Ugo Mmirikwe

Polyglot Software Developer

• Website ugommirikwe.com • Email me@ugommirikwe.com • GitHub https://github.com/ugommirikwe
• Google Developer Group Organizer Profile https://gdg.community.dev/u/ugo/#/about

Experience

With over a decade in building software applications for various digital platforms as well as integrating and administering Linux-based enterprise systems, I've built up skills and practical experience in the following areas:

- **iOS Application Development**, using native tools:
  - *Xcode* and *AppCode* IDEs;
  - *Swift* (a little of Objective-C) programming language;
  - *UIKit* and *SwiftUI* frameworks for UI development;
  - Storyboard and Storyboard-less *UIKit* app development.
  - Other native SDKs & *Platform Tools*: Combine, *CoreData*, *MapKit*, etc;
  - Design patterns: MVVM and Redux-inspired Unidirectional State Management;
  - Codebase modularisation using Swift Packages;
  - Automated Testing—*XCTest*, *XCUITest*;
  - *Fastlane*-driven CI/CD Pipeline Processes.

- **Android Application Development**, using native tools:
  - *Android Studio* IDE;
  - *Kotlin* and *Java* programming languages;
  - Legacy Android View and *Jetpack Compose* UI development;
  - The native Android SDK & Android Jetpack suite of tools and APIs: *Dagger Hilt*, *Room*, *Navigation*, etc;
  - Kotlin Coroutines, with *Flow* data structures;
  - 3rd-Party Libraries: *Retrofit*, *OkHTTP, Firebase*;
  - Automated Testing: Unit & Instrumentation, with *JUnit, Espresso, UIAutomator* and *Roboelectric.*
  - Protocol/Interface-driven development.

- **Flutter** mobile app development, using:
  - *Dart* programming language;
  - State Management & Dependency Injection with **RiverPod Framework**;
  - Automated Testing: unit, widget and integration testing.

- **Firebase** Platform:
  - Firebase Dynamic Links
  - Firestore Database
  - Firebase Remote Config
  - Firebase Crashlytics
  - Firebase App Distribution
  - Firebase A/B Testing

- **Web Front-End/Browser-side Development**:
  - Modern JavaScript Frameworks & Libraries: *VueJS, ReactJS, Redux, TypeScript;*
  - CSS Frameworks & Libraries: *Twitter Bootstrap, Material Design Lite, Semantic UI, Tailwind CSS*;
  - CSS Preprocessors: *LESS* and *SASS/SCSS*;
  - CSS Coding Methodologies: *BEM* and OOCSS.

- **Back-end/Server-Side Development**:
  - NodeJS with *Express* + *Loopback*/*Parse* Framework;
  - PHP Frameworks & Tools: *Laravel*, *WordPress*;
  - Python Framework: *Django* (very limited experience working with this);
  - HATEOAS-compliant REST Web Services API Development;
  - Database Systems: MongoDB, MySQL, PostgreSQL;
  - Web Servers: *Nginx* and *Apache*.

- **DevOps, Agile Software Development Management & Application Lifecycle Management:**
  - User Stories development & management;
  - *GitFlow* Distributed Workflow Management, with Team Code Reviews & Pull-Request Management;
  - *Docker*-based Development Environment Tooling;
  - Continuous Integration & Delivery (CI & CD), with Fastlane, Docker and similar orchestration tools;
  - Release Management using Apple iTunes Connect, *Microsoft AppCenter, Firebase Distribution, etc.*;
  - Issues Management with *Microsoft Azure Devops Services, Atlassian JIRA, BitBucket, GitHub, Gitlab*

Recent Engagements

(October 2020 - now)
Freelance Software Development

- https://www.youtube.com/watch?v=jupgwnsjNIY&t=2500s
  Speaking at the Google Developer Group MeetUp event on the topic "**Concurrency in Dart**"

- **https://github.com/ugommirikwe/clearscore**
  Small teaser project where I demonstrate best practices in iOS development. (While it replicates a very tiny subset of features of the ClearScore app, it is in no way affiliated nor endorsed by the ClearScore company and can not be used for anything order than for demonstration purposes):
  - UIKit-based programmatic UI—no Storyboards except for the LaunchScreen;
  - MVVM UI unidirectional data flow pattern; also uses Swift Combine to enable reactive data-binding for the views;
  - Localized strings: Uses standard *Localizable.strings* file to set up an easily extensible internationalization (i18n) configuration to ease incremental localization (l10n) of the app, but then wrapping the strings in a Swift *Enum* to ensure compile-time access safety anywhere else in the app;
  - Simple Unified Remote Network Layer, built on modern Swift concurrency constructs: *Async/Await*, *@MainActor*;
  - Dependency Injection and Protocol-Driven development;
  - Automated Tests: unit and UI tests (implemented with Page Object pattern for a more expressive test structure);
  - Guided by standard Swift linting conventions, the codebase aims to be as neat and easy to read as possible.

- **Devops Set Up With Fastlane & Gitlab: Automated iOS App Release Implementation**
  I implemented a Gitlab (https://gitlab.com) + Fastlane-driven (https://fastlane.tools) CI/CD workflow for an in-house enterprise iOS Project for a German electricity company **to automate their iOS app release process**. The workflow is automatically triggered by merge requests into a major/release branch: lint -> build -> test -> upload to TestFlight:
  - Uses Fastlane's ***match*** (*https://docs.fastlane.tools/actions/match/*) tool for automating code-signing of the app—generating and managing app distribution certificates & provisioning profiles;
  - Uses Fastlane's ***gym*** (*https://docs.fastlane.tools/actions/gym/*) tool to build the app and generate the installable IPA file;
  - Uses Fastlane's ***scan*** (*https://docs.fastlane.tools/actions/scan/*) tool to run the XCTest and XCUITests automated tests bundled with the app's codebase;
  - Uses Fastlane's ***pilot*** (*https://docs.fastlane.tools/actions/pilot/*) to upload the built artifacts--IPA, dSYM files, etc--to iTunes Connect's TestFlight console for onward distribution to testers and end users;
  - Uses Gitlab Runner on remote macOS machines to execute the above Fastlane commands and report back to the Gitlab console.

- **https://github.com/ugommirikwe/AeroboticsFarms**
  Small teaser project where I demonstrate best practices in iOS development:
  - SwiftUI-based UI,
  - MVVM UI data flow pattern, combined with data repository pattern that simplifies offline-first experience—remote data fetching and persistence with **CoreData** framework, so data is available even without Internet connection.
  - Also uses the **Apple Keychain framework** for secure persistence of sensitive data (i.e. API Token).
  - The code also demonstrates protocol-driven development and dependency injection (using a rudimentary IoC container), making it easy to swap-out code dependency and facilitate easy testability of code.
  - Proper modularisation of code using Swift Packages. Among other benefits, this reduces code compile times and contributes to a robust developer experience (DX), especially when working in a large team.

- **https://ugommirikwe.com/lokator** ***(In Progress)***
  Cross-platform iOS+watchOS and Android apps I'm working on for sharing users' live location for a short time span. This is modeled after the WhatsApp live location sharing feature, where users can share their movements with specific people for a specific period. It's aimed at people who are privacy sensitive, as it doesn't use or retain any personally identifiable data. (*Apps should be in the AppStore and PlayStore soon.*):

    - Uses the **Apple MapKit** framework, on iOS+watchOS
    - Uses **Mapbox Map SDK** (https://docs.mapbox.com/android/maps/guides/) for the Android app.
    - **Firebase Firestore** Platform for real-time data storage.
    - **Firebase Dynamic Links** library (https://firebase.google.com/docs/dynamic-links) for creating sharable cross-platform deep-links.
    - Also uses my **Turnstate** library (https://github.com/ugommirikwe/turnstate) mentioned below.
    - App preview here: https://drive.google.com/file/d/1uhR27PIfczP7ssO2z7jfCkvSpo6zH209/view?usp=sharing

- **https://github.com/ugommirikwe/turnstate**
  I built a Redux-inspired unidirectional state management library for Swift, which has no external dependency aside the standard Swift library, meaning it can be used for developing apps across (even old) versions of Swift-supported platforms, unlike other similar solutions out there. (I'm also working on an Kotlin version of the state management library.)

(November 2019 – October 2020)
Mobile App Development Contractor,
WhereIsMyTransport (www.whereismytransport.com)

- My first task on joining was to implement a suite of automated tests (UI) tests for the **Collector** app (https://play.google.com/store/apps/details?id=whereismytransport.com.collector.app&hl=en_ZA&gl=US) to improve the quality assurance process for, and reliability of, the app used by hundreds of users to collect geo-spatial data on informal transport routes in several cities across the world.
- I then led the project to rewrite the **Collector** app using the Kotlin language and modern Android Architecture Components, complete with implementing modern Android unit and instrumentation testing.
- As a weekend project, I also started building a native UIKit-based native iOS edition of the **Collector** app.
- Between working on the Collector app, I worked in a two-man team on building, releasing and pushing updates to the Flutter-based **Rumbo** app: https://play.google.com/store/apps/details?id=mx.mirumbo.rumbo&hl=en_ZA&gl=MX

(April 2017 – November 2019)
Software Development Contractor,
Double-Eye (www.double-eye.com)

I built a cross-platform (web, iOS, Android and Windows Desktop) browser-based single-page application for viewing content packaged in PDF and ePub format and served from a purpose-built content management system. I built the app entirely from scratch without using any of the cross-platform development tools and frameworks like Flutter or ReactNative.

- The application works similarly to Apple iBooks, Google Books, Amazon Kindle and the major ebook apps. It allows text highlighting and annotating, bookmarking, search, etc, with much of the content and functionalities still available and accessible when offline.

- While browser-based, the application adapts to its runtime enclosure—native Windows, iOS, or Android apps, or in an iFrame within a larger browser app—and interacts with the resources available from the enclosure, including storing and loading content and user-generated content enrichments using persistent local storage systems such as ***IndexedDB***, falling back to ***localStorage*** (in a browser enclosure), and ***SQLite***, in the native apps' enclosures.

- The application was built with the ***VueJS*** 2 Javascript framework. Uses ***Vuex*** for state management, ***Axios*** for XHR requests. ***Vue Test Utils*** and ***Jest*** for automated unit tests. I also wrote the Android (**Java**), iOS (**Objective-C**) and Windows Desktop (**C#**) native components of the app.
