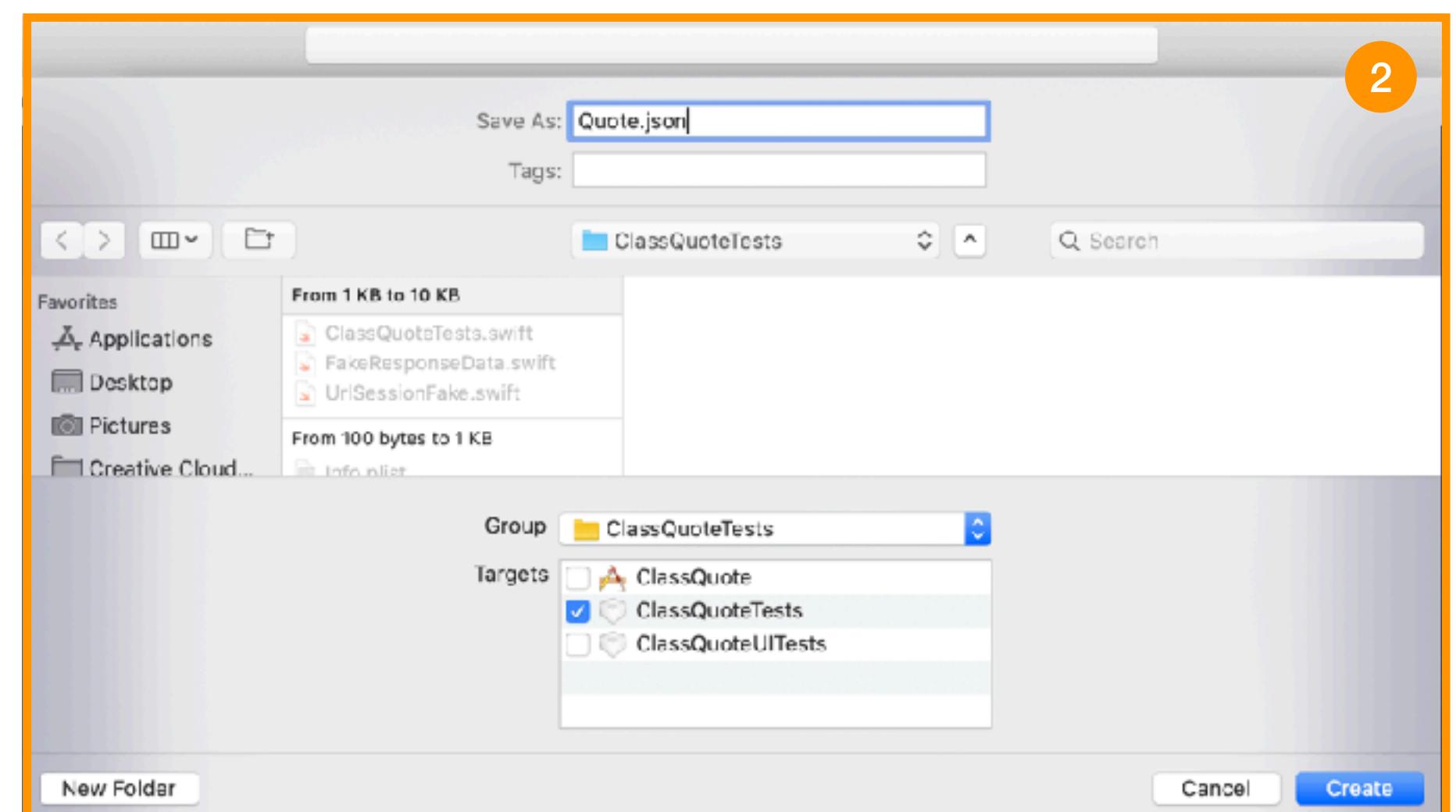
```
1
2 {
3     "quoteText": "Be not afraid of greatness: some are born great, some achieve greatness, and some have greatness thrust upon them. ",
4     "quoteAuthor": "William Shakespeare",
5     "senderName": "",
6     "senderLink": "",
7     "quoteLink": "http://forismatic.com/en/c49351b049/"
8 }
```

A yellow circle with the number 4 is in the top right corner of the editor.

4. On colle la réponse récupérée dans ce fichier Quote.json



1. On crée un nouveau fichier et on choisit Empty



2. On nomme le Quote.json et on sauvegarde le du coté des tests

The screenshot shows the Postman interface. A POST request is made to <http://api.forismatic.com/api/1.0/?method=getQuote&format=json&lang=en>. The response body is displayed in JSON format:

```
1 {
2     "quoteText": "I think somehow we learn who we really are and then live with that decision. ",
3     "quoteAuthor": "Eleanor Roosevelt",
4     "senderName": "",
5     "senderLink": "",
6     "quoteLink": "http://forismatic.com/en/1ab1e78827/"
7 }
```

A yellow circle with the number 3 is in the top right corner of the Postman window.

3. On va Postman et on fait **copier / coller** ceci en bleu

ClassQuote > iPhone SE (2nd generation)

ClassQuoteTests > FakeResponseData.swift > No Selection

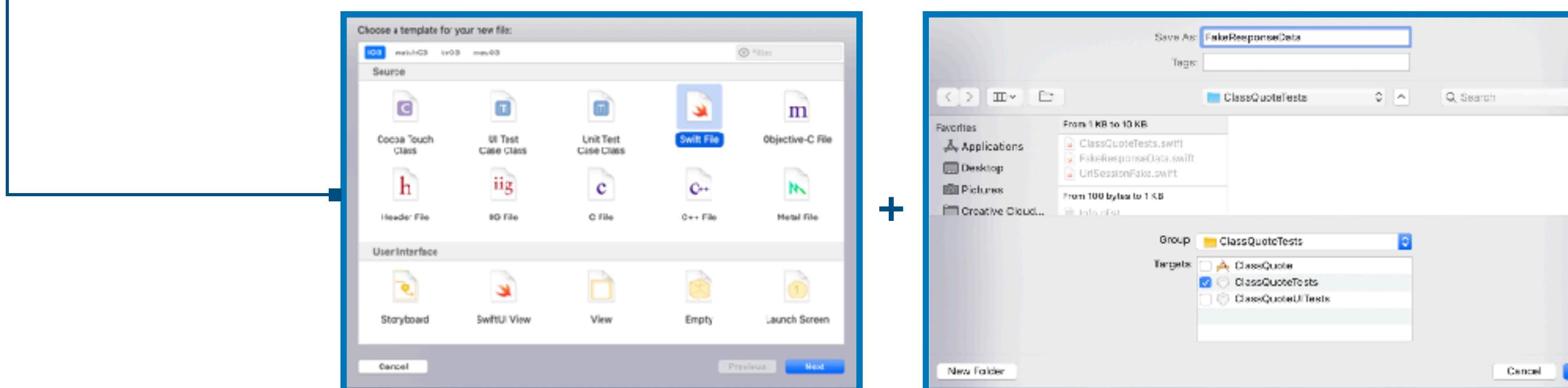
```

1
2
3 import Foundation
4
5 class FakeresponseData {
6     // MARK: - Data
7     static var quoteCorrectData: Data? {
8         let bundle = Bundle(for: FakeresponseData.self)
9         let url = bundle.url(forResource: "Quote", withExtension: "json")!
10        return try! Data(contentsOf: url)
11    }
12
13    static let quoteIncorrectData = "erreur".data(using: .utf8)!
14
15    static let imageData = "image".data(using: .utf8)!
16
17    // MARK: - Response
18    static let responseOK = HTTPURLResponse(
19        url: URL(string: "https://openclassrooms.com")!,
20        statusCode: 200, httpVersion: nil, headerFields: [:]!
21    )
22
23    static let responseKO = HTTPURLResponse(
24        url: URL(string: "https://openclassrooms.com")!,
25        statusCode: 500, httpVersion: nil, headerFields: [:]!
26
27    // MARK: - Error
28    class QuoteError: Error {}
29    static let error = QuoteError()
30
31
32
33

```

La réponse contient 3 paramètres data, response et error. Pour simuler la réponse des 2 APIs, on va avoir besoin de simuler ces 3 paramètres pour chaque appel.

1. Pour simuler **response**, on va créer deux instances de `HTTPURLResponse`:
  - Une qui a pour code 200, c'est le cas où tout va bien
  - Une avec le code 500, quand ça ne fonctionne pas
2. Pour simuler **error**, on ne s'intéresse pas à l'erreur en elle-même mais seulement à la présence ou non d'une erreur. Donc on va juste créer une erreur tout simple.
3. Pour simuler **data**, on va simuler trois données différentes:
  - a) Simuler le json renvoyé par **Forismatic** : *pour récupérer les données du fichier*
  - b) Simuler un JSON endommagé : *la méthode renvoie une valeur de type data*
  - c) Simuler les données de l'image : on vérifie les données dans le model



On va créer une classe qu'on va appeler **FakeresponseData** dont le rôle va être de gérer les données de test.