let url = URL(string: "https://example.com/api/resource")!

let task = URLSession.shared.dataTask(with: url) { data, response, error in

guard let data = data, error == nil else {

print(error ?? "Unknown error")

return

}

// Process the data

}

task.resume()

var request = URLRequest(url: URL(string: "https://example.com/api/resource")!)

request.httpMethod = "POST"

request.setValue("application/json", forHTTPHeaderField: "Content-Type")

let body: [String: Any] = ["name": "New Item", "value": 123]

let jsonData = try? JSONSerialization.data(withJSONObject: body)

request.httpBody = jsonData

let task = URLSession.shared.dataTask(with: request) { data, response, error in

// Handle response here

}

task.resume()

func retryRequest(with delay: TimeInterval, completion: @escaping () -> Void) {

DispatchQueue.global().asyncAfter(deadline: .now() + delay) {

completion()

}

}

// Usage

if let httpResponse = response as? HTTPURLResponse, httpResponse.statusCode == 429 {

retryRequest(with: 60) { // Retry after 60 seconds

// Make the API call again

}

}

func fetchResource(from urlString: String, completion: @escaping (Result<Data, Error>) -> Void) {

guard let url = URL(string: urlString) else { return }

let task = URLSession.shared.dataTask(with: url) { data, response, error in

if let error = error {

completion(.failure(error))

return

}

guard let data = data else {

completion(.failure(URLError(.badServerResponse)))

return

}

completion(.success(data))

}

task.resume()

}

protocol APIDelegate: AnyObject {

func didReceiveData(\_ data: Data)

func didFailWithError(\_ error: Error)

}

class APIClient {

weak var delegate: APIDelegate?

func fetchResource(from urlString: String) {

// Perform the URLSession data task and handle responses

// On success:

self.delegate?.didReceiveData(receivedData)

// On failure:

self.delegate?.didFailWithError(error)

}

}

{

"user": {

"id": "123",

"name": "John Doe",

"address": {

"street": "123 Apple St",

"city": "Cupertino",

"zip": "95014"

}

}

}

struct User: Codable {

var id: String

var name: String

var address: Address

}

struct Address: Codable {

var street: String

var city: String

var zip: String

}

// Decoding

let jsonDecoder = JSONDecoder()

let user = try jsonDecoder.decode(User.self, from: jsonData)

struct Product: Codable {

var id: String

var name: String

var price: Double?

}

// A missing price in the JSON will not cause a decoding failure.

struct Employee: Codable {

var employeeId: String

var employeeName: String

enum CodingKeys: String, CodingKey {

case employeeId = "id"

case employeeName = "name"

}

}

// The JSON keys "id" and "name" will map to "employeeId" and "employeeName" in Swift.

struct Event: Codable {

var name: String

var date: Date

}

let jsonDecoder = JSONDecoder()

jsonDecoder.dateDecodingStrategy = .iso8601

let event = try jsonDecoder.decode(Event.self, from: jsonData)

struct CustomData: Codable {

var id: String

var attributes: [String: String]

init(from decoder: Decoder) throws {

let container = try decoder.container(keyedBy: CodingKeys.self)

id = try container.decode(String.self, forKey: .id)

let additionalInfo = try container.decode([String: String].self,

forKey: .attributes)

attributes = additionalInfo

}

}

let username = "username"

let password = "password"

let loginString = String(format: "%@:%@", username, password)

let loginData = loginString.data(using: String.Encoding.utf8)!

let base64LoginString = loginData.base64EncodedString()

var request = URLRequest(url: URL(string: "https://api.example.com")!)

request.httpMethod = "GET"

request.setValue("Basic \(base64LoginString)", forHTTPHeaderField: "Authorization")

let task = URLSession.shared.dataTask(with: request) { data, response, error in

if let error = error {

// Handle connection errors here

return

}

guard let httpResponse = response as? HTTPURLResponse,

(200...299).contains(httpResponse.statusCode) else {

// Handle HTTP errors here

return

}

// Process the response data

}

task.resume()

import Network

let monitor = NWPathMonitor()

monitor.pathUpdateHandler = { path in

if path.status == .satisfied {

print("Connected to the internet")

} else {

print("No internet connection")

}

}

monitor.start(queue: DispatchQueue.global(qos: .background))

struct User: Codable {

let id: Int

let name: String

let email: String

}

import Foundation

import Combine

class NetworkManager {

// A shared instance for simplicity

static let shared = NetworkManager()

private init() {}

// Function to fetch users

func fetchUsers() -> AnyPublisher<[User], Error> {

// URL for the API endpoint

guard let url = URL(string: "https://api.example.com/users") else {

fatalError("Invalid URL")

}

return URLSession.shared.dataTaskPublisher(for: url)

.map(\.data) // Extract data from the response

.decode(type: [User].self,

decoder: JSONDecoder()) // Decode JSON to User array

.receive(on: RunLoop.main) // Ensure we're on the main thread

.eraseToAnyPublisher() // Return AnyPublisher

}

}

import UIKit

import Combine

class UsersViewController: UIViewController {

private var users: [User] = []

private var cancellables: Set<AnyCancellable> = []

override func viewDidLoad() {

super.viewDidLoad()

fetchUserData()

}

private func fetchUserData() {

NetworkManager.shared.fetchUsers()

.sink(receiveCompletion: { completion in

switch completion {

case .finished:

break

case .failure(let error):

print("Error: \(error.localizedDescription)")

}

}, receiveValue: { [weak self] users in

self?.users = users

self?.updateUI()

})

.store(in: &cancellables)

}

private func updateUI() {

// Update your UI with the fetched users

}

}