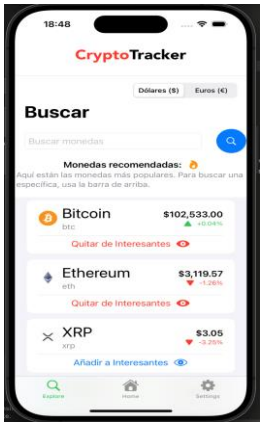
INFORME TRABAJO FINAL APLICACIONES AVANZADAS: CRYPTOTRACKER

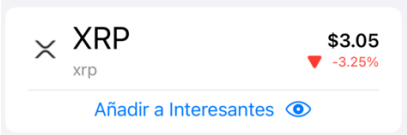
Diego Lobo Mirón 72259918J.

Voy a explicar un poco mi trabajo, desde las vistas que he usado hasta el manejo de datos a través de la api, primero comenzare por las vistas y acabare hablando del servicio de la API.

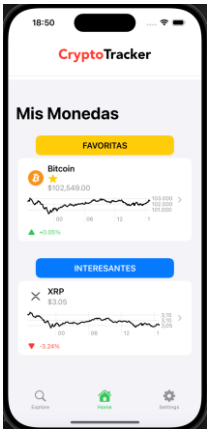
1. **ContentView**
    a. Su objetivo es mostrar una interfaz con tres pestañas: **Buscar**, **Mi contenido** y **Ajustes**, donde cada pestaña tiene su propia vista.
    b. Usa un TabView, que es el componente de SwiftUI que permite mostrar **pestañas** en la aplicación.

2. **SearchPopularListView**
    a. Se llama desde la pestaña Buscar
    b. La vista SearchPopularListView permite buscar criptomonedas o explorar las más populares.
    c. **Busqueda**: Cuando el usuario introduce un término y presiona el botón de buscar, performSearch() filtra las criptomonedas y muestra los resultados en una lista, donde cada fila se representa con SearchPopularRowView.
    d. **Monedas populares**: Si no hay búsqueda activa, loadPopularCoins() carga y muestra una lista de criptos recomendadas desde el viewModel.
    e. **Funciones auxiliares:** clearSearch() borra el texto de búsqueda y restaura la vista inicial con las monedas populares.
    f. 

1. **SearchPopularRowView**
    a. Es la representación de una fila en la lista de búsqueda y monedas populares. Carga los precios históricos de la criptomoneda y muestra su variación en las últimas 24 horas, indicando si ha subido o bajado.



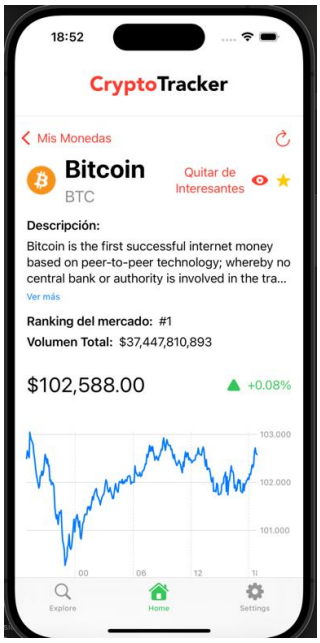
2. **InterestingCryptoListView**
    a. Esta vista muestra las criptomonedas que el usuario ha marcado como interesantes o favoritas. Permite **agregar, eliminar y organizar** criptomonedas en dos secciones: favoritas e interesantes
    b. Ordena dichas monedas con dos funciones , y las ordena según su precio, también se pueden marcar o desmarcar como favoritas mediante una animación



3. **InterestigCryptoRowView**
    a. La vista InterestingCryptoRowView representa **una fila en la lista de criptomonedas interesantes**. Muestra el **nombre, imagen, precio, gráfico de evolución y cambio porcentual en 24h**. Además, al hacer clic, redirige a CryptoDetailView, donde se ve información más detallada.

b. 

4. **CryptoDetailView**
    a. CryptoDetailView es la vista que **muestra información detallada** de una criptomoneda específica.
    Incluye **imagen, descripción, precio, ranking, volumen de mercado, gráfico de precios y opciones para marcar como favorita**.
    b. La función fetchData(days: Int, currency: String) se encarga de descargar los precios históricos desde la API, utilizando el identificador único de la criptomoneda (crypto.APIid).
    c. El número de días seleccionado (selectedDays) determina el período del historial a consultar (24h, 7 días, 1 mes o 90 días).
    d. Cuando la API devuelve los datos, se almacenan en prices y se calcula el cambio porcentual con calculatePercentageChange().
    e. Una vez que los datos están disponibles:
        i. Se muestra el **precio actual** de la criptomoneda en euros o dólares.
        ii. Se calcula y presenta la **variación porcentual del precio** en el período seleccionado.
        iii. Se dibuja un **gráfico de línea** con los precios históricos, ajustando los ejes
        mediante xScaleDomain() y yScaleDomain().



EXPLICACION DEL FUNCIONAMIENTO DE LA API:

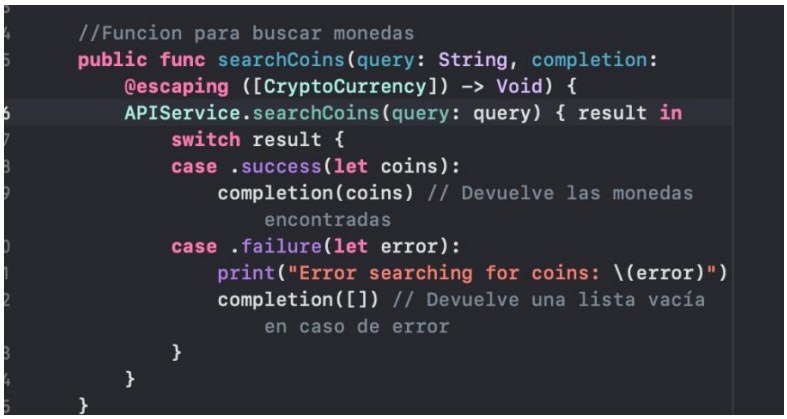
La clase CryptoCurrency representa una criptomoneda dentro de la aplicación. Sus datos se actualizan dinámicamente a través de la API y se almacenan en la base de datos con SwiftData.

```swift
@Model
class CryptoCurrency {
    @Attribute var name: String
    @Attribute var APIid: String
    @Attribute var symbol: String
    var coinDescription: String?
    var isFavorite: Bool = false
    var marketCapRank: Int?
    var totalVolume: [String: Double]?
    var imageURL: URL?
```

Cuando SearchPopularListView aparece, llama a loadPopularCoins() para obtener criptomonedas populares. Si los datos ya fueron cargados, los recupera con getPopularCoins() en el viewModel. Si no, loadPopularCoins() solicita los datos a la API a través de viewModel.loadPopularCoins(). La API responde con las criptomonedas más importantes, las almacena en popularList, y la vista se actualiza automáticamente

```swift
    //Funcion para cargar las monedas mas populares
    public func loadPopularCoins(completion: @escaping ()
        -> Void) {
        // Llama a la API para obtener monedas populares
        APIService.getPopularCoins { result in
            switch result {
            case .success(let coins):
                self.popularList = coins // Almacena las
                    monedas populares directamente en
                    recommended
                completion() // Llama al callback cuando
                    termina
            case .failure(let error):
                print("Error fetching popular coins:
                    \(error)")
                completion() // Llama al callback incluso
                    en caso de error
            }
        }
    }
```

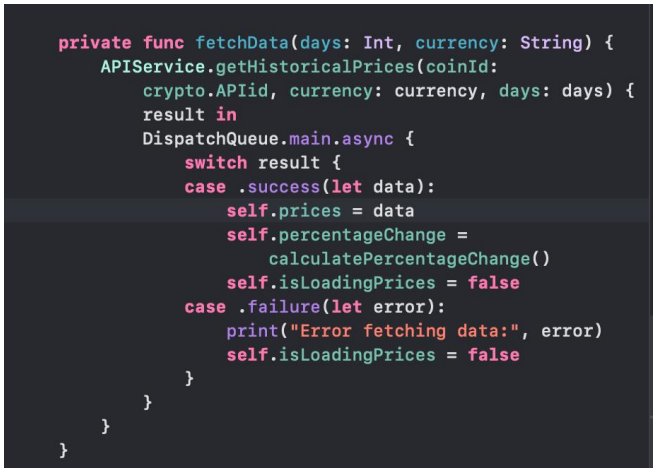
Cuando el usuario introduce un texto en la barra de búsqueda y pulsa el botón, SearchPopularListView llama a viewModel.searchCoins(query:). Esta función solicita los datos a la API mediante APIService.searchCoins(query:), que devuelve un JSON con una lista de criptomonedas dentro de SearchResult. Los datos se decodifican en un array de SearchCoin, que luego se transforma en objetos CryptoCurrency. Finalmente, los resultados se devuelven a SearchPopularListView, actualizando la lista y mostrando las criptomonedas encontradas en la interfaz.



```
//Funcion para buscar monedas
public func searchCoins(query: String, completion:
    @escaping ([CryptoCurrency]) -> Void) {
    APIService.searchCoins(query: query) { result in
        switch result {
        case .success(let coins):
            completion(coins) // Devuelve las monedas
                encontradas
        case .failure(let error):
            print("Error searching for coins: \(error)")
            completion([]) // Devuelve una lista vacía
                en caso de error
        }
    }
}
```

Cuando CryptoDetailView ejecuta fetchData(days, currency), esta función llama a APIService.getHistoricalPrices(coinId, currency, days)

Si la API responde correctamente, los datos se decodifican en una lista de precios históricos (Price(timestamp, value)) y se devuelven a fetchData(), que los almacena en prices y calcula la variación porcentual con calculatePercentageChange(). Finalmente, CryptoDetailView usa estos datos para actualizar el gráfico de precios



```
private func fetchData(days: Int, currency: String) {
    APIService.getHistoricalPrices(coinId:
        crypto.APIid, currency: currency, days: days) {
        result in
        DispatchQueue.main.async {
            switch result {
            case .success(let data):
                self.prices = data
                self.percentageChange =
                    calculatePercentageChange()
                self.isLoadingPrices = false
            case .failure(let error):
                print("Error fetching data:", error)
                self.isLoadingPrices = false
            }
        }
    }
}
```