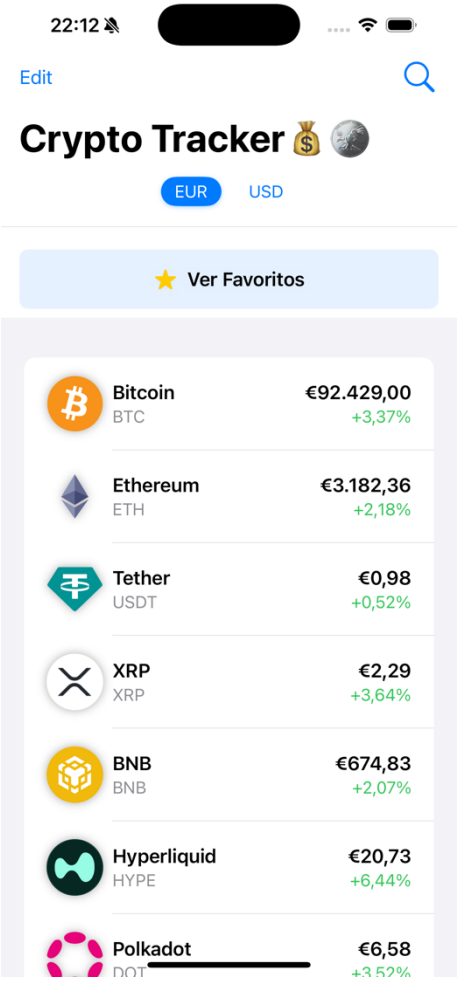
Contenido

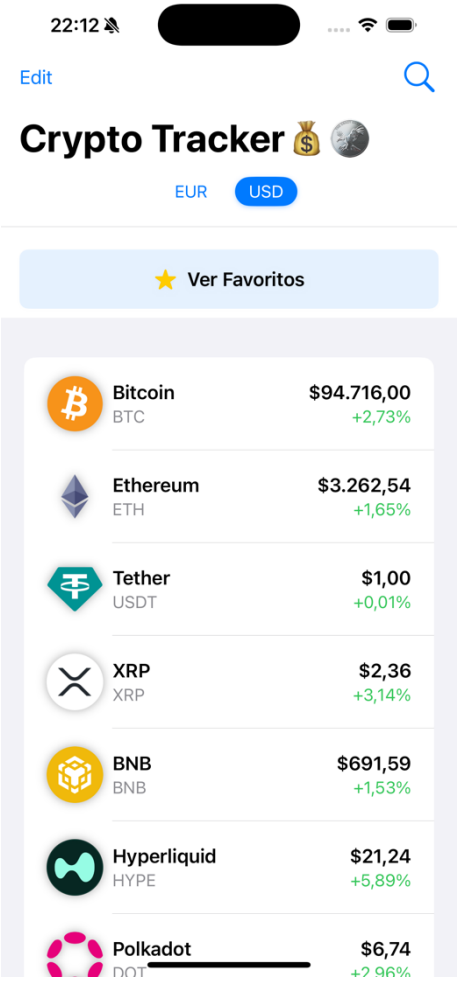
Manual de usuario

Inicio y Selección de moneda

Al abrir la aplicación, se muestra una lista de las 5 criptomonedas más destacadas ordenadas por capitalización de mercado y las añadidas por el usuario manualmente

Arriba de la pantalla, puedes elegir entre las dos principales divisas: **EUR** y **USD**, según la divisa seleccionada podrás ver los datos detallados al pulsar sobre ellas en la moneda decidida por el usuario





Lista de Criptomonedas

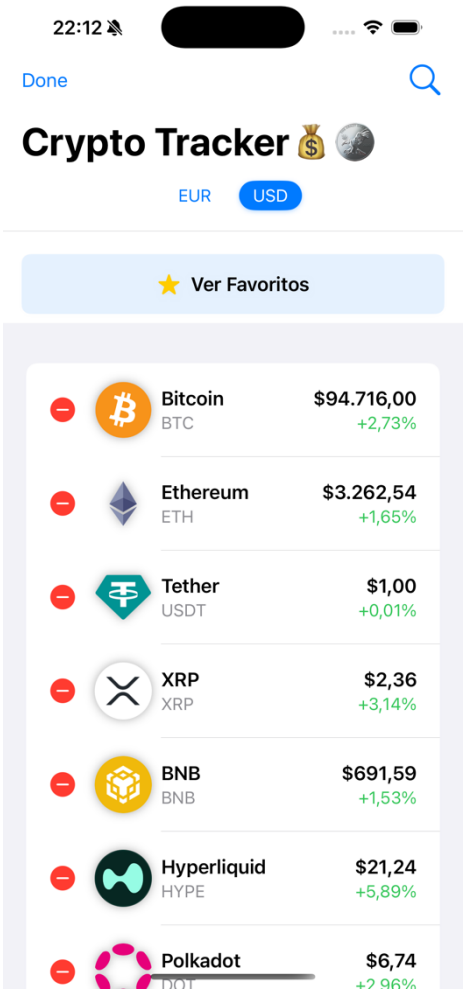
La lista muestra las criptomonedas destacadas y las manualmente añadidas.

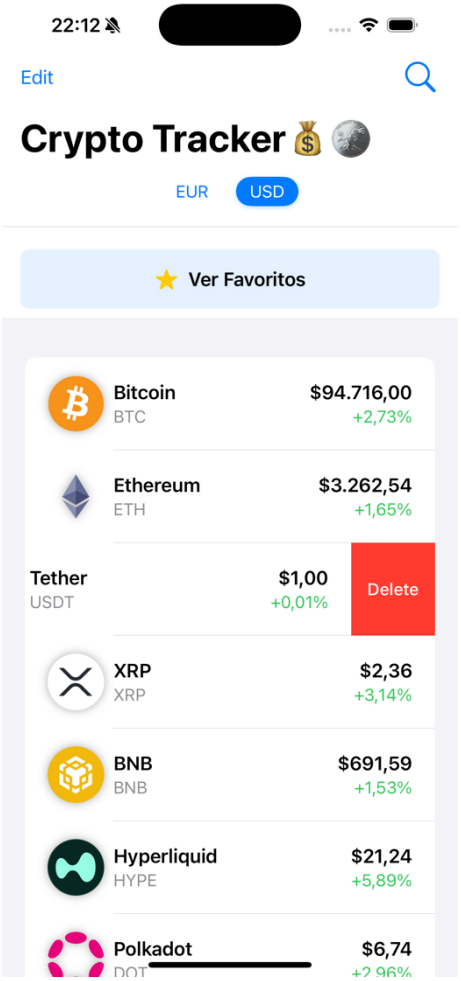
Cada fila incluye:

- Imagen del logo de la criptomoneda.
- Nombre y símbolo.
- Precio actual en la moneda seleccionada.
- Variación porcentual del precio en las últimas 24 horas (positiva o negativa).

Puedes realizar las siguientes acciones:

- **Deslizar hacia la izquierda** para eliminar una criptomoneda manualmente añadida.
- **Pulsar sobre cualquier fila** para ver detalles ampliados.





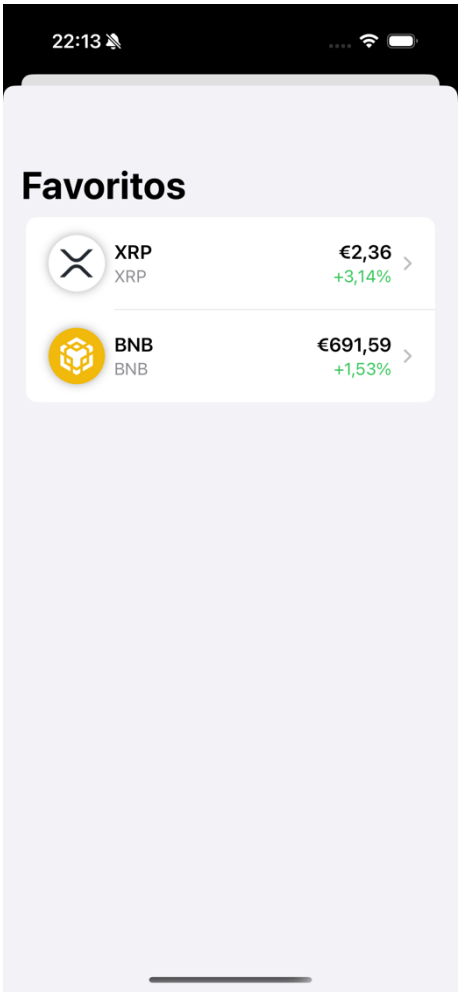
Lista de favoritos

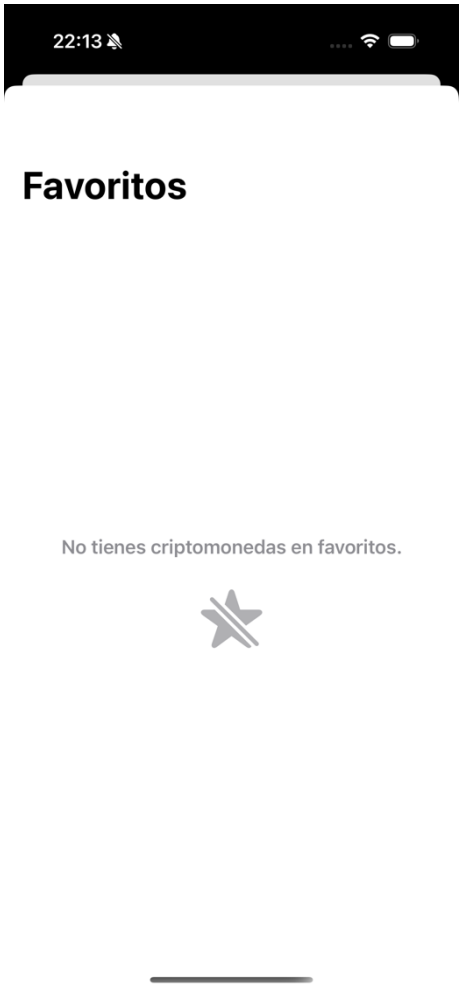
Pulsa en el botón **"Ver Favoritos"** para acceder a la lista de tus criptomonedas favoritas.

La lista de favoritos incluye:

- Todas las criptomonedas marcadas como favoritas desde la vista detallada.

Si no tienes favoritos guardados, la aplicación te muestra un mensaje indicando que no hay elementos disponibles.





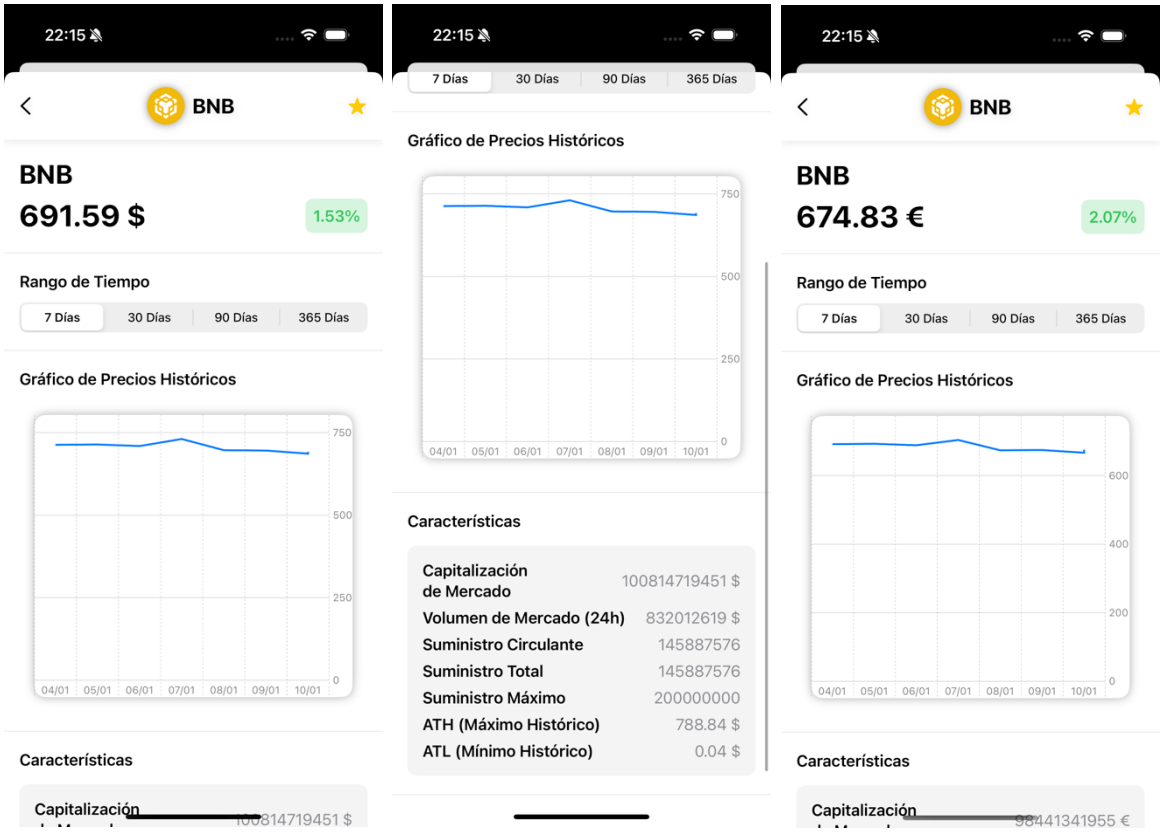
Vista Detallada

Al pulsar sobre cualquier criptomoneda en la lista principal o en favoritos, se abrirá una vista con los datos detallados que incluyen:

- Precio actual.
- Gráfica histórica de precios (30, 90, 365 días).
- Datos adicionales como capitalización de mercado, volumen de transacciones, y máximos/mínimos históricos.

Desde esa vista puedes:

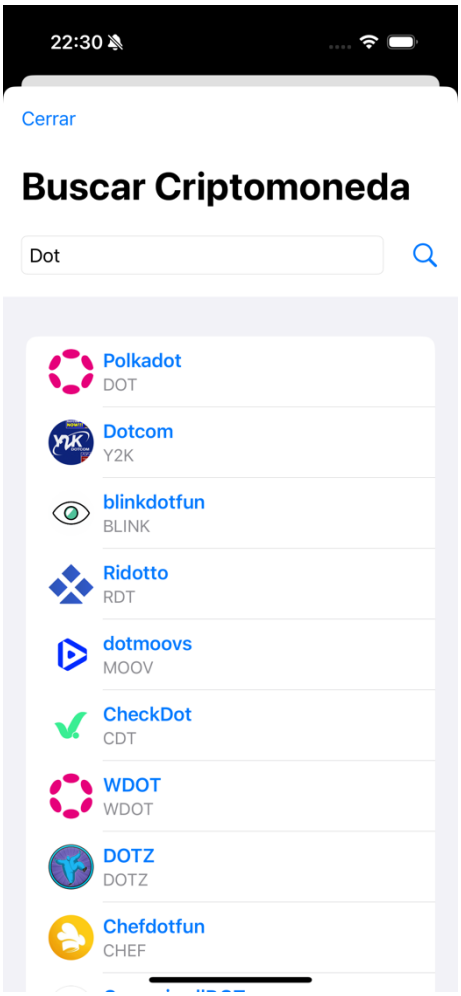
- Añadir o eliminar la criptomoneda de favoritos pulsando el ícono de estrella.
- Seleccionar el rango de tiempo para actualizar la gráfica histórica.

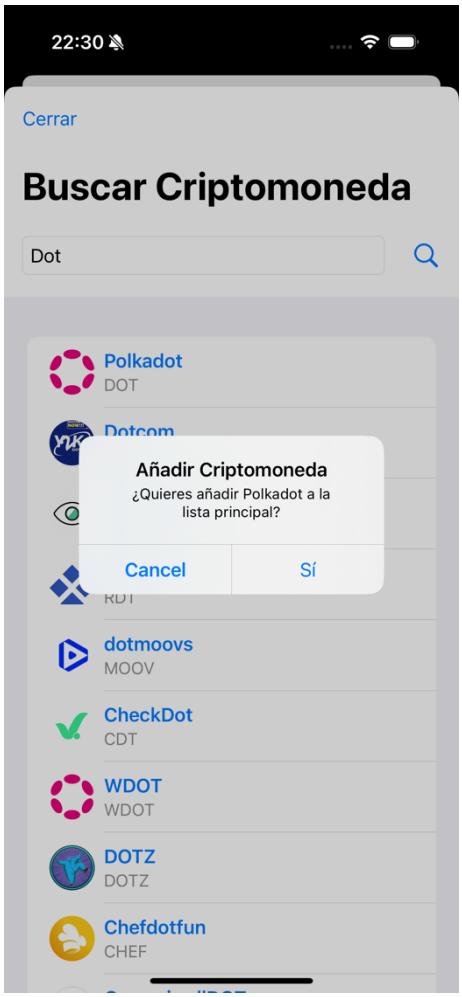


Búsqueda Manual de Criptomonedas

La búsqueda manual permite añadir nuevas criptomonedas a la vista principal. Al pulsar el icono de lupa, puedes buscar por nombre o símbolo de la criptomoneda. Los resultados se muestran en tiempo real. Una vez encontrada, puedes seleccionar una criptomoneda para añadirla a la lista principal mediante la opción correspondiente.

Las criptomonedas añadidas manualmente se guardan y estarán disponibles en ambas monedas (EUR y USD), manteniéndose en la lista incluso después de cerrar la aplicación. Si la criptomoneda ya está en la lista, no será duplicada. Esta función utiliza la API de CoinGecko y requiere conexión a internet para buscar y recuperar los datos actualizados.





Manual del programador

Este manual del programador está creado para explicar más detalladamente el flujo de datos en la aplicación, cómo se realizan las peticiones a la API, cómo se gestionan las respuestas, y cómo se almacenan tanto las criptomonedas guardadas manualmente como las favoritas.

Inicialmente la aplicación está configurada para mostrar las 5 criptomonedas más importantes en cuanto a capitalización de mercado se refiere y las añadidas por el usuario, haciendo la solicitud a la api de CoinGecko para obtener los datos actualizados de cada una de ellas.

Peticiones a la API

Petición principal

Todas las peticiones a la API se gestionan mediante la clase “API.swift”. Como comentábamos antes al iniciarse la app se muestran las 5 primeras monedas y las guardadas por el usuario en swiftdata, aquí te muestro la llamada utilizada para cargar las principales.

```swift
func fetchTopCryptocurrencies(vsCurrency: String, completion: @escaping (Result<[Cryptocurrency], Error>) -> Void) {
        let endpoint = "coins/markets"
        let urlString = "\(baseURL)\(endpoint)?vs_currency=\(vsCurrency)&order=market_cap_desc&per_page=5&page=1"

         //comprobar si url valida
        guard let url = URL(string: urlString) else {
            completion(.failure(APIError.invalidURL))
            return
        }

        var request = URLRequest(url: url)
        request.setValue("Bearer \(Private.coinGeckoAPIKey)", forHTTPHeaderField: "Authorization")
        request.setValue("application/json", forHTTPHeaderField: "Content-Type")

        //realizar solicitud
        let task = URLSession.shared.dataTask(with: request) { data, response, error in
            if let error = error {
                completion(.failure(error))
                return
            }

            guard let data = data else {
                completion(.failure(APIError.noData))
                return
            }

            //decodificar datos
            do {
                let cryptocurrencies = try JSONDecoder().decode([Cryptocurrency].self, from: data)
                completion(.success(cryptocurrencies))
            } catch {
                completion(.failure(error))
            }
        }
        task.resume()
    }
```

Cargar Criptomonedas Guardadas

Durante la inicialización de la aplicación, los IDs guardados manualmente se obtienen de SwiftData y se sincronizan con los datos actuales de la API.

```
private func loadManualCryptocurrencies(modelContext: ModelContext, vsCurrency: String, completion: @escaping ([Cryptocurrency]) -> Void) {
    let fetchDescriptor = FetchDescriptor<CryptoID>()
    do {
        let cryptoIDs = try modelContext.fetch(fetchDescriptor).map { $0.id }

        // Usar el nuevo método con múltiples IDs
        API.shared.fetchCryptocurrencyDetails(ids: cryptoIDs, vsCurrency: vsCurrency) { result in
            DispatchQueue.main.async {
                switch result {
                case .success(let cryptos):

                    completion(cryptos)
                case .failure(let error):
                    self.errorMessage = ErrorMessage(message: "Error al cargar criptomonedas manuales: \(error.localizedDescription)")
                    completion([])
                }
            }
        }
    } catch {
        errorMessage = ErrorMessage(message: "Error al cargar IDs: \(error.localizedDescription)")
        completion([])
    }
}
```

Para no saturar la api al iniciar la aplicación se precargaran dichas criptomonedas con las añadidas por el usuario obtenidas desde swiftdata a dos diferentes arrays depende de la divisa seleccionada (eur/usd), esta es la funcion que se ejecuta al iniciar la app.

```
func loadInitialData(modelContext: ModelContext) {
    let group = DispatchGroup()

    // Cargar criptomonedas top en EUR
    group.enter()
    fetchTopCryptocurrencies(vsCurrency: "eur") { cryptos in
        DispatchQueue.main.async {
            self.cryptocurrenciesEUR = cryptos
            group.leave()
        }
    }

    // Cargar criptomonedas top en USD
    group.enter()
    fetchTopCryptocurrencies(vsCurrency: "usd") { cryptos in
        DispatchQueue.main.async {
            self.cryptocurrenciesUSD = cryptos
            group.leave()
        }
    }

    // Cargar criptomonedas manuales y favoritas
    group.enter()
    loadManualCryptocurrencies(modelContext: modelContext, vsCurrency: "eur") { cryptos in
        DispatchQueue.main.async {
            self.manualCryptocurrenciesEUR = cryptos
            group.leave()
        }
    }

    group.enter()
    loadManualCryptocurrencies(modelContext: modelContext, vsCurrency: "usd") { cryptos in
        DispatchQueue.main.async {
            self.manualCryptocurrenciesUSD = cryptos
            group.leave()
        }
    }

    group.notify(queue: .main) {
        self.switchCurrency(to: self.selectedCurrency)
    }
}
```

Decodificación de Datos

Los datos obtenidos de la API son decodificados en modelos utilizando JSONDecoder.

```swift
struct Cryptocurrency: Identifiable, Codable, Hashable {
    let id: String
    let symbol: String
    let name: String
    let image: String
    let currentPrice: Double
    let marketCap: Double
    let marketCapRank: Int?
    let fullyDilutedValuation: Double?
    let totalVolume: Double       // Volumen de mercado 24h
    let high24h: Double?          // Máximo en las últimas 24h
    let low24h: Double?           // Mínimo en las últimas 24h
    let priceChange24h: Double?   // Cambio de precio en 24h
    let priceChangePercentage24h: Double
    let marketCapChange24h: Double?
    let marketCapChangePercentage24h: Double?
    let circulatingSupply: Double // Suministro circulante
    let totalSupply: Double?      // Suministro total
    let maxSupply: Double?        // Suministro máximo
    let ath: Double?              // Precio más alto de la historia
    let athChangePercentage: Double? // Cambio de porcentaje respecto al ATH
    let athDate: String?          // Fecha del ATH
    let atl: Double?              // Precio más bajo de la historia
    let atlChangePercentage: Double? // Cambio de porcentaje respecto al ATL
    let atlDate: String?          // Fecha del ATL
    let roi: ROI?                 // Retorno de la inversión
    let lastUpdated: String?      // Última actualización
    var isFavorite: Bool = false  // Para marcar como favorito
```

Almacenamiento de Criptomonedas manuales y favoritas

- **Criptomonedas Manuales**

El modelo utilizado para almacenar criptomonedas manuales es CryptoID. Este modelo contiene únicamente el identificador (id) de la criptomoneda seleccionada manualmente.

El propósito de este modelo es guardar los IDs en la base de datos local y usarlos para sincronizar los datos completos de las criptomonedas desde la API.

```swift
import Foundation
import SwiftData

@Model
final class CryptoID: Identifiable {
    @Attribute(.unique) var id: String

    init(id: String) {
        self.id = id
    }
}
```

Cuando el usuario selecciona una criptomoneda desde la búsqueda, su ID se guarda en SwiftData utilizando el modelo CryptoID. Posteriormente, se realiza una solicitud a la API para obtener los datos completos de la criptomoneda y mostrarlos en la vista principal

```
    func addFavorite(for crypto: Cryptocurrency, modelContext: ModelContext) {
        let newFavorite = CryptoFavorite(id: crypto.id)
        modelContext.insert(newFavorite)
        do {
            try modelContext.save()
            if !favoriteCryptocurrencies.contains(where: { $0.id == crypto.id }) {
                favoriteCryptocurrencies.append(crypto)
            }
            print("Añadido a favoritos: \(crypto.name)")
        } catch {
            errorMessage = ErrorMessage(message: "Error al añadir a favoritos: \(error.localizedDescription)")
        }
    }
}
```

Durante la inicialización de la aplicación, los IDs almacenados en CryptoID se obtienen desde SwiftData. Luego, se realiza una solicitud a la API para sincronizar los datos completos de las criptomonedas manuales.

```
private func loadManualCryptocurrencies(modelContext: ModelContext, vsCurrency: String, completion: @escaping ([Cryptocurrency]) -> Void) {
    let fetchDescriptor = FetchDescriptor<CryptoID>()
    do {
        let cryptoIDs = try modelContext.fetch(fetchDescriptor).map { $0.id }

        // Usar el nuevo método con múltiples IDs
        API.shared.fetchCryptocurrencyDetails(ids: cryptoIDs, vsCurrency: vsCurrency) { result in
            DispatchQueue.main.async {
                switch result {
                case .success(let cryptos):

                    completion(cryptos)
                case .failure(let error):
                    self.errorMessage = ErrorMessage(message: "Error al cargar criptomonedas manuales: \(error.localizedDescription)")
                    completion([])
                }
            }
        }
    } catch {
        errorMessage = ErrorMessage(message: "Error al cargar IDs: \(error.localizedDescription)")
        completion([])
    }
}
```

- **Criptomonedas Favoritas**

El modelo utilizado para las criptomonedas marcadas como favoritas es CryptoFavorite. Al igual que en el caso de las manuales, este modelo almacena únicamente el id de la criptomoneda.

Estos IDs se utilizan para identificar las favoritas y cargarlas al iniciar la aplicación.

```
import SwiftData

@Model
class CryptoFavorite {
    @Attribute(.unique) var id: String

    init(id: String) {
        self.id = id
    }
}
```

Agregar una Criptomoneda Favorita

Cuando un usuario marca una criptomoneda como favorita, su ID se guarda en SwiftData utilizando el modelo CryptoFavorite.

```
    func addFavorite(for crypto: Cryptocurrency, modelContext: ModelContext) {
        let newFavorite = CryptoFavorite(id: crypto.id)
        modelContext.insert(newFavorite)
        do {
            try modelContext.save()
            if !favoriteCryptocurrencies.contains(where: { $0.id == crypto.id }) {
                favoriteCryptocurrencies.append(crypto)
            }
            print("Añadido a favoritos: \(crypto.name)")
        } catch {
            errorMessage = ErrorMessage(message: "Error al añadir a favoritos: \(error.localizedDescription)")
        }
    }
}
```

Eliminar Criptomonedas Favoritas

Cuando se desmarca una criptomoneda como favorita, su ID se elimina de SwiftData:

```
    }

    func removeFavorite(for crypto: Cryptocurrency, modelContext: ModelContext) {
        let fetchDescriptor = FetchDescriptor<CryptoFavorite>(predicate: #Predicate { $0.id == crypto.id })
        do {
            if let existingFavorite = try modelContext.fetch(fetchDescriptor).first {
                modelContext.delete(existingFavorite)
                try modelContext.save()
                favoriteCryptocurrencies.removeAll { $0.id == crypto.id }
                print("❌ Eliminado de favoritos: \(crypto.name)")
            }
        } catch {
            errorMessage = ErrorMessage(message: "Error al eliminar de favoritos: \(error.localizedDescription)")
        }
    }
```

API CoinGecko

La aplicación opera con una cuenta demo de la API de CoinGecko, lo que implica un límite estricto en el número de solicitudes permitidas por minuto. Si se alcanza este límite, las solicitudes serán rechazadas temporalmente y aparecerá un mensaje informando al usuario que debe esperar unos segundos antes de intentar nuevamente. Este límite afecta tanto la carga inicial de datos como las operaciones relacionadas con favoritos o búsquedas manuales. Para un mejor rendimiento y evitar interrupciones, se ha intentado solucionar usando una manejadora para elegir entre varias claves pero CoinGecko no contempla esa opción, para un correcto funcionamiento de la aplicación obligatoriamente tenemos que tener una cuenta un limite superior de solicitudes al de la cuenta demo.