Lab: Prepare a Flutter App for Publishing

Estimated time: 30 minutes

Welcome to this hands-on lab, where you will embark on the journey of preparing your app for submission to the Apple App Store and Google Play Store. This lab will guide you through gathering essential app information, managing iOS certificates and provisioning profiles, and setting up app signing for Android.

Objectives

After completing this lab, you will be able to:

- Gather and organize necessary app information for submission
- Understand and manage certificates and provisioning profiles for iOS app submission
- Set up and manage app signing for releasing an Android app

About Cloud IDE

This lab is designed to be completed using a Cloud IDE environment, where you will engage directly with code and configurations.

About Skills Network Cloud IDE

Skills Network Cloud IDE, based on Theia and Docker, offers a platform for hands-on labs across courses, facilitating learning in an interactive setting.

Important note about this lab environment

Each session in this lab environment starts fresh upon connection. Ensure to plan your activities to complete these labs in a single session to prevent data loss.

Step 1: Gathering app information

Embarking on the app store submission journey begins with collecting essential details about your app. This is common for both Apple Store and Google Play Store. This first step is akin to packing your suitcase for an international trip—every item needs a purpose, and forgetting something crucial can complicate your journey. Let's imagine you are building a travel application that provides users with important tips and considerations when embarking on a vacation.

1. Let's create a basic flutter application first using the `flutter` command.

   ```
   flutter create hello_world
   ```

   If you already have a personal app or an app from previous labs, you can take these steps on that as well.

2. **Delve into the core of your app's identity**:

   - **App name**: Consider a name like "World Explorer," which instantly evokes a sense of adventure and discovery. Your app's name is the first thing users will see, so make it memorable and relevant.
   - **Description**: Craft a description that tells a compelling story. For "World Explorer," you might write, "Embark on thrilling adventures with World Explorer! Discover hidden gems, book unique experiences, and create memories that last a lifetime. Your journey begins here."
   - **Keywords**: Choose keywords that potential users might enter when searching for an app like yours. For a travel app, consider "travel, adventure, explorer, bookings, world tour." These keywords help your app appear in relevant search results, increasing its visibility.
   - **Privacy policy URL**: Provide a link such as "yourdomain/privacy." This assures users that their data is handled with care, fulfilling legal requirements and building trust.
   - **Support URL**: Offer a pathway for support with a link like "yourdomain/support." This helps users resolve issues and find more information, enhancing their experience and satisfaction with your app.

3. **Prepare graphical assets with precision and creativity**:

   This step is like creating a visual brochure for your app. Design icons, screenshots, and promotional graphics that not only meet the technical specifications of app stores but also allure and engage your potential users. Think of these visuals as the decor of your app's storefront. This is what pulls a casual user into your application!

   - **Core visual elements**: Develop and upload key graphic assets for your app store listing, including the app icon, feature graphic, and promotional video. Adhere to the specified graphic requirements for each element, such as dimensions, file format, and color depth.

   - **Consistency in branding**: Achieve a consistent brand experience across all visual assets by using elements, styles, and color themes that complement each other. This approach helps reinforce your app's identity and enhances user recognition.

- **Design considerations**: When designing graphics, minimize the use of text to keep visuals clean and focused. Design with scalability in mind to ensure graphics look appealing on various device screens. Place crucial visual elements centrally within the graphics to prevent them from being cropped on different displays.
- **Promotional video**: Include a promotional video, hosted on YouTube, to highlight your app's key features and engage potential users effectively. A well-crafted video can significantly boost user interest by showcasing the app's functionality and user experience.
- **Compliance with policies**: Before uploading your graphics, ensure they comply with the app store's Impersonation and Intellectual Property policies to avoid potential issues with copyright and trademark infringement.

4. **Strategize app pricing and regions with a global mindset**:

- **Free**: Apps labeled as "free" are available to download at no cost to the user. These apps often rely on other revenue streams such as advertising, in-app purchases, or selling user data (in accordance with privacy laws). Free apps can attract a large user base quickly due to the lack of a download barrier.
- **Paid**: Paid apps require users to make a purchase before downloading the app. The upfront cost can act as a barrier to entry, which might limit the initial user base but can ensure that the users are genuinely interested or see value in the app. Revenue is generated directly through these purchases, providing a straightforward business model.
- **Freemium**: Freemium apps are available for free download but include optional in-app purchases or subscriptions that unlock additional features, content, or capabilities. This model allows users to engage with the app at no cost initially and then pay for enhanced functionality or content if they find value in it. Freemium apps can effectively balance attracting a large user base with the potential for significant revenue from committed users.

Step 2: Managing certificates and provisioning profiles for iOS

Managing certificates and provisioning profiles is like preparing your official documents before an international trip. These components are crucial for ensuring that the iOS platform recognizes you as a legitimate developer and grants your app the necessary permissions to function properly on user devices.

Enrolling in the Apple Developer Program is required to generate certificates for an iOS app. This involves creating an account and purchasing an annual subscription to access tools like certificates, provisioning profiles, and App Store Connect. While we provide a step-by-step guide, completing this part of the lab is optional.

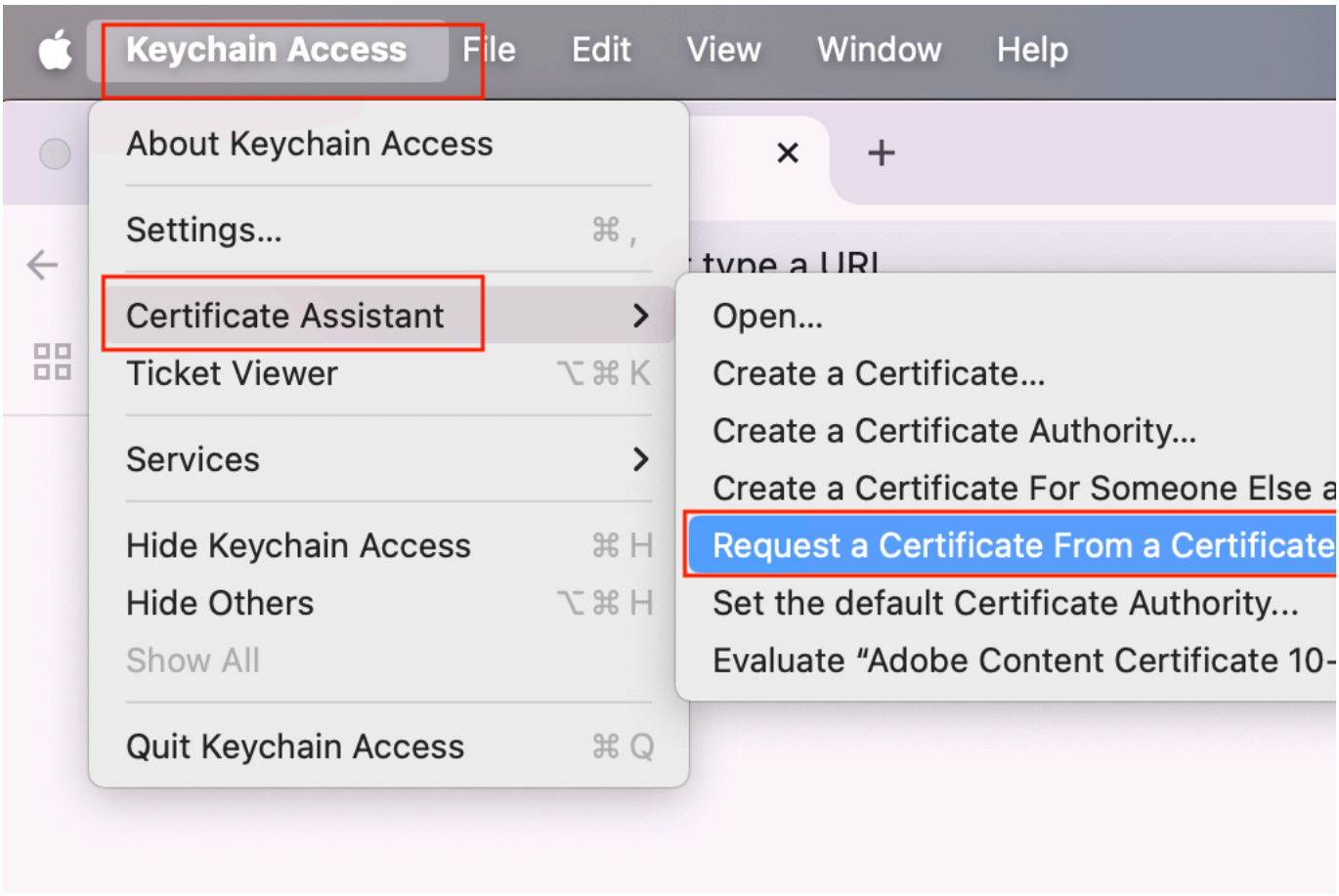
Optional - generate iOS certs

1. **Understanding certificates and provisioning profiles**:

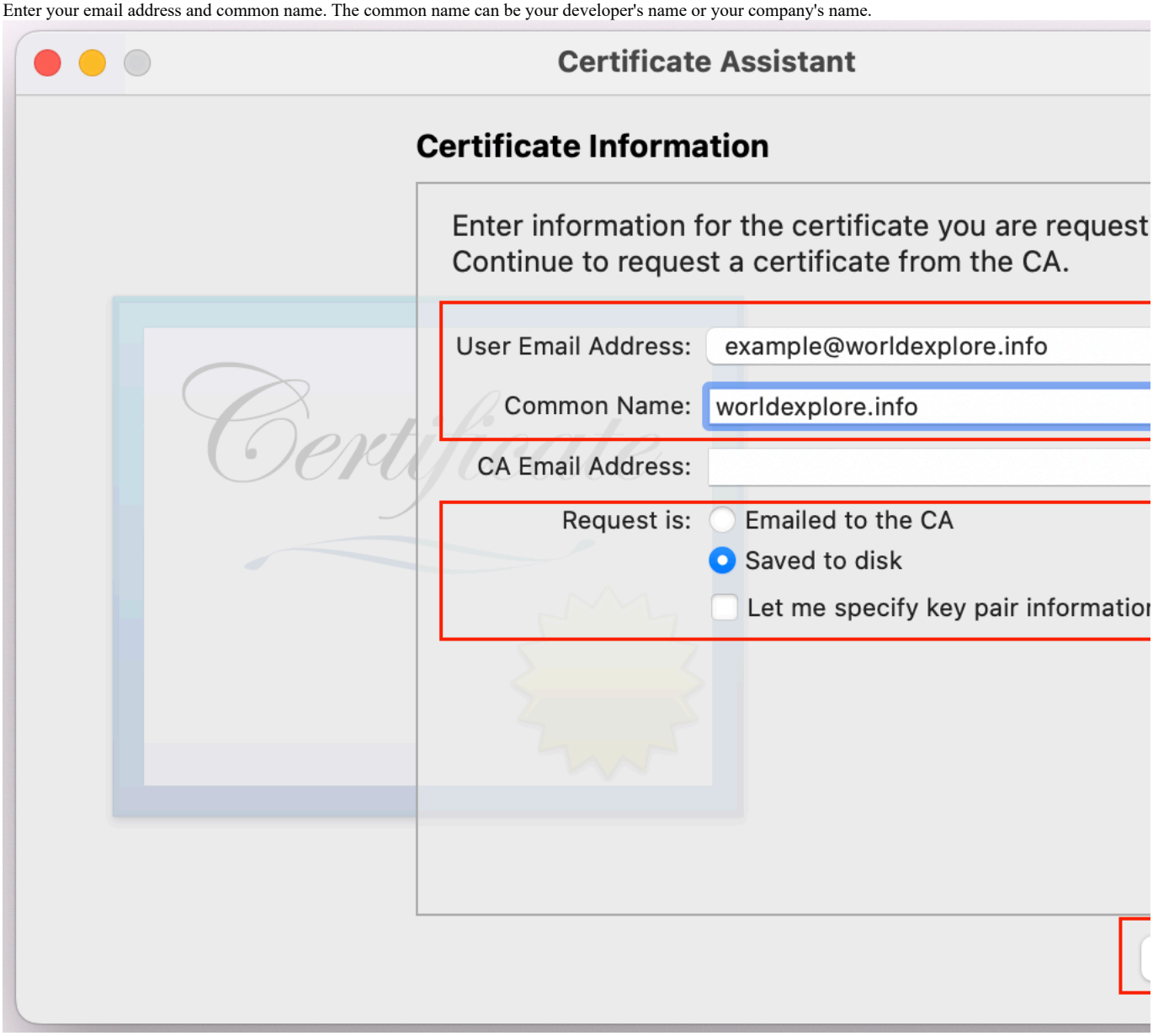
- **Certificates**: These digital certificates verify your identity as a developer and your authorization to submit apps to the App Store. You will need a development certificate for testing your app and a distribution certificate for releasing it.
- **Provisioning profiles**: These profiles are essentially permissions slips that tell your app what services it can access and on which devices it can run during testing. They bind your app's ID to your development certificate and list the devices authorized for testing.

2. **Create a certificate signing request (CSR)**:

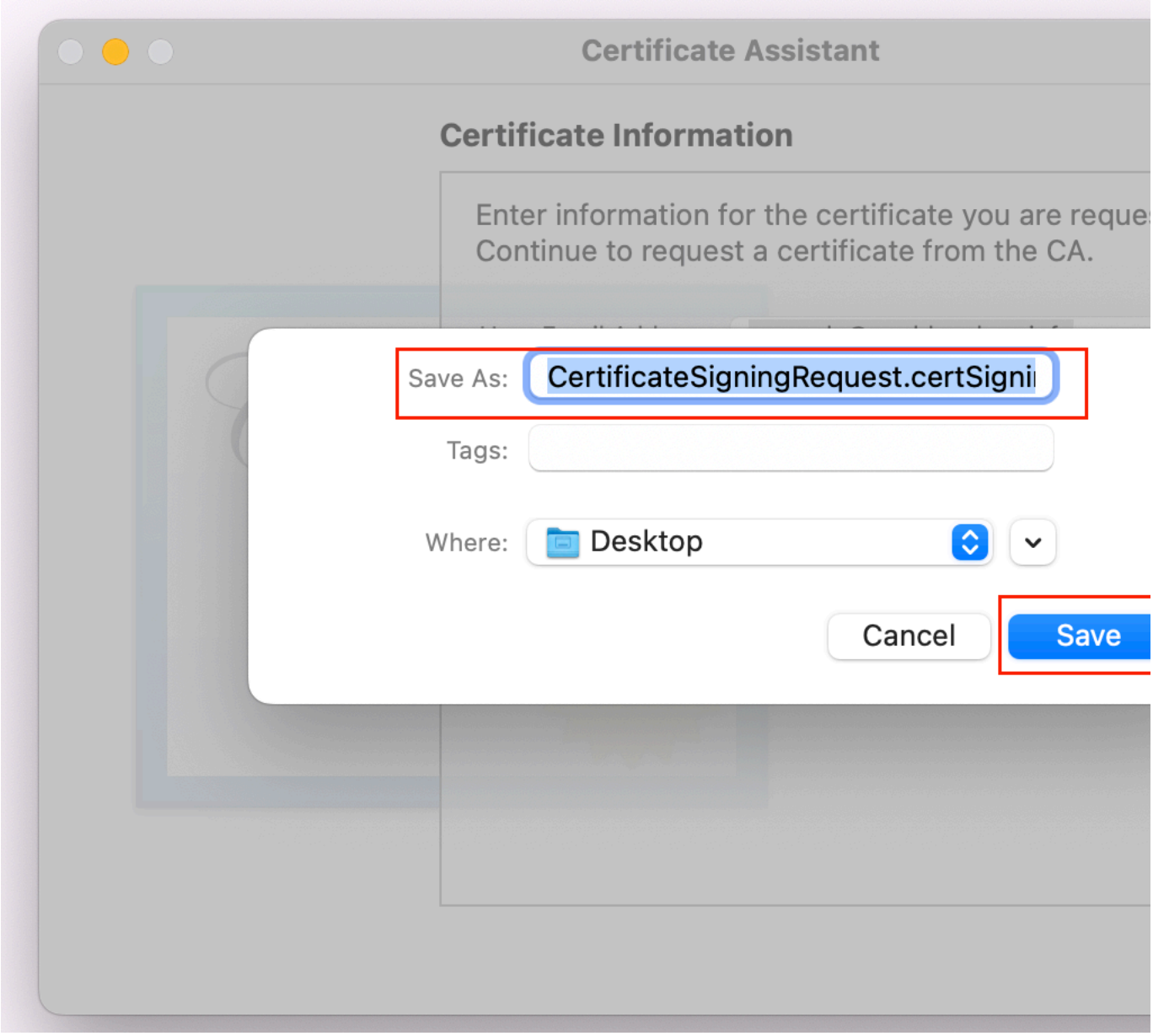
- Open the **Keychain Access** tool on your Mac.
- Navigate to **Keychain Access > Certificate Assistant > Request a Certificate From a Certificate Authority**.


Keychain Access
File
Edit
View
Window
Help
About Keychain Access
Settings... ⌘ ,
Certificate Assistant
Ticket Viewer ⌥⌘K
Services
Hide Keychain Access ⌘H
Hide Others ⌥⌘H
Show All
Quit Keychain Access ⌘Q
Open...
Create a Certificate...
Create a Certificate Authority...
Create a Certificate For Someone Else a
Request a Certificate From a Certificate
Set the default Certificate Authority...
Evaluate "Adobe Content Certificate 10-

- Enter your email address and common name. The common name can be your developer's name or your company's name.



- Save the CSR to disk.



3. **Generating certificates**:
   - Log in to the Apple Developer Portal.
   - Navigate to **Certificates, Identifiers & Profiles**.
   - Under **Certificates**, select + to add a new certificate.
   - Choose the appropriate certificate type (iOS App Development for testing, iOS Distribution for App Store distribution), and upload your CSR.
   - Download the generated certificate and double-click it to install in your Keychain.
4. **Create an app ID**:
   - In the Apple Developer Portal, navigate to **Identifiers** > +.
   - Select **App IDs** as the type, enter a description, and specify an explicit App ID or use a wildcard App ID if you plan to use this ID for multiple apps.
5. **Generating and installing provisioning profiles**:
   - Still in the Apple Developer Portal, go to **Provisioning Profiles** > +.
   - Select the type of provisioning profile you need (development or distribution).
   - Associate the profile with the App ID you created, select the certificates to include in this profile, and define the devices for a development profile.
   - Download the provisioning profile and double-click it to install it in Xcode.
6. **Use the provisioning profile in Xcode**:
   - Open your project in Xcode.
   - Navigate to the project settings, then to the **Signing & Capabilities** section.
   - Select the correct provisioning profile from the list to ensure your app is signed with the correct credentials.

By carefully following these steps, you can ensure that your iOS app is set up correctly for development and distribution, aligning with Apple's requirements for app submissions. These preparations are essential to ensure a smooth passage through app review and onto users' devices.

Step 3: Setting up app signing for Android

1. **Decide on a signing method**:
   - **Google Play app signing**: Recommended for most developers as it offers enhanced security by Google managing your app signing keys. It also simplifies the update process and prevents loss of keys.
   - **Manage your own keys**: Choose this if you need to retain control over your keys for specific reasons. This method requires careful management and secure storage of your keys.

2. **Generate a keystore file**:
   - Open your terminal.
   - Run:

     ```
     keytool -genkey -v -keystore release-key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias key
     ```

   - This command generates a secure file that stores your keys. You will be asked some identifying questions in the process. Here is an example:

     ```
     Enter keystore password:
      Enter the distinguished name. Provide a single dot (.) to leave a sub-component empty or press ENTER to use the default value in braces.
      What is your first and last name?
        [Unknown]:  first last
      What is the name of your organizational unit?
        [Unknown]:  OU
      What is the name of your organization?
        [Unknown]:  Tech
      What is the name of your City or Locality?
        [Unknown]:  Montreal
      What is the name of your State or Province?
        [Unknown]:  Quebec
      What is the two-letter country code for this unit?
        [Unknown]:  CA
      Is CN=first last, OU=OU, O=Tech, L=Montreal, ST=Quebec, C=CA correct?
        [no]:  yes
      Generating 2,048 bit RSA key pair and self-signed certificate (SHA384withRSA) with a validity of 10,000 days
              for: CN=first last, OU=OU, O=Tech, L=Montreal, ST=Quebec, C=CA
      [Storing release-key.jks]
     ```

3. **Configure signing in build.gradle**:
   - Open `/home/project/hello_world/android/app/build.gradle`.
   - Add the following signing configuration under the `android` block. The complete code is hidden for brevity.

     ```
     android {
         ...
         signingConfigs {
             release {
                 keyAlias 'key'
                 keyPassword 'your-key-password'
                 storeFile file('path/to/your/release-key.jks')
                 storePassword 'your-keystore-password'
             }
         }
         buildTypes {
             release {
                 signingConfig signingConfigs.release
             }
         }
         ...
     }
     ```

- Replace `your-key-password` and `your-keystore-password` with the actual passwords. Ensure the path to your keystore file is correct.

Conclusion

Congratulations on navigating the initial stages of preparing your app for the App Store and Play Store! You've packed your "suitcase" correctly, prepared your "travel documents," and even planned your "itinerary" for the app submission journey. These foundational skills are crucial for a successful app launch and will guide you as you continue to develop and refine your applications.

Author(s)

Skills Network