# URLManager (Networking)

Librería ligera para **construir y ejecutar requests HTTP** con `async/await`, `Codable`, middlewares y **reintentos exponenciales**. Sin dependencia de UIKit/AppKit: usable en **iOS, macOS, tvOS, watchOS** y librerías puras de Swift.

> Este README está alineado con la API actual del repo `BurgerMike/URLManager` y agrega ejemplos prácticos.

---

## ✅Requisitos

- **Swift**: 6.1+
- **Plataformas** (según `Package.swift`): iOS 16+, macOS 13+, tvOS 16+, watchOS 10+
- Concurrencia moderna (`actor`, `async/await`).

---

## ♒Instalación (Swift Package Manager)

1. Xcode → **File → Add Packages…**
2. URL: `https://github.com/BurgerMike/URLManager.git`
3. Agrega el producto **URLManager** a tu target.

**Package.swift**:

```swift
// swift-tools-version: 6.1
import PackageDescription

let package = Package(
  name: "MyApp",
  platforms: [.iOS(.v16), .macOS(.v13), .tvOS(.v16), .watchOS(.v10)],
  dependencies: [
    .package(url: "https://github.com/BurgerMike/URLManager.git", from: "0.1.0")
  ],
  targets: [
    .target(
      name: "MyApp",
      dependencies: ["URLManager"]
    )
  ]
)
```

---

## 🟩Conceptos

- `RequestManager` *(actor)*: ejecuta requests (`URLRequest`), maneja reintentos y decodifica respuestas.
- `Endpoint<Response>`: describe un endpoint tipado (ruta, método, query, headers y body).
- `URLBuilder`: compone URLs desde una base + path + query.
- **Middlewares**: mutan/observan `URLRequest`/respuesta (log, auth, etc.).
- **Errores tipados**: `URLManagerError` para red/servidor/decoding/cancelaciones.

---

## 🚯Arranque rápido

```swift
import URLManager

struct User: Decodable { let id: Int; let name: String }

let base = URL(string: "https://api.ejemplo.com")!
let manager = RequestManager(url: base, middlewares: [LoggingMiddleware()])

let getUser = Endpoint<User>(
  path: "/v1/users/42",
  method: .get
)

Task {
  do {
    let user = try await manager.run(base: base, getUser)
    print(user.name)
  } catch {
    print("❌", error)
  }
}
```

---

## 🔯Construcción de URL (URLBuilder)

```swift
let u = try URLBuilder(base: base)
  .adding(path: "/v1/search")
  .adding(query: [
    URLQueryItem(name: "q", value: "metal"),
    URLQueryItem(name: "page", value: "1")
  ])
  .build()
```

## 💜GET / POST con JSON

```swift
// GET tipado
struct Repo: Decodable { let id: Int; let full_name: String }
let list = Endpoint<[Repo]>(path: "/repos", method: .get)
let repos = try await manager.run(base: base, list)

// POST con body Encodable
struct CreateUser: Encodable { let name: String }
struct CreatedUser: Decodable { let id: Int; let name: String }

let body = try JSONCoder.encoder.encode(CreateUser(name: "Ada"))
let create = Endpoint<CreatedUser>(
  path: "/v1/users",
  method: .post,
  headers: ["Content-Type": "application/json"],
  body: body
)
let created = try await manager.run(base: base, create)
```

`JSONCoder` ya viene configurado con ISO-8601 y `snake_case/camelCase` convenientes.

## 📵Auth con Bearer + refresh

```swift
let store = TokenStore(initial: "<token>") {
  // Bloque opcional para refrescar el token (si el server devuelve 401)
  return "<nuevo-token>"
}
let auth = BearerAuthMiddleware(provider: store)
let authed = RequestManager(url: base, middlewares: [auth, LoggingMiddleware()])

let me = Endpoint<User>(path: "/v1/me", method: .get)
let user = try await authed.run(base: base, me)
```

## 🔁Reintentos automáticos (exponential backoff)

```swift
let policy = RetryPolicy(maxRetries: 3, baseDelay:
0.4) // reintenta 429/5xx por defecto
```

```
let resilient = RequestManager(url: base, retry: policy)
let user = try await resilient.run(base: base, getUser)
```

---

## ❤️Manejo de errores

```
catch let err as URLManagerError {
  switch err {
  case .invalidURL: /* ... */
  case .invalidResponse(let code): /* códigos no 2xx */
  case .serverError(let status, let data): /* mapear APIProblem */
  case .decodingError(let underlying): /* JSON mal formado */
  case .networkError(let underlying): /* sin conexión, timeout */
  case .cancelled: /* Task cancelada */
  case .custom(let reason): /* tu caso */
  }
}
```

Si tu backend usa **Problem+JSON**, mapea con `APIProblem` + `mapServerError(_:)` y muestra mensajes amigables.

---

## 🛄Tests (XCTest) sin red real

```
import XCTest
@testable import URLManager

final class URLManagerTests: XCTestCase {
  func testBuildSearchURL() throws {
    let base = URL(string: "https://api.ejemplo.com")!
    let url = try URLBuilder(base: base)
      .adding(path: "/v1/search")
      .adding(query: [URLQueryItem(name: "q", value: "swift")])
      .build()
    XCTAssertEqual(url.absoluteString, "https://api.ejemplo.com/v1/search?
q=swift")
  }
}
```

Para tests de red, considera inyectar un `URLProtocol` custom o un `URLSession` stub.

---

## 🛅Middlewares personalizados

```
struct Header: RequestMiddleware {
  let key: String; let value: String
  func prepare(_ request: inout URLRequest) async throws {
    request.addValue(value, forHTTPHeaderField: key)
  }
}

let m = RequestManager(url: base, middlewares: [Header(key: "X-App", value:
"MyApp")])
```

Puedes observar la respuesta:

```
struct CaptureStatus: RequestMiddleware {
  func didReceive(data: Data, response: HTTPURLResponse) async {
    print("Status:", response.statusCode)
  }
}
```

---

## ♒Cliente API recomendado (envoltura)

```
public struct APIClient {
  let base: URL
  let manager: RequestManager

  public init(base: URL, manager: RequestManager) {
    self.base = base; self.manager = manager
  }

  public func user(id: Int) async throws -> User {
    try await manager.run(base: base, Endpoint<User>(path: "/v1/users/\(id)",
method: .get))
  }
}
```

---

## 🛅Utilidades y tips

- **Raw response**: `ActionResponse()` devuelve `(Data, HTTPURLResponse)` si prefieres parsear manualmente.

- **Cancelación**: almacena `Task` para cancelar requests en navegación.
- **Thread safety**: `RequestManager` es `actor` → seguro en concurrencia.
- **Sin UIKit**: úsalo en ViewModels/observables y llama desde SwiftUI (`Task {}`) o Combine/AsyncSequence.

---

## 🕎Tabla de ejemplos

| Caso | Snippet |
|---|---|
| GET tipado | `run(base:, Endpoint<Response>(path:"/v1/...", .get))` |
| POST JSON | `body = JSONCoder.encoder.encode(Body); headers["Content-Type"]` |
| Auth Bearer | `BearerAuthMiddleware(TokenStore(...))` |
| Reintentos | `RetryPolicy(maxRetries:, baseDelay:)` |
| URLBuilder | `.adding(path:) + .adding(query:) + .build()` |

---

## 🔙Pendiente / Roadmap (ideas)

- `multipart/form-data` helpers.
- Descargas con progreso y persistencia.
- Métricas/telemetría con `OSLog` y signposts.

---

## ♌Licencia

MIT

---

# 🛄Proyecto de ejemplo listo para GitHub

A continuación tienes un **mini-demo SwiftUI multiplataforma** (iOS/macOS) que usa tu `URLManager` real. Copia la carpeta `Examples/URLManagerDemo` tal cual a tu repo y ábrela en Xcode como proyecto (o crea uno nuevo y arrastra los archivos).

> El demo consume la **GitHub API pública** (usuarios) para mostrar lista → detalle, usa `BearerAuthMiddleware` (token opcional), `RetryPolicy`, `LoggingMiddleware` y pruebas con `URLProtocol` stub.

## ☢️Estructura

```
Examples/
  URLManagerDemo/
    README.md
    URLManagerDemo.xcodeproj/   # (créalo en Xcode; los fuentes ya están listos)
    Sources/
      App/
        URLManagerDemoApp.swift
        DI.swift
      API/
        APIClient.swift
        Endpoints.swift
        Models.swift
      ViewModels/
        UserListVM.swift
        UserDetailVM.swift
      Views/
        UserListView.swift
        UserDetailView.swift
        RetryBadge.swift
    Tests/
      URLManagerDemoTests.swift
```

---

## ☪️README.md (del demo)

```
# URLManagerDemo
Demo SwiftUI (iOS/macOS) que muestra cómo usar `URLManager` con endpoints
tipados, middlewares y reintentos.

## Requisitos
- iOS 16+/macOS 13+
- Swift 6.1+

## Ejecutar
1) Abre el proyecto en Xcode (o crea uno y agrega `Sources/` y `Tests/`).
2) (Opcional) En `DI.swift` coloca un token personal de GitHub si quieres elevar
el rate-limit.
3) Run en iPhone Simulator o macOS.
```

---

## # Sources/App/URLManagerDemoApp.swift

```
import SwiftUI

@main
struct URLManagerDemoApp: App {
    @StateObject private var container = DI.shared

    var body: some Scene {
        WindowGroup {
            NavigationStack {
                UserListView()
            }
            .environmentObject(container)
        }
    }
}
```

## # Sources/App/DI.swift

```
import Foundation
import URLManager

final class DI: ObservableObject {
    static let shared = DI()

    let baseURL = URL(string: "https://api.github.com")!
    let client: APIClient

    private init() {
        // Middlewares
        let logger = LoggingMiddleware()
        // Coloca tu token si quieres evitar límites: Settings → Developer
settings → Fine-grained PAT
        let tokenProvider = TokenStore(initial: nil) {
            // refresh opcional si tu backend lo requiere; para GitHub no
aplica.
            return nil
        }
        let auth = BearerAuthMiddleware(provider: tokenProvider)

        // Reintentos exponenciales (429/5xx)
        let retry = RetryPolicy(maxRetries: 2, baseDelay: 0.5)

        let manager = RequestManager(url: baseURL,
                                    middlewares: [auth, logger],
                                    retry: retry)
        self.client = APIClient(base: baseURL, manager: manager)
    }
}
```

---

## # Sources/API/Models.swift

```
import Foundation

struct GHUser: Decodable, Identifiable, Hashable {
    let id: Int
    let login: String
    let avatar_url: URL?
}

struct GHUserDetail: Decodable, Hashable {
    let id: Int
    let login: String
    let name: String?
    let followers: Int
    let following: Int
    let public_repos: Int
    let bio: String?
}
```

## # Sources/API/Endpoints.swift

```
import Foundation
import URLManager

enum GitHub {
    static func listUsers(since: Int? = nil, perPage: Int = 30) ->
Endpoint<[GHUser]> {
        var query: [URLQueryItem] = [
            .init(name: "per_page", value: String(perPage))
        ]
        if let since { query.append(.init(name: "since", value:
String(since))) }
        return Endpoint<[GHUser]>(path: "/users", method: .get, query: query)
    }

    static func userDetail(_ login: String) -> Endpoint<GHUserDetail> {
        Endpoint<GHUserDetail>(path: "/users/\(login)", method: .get)
    }
}
```

## # Sources/API/APIClient.swift

```
import Foundation
import URLManager

public struct APIClient {
    let base: URL
    let manager: RequestManager

    public init(base: URL, manager: RequestManager) {
        self.base = base
        self.manager = manager
    }

    public func users(since: Int? = nil, perPage: Int = 30) async throws ->
[GHUser] {
        try await manager.run(base: base, GitHub.listUsers(since: since,
perPage: perPage))
    }

    public func user(login: String) async throws -> GHUserDetail {
        try await manager.run(base: base, GitHub.userDetail(login))
    }
}
```

---

## # Sources/ViewModels/UserListVM.swift

```
import Foundation

@MainActor
final class UserListVM: ObservableObject {
    @Published private(set) var users: [GHUser] = []
    @Published private(set) var isLoading = false
    @Published private(set) var error: String?
    private var since: Int? = nil

    func loadMore(_ client: APIClient) async {
        guard !isLoading else { return }
        isLoading = true; defer { isLoading = false }
```

```swift
        do {
            let next = try await client.users(since: since, perPage: 30)
            if let last = next.last { since = last.id }
            users.append(contentsOf: next)
        } catch {
            self.error = String(describing: error)
        }
    }
}
```

## Sources/ViewModels/UserDetailVM.swift

```swift
import Foundation

@MainActor
final class UserDetailVM: ObservableObject {
    @Published private(set) var detail: GHUserDetail?
    @Published private(set) var isLoading = false
    @Published private(set) var error: String?

    func load(_ client: APIClient, login: String) async {
        isLoading = true; defer { isLoading = false }
        do { detail = try await client.user(login: login) }
        catch { self.error = String(describing: error) }
    }
}
```

---

## Sources/Views/UserListView.swift

```swift
import SwiftUI

struct UserListView: View {
    @EnvironmentObject private var di: DI
    @StateObject private var vm = UserListVM()

    var body: some View {
        List(vm.users) { user in
            NavigationLink(value: user.login) {
                HStack(spacing: 12) {
                    AsyncImage(url: user.avatar_url) { img in img.resizable() }
placeholder: { ProgressView() }
                        .frame(width: 40, height: 40)
                        .clipShape(.circle)
                    Text(user.login).font(.headline)
                }
            }
        }
        .navigationTitle("GitHub Users")
        .toolbar { RetryBadge() }
        .task { await vm.loadMore(di.client) }
        .refreshable { await vm.loadMore(di.client) }
        .navigationDestination(for: String.self) { login in
            UserDetailView(login: login)
        }
    }
}
```

## # Sources/Views/UserDetailView.swift

```
import SwiftUI

struct UserDetailView: View {
    let login: String
    @EnvironmentObject private var di: DI
    @StateObject private var vm = UserDetailVM()

    var body: some View {
        Group {
            if let d = vm.detail {
                VStack(alignment: .leading, spacing: 12) {
                    Text(d.name ?? d.login).font(.largeTitle).bold()
                    HStack { Label("Followers: \(d.followers)", systemImage:
"person.2"); Label("Following: \(d.following)", systemImage:
"arrow.uturn.right") }
                    Label("Repos: \(d.public_repos)", systemImage:
"shippingbox")
                    if let bio = d.bio { Text(bio).italic() }
                    Spacer()
                }
                .padding()
            } else if vm.isLoading {
                ProgressView()
            } else if let err = vm.error {
                Text(err).foregroundStyle(.red)
            }
        }
        .navigationTitle(login)
        .task { await vm.load(di.client, login: login) }
    }
}
```

## # Sources/Views/RetryBadge.swift

```
import SwiftUI
import URLManager

struct RetryBadge: View {
    // Pequeño indicador educativo: muestra la política configurada
    var body: some View {
        HStack(spacing: 6) {
            Image(systemName: "arrow.clockwise")
            Text("Retries: 2")
        }
        .font(.footnote)
        .padding(6)
        .background(.quaternary, in: .capsule)
    }
}
```

---

## 🛄Tests/URLManagerDemoTests.swift

```
import XCTest
@testable import URLManagerDemo
import URLManager

final class URLManagerDemoTests: XCTestCase {
    func testListUsersURLBuilding() throws {
        let base = URL(string: "https://api.github.com")!
        let e = GitHub.listUsers(since: 100, perPage: 10)
        // Si tu Endpoint expone URL, puedes validarla; si no, prueba integrando
a través del manager con un URLProtocol stub.
        XCTAssertEqual(e.path, "/users")
    }
}
```

---

## ♌Notas

- El demo **no depende de UIKit**.
- Para builds de macOS, cambia el destino de la app a macOS en el proyecto.

- Si tu `Endpoint` en el repo tiene propiedades distintas (p.ej. `query`/`headers`), ajusta en `Endpoints.swift`.
- Puedes añadir un `Problem+JSON` model y mapearlo en `RequestManager.onError` según tu backend. ```