Project 3: Fetcher Refactor & Mocked API Testing

Project Title

Project 3 - Fetcher Refactor & API Mocking with Tests

Goal

Refactor your HTTP fetching logic (currently in httpFetcher.go) into a more testable, modular form, and write unit tests using httptest.Server to simulate responses from Magic Eden.

Why This Matters

- Real-world APIs fail, change, or rate-limit. Mocking lets you test behavior without relying on live APIs.
- Decoupling HTTP logic = easier testing, debugging, and reuse in desktop/Wails apps.

Requirements

1. Refactor fetcher:

- Move the raw URL + HTTP logic into an internal `fetch` or `client` package
- Let it return ([]byte, error) instead of using a channel

2. Create a test file for the fetcher

- Use `httptest.NewServer` to simulate a working Magic Eden listings endpoint
- Validate URL construction based on input parameters
- Test edge cases (empty data, malformed response, timeout)

3. Use interfaces (optional but encouraged):

- Define an interface (e.g. `type ListingsFetcher interface`) and implement it with your real + mock clients
- This makes testing and future wallet integrations easier

Stretch Goals

- Add retry logic with backoff on 5xx or timeout responses
- Parameterize the base API URL to support test/local APIs

Testing Tasks

- Simulate successful responses
- Simulate empty JSON array return

Project 3: Fetcher Refactor & Mocked API Testing

- Simulate broken JSON return (test JSON error path)
- Simulate delayed response (timeout)
- Validate that constructed URL matches expected

Recommended Structure

- fetcher/
  - fetcher.go
  - fetcher_test.go
  - client.go (if you abstract base API URL logic)

- Each test should not rely on external API.
- Use Go's stdlib `httptest`, not third-party mocking libs.