iOS Coding Challenge

What?

Create an app that has 2 views.

1. A list view of all possible books handled by Bitso
    a. Ignore fees object
    b. Book name should be in an user friendly format (remove underscore and uppercase it. i.e BTC MXN)
    c. Display maximum_price highlighted (values should be formatted taking into consideration the Locale)
    d. Display maximum_value - minimum_value as secondary information (values should be formatted taking into consideration the Locale)
    e. Make sure to add pull to refresh
    f. Handle errors gracefully
    g. Keep the screen refreshing every 30 seconds

2. A detail view for a particular book showing information about it
    a. Handle errors
    b. Display (volume, high, change_24) (values should be formatted taking into consideration the Locale)
    c. Display ask and bid in a different section (values should be formatted taking into consideration the Locale)

How?

- Use Dependency Injection
- Use Modularization (You can choose CocoaPods, SPM or native Xcode Frameworks/Libraries)
- Create a simple networking layer
- Make sure to include unit tests
- Work on a Github public repo (with detailed commits messages)

General instructions

- Write a README file explaining your decisions, architecture and instructions for running the project and test suite
- The code must be authorial and created based on the scope of the coding exercise
- Over-engineering and/or unnecessary complexity will not be accepted
- Non-authorial code will not be accepted
- You can use SwiftUI or UIKit
- Avoid using third-party dependencies - maximum limit: 2 third-party dependencies
- Mandatory: the project and tests must compile and work as expected

Endpoints

List: https://api.bitso.com/v3/available_books/
Detail: https://api.bitso.com/v3/ticker?book=\(book)
Full information about API
https://docs.bitso.com/bitso-api/docs/list-available-books

Tips & Tricks!

- ➔ Use Dependency Injection
- ➔ Be aware of couplings between modules
- ➔ Isolate responsibilities in specific layers or classes
- ➔ You must write tests for the entire application (if possible)
- ➔ You must avoid over-engineering
- ➔ You should use generics when possible