Clase 6

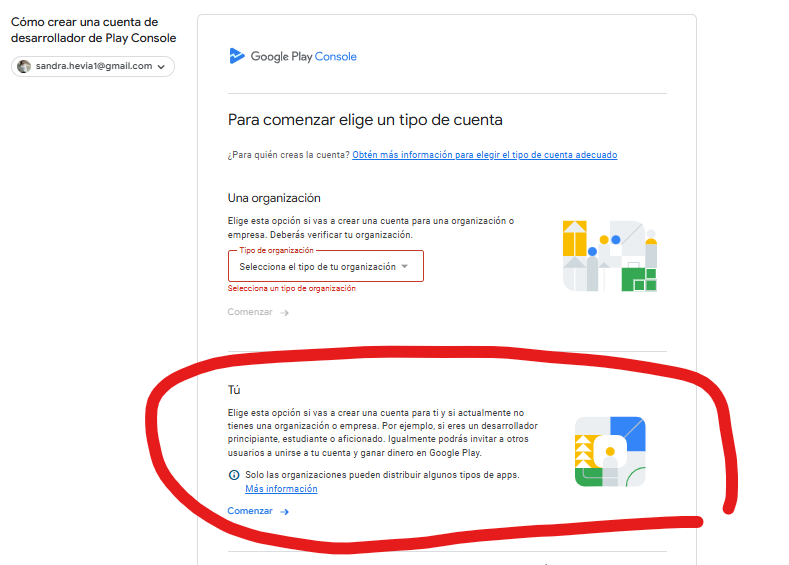
Ya teniendo una aplicación funcional, debemos comenzar el proceso para su distribución. Bien podríamos hacerlo a la antigua de persona n persona y no hay problema. Pero si queremos llegar a mas usuarios, debemos registrar nuestra aplicación en una tienda de apps. En este caso trabajaremos con Google play.

Vamos a la pagina de Google play:

<https://play.google.com/console/u/0/signup>

y de una vez tendremos para crear una cuenta. NOTA: la cuenta tiene un costo de un pago único de 25$

elegimos la opcion de hacer una cuenta personal:

­

Luego nos indicaran los requisitos:

1. Correo electrónico para la cuenta.
2. Número de teléfono de contacto y correo de contacto para Google.
3. Forma de pago (tarifa única de 25$).

Llenas la información que te van pidiendo…

Al final, cuando vayas a hacer el pago, asegúrate de tener el dinero y la tarjeta lista (ya que si hay un error al momento de hacer el pago, tardara hasta 15 dias en solucionarse).

Una vez se tenga la cuenta de desarrollador, debes verificar tu identidad con una foto tuya y otra de tu documento de identidad (este proceso puede durar 72 h).

Listo todo lo anterior, para poder subir tu app, debe ser una versión firmada. Para ello seguiremos los siguientes pasos:

1. Generar el keystore:
   1. En el CMD, ve hasta la carpeta de tu proyecto y luego ingresa lo siguiente:

keytool -genkey -v -keystore nombreApp.keystore -keyalg RSA -keysize 2048 -validity 10000 -alias aliasApp

(lo rojo es lo que se puede modificar)

Luego te va a pedir una clave y otros datos que deberás guardar en un lugar seguro.

1. Obtener el .aab:
   1. Tenemos que compilar en modo reléase:

cordova build android –release

* 1. Copia la ruta que te genera y te indica donde se guardo el archivo .aab

1. Firmar:
   1. Usamos el siguiente comando:

jarsigner -keystore nombre -storepass clave -keypass clave C:\xampp\htdocs\Github\MataNotas\platforms\android\app\build\outputs\bundle\release\app-release.aab alias

nombre: es el nombre completo del archivo keystore que acabamos de crear.

clave: es la clave que ingresaste al crear el keystore (es la misma en los dos campos).

Alias: el alias del keystore.

Ruta: la ruta completa que debimos copiar al compilar la app.

1. Verificar fima:

jarsigner -verify -verbose -certs C:\Users\hp\Music\reactApp\platforms\android\app\build\outputs\bundle\release\app-release.aab

Ruta: la ruta completa que debimos copiar al compilar la app.

Puede que aparezca un error con respecto al tiempo, si preocupación, no afecta, pero también puedes ingresar lo siguiente:

1. Añadir marca de tiempo:

jarsigner -keystore nombre -storepass clave -keypass clave -tsa http://timestamp.digicert.com C:\xampp\htdocs\Github\MataNotas\platforms\android\app\build\outputs\bundle\release\app-release.aab alias

ya estamos listos… ahora vamos a Google play console…

1. Creamos una nueva app: (nota: este proceso es un poco largo y puede tomar tiempo)
   1. Hay que tener listo:
      1. Descripción de la app
      2. Un link con las políticas de privacidad (<https://app.privacypolicies.com/> )
      3. Icono de la app
      4. Captures de como se ve la app en diferentes dispositivos (por lo menos 1)
      5. 1 imagen de 1024x500px donde se muestren las principales funcionabilidades de la app.
      6. Público objetivo.
      7. 20 personas que deseen ser nuestros verificadores.
   2. Hay que llenar toda la información que nos piden.

Creada la app, podemos trabajar diferentes tipos de pruebas:

1. Interna: la más básica, donde participa un grupo de verificadores y no tiene mayor incidencia en tu cuenta.
2. Cerrada (Alpha): solo participa un grupo de verificadores y es el inicio para poder entrar en producción. Nota: la 1ra vez que subes una app, debes pasar mínimo 14 días de pruebas cerradas con por lo menos 20 verificadores para poder acceder a producción).
3. Abierta (beta): prueba opcional, cualquiera podría inscribirse y participar.

Cada prueba, de ser satisfactoria, te da la opción de accender esa versión al siguiente escalafon, siendo producción el ultimo (ya la aplicación en la tienda, disponible para todos).

Realmente, Google play console es muy intuitivo y claro, de hecho, cuando Google hace las revisiones te da consejos de colores, contrastes y tamaños ideales. Lo mas complicado es hacer la app y tenerla lista para compartirla en Google.

Lo que si hay que tener cuidado es con la descripción de la app y los botones o funciones que ponemos, ya que si en la descripción decimos que se pueden agrupar elementos y no es asi, Google no aceptara esa versión de la app. Así mismo pasa cuando tenemos botones que no hacen nada.

React con cordova (también pueden revisar el siguiente link <https://medium.com/@pshubham/using-react-with-cordova-f235de698cc3> )

Si tenemos algo desarrollado con react, por supuesto que podemos convertirlo en una app.

1. Fusionar los proyectos:
   1. Copie los directorios src/ y public/ de la carpeta del proyecto de react a la carpeta del proyecto de cordova
   2. También es necesario fusionar los archivos package.json. Copia las claves “scripts”, “dependencies” y “browserList” del proyecto de react/package.json a proyecto de cordova /package.json.
   3. Dado que package.json se actualizó manualmente, debemos asegurarnos de que las dependencias estén instaladas: entramos a la carpeta del proyecto de cordova con el CMD y corremos: npm install.
   4. Buscamos public/index.html y agregamos:

<meta http-equiv="Content-Security-Policy" content="default-src 'self' data: https://ssl.gstatic.com; script-src 'self' 'unsafe-inline' 'unsafe-eval'; style-src 'self' 'unsafe-inline'; media-src \*; img-src 'self' data: content:;">

        <meta name="format-detection" content="telephone=no">

        <meta name="msapplication-tap-highlight" content="no">

        <meta name="viewport" content="initial-scale=1, width=device-width, viewport-fit=cover">

        <meta name="color-scheme" content="light dark">

* 1. Agregue la siguiente etiqueta de script justo antes de </body> en public/index.html

<script src="cordova.js" tipo="texto/javascript"></script>

* 1. Configurar react dom para que se cargue después del evento ondevice ready de cordova:



* 1. agregar una propiedad de “página de inicio” en package.json como se menciona a continuación: "homepage": "./"
  2. definir ganchos en el archivo config.xml del proyecto. Agregue la siguiente entrada en <widget>:

<hook type="before\_prepare" src="scripts/prebuild.js" />

* 1. Ahora que el gancho está definido, crea una carpeta de scripts dentro del directorio del proyecto. Dentro de la carpeta de scripts, crea también un archivo prebuild.js .
  2. El archivo creado tendrá dentro:

const path = require('path');

const { exec } = require('child\_process');

const fs = require('fs');

const { promisify } = require('util');

// Promisificar métodos de fs para usar con async/await

const fsRename = promisify(fs.rename);

const fsRmDir = promisify(fs.rmdir);

const fsUnlink = promisify(fs.unlink);

const fsReaddir = promisify(fs.readdir);

const fsStat = promisify(fs.stat);

async function deleteFolderRecursive(directory) {

    const entries = await fsReaddir(directory);

    await Promise.all(entries.map(async (entry) => {

        const fullPath = path.join(directory, entry);

        const stats = await fsStat(fullPath);

        if (stats.isDirectory()) {

            await deleteFolderRecursive(fullPath);

        } else {

            await fsUnlink(fullPath);

        }

    }));

    await fsRmDir(directory);

}

function renameOutputFolder(buildFolderPath, outputFolderPath) {

    console.log(`Renombrando carpeta de ${buildFolderPath} a ${outputFolderPath}`);

    return fsRename(buildFolderPath, outputFolderPath)

        .then(() => '¡Construcción exitosa!')

        .catch(err => Promise.reject(err));

}

function execPostReactBuild(buildFolderPath, outputFolderPath) {

    return new Promise(async (resolve, reject) => {

        try {

            if (fs.existsSync(buildFolderPath)) {

                if (fs.existsSync(outputFolderPath)) {

                    console.log(`Eliminando carpeta existente en ${outputFolderPath}`);

                    await deleteFolderRecursive(outputFolderPath);

                }

                const result = await renameOutputFolder(buildFolderPath, outputFolderPath);

                resolve(result);

            } else {

                reject(new Error('La carpeta de construcción no existe'));

            }

        } catch (error) {

            reject(error);

        }

    });

}

module.exports = () => {

    const projectPath = path.resolve(process.cwd(), './node\_modules/.bin/react-scripts');

    return new Promise((resolve, reject) => {

        console.log(`Ejecutando comando de construcción: ${projectPath} build`);

        exec(`${projectPath} build`, (error) => {

            if (error) {

                console.error(error);

                reject(error);

                return;

            }

            execPostReactBuild(path.resolve(\_\_dirname, '../build/'), path.join(\_\_dirname, '../www/'))

                .then((s) => {

                    console.log(s);

                    resolve(s);

                })

                .catch((e) => {

                    console.error(e);

                    reject(e);

                });

        });

    });

};

* 1. Agregamos la plataforma
  2. Listo, compilamos