All in Mobile

Few words about project sample

The sample project is the part of our newest Swift project which is still inside implementation process. This sample contains one module which is used to displays user content (statements) on 3 screens. The sample also contains the screen to create new content as a user.

Keep that in mind that **All request are based on static .json files** just to allow you to compile the project and see the content inside the app.

Rx - Reactive programming

The app implements MVVM architecture instead of MVC, which is recommended by Apple. MVVM approach recently is strongly used by the iOS community.

The app uses heavily the `RxSwift` library, which is the Swift implementation of `ReactiveX` .

Code style & lint

Inside AIM we use SwiftLint to keep the same style guide in our swift projects.

Continuous Integration in All in Mobile

First of all our Continuous Integration is based on the Jenkins platform which is installed locally on our machine. Each `job` inside Jenkins represents version-type which has to be created. Usually, one project has 2 jobs inside Jenkins. The first one is used for the `Crashlytics` builds for our testers and internal usage. The second job builds and sends version to the iTunesConnect TestFlight platform from where it is possible to publish the app on the market.

GitFlow

Continuous Integration it has to be fully automated and supported by branching model which will allow releasing new versions, hot-fixes or test builds. For that, we use GitFlow branching model which is described here.

The basic setup of our branches is to have at least 2 main branches:
- `development` - it is the place where developers merge their work via pull requests. Every push on this branch fire the Jenkins job and sends version to the `Crashlytics`
- `master` - it represents the version which is available for users. Each push on this branch fire the Jenkins job to create the AppStore version and sends it there.

Fastlane

Jenkins and GitFlow represent **when** a version should be built and released. `Fastlane` defines **how** to build it. Fastlane is written by Twitter team and allows to write scripts to create desired delivery process. Our basic setup covers:
- Increasing the build number
- Running unit tests (and fails on unit test fail)
- Slack notifications
- Building the version (including signing with proper certificates)
- Sending the version to the `Crashlytics` or Testflight/AppStore

More advanced setups allow also to create app screenshots and uploading them to the iTunesConnect (in every language which the app supports)