Ho to run things concurrently

Async Let

- Spawns a new Task behind the scenes, inheriting the task local values and an actor
- You must await the task created with async let

```
struct FetchTrackAndArtistUseCase {

    private let spotifyRepo: SpotifyRepo

    init(spotifyRepo: SpotifyRepo) {
        self.spotifyRepo = spotifyRepo
    }

    func execute(trackName: String, artistName: String) async throws -> (track: Track, artist: Artist) {
        async let trackTask = spotifyRepo.getTrack(by: trackName)
        async let artistTask = spotifyRepo.getArtist(by: artistName)

        return try await (trackTask, artistTask)
    }
}
```

Tasks Group

- Can be throwing or non-throwing
- Child tasks must respect throwing vs non-throwing condition
- No child task can live longer than it’s parent group
- Error handling is a crucial thing here
- The order of execution is not determined

```
struct FetchTracksUseCase {

    private let spotifyRepo: SpotifyRepo

    init(spotifyRepo: SpotifyRepo) {
        self.spotifyRepo = spotifyRepo
    }

    func execute(by names: [String]) async -> [Track] {
        // initiate a task group
        await withTaskGroup(of: Track?.self) { taskGroup in
            // for each track name we need to spawn a task
            names.forEach { name in
                taskGroup.addTask {
                    do {
                        return try await spotifyRepo.getTrack(by: name)
                    } catch {
                        // or any other error handling
                        print(error)
                        return nil
                    }
                }
            }
            // now we await all the child tasks
            var tracks: [Track] = []
            for await track in taskGroup {
                if let track {
                    tracks.append(track)
                }
            }
            // return the tracks
            return tracks
        }
    }
}
```