Control de canvis

DATA	CANVI	DESCRIPCIÓ	AUTOR
04/10/18		Creació document	THR
25/10/18	Actualització	Modificació directori config.key en Android i iOS	THR
31/10/18	Actualització	Exemples format fitxers	THR
13/11/18	Actualització	Exemple format tag commit	THR
21/11/18	Novetat	Generació automàtica del versionCode	THR
8/2/19	Actualització	Millores integració continua	CRT
15/04/19	Actualització	Integració key crashlytics Indicació de punts obsolets	THR
31/07/19	Actualització	Adaptació key crashlytics	ECR
18/02/20	Actualització i novetat	Obsolescència de fabric i GA Warnings Android	HGV
19/02/2020	Actualització i novetat	Adaptació de l'arxiu "CHANGELOG.txt", millora de l'arxiu gitignore, migració de Fabric a Firebase, nous requisits de test	RGR
14/07/2020	Novetat i actualització	Pujada Dsym i actualització nomenclatura buildNumber	THR
18/06/2021	ESPOSAMOT-2978	Creació AppBundle per les apps d'Android	ECR
03/12/2021	ESPOSAMOT-3623	Actualització FASE 1 amb nota important del tag	ECR

08/02/2022	**ESPOSAMOT-3791**	**Actualització documentació mòdul comú: afegida variable de configuració amb la URL de l'entorn de desenvolupament**	**JPG**
14/02/2022	**ESPOSAMOT-3668**	**Actualitzar l'obtenció del codi del projecte a través del TAG en el repositori**	**ECR**
14/07/2022	**OSAM-329** **OSAM-330**	**Actualització de les validacions: fer referència als requeriments**	**ECR**
14/10/2022	**OSAM-10425**	**Actualitzar les llibreries que es validen en les validacions**	**ECR**
05/12/2022	**OSAM-11203**	**Afegir nova configuració per als projectes Flutter**	**ECR**
08/06/2023	**OSAM-12318**	**Executar Jobs Jenkins per a versions PRE**	**ECR**
21/06/2023	**OSAMWF-5093**	**Refactor de tot el document: reordenar tots els apartats**	**ECR**

Manual Integració Continua per a proveïdors

Contingut

Objectiu

L'objectiu d'aquest document és definir el processos que es duran a terme en la integració continua de la OSAM, la qual involucrarà als proveïdors.
En els següents apartats es definiran les fases d'aquesta implantació i els requeriments que hauran de complir els proveïdors per poder-se integrar amb els desenvolupaments realitzats per part de l'equip tècnic de la OSAM.

La informació està orientada al Framework de Flutter i els projectes natius d'Android i iOS, però en la majoria de situacions també aplica, no de la mateixa manera, a la resta de Frameworks disponibles (react, ionic...).

Fase 1: Obtenció codi font del Gitlab

Des de l'IMI, s'han de crear els usuaris necessaris per cada proveïdor i donar-li accés a la branca "jenkins" del projecte corresponent al repositori del Gitlab.

Un cop el proveïdor tingui una versió disponible per distribuir i/o iniciar el procés OSAM, haurà de fer push a la branca "jenkins" del repositori. Aquesta branca serà la que utilitzarà el Jenkins per realitzar el procés de compilació i distribució.

A més de pujar el codi a la branca "jenkins", s'ha d'afegir el TAG al commit corresponent. Aquest tindrà el següent format: "**nom_projecte_X.Y.Z_nom_proveïdor**".
On:

- **nom_projecte**: el mateix indicat en el **BundleID** o **ApplicationID**.
  Ex: per al projecte *Barcelona a la butxaca* seria "butxaca" (cat.bcn.**butxaca**)
- **X.Y.Z**: versió de l'app.
  Ex: 1.0.2
- **nom_proveïdor**: el nom del proveïdor. Ex: worldline

En el nostre exemple quedaria de la següent manera: **butxaca_1.0.2_worldline**

IMPORTANT: és molt important informar aquest TAG a la fitxa de Palàntir, ja que s'utilitza per obtenir el codi del projecte i generar les versions.

Fase 2: Modificacions codi font

A continuació s'explicarà la configuració necessària que han de tenir els projectes. Com a l'OSAM arriben moltes apps, si els aspectes generals que han de tenir totes les apps estan normalitzats, ens permet treballar de manera més àgil.

Codi font de les apps

1. **Android**

El projecte d'estar desenvolupat amb Android Studio (o IntelliJ) i la darrera versió estable, ja que aquest és el IDE instal·lat a la màquina que executa el Jenkins.

El codi font de la app ha d'estar al directori /app i aquest és preferible que no tingui un nom customitzat.

Build.gradle

Al fitxer "build.gradle" del projecte s'haurà de crear un productFlavor amb el nom "jenkins".

NOTA IMPORTANT: el productFlavor "jenkins" ha de tenir la mateixa informació que el productFlavor que s'utilitzi, si hi ha, de producció, ja que la compilació es farà amb aquest productFlavor. Els projectes hauran de tenir-ho en compte per modificar si utilitzen comprovacions que necessitin el nom del productFlavor.

VersionCode i VersionName:

Per poder actualitzar el versionCode i VersionName automàticament amb el format indicat als requeriments de la OSAM, cal que el proveïdor:

1. Creï/actualitzi al directori arrel de l'app el fitxer jenkins.properties declarant la variable APP_VERSION amb el versionName. Aquest arxiu també ha de declarar el nom de l'app com a APP_NAME ja que jenkins necessita aquesta variable en el job. El nom de l'app ha d'anar entre cometes invertides:

```
APP_NAME=`On Aparcar Residents Android`
APP_VERSION=1.3.1
```

2. inclogui les següents funcions al fitxer de build.gradle:

```
def getDate() {
        return new Date().format('yyyyMMddHH').toInteger()
}

def getVersionNameFromProperties() {
        def jenkinsProperties = new Properties()
        jenkinsProperties.load(new
FileInputStream(rootProject.file('jenkins.properties')))
        return jenkinsProperties['APP_VERSION']
}
```

3. Posteriorment les podrà utilitzar de la següent manera:

```
defaultConfig {
        applicationId "cat.bcn.nomapp"
        minSdkVersion 19
        targetSdkVersion 28
        versionCode getDate()
        versionName getVersionNameFromProperties()
}
```

Git Ignore

El contingut del fitxer .gitignore de la carpeta arrel del projecte ha de ser aquest (pot ser que alguna app en particular hagi d'afegir més arxius a la llista):

Android	React
AndroidStudio, Settings & build *.apk *.ap_ *.dex *.class bin/ gen/ local.properties Thumbs.db .DS_Store *.iml .idea .gradle .navigation build/ captures/ output.json obj/ .externalNativeBuild *.qicon	**Yarn** yarn-error.log yarn.lock **Watchman** .watchmanconfig **Buck** .buckd buck-out **Node** node_modules *.log .nvm /bots/node_modules/ package-lock.json npm-debug.log

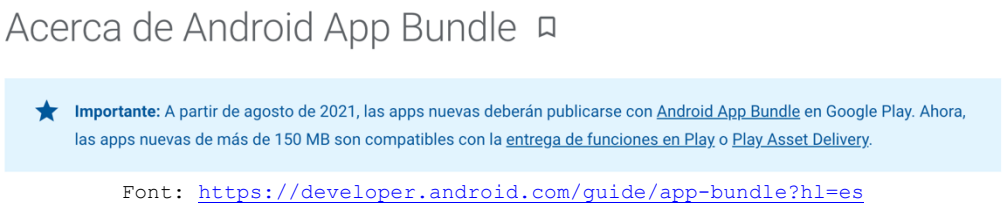
Aquests fitxers a afegir al .gitignore s'han d'entendre com uns mínims sense perjudici dels que el proveïdor necessiti.

Firebase

El proveïdor ha de tenir a l'arrel de la carpeta "/app/" del projecte el fitxer "google-service.json", corresponent al servei de Firebase. El json serà el de desenvolupament i en el moment de la compilació es substituirà per el de producció.

App Bundle

Per a noves apps, i per a actualitzacions en el futur, GooglePlay demana obligatòriament des d'agost de 2021 la pujada de les apps en el format d'app bundle:



Font: https://developer.android.com/guide/app-bundle?hl=es

Per poder pujar la versió de l'app en format Bundle, s'haurà d'afegir al build.gradle de l'app (no del projecte) la següent configuració:

```
        android {
            bundle {
                language {
                    // This property is set to true by default.
                    // You can specify `false` to turn off
                    // generating configuration APKs for language resources.
                    // These resources are instead packaged with each base and
                    // feature APK.
                    // Continue reading below to learn about situations when an app
                    // might change setting to `false`, otherwise consider leaving
                    // the default on for more optimized downloads.
                    enableSplit = false
                }
                density {
                    // This property is set to true by default.
                    enableSplit = false
                }
                abi {
                    // This property is set to true by default.
                    enableSplit = false
                }
            }
}
```

2. iOS

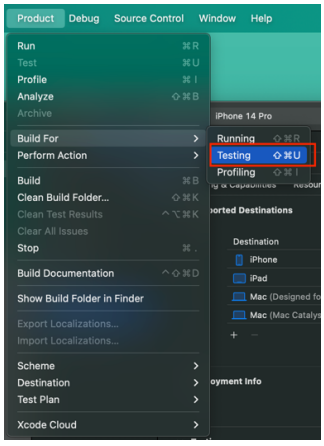
En el cas d'iOS, la IDE instal·lada és XCode, i el codi ha d'estar desenvolupat amb la darrera versió disponible i estable. D'aquesta manera es podrà compilar qualsevol versió d'Objective C i versions de Swift 3 o superiors.

Build Configuration

El proveïdor ha de crear la Build Configuration "jenkins", amb la qual es generarà la versió que s'entregarà a la OSAM i amb la que es compilarà per distribuir a l'AppStore.

SonarQube: targets Test/UITest

Quan s'executa l'anàlisi del SonarQube, cal que els targets de Tests i de UITests compilin i funcionin. Si aquests no es mantenen ni compilen, s'han de treure de la build de Test:



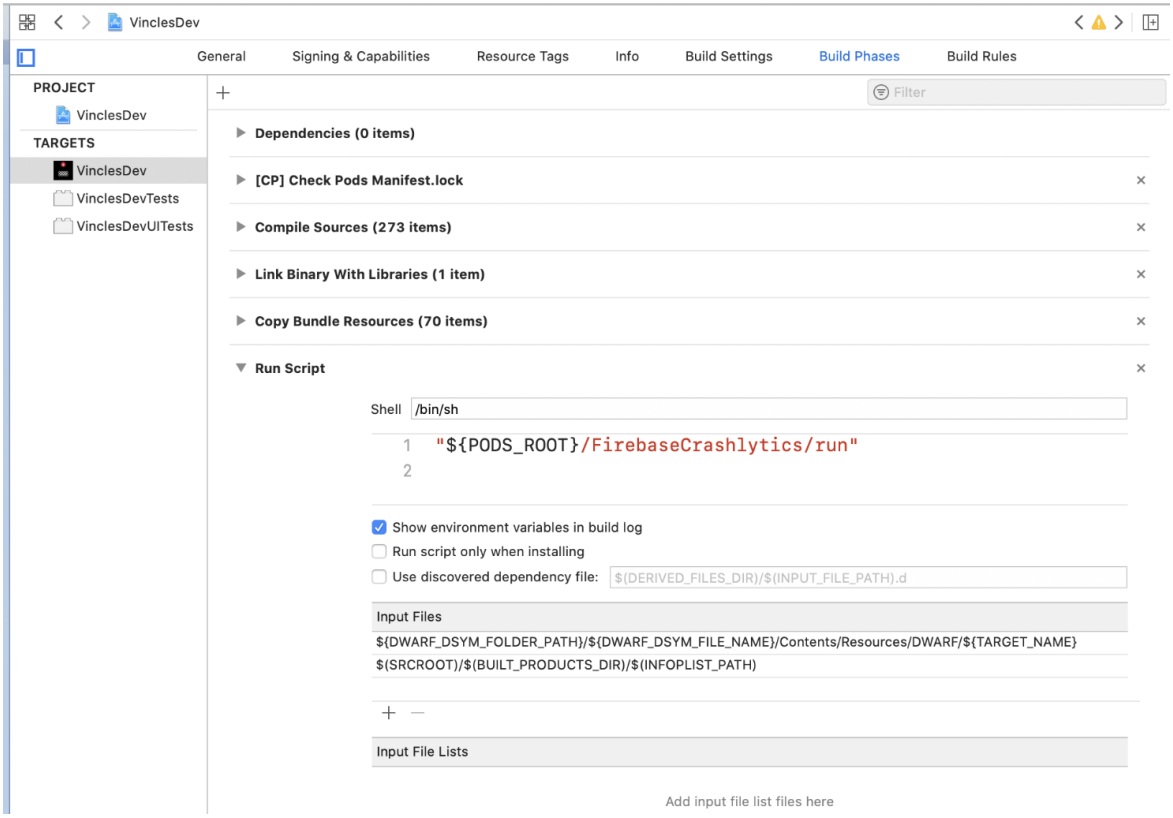
Firebase

En la correcta configuració de Firebase s'ha de tenir en compte:

- Fitxer de configuració
- DSYMs

El fitxer de configuració de Firebase "GoogleService-Info.plist" es posarà a l'arrel del projecte. El plist serà el del projecte de DEV/PRE amb el que es faci el desenvolupament

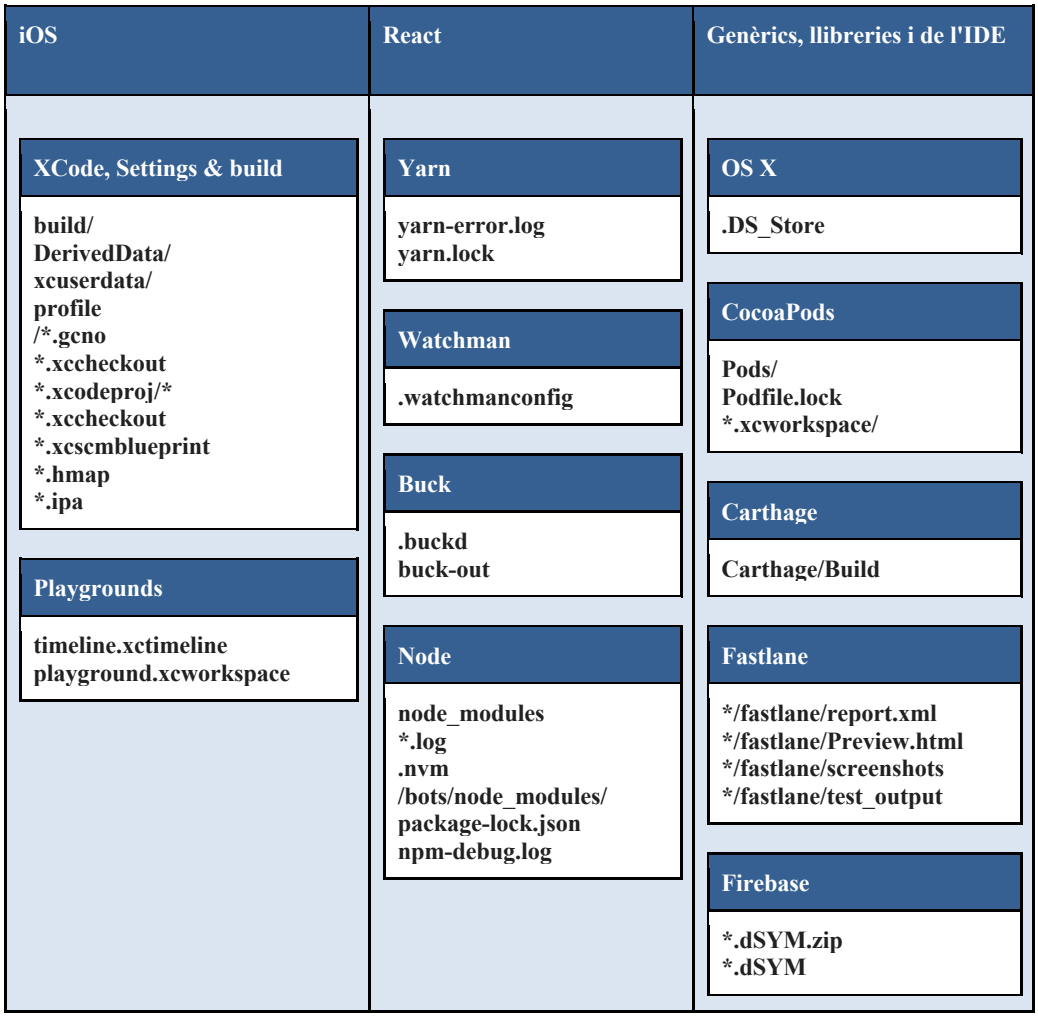
Per altra banda tenim els DSYMs. Per poder pujar-los automàticament a Firebase Crashlytics cal realitzar la següent configuració al projecte:



Per més informació s'ha de seguir la pròpia documentació proporcionada per Firebase: https://firebase.google.com/docs/crashlytics/get-deobfuscated-reports?hl=es-419&platform=ios

Git Ignore

El contingut del fitxer .gitignore de la carpeta arrel del projecte ha de ser aquest (pot ser que alguna app en particular hagi d'afegir més arxius a la taula):



iOS	React	Genèrics, llibreries i de l'IDE
XCode, Settings & build build/ DerivedData/ xcuserdata/ profile /*.gcno *.xccheckout *.xcodeproj/* *.xccheckout *.xcscmblueprint *.hmap *.ipa	**Yarn** yarn-error.log yarn.lock	**OS X** .DS_Store
Playgrounds timeline.xctimeline playground.xcworkspace	**Watchman** .watchmanconfig	**CocoaPods** Pods/ Podfile.lock *.xcworkspace/
	Buck .buckd buck-out	**Carthage** Carthage/Build
	Node node_modules *.log .nvm /bots/node_modules/ package-lock.json npm-debug.log	**Fastlane** */fastlane/report.xml */fastlane/Preview.html */fastlane/screenshots */fastlane/test_output
		Firebase *.dSYM.zip *.dSYM

Aquests fitxers a afegir al .gitignore s'han d'entendre com uns mínims sense perjudici dels que el proveïdor necessiti.

3. Flutter

Es realitzaran les mateixes accions que s'han comentat prèviament:

- Android: Build.gradle i App Bundle
- iOS: Build Configuration i Targets Tests/UITests

Firebase

En Flutter s'uneixen els dos mons d'Android i iOS. S'haurien de seguir les mateixes passes que amb Android i la mateixa configuració que a iOS.

Informació més detallada a la documentació de Firebase:
https://firebase.google.com/docs/crashlytics/get-deobfuscated-reports?hl=es-419&platform=flutter

Git Ignore

Un cop més, a Flutter es junten els dos mons d'Android i iOS degut a que l'estructura del codi incorpora un projecte per a cada plataforma.
El contingut del fitxer .gitignore de la carpeta arrel del projecte ha de ser el que s'ha comentat per a Android, iOS i, a més:

Flutter

AndroidStudio o VisualStudio, Settings & build

```
*.class
*.log
*.pyc
*.swp
.DS_Store
.atom/
.buildlog/
.history
.svn/
.fvm/flutter_sdk
*.iml
*.ipr
*.iws
.idea/
.vscode/
**/doc/api/
**/ios/Flutter/.last_build_id
.dart_tool/
.flutter-plugins
.flutter-plugins-dependencies
.packages
.pub-cache/
.pub/
/build/
lib/generated_plugin_registrant.dart
app.*.symbols
app.*.map.json
!/packages/flutter_tools/test/data/dart_dependencies_test/**/.packages
```

Signar les apps

1. Android

En el fitxer "build.gradle" del mòdul de la app, abans de l'apartat "android", es posen les següents línies:

```
def keystorePropertiesFile = rootProject.file("keystore.properties")
def keystoreProperties = new Properties()
keystoreProperties.load(new FileInputStream(keystorePropertiesFile))
```

Això implica tenir un fitxer anomenat "keystore.properties" a l'arrel del projecte (al mateix nivell que el gradle.properties). L'estructura del "build.gradle" quedaria de la següent manera:

```
def keystorePropertiesFile = rootProject.file("keystore.properties")
def keystoreProperties = new Properties()
keystoreProperties.load(new FileInputStream(keystorePropertiesFile))

android {
          //compileSdkVersion, defaultConfig…
}
```

En el fitxer "build.gradle" del mòdul de la app, a l'apartat "android", s'ha d'afegir la configuració de signatura de l'app:

```
signingConfigs {
            release {
                keyAlias keystoreProperties['keyAlias']
                keyPassword keystoreProperties['keyPassword']
                storeFile file(keystoreProperties['storeFile'])
                storePassword keystoreProperties['storePassword']
            }
        }
```

En el fitxer "build.gradle" dels corresponents mòduls caldrà introduir al productFlavors perquè Jenkins pugui signar amb la keystore de producció:

```
jenkins {signingConfig signingConfigs.release}
```

El "productFlavors" quedaria de la següent manera:

```
productFlavors {
    jenkins {
        signingConfig signingConfigs.release
    }
}
```

En l'arrel del projecte s'haurà de crear el fitxer "keystore.properties" que contindrà les variables:

- storePassword=*****
- keyAlias=*****
- keyPassword=*****
- storeFile=*****

Finalment, el build.gradle de la app hauria de quedar amb aquesta estructura:

```
def keystorePropertiesFile = rootProject.file("keystore.properties")
def keystoreProperties = new Properties()
keystoreProperties.load(new FileInputStream(keystorePropertiesFile))

android {

    signingConfigs {
        release {
            keyAlias keystoreProperties['keyAlias']
            keyPassword keystoreProperties['keyPassword']
            storeFile file(keystoreProperties['storeFile'])
            storePassword keystoreProperties['storePassword']
        }
    }

    productFlavors {
        jenkins {
```

```
            //Veure NOTA IMPORTANT
                signingConfig signingConfigs.release
            }
        }
    }
```

Aquí ja s'afegirien les dependències que tingui cada projecte, buildTypes, applicationId, buildScript...

2. iOS

En el cas d'iOS el proveïdor haurà d'activar la compartició del scheme, per a que Jenkins pugui compilar el codi, i que no tingui activada l'opció de "automatically manage signing", ja que serà Jenkins qui signi les apps.

3. Flutter

Es realitzen les mateixes accions tant d'Android com d'iOS a les carpetes corresponents.

Fitxers de configuració

1. Android

En el directori **app/src/main/res/values/** s'ha de crear el fitxer "**config_keys.xml**" amb les variables "maps_key", i "common_module_endpoint". En el codi font s'hauran de referenciar a aquestes variables per utilitzar els seus valors. (Els valors seran els de desenvolupament). Un exemple:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="common_module_endpoint">https://dev-osam-modul-comu.dtibcn.cat</string>
    <string name="maps_key">valor_key</string>
</resources>
```

Nota: Per poder referenciar les variables d'aquest fitxer al AndroidManifest, s'ha de fer de la següent manera:

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="@string/maps_key" />
```

MapBox

En cas de que s'utilitzi MapBox en comptes de Google Maps, en comptes de tenir la variable "maps_key", s'haurà de posar la variable "`mapbox_access_token`" tal i com diu la documentació oficial de Mapbox per a Android a l'apartat "Configure your públic token": https://docs.mapbox.com/android/navigation/guides/get-started/install/

2. iOS

S'ha de crear el fitxer "**config_keys.plist**" al directori **"/res/config/**" del projecte. Aquest contindrà les variables, "maps_key" " i "common_module_endpoint". En el codi font s'hauria de referenciar a aquestes variables per utilitzar els seus valors. (Els valors seran els de desenvolupament). Un exemple:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>maps_key</key>
        <string></string>
        <key>common_module_endpoint</key>
        <string> https://dev-osam-modul-comu.dtibcn.cat</string>
    </dict>
</plist>
```

MapBox

En cas de que s'utilitzi MapBox en comptes de Google Maps, per carregar el token des del "config_keys.plist" és pot fer de la següent manera:

```
let configKeysPlist = Bundle.main.path(forResource: configKeysFile, ofType: "plist")!
if let configDict = NSDictionary(contentsOfFile: configKeysPlist) as? [String: String], let token = configDict["maps_key"] {
    return  token
}
return ""
```

Un cop s'aconsegueix el token, se li pot assignar a les propietats del mapa:

```
private func setupMapView() {
    let mapResourceOptions = ResourceOptions(accessToken: getMapBoxAccessToken())
    let cameraOptions = CameraOptions(center: CLLocationCoordinate2D(latitude: 41.3866352, longitude: 2.1649013), zoom: zoom)

    var mapInitOptions: MapInitOptions!
    if let url = URL(string: "https://osam.bcn.cat/apps/mapbox/bcn.json") {
        mapInitOptions = MapInitOptions(resourceOptions: mapResourceOptions, cameraOptions: cameraOptions, styleURI: StyleURI(url: url))
    } else {
        mapInitOptions = MapInitOptions(resourceOptions: mapResourceOptions, cameraOptions: cameraOptions)
    }
    mapView = MapView(frame: view.bounds, mapInitOptions: mapInitOptions)
```

Més informació a la documentació oficial de MapBox:
https://docs.mapbox.com/ios/maps/guides/install/#add-a-map

3. Flutter

Flutter té algunes diferències respecte les versions natives.

En Flutter no es poden carregar les dades de producció des del "config_keys". Per això en Flutter carreguem les dades en el fitxer ".env". Aquest fitxer estarà a l'arrel del projecte i tindrà les següents dades:

- COMMON_MODULE_URL: url base del mòdul comú (https://dev-osam-modul-comu.dtibcn.cat)
- MAPS_KEY: token del mapa, ja sigui de GoogleMaps o Mapbox

Per carregar aquesta informació es pot utilitzar el següent plugin de Flutter:
https://pub.dev/packages/flutter_dotenv
El procés de la integració continua ja s'encarrega de substituir aquest fitxer amb les dades de producció corresponents.

Validacions

1. Android

Abans de que comenci la compilació des del Jenkins, un script revisarà el codi. Totes les validacions que s'executin s'adjuntaran al cos de l'email del resultat de la compilació. Hi haruàn de tres tipus: s'informarà de quines validacions són correctes, quines són un "warning" (indicades amb el text "Atenció") i quines són un KO (indicades amb el text "**Error**").

Les validacions que es fan són les següents:

- Existeixen els fitxers /app/**build.gradle**, app/src/main/**AndroidManifest.xml**, **jenkins.properties, google-services.json** i **config_keys**
- El nom del paquet d'Android comença amb **cat.bcn.*** tal i com es demana en els requeriments de la OSAM (R3.2)
- El nom de la versió ha d'estar en format **X.Y.Z** (on X,Y,Z son nombres naturals) tal i com ho dicten els requeriments de les apps (R1.9)
- El codi de la versió es genera automàticament amb **Date().format('yyyyMMddHH').toInteger()** tal i com s'indica en aquest document
- La versió mínima del SDK ha de ser igual o major a la exigida per els requeriments de la OSAM (R3.1.1)
- La versió target de SDK ha de ser igual o major a la exigida per els requeriments de GooglePlay (R.3.1.1)
- S'ha implementat correctament la llibreria de l'ajuntament **modul_comu_osam.** És a dir, ha d'aparèixer al build.gradle amb la última versió tal i com s'informa en els requeriments de la OSAM (R1.7)
- Les dependències de Firebase estan implementades correctament tal i com es demana en els requeriments de la OSAM (R1.5):
    - Firebase Crashlytics
    - Firebase Analytics
    - Firebase Messaging

2. iOS

Abans de que comenci la compilació des de Jenkins un script revisarà el codi. Totes les validacions que s'executin s'adjuntaran al cos de l'email del resultat de la compilació. Hi haruàn de tres tipus: s'informarà de quines validacions són correctes, quines són un "warning" (indicades amb el text "Atenció") i quines són un KO (indicades amb el text "**Error**").
Les validacions que es fan són les següents:

- Existeixen els fitxers **GoogleServices-Info.plist** i **config_keys**
- El projecte té la versió **mínima de l'sdk** exigida per els requeriments de la OSAM (R2.1)
- El projecte té correcte el format del **bundleID** (cat.bcn.nom_app) tal i com es demana en els requeriments de la OSAM (R2.2)

- El codi de versió ha de tenir el format **yyyyMMdd** o **yyyyMMddHH** tal i com es demana en els requeriments de la OSAM (R2.2)
- El nom de la versió ha d'estar en format **X.Y.Z** (on X,Y,Z son nombres naturals) tal i com ho dicten els requeriments de les apps (R1.9)
- El projecte té la versió de **Swift** igual o superior a la exigida per els requeriments de la OSAM (R2.1)
- Els idiomes que té l'app
- Veure si les dependències exigides per els requeriments de la OSAM (R1.5) i (R1.7) estan afegides correctament:
    - Firebase Crashlytics
    - Firebase Analytics
    - Firebase Messaging
    - Mòdul comú
- El projecte conté el **URL Scheme** correcte com s'exigeix en els requeriments de la OSAM (R2.2)
- Revisar si l'aplicació té la **icona obligatòria** de 1024.

3. **Flutter**

Per a Android i iOS es realitzen les mateixes validacions que en natiu tret de les llibreries. Aquestes, en comptes de validar les llibreries natives, es validen els plugins utilitzats en el fitxer Pubspec.

Compilació de versions al Jenkins de l'OSAM

Per validar que el codi pujat de la nova versió sigui correcte, les versions de PRE que validarà el cap de projecte (el contacte amb l'OSAM), podran ser compilades amb el Jenkins de l'OSAM. Això podrà garantir que la versió que iniciarà el cicle de publicació esta pujada correctament al repositori i no hi ha cap problema de compilació (error en el Framework per compilar, error en la pujada de codi...).
Aquesta funcionalitat no està configurada per defecte. Per demanar la creació del procés CI/CD en l'entorn de PRE s'haurà de contactar amb el Cap de Projecte de l'OSAM que té assignada l'app. D'aquesta manera l'OSAM pot planificar l'activació d'aquesta funcionalitat per a l'app demandada. A continuació s'expliquen les passes que s'han de realitzar.

Creació de l'usuari

El Jenkins de l'OSAM es troba en la següent URL: https://cicd-osam.dtibcn.cat/
Per accedir s'ha de contactar amb el responsable del projecte a l'OSAM i demanar accés.
Les dades necessàries són:

- Nom
- Cognoms
- Correu electrònic
- Empresa

Un cop l'usuari estigui creat es facilitaran les credencials per acceder als vostres Jobs per compilar les versions a l'entorn PRE.

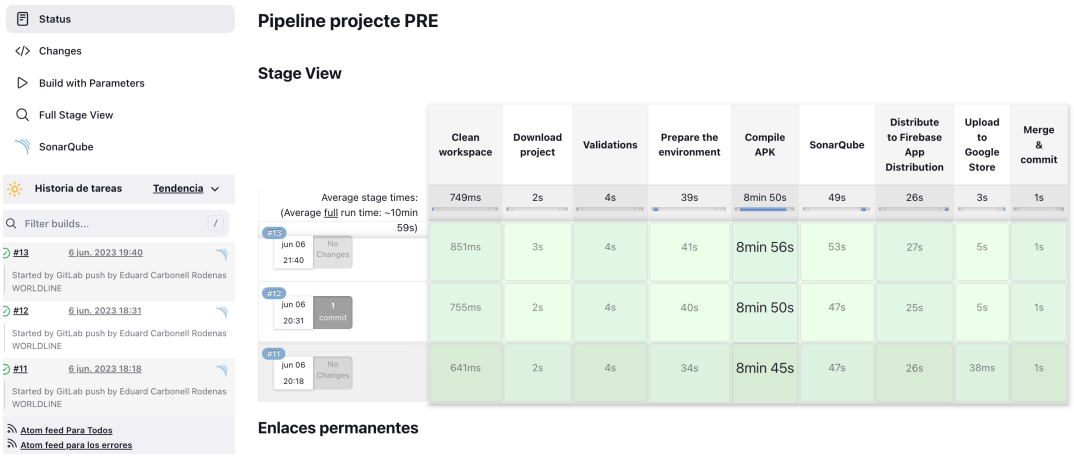
Preparar el codi al repositori

Com hi ha diverses aplicacions, s'ha de normalitzar on pujar el codi a compilar per a totes les apps. Tal i com es comenta a l'apartat "**Fase 1: Obtenció del codi font**" d'aquest propi document, el codi de la nova versió de l'app s'ha de pujar a la branca "jenkins" del repositori de l'ajuntament. D'aqui s'agafaràn les dades del projecte per validar la seva compilació i correcte funcionament. En aquest cas no cal afegir el TAG de la versió.

Executar els Jobs de les apps

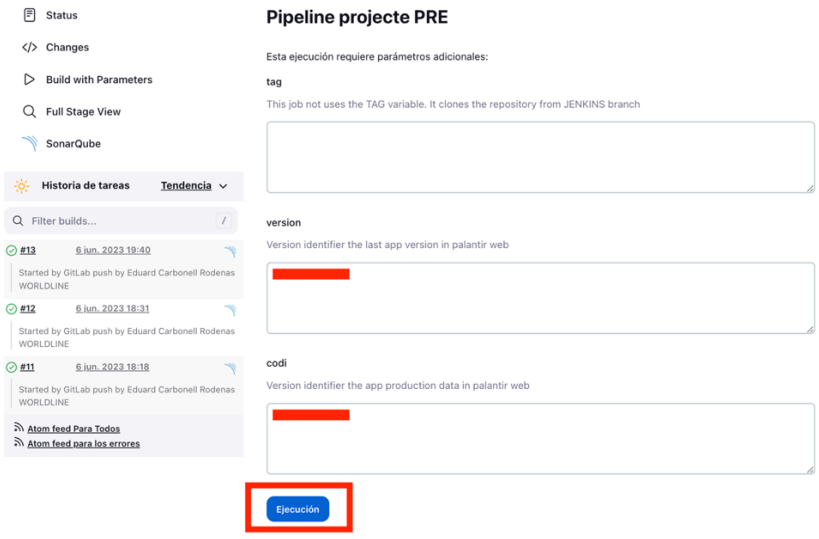
Un cop s'accedeix al Jenkins es veuran els Jobs als quals es té accés.

Per executar el Job desitjat s'ha de seleccionar i veurem la següent pantalla:



En aquesta pantalla podem veure els resultats de compilacions anteriors. Però l'acció que ens interessa és la de "Build with Parameters" situada al menú superior esquerra

Si premem aquesta opció ens preguntarà amb quins valors de paràmetres volem executar el job. **Aquests valors no s'han de modificar i sobretot, molt important, deixar la variable "tag" buida.**



La Pipeline començarà a executar-se i, si tot és correcte, la versió compilada es distribuirà per Firebase App Distribution.

Accions que es realitzaran en el procés de CI/CD

Quan s'executa el job, es realitza un seguit d'accions:

1. Descàrrega del repositori a partir de la branca "jenkins" del repositori
2. Validacions del projecte
    a. S'executaran les validacions però no seran restrictives. En cas de que hi hagi algun requeriment de l'OSAM que l'app no compleixi, s'indicarà en el correu de la compilació informant que a la compilació de PRO serà un error.
3. Compilació del codi
4. Anàlisi de la qualitat del codi amb SonarQube
    a. Encara que no compleixi els criteris de qualitat de l'OSAM no es tindrà en compte en el procés, només a PRO.
5. Distribució de la versió per Firebase App Distribution