Semaine 5 : Réseautage et APIs dans le développement iOS

Oussama ELMIR

February 11, 2025

1 Introduction au réseautage dans iOS

Le réseautage est une partie fondamentale des applications iOS modernes, leur permettant de communiquer avec des serveurs et de récupérer des données dynamiquement. Cette semaine, nous allons couvrir :

- Effectuer des requêtes HTTP avec `URLSession`
- Analyser du JSON en utilisant le protocole `Codable` de Swift
- Afficher les données d'une API dans un `UITableView`
- Gérer les erreurs et optimiser les performances

2 Effectuer des requêtes API avec URLSession

2.1 Comprendre les requêtes HTTP

Les APIs communiquent généralement en utilisant les méthodes HTTP suivantes :

- **GET** - Récupérer des données depuis un serveur
- **POST** - Envoyer des données à un serveur
- **PUT** - Mettre à jour des données existantes
- **DELETE** - Supprimer des données depuis un serveur

2.2 Requête API de base en Swift

Voici un exemple simple d'une requête GET en utilisant `URLSession` :

```swift
import UIKit

struct Post: Codable {
    let userId: Int
    let id: Int
    let title: String
    let body: String
}

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        fetchPosts()
    }

    func fetchPosts() {
        let urlString = "https://jsonplaceholder.typicode.com/posts"
        guard let url = URL(string: urlString) else { return }

        let task = URLSession.shared.dataTask(with: url) { data, response, error in
            if let error = error {
                print("Erreur lors de la récupération des données: \(error)")
                return
            }
            guard let data = data else { return }

            do {
                let posts = try JSONDecoder().decode([Post].self, from: data)
                print(posts) // Affiche les données récupérées
            } catch {
                print("Échec du décodage JSON: \(error)")
            }
        }
        task.resume()
    }
}
```

3 Analyser JSON et afficher les données dans un UITableView

Pour analyser les données JSON, nous utilisons le protocole `Codable` de Swift. La structure suivante correspond aux champs JSON :

```swift
struct Post: Codable {
    let userId: Int
    let id: Int
    let title: String
    let body: String
}
```

3.1 Mise à jour du Table View

Pour afficher les données API dans un tableau :

```
class PostsViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {
    @IBOutlet weak var tableView: UITableView!
    var posts: [Post] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        tableView.delegate = self
        tableView.dataSource = self
        fetchPosts()
    }

    func fetchPosts() {
        let urlString = "https://jsonplaceholder.typicode.com/posts"
        guard let url = URL(string: urlString) else { return }

        URLSession.shared.dataTask(with: url) { data, response, error in
            if let error = error {
                print("Erreur : \(error)")
                return
            }
            guard let data = data else { return }

            do {
                let decodedData = try JSONDecoder().decode([Post].self, from: data)
                DispatchQueue.main.async {
                    self.posts = decodedData
                    self.tableView.reloadData()
                }
            } catch {
                print("Erreur de d codage : \(error)")
            }
        }.resume()
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return posts.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "PostCell", for: indexPath)
        cell.textLabel?.text = posts[indexPath.row].title
        return cell
    }
}
```

4 Bonnes pratiques pour le réseautage dans iOS

4.1 Gestion des erreurs et codes de statut

Utilisez les codes de réponse HTTP pour gérer correctement les erreurs :

```
if let httpResponse = response as? HTTPURLResponse {
    switch httpResponse.statusCode {
    case 200...299:
        print("Succ s")
    case 400...499:
        print("Erreur␣client␣:␣\(httpResponse.statusCode)")
    case 500...599:
        print("Erreur␣serveur␣:␣\(httpResponse.statusCode)")
    default:
        print("Erreur␣inconnue")
    }
}
```

4.2 Utilisation d'Alamofire pour simplifier les requêtes API

Alamofire est une bibliothèque populaire pour le réseautage en Swift :

```
import Alamofire

AF.request("https://jsonplaceholder.typicode.com/posts").
    responseDecodable(of: [Post].self) { response in
    switch response.result {
    case .success(let posts):
        print(posts)
    case .failure(let error):
        print(" chec ␣de␣la␣requ te␣:␣\(error)")
    }
}
```