Github Actions deploy Flutter iOS app

When I encountered challenges with deploying A flow manually through GitHub Actions, I turned to the invaluable insights shared by Duco Fronik in their expertly crafted article. Through their guidance, I was able to gain a deeper understanding of the fundamental principles of A flow and learn how to navigate Apple's unique limitations for a smoother deployment process. I found the article particularly useful as it provided practical tips on what factors to consider and how to overcome common obstacles without resorting to third-party solutions like Codemagic or Fastlane. Fronik's article proved to be an indispensable resource for anyone looking to streamline their A flow deployment process with GitHub Actions.

GitHub actions workflow:
https://github.com/PrimozRatej/app/blob/master/.github/workflows/version-release.yml

Build IPA

For building an IPA file that will be deployed with pipe we need these 4 values:

```
- name: 'Install the Apple certificate and provisioning profile'
  env:
    BUILD_CERTIFICATE_BASE64: ${{ secrets.BUILD_CERTIFICATE_BASE64 }}
```

```
    P12_PASSWORD: ${{ secrets.P12_PASSWORD }}
    BUILD_PROVISION_PROFILE_BASE64: ${{ secrets.BUILD_PROVISION_PROFILE_BASE64 }}
    KEYCHAIN_PASSWORD: ${{ secrets.KEYCHAIN_PASSWORD }}
```

1. BUILD_CERTIFICATE_BASE64 and P12_PASSWORD

   1.1 This is the Base64 representation of the building certificate of a TYPE ‘Distribution’ it can be found in developer.apple.com/ ⇒ **Certificates, Identifiers & Profiles ⇒ Certificates**

   1.2 After we download the certificate we need to convert it to password protected p12 file this can be done from Keytool on an Apple machine or with `openssl` lib. instructions here

   1.3 Then we end up with a .p12 file that represents our signing cert and a password, now we need to convert that p12 file to a base64 string.

   ```
   openssl base64 -in [filename] | pbcopy // Mac, Linux
   openssl base64 -in [filename] | clip // Win
   ```

   1.4 At last we can now copy the values to Github Secrets.

2. BUILD_PROVISION_PROFILE_BASE64

   A provisioning profile **authorizes your app to use certain app services and ensures that you're a known developer developing, uploading, or distributing your app**
   . A provisioning profile contains a single App ID that matches one or more of your apps and a distribution certificate.

   Download a a provisioning profile from developer.apple.com/ ⇒ **Certificates, Identifiers & Profiles ⇒ Profiles**

   Convert it to base64 string and import that to GitHub Secrets under the BUILD_PROVISION_PROFILE_BASE64

3. KEYCHAIN_PASSWORD

   When building an IPA file, the Keystore and certificates are always removed from the build at the end. However, to ensure its security during the build process, we

utilize a randomly generated string (password) as a protective measure for the Keystore.

Select any password that you would like.

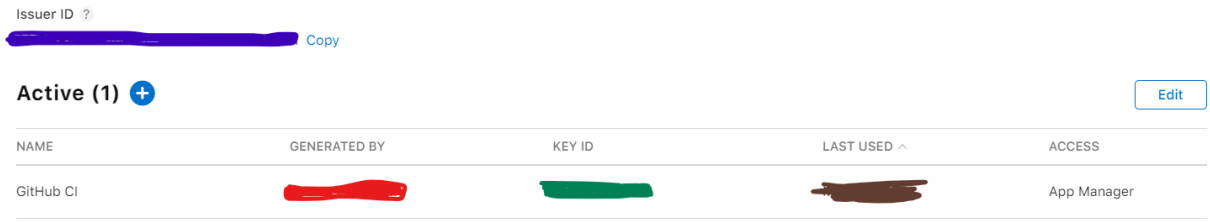
And that’s it for the build let's continue to Release.

Release IPA to TestFlight

For the release, we are using the apple-actions/upload-testflight-build@v1

```
- name: 'Upload app to TestFlight'
  uses: apple-actions/upload-testflight-build@v1
  with:
    app-path: build/ios/ipa/humhub.ipa
    issuer-id: ${{ secrets.APP_API_ISSUER_ID }}
    api-key-id: ${{ secrets.APPSTORE_API_KEY_ID }}
    api-private-key: ${{ secrets.APP_API_PRIVATE_KEY }}
```

Go to https://appstoreconnect.apple.com/access/api



APP_API_ISSUER_ID (Blue) and APPSTORE_API_KEY_ID (Brown) the last APP_API_PRIVATE_KEY is a String that you previously downloaded as a file copy it as it is to Github secrets together with the other 2 values.

Now, this is all that we should need to successfully deploy the app to Testflight with GitHub Actions with a current pipe.

Author: Primož Ratej Cvahte