Integracija zaganjalnika mobilnih aplikacij v lasten sistem za upravljanje mobilnih naprav

Alen Granda

Alcad, Slovenska Bistrica, Slovenija
alen.granda@alcad.si

V svetu industrije 4.0 je vključenost mobilnih naprav vse bolj pogost pojav. Od podpore pri načrtovanju virov podjetja, skladiščnega poslovanja, usmerjanja delavcev, vse do optimizacije procesov, opazimo, da so nepogrešljiv del modernih industrij. Na njih je naloženih vse več aplikacij po meri, katere je potrebno vzdrževati. Posledično je smiselno razmišljati o sistemu za upravljanje mobilnih naprav MDM (ang. Mobile device management). V prispevku predstavimo postopek implementacije lastnega sistema MDM znotraj privatnega podjetja. Osredotočili smo se predvsem v razvoj zaganjalnika na mobilni napravi, ki skrbi za zagon, nalaganje in posodabljanje mobilnih aplikacij. Zaganjalnik je implementiran na osnovi ogrodja večplatformnih uporabniških vmesnikov aplikacije .NET MAUI. Zraven implementacijskih podrobnosti opišemo še cevovod v platformi GitLab, ki poskrbi za samodejno posodabljanje mobilnih aplikacij. Na koncu sledi analiza koristnih napotkov ter možnosti izboljšav sistema.

Ključne besede:
MDM
.NET MAUI
mobilna aplikacija
CI/CD
upravljanje mobilnih naprav

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.6

1 Uvod

V današnjem dinamičnem okolju razvoja programske opreme, za katerega je značilen agilen pristop, se soočamo s stalnim in hitrim uvajanjem novih verzij programske opreme. Moderni pristopi k razvoju programske opreme se zgledujejo po modelu SCRUM [16], ki priporoča redno in pogosto izdajo posodobitev aplikacij. Posledično postane ročno vzdrževanje mobilnih naprav časovno potratno in podvrženo napakam, nasploh v primeru številnih poslovnih aplikacij ter fizičnih naprav. V tem kontekstu naglih sprememb postaja vse bolj pomembno sledenje različnim verzijam aplikacij, saj lahko ena aplikacija obstaja v več različicah, vsaka prilagojena specifičnim potrebam ali okoljem. Zato je vzpostavitev učinkovitega sistema sledenja, ki vključuje zgodovino naložitev programske opreme na odjemalce, odgovorne osebe in podrobnosti o naloženih verzijah, ključnega pomena za uspešno upravljanje razvoja in vzdrževanja aplikacij v agilnem razvojnem okolju.

V prispevku na kratko razložimo arhitekturno zasnovo lastnega sistema za upravljanje, nadzor in registracijo mobilnih naprav MDM (ang. Mobile device management) v organizacijah z več poslovnimi napravami in aplikacijami. Prednost v sistemu vidimo predvsem v avtomatizaciji izvedbe posodobitev, varnosti, centraliziranem nadzoru nad vsemi registriranimi mobilnimi napravami in v prikazu njihovih osnovnih informacij. Tako je omogočena večja učinkovitost in varnost pri uporabi mobilnih naprav, izboljšano zadovoljstvo končnih uporabnikov, poenostavljen razvoj in vzdrževanje mobilnih aplikacij ter lažja in hitrejša avtomatizacija proizvodnih procesov.

Sistem temelji na vrhovni mobilni aplikaciji, ki deluje kot zaganjalnik le izbrane programske opreme, naložene na izbrani napravi. Ob zagonu se kličoči aplikaciji posredujejo podatki, s katerimi lahko v dnevniških sistemih enolično določimo napravo, katera poroča zapise. Zaganjalnik je razvit tako, da omogoča zagon mobilnih aplikacij neodvisno od ogrodja, v katerem je le-ta zasnovana. Dodatno je mogoče filtrirati prikaz poslovnih aplikacij glede na organizacijo in okolje. V zaganjalnik smo integrirali tehnologijo SignalR [1], s pomočjo katere komunicira z zalednim sistemom. Potrebuje se na primer za posredovanje lokacije naprave na zahtevo.

Programsko opremo je mogoče ob predhodni prijavi tudi posodabljati in brisati. Posodobitve potekajo popolnoma avtomatizirano, saj gradnja novih različic mobilnih aplikacij poteka na podlagi pristopa neprekinjene integracije in postavitve. Cevovod poleg ustvarjenega paketa poskrbi še za posredovanje informacije o novi različici, kar izkorišča zaganjalnik na mobilni napravi in uporabniku ponudi avtomatsko posodobitev izbrane aplikacije. Posledično smo s sistemom razbremenili razvijalca od ročnega posodabljanja programske opreme ter poskrbeli za minimizacijo napak, ki lahko v tem postopku nastanejo.

Na koncu opišemo še izzive, s katerimi smo se soočili ob zagonu, posodabljanju in brisanju paketov znotraj ogrodja MAUI, koristne predloge in morebitne izboljšave implementiranega sistema.

2 Stanje obstoječih rešitev

Sistem za upravljanje mobilnih naprav je postal ključna komponenta v sklopu industrije 4.0. To je posledica vse večje vključenosti mobilnih naprav v proces digitalizacije industrije. Uporabljajo se za povezovanje, nadzor in upravljanje proizvodnih procesov ter opreme. Pomembno je tudi naraščajoče število mobilnih aplikacij, ki jih uporabljajo zaposleni, pri čemer je vsaka aplikacija pogosto na voljo v več različicah. Vse to poudarja potrebo po sistemu, ki omogoča nadzor nad temi elementi.

Sistem MDM omogoča centralizirano upravljanje tovrstnih naprav, vključno z namestitvijo in posodabljanjem aplikacij, nadzorom dostopa, varnostnimi ukrepi in spremljanjem delovanja. Zaradi agilnega pristopa k razvoju programske opreme je pomembno, da so rešitve MDM prilagodljive in omogočajo hitro prilagajanje novim zahtevam ter spremembam v poslovnem okolju. V industrijskem okolju je ključna tudi zanesljivost in razpoložljivost naprav, kar MDM zagotavlja s spremljanjem stanja naprav, diagnostiko in oddaljenimi popravili.

V nadaljevanju je predstavljenih nekaj trenutno najbolj uveljavljenih rešitev, skupaj s prednostmi in slabostmi vsake ter skupno analizo.

2.1. Microsoft Intune

Microsoft Intune [2] je celovita rešitev za upravljanje mobilnih naprav in aplikacij v organizacijah in temelji na storitvah v oblaku. Nudi širok nabor funkcij, kot so oddaljeno upravljanje naprav, avtomatizirano nameščanje in posodabljanje aplikacij ter upravljanje varnosti naprav. Prednost sistema je njegova integracija z drugimi storitvami podjetja Microsoft, kot je aktivni imenik (ang. Active directory), kar omogoča enostavno upravljanje uporabniških identitet in dostopov. Intune omogoča tudi upravljanje naprav na različnih platformah, vključno z operacijskimi sistemi Windows, iOS in Android. Vključuje obsežne varnostne funkcije, kot so upravljanje varnostnih pravil, šifriranje podatkov, oddaljeno brisanje naprav in nadzor dostopa do poslovnih virov. Rešitev je prilagodljiva in omogoča organizacijam, da individualizirajo politike upravljanja glede na svoje specifične potrebe in zahteve. Slabosti vključujejo omejitve pri podpori sledenju lokacij naprav in zahtevna uvedba ter konfiguracija sistema za manj izkušene administratorje, kar lahko povzroči dodatne stroške in časovne zamude.

2.2. SOTI MobiControl

SOTI MobiControl [3] je platforma za upravljanje mobilnih naprav in aplikacij. Ponuja oddaljeno upravljanje naprav, dodeljevanje funkcionalnosti na podlagi trenutne lokacije naprave, nameščanje in posodabljanje aplikacij ter upravljanje varnosti na mobilnih napravah. Sistem omogoča upravljanje velikega števila naprav v realnem času in ponuja obsežne možnosti za avtomatizacijo procesov, kar olajša upravljanje in zmanjšuje potrebo po ročnem posredovanju administratorjev. Podprta je tudi integracija postavitve aplikacij z rešitvami tretjih oseb. Prednosti vključujejo visoko stopnjo varnosti in poročanje zadnjih lokacij naprav z uporabo signalov globalnega umeščanja GPS ter upravljanje naprav na več platformah kot so Android, iOS, Windows in Linux. Slabosti so visoki stroški licenciranja in vzdrževanja, kar je lahko finančni izziv za manjše organizacije, ter zahtevna konfiguracija in implementacija, ki zahtevata dodatne vire in strokovno znanje.

2.3. Zebra Mobility DNA

Zebra Mobility DNA [4] je celovita platforma za upravljanje mobilnih naprav, razvita za industrijska in poslovna okolja. Ponuja napredne funkcije za upravljanje naprav, polavtomatizirano nameščanje in posodabljanje aplikacij ter izboljšano varnost. Sistem je specializiran za industrijsko uporabo, kar omogoča upravljanje naprav v zahtevnih okoljih, kot so proizvodnja, logistika in distribucija. Ponuja tudi široko paleto orodij za optimizacijo delovnih procesov, kar povečuje produktivnost in učinkovitost zaposlenih. Med slabostmi so nepopolnoma avtomatizirano nameščanje oziroma nadgradnja aplikacij, kjer je potrebno ročno ustvariti črtne kode za nove različice programske opreme, kar je časovno zahtevno ter podvrženo napakam. Prav tako so visoki stroški licenciranja in vzdrževanja dodatna slabost sistema.

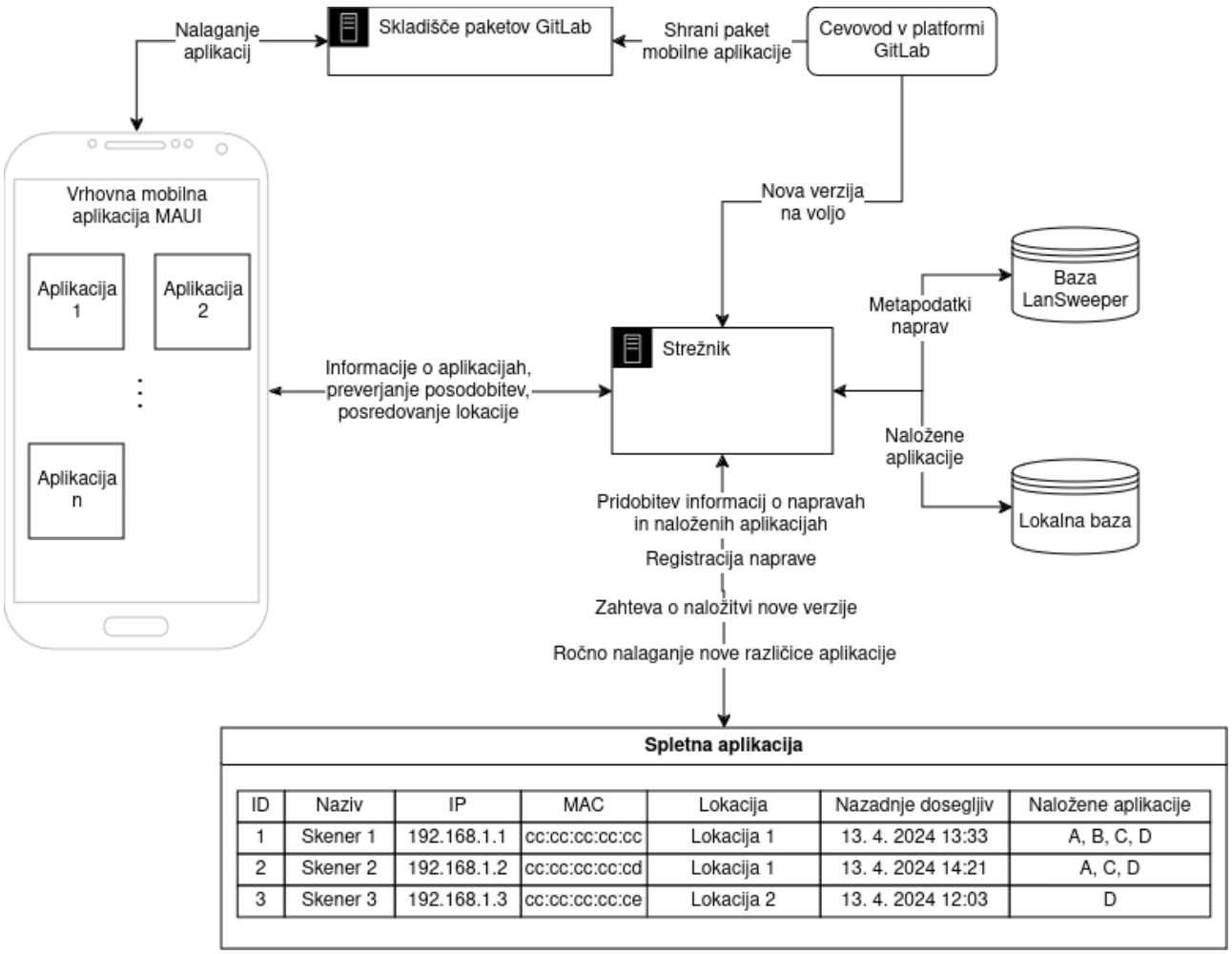
2.4. Povzetek

Vse omenjene platforme omogočajo centralizirano upravljanje mobilnih naprav in aplikacij, kar olajša administracijo in zagotavlja doslednost. Ponujajo funkcije, kot so oddaljeno brisanje aplikacij, nadzor dostopa in šifriranje, s čimer skrbijo za varnost. Poleg tega so prilagodljive in skalabilne, kar omogoča prilagajanje politik upravljanja glede na specifične potrebe organizacije. Vendar pa organizacije pogosto ne potrebujejo vseh funkcionalnosti, ki jih ponujajo te platforme. Dodatno je potrebno specifično znanje za upravljanje tovrstnih sistemov, kar lahko podaljša čas uvedbe in poveča stroške.

Opisane pomanjkljivosti so nas spodbudile k razvoju lastnega sistema MDM, ki vključuje večino skupnih lastnosti in ne zahteva dodatnega strokovnega znanja. Sistem omogoča centraliziran nadzor nad mobilnimi napravami organizacije, njihovim statusom, naloženimi aplikacijami ter popolnoma avtomatizirano posodabljanje aplikacij, ki je integrirano v proces neprekinjene integracije in postavitve CI/CD, kar razvijalcem olajša delo brez potrebe po dodatnih znanjih.

3 Vključenost zaganjalnika v sistem MDM

Na Slika vidimo globalno shemo celotnega sistema MDM. Osredotočimo se na jedro – to je osrednja mobilna aplikacija. Njena glavna naloga je prikazovanje, upravljanje in izvajanje izbranih poslovnih aplikacij na mobilnih napravah, s poudarkom na samodejnih posodobitvah in vzdrževanju programske opreme. Aplikacija za uspešno izvajanje nalog komunicira s centralnim zalednim sistemom. Pomemben vidik komunikacije je prenos unikatnega identifikatorja naprave kot metapodatka, kar omogoča natančno in zanesljivo sledenje vsaki napravi v omrežju. Na podlagi identifikatorja je omogočen prikaz seznama vseh naloženih aplikacij ter upravljanje in omejevanje drugih funkcionalnosti na posamezni napravi.

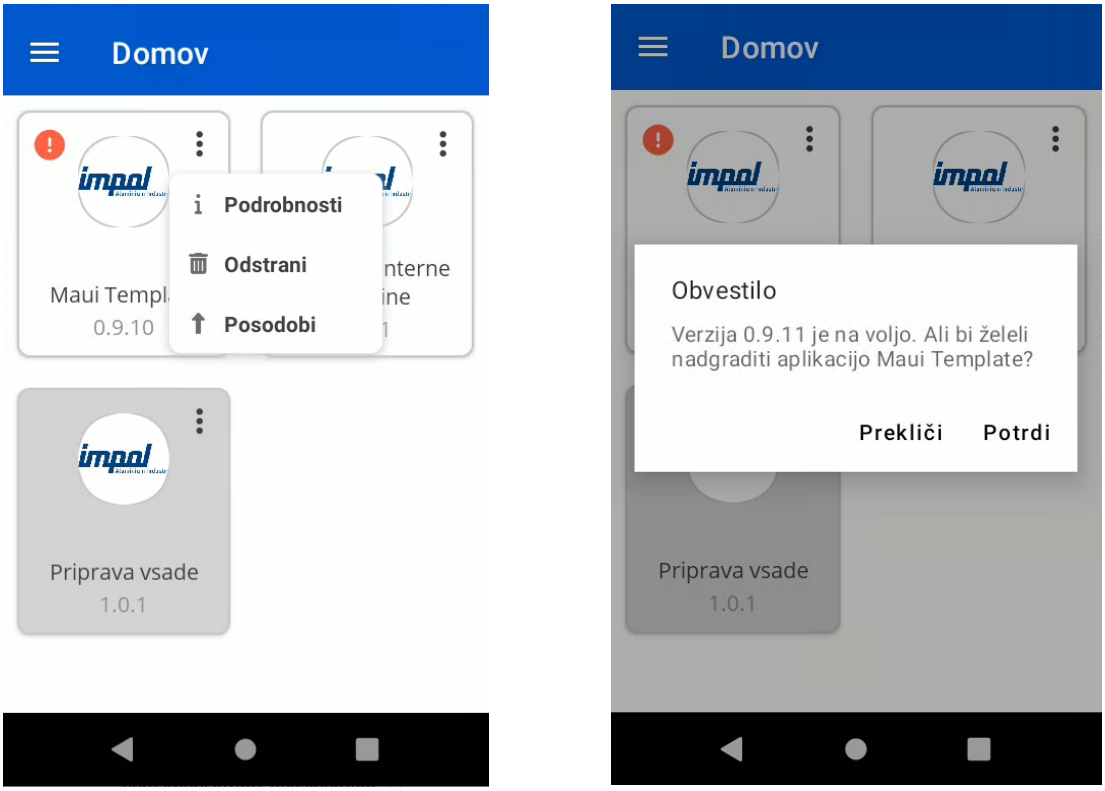


ID	Naziv	IP	MAC	Lokacija	Nazadnje dosegljiv	Naložene aplikacije
1	Skener 1	192.168.1.1	cc:cc:cc:cc:cc	Lokacija 1	13. 4. 2024 13:33	A, B, C, D
2	Skener 2	192.168.1.2	cc:cc:cc:cc:cd	Lokacija 1	13. 4. 2024 14:21	A, C, D
3	Skener 3	192.168.1.3	cc:cc:cc:cc:ce	Lokacija 2	13. 4. 2024 12:03	D

Slika 1: Globalna shema sistema MDM.

Priprava nove različice mobilne aplikacije sledi pristopu neprekinjene integracije in postavitve CI/CD. V platformi GitLab [5] je vzpostavljen cevovod, ki se sproži ob združitvi posameznih vej, kar omogoča postopno testiranje in uvajanje novih verzij aplikacij ter zmanjšuje tveganje za napake v produkcijskem okolju. Cevovod nato samodejno

začne postopek gradnje programske kode za izbrane organizacije ter izdelavo paketov mobilnih aplikacij. Ko je paket mobilne aplikacije uspešno izdelan, se shrani v register paketov na platformi GitLab v vnaprej določenem projektu. Ob tem se na zaledni sistem pošlje informacija o novo ustvarjeni različici aplikacije. Zaledni sistem nato sproži signal o potrebi po posodobitvi mobilne aplikacije na vseh napravah, ki jo imajo nameščeno. Ob naslednji osvežitvi seznama naloženih aplikacij zaganjalnik na mobilni napravi prikaže indikator, ki označuje potrebo po posodobitvi aplikacije (Slika). Na ta način omogočimo uporabnikom enostavno in pregledno posodabljanje aplikacij na njihovih napravah ter razbremenimo razvijalca od potrebe po ročnem nadgrajevanju mobilnih aplikacij.



Slika 2: Prikaz aplikacij na zaganjalniku ter ponudba posodobitve.

V nadaljevanju podrobneje opišemo implementacijske podrobnosti razvoja in integracije vrhovne mobilne aplikacije v ogrodju MAUI, vključno z opisom samega ogrodja, načini komunikacije z zalednim sistemom, upravljanjem identifikatorja naprave, posredovanjem lokacijske informacije ter razlago zagona in samodejnega posodabljanja aplikacij.

4 Implementacija zaganjalnika

4.1. Ogrodje MAUI

Ogrodje večplatformnih uporabniških vmesnikov aplikacije .NET MAUI (ang. Multi-platform app user interface) je zasnovano za ustvarjanje izvornih mobilnih in namiznih aplikacij, kar omogoča večplastni pristop k razvoju z uporabo programskega jezika C# in razširljivega označevalnega jezika aplikacij XAML [6]. Uporabno je za reševanje različnih izzivov, kot so spremenljive zahteve uporabnikov, nove poslovne priložnosti in stalne povratne informacije med razvojem. Cilj je ustvariti prilagodljive aplikacije, ki jih je mogoče enostavno prilagajati in razširjati skozi čas.

Ogrodje je odprtokodno in temelji na priljubljenem ogrodju za razvoj izvornih mobilnih aplikacij Xamarin [8]. Microsoft, v skladu s svojo strategijo združevanja vseh orodij, izvajalnih okolij in programskih knjižnic, je prepoznal potrebo po nadgradnji ogrodja Xamarin. Zato so maja 2022 javnosti predstavili ogrodje .NET MAUI

kot njegovega naslednika [7]. Nova generacija razvoja večplatformnih poslovnih rešitev je prinesla številne zanimive spremembe. Ena najpomembnejših novosti za avtomatizacijo posodobitev mobilnih aplikacij je podpora uporabi vmesnika ukazne vrstice .NET CLI (ang. .NET command line interface). Orodje omogoča razvoj, gradnjo, zagon in izdajo aplikacij .NET na več platformah, kar omogoča vključitev v proces neprekinjene integracije in postavitve CI/CD mobilnih aplikacij v zabojnikih, ki delujejo pod operacijskim sistemom Linux. Podrobnosti bomo opisali v nadaljevanju.

Poleg možnosti avtomatizacije posodobitev nas je ogrodje MAUI pritegnilo tudi zaradi vedno večje skupnosti, stabilnosti, ki jo prinaša tehnologija .NET 8, in celovite integracije v ekosistem tehnologije .NET. Dodatno so privlačne tudi kakovostne knjižnice za razvoj uporabniških vmesnikov, kot sta komplet orodij skupnosti [17] in knjižnica modulov uporabniških orodij za ogrodje MAUI podjetja DevExpress [18].

4.2. Lokacija mobilne naprave in posredovanje z uporabo tehnologije SignalR

Lokacije mobilne naprave ne osvežujemo konstantno, ampak na zahtevo, saj je takšna operacija energijsko potratna. Dvosmerno komunikacijo z zalednim sistemom smo vzpostavili s pomočjo tehnologije SignalR. Tako lahko zaledni sistem pošlje ukaz mobilni aplikaciji za osvežitev njene lokacije. Mobilna aplikacija ob prejemu ukaza pridobi trenutno lokacijo z uporabo signalov GPS ter jo posreduje nazaj zalednemu sistemu.

Za pridobitev lokacije je bilo znotraj mobilne aplikacije potrebno pridobiti pravice za dostop do lokacijskih storitev naprave. To smo dosegli z dodanimi zahtevami za dovoljenje v datoteki nastavitev AndroidManifest.xml. Poleg tega mora uporabnik eksplicitno privoliti v branje lokacije naprave skozi uporabo zaganjalnika. Po pridobitvi vseh dovoljenj lokacijo preberemo z uporabo višjenivojskih ukazov ogrodja MAUI, kot je prikazano v Slika .

Odjemalec tehnologije SignalR je implementiran s pomočjo uradne knjižnice podjetja Microsoft [9]. Ker je funkcionalnost komunikacije čez tehnologijo SignalR uporabna za več mobilnih aplikacij, smo implementacijo zapakirali v lastno knjižnico in jo objavili v internem registru knjižnic. Znotraj mobilne aplikacije je tako le storitev za registracijo metod, katere se sprožijo ob prejemu določene zahteve. Primer metode, ki registrira poslušalca na zahtevo za posredovanje sveže lokacije in implementira prej opisano logiko, je prikazana v Slika .

```
public async Task<Location?> GetGpsLocationAsync()
{
    var status = await Permissions.CheckStatusAsync<Permissions.LocationWhenInUse>();

    if (status != PermissionStatus.Granted)
    {
        await MainThread.InvokeOnMainThreadAsync(funcTask: async () =>
            status = await Permissions.RequestAsync<Permissions.LocationWhenInUse>());

        if (status != PermissionStatus.Granted)
        {
            return null;
        }
    }

    var location = await Geolocation.Default.GetLastKnownLocationAsync();

    if (location != null)
    {
        return location;
    }

    var request = new GeolocationRequest(GeolocationAccuracy.Default, timeout: TimeSpan.FromSeconds(10));
    location = await Geolocation.Default.GetLocationAsync(request);

    return location;
}
```

Slika 3: Pridobitev lokacije z uporabo signalov GPS.

4.3. Zagon in posodobitev mobilnih aplikacij

Aplikacije, ki so na voljo za specifično napravo, pridobimo s klicem na zaledni sistem. V odgovoru dobimo nazive paketov, njihove naložene verzije ter zadnjo razpoložljivo različico. Če aplikacija na napravi še ne obstaja, ponudimo namestitev. V primeru starejše različice predlagamo posodobitev. V kolikor uporabnik klikne na ikono aplikacije, jo zaženemo.

Podobno kot pri pridobitvi lokacije mobilne naprave, tudi za zagon, posodabljanje ali brisanje aplikacij potrebujemo ustrezne pravice, določene v datoteki AndroidManifest.xml. Programsko kodo za te akcije smo morali zapisati za vsako platformo posebej, saj ogrodje ne ponuja abstrakcij za tovrstne funkcije. Posodabljanje, nalaganje in brisanje aplikacij smo reševali z uporabo namenov (ang. Intent) operacijskega sistema Android. Postopek zagona aplikacije je prikazan v Slika in sledi naslednjim korakom: najprej preverimo, ali aplikacija z izbranim nazivom paketa obstaja na napravi. Če obstaja, pridobimo njen namen zagona in kot metapodatek pošljemo identifikator naprave. Aplikacija ga uporablja pri pisanju dnevnikov, kar omogoča enolično identifikacijo naprav pri pregledu dnevnikov.

```
public void RegisterAppHubListeners()
{
    _hubService.RegisterListener( methodName: HubAction.RequestForGps, [typeof(int)], handler: async objects :object?[] =>
    {
        try
        {
            if (objects.Length <= 0 || objects[0] is not int receivedAssetId)
            {
                return;
            }

            var assetId :int = await _settingsService.GetAssetId();
            if (receivedAssetId == assetId)
            {
                var location = await _gpsService.GetGpsLocation();

                if (location != null)
                {
                    await _mdmHttpClientService.PatchAssetGpsLocation(assetId, (decimal)location.Latitude, (decimal)location.Longitude);
                }
            }
        }
        catch (Exception e)
        {
            LogService.LogError(e);
        }
    });
}
```

Slika 4: Logika za registracijo poslušalca na zahtevo osvežitve lokacije mobilne naprave.

```
public async Task RunApp(string packageName)
{
    var assetId :int = await settingsService.GetAssetId();
    try
    {
        var pm :PackageManager? = Application.Context.PackageManager;

        if (IsAppInstalled(packageName))
        {
            var intent = pm?.GetLaunchIntentForPackage(packageName);
            if (intent != null)
            {
                intent.PutExtra( name: Common.Enums.Constants.AssetIdIdentifier, assetId);
                intent.SetFlags(ActivityFlags.NewTask);
                Application.Context.StartActivity(intent);
            }
        }
        else
        {
            throw new ApplicationNotFoundException();
        }
    }
    catch (ActivityNotFoundException e)
    {
        logService.LogError(e);
        throw;
    }
}
```

Slika 5: Zagon izbrane mobilne aplikacije.

4.4. Avtomatizacija posodobitev

Nazadnje opišimo še postopek avtomatizacije posodobitev aplikacij na mobilnih napravah. V kolikor sledimo diagramu na Slika , opazimo, da se lahko nova verzija aplikacije naloži na 2 načina. Na eni strani se nova različica naloži skozi cevovod v platformi GitLab. Opozoriti moramo, da je cevovod pripravljen le za aplikacije, razvite v ogrodju MAUI. Zato je podprt tudi ročni način nalaganja verzij. Tako smo omogočili nalaganje poljubnih aplikacij v sistem MDM.

Ob registraciji nove različice aplikacije se paket shrani v register paketov v platformi GitLab. Na zaledni sistem zapišemo le metapodatke o različici, kot so verzija, naziv paketa, spletni naslov za prenos paketa, organizacija, za katerega je zgrajen, ipd. Ko zaganjalnik na mobilni napravi ponovno zahteva pridobitev aplikacij, ki so zanj na voljo, se zraven aplikacij vrne še obvestilo, da je na voljo nadgradnja. Obvestilo prikažemo uporabniku in ponudimo posodobitev.

V kombinaciji s cevovodom v platformi GitLab, ki sledi pristopu neprekinjene integracije in postavitve, smo poskrbeli za popolno razbremenitev razvijalca od skrbi za ročno posodabljanje aplikacij. Ker je to ena izmed pomembnejših funkcionalnosti sistema MDM, jo podrobneje predstavimo. Cevovod se zažene ob združitvi posameznih vej. Najprej poskrbimo za procesiranje metapodatkov aplikacije (Slika). Metapodatke preberemo iz datoteke projekta s končnico .csproj. V njem se nahajata verzija ter naziv aplikacije. Nato poskrbimo za menjavo vnaprej definiranih spremenljivk v aplikacijskih nastavitvah z dejanskimi vrednostmi, ki so zapisane v skrivnostih (ang. Secret) platforme GitLab. Sledi priprava datotek za definiranje izvorov registrov knjižnic ter nastavitev osnovnega spletnega naslova za dostop do zalednega sistema. Naslov je namreč različen glede na okolje, za katerega se aplikacija gradi.

Po postopku predprocesiranja sledi še grajenje (Slika). Grajenje je omogočeno za več organizacij hkrati. Pred tem moramo seveda preveriti, ali aplikacija s tovrstno verzijo že obstaja na zalednem sistemu. V kolikor obstaja, se cevovod zaključi, saj je razvijalec najverjetneje pozabil povišati različico. V nasprotnem primeru se aplikacija zgradi z uporabo vmesnika ukazne vrstice .NET CLI in podpiše z vnaprej pripravljeno shrambo ključev (ang. keystore). Nazadnje paket shranimo v register paketov ter zabeležimo novo različico aplikacije na zaledni sistem.

```
script:
  # app metadata
  - |
    APPLICATION_ID=$(xmlstarlet sel -t -v "//ApplicationId" $PROJECT_PATH)
    APPLICATION_DISPLAY_VERSION=$(xmlstarlet sel -t -v "//ApplicationDisplayVersion" $PROJECT_PATH)
    APPLICATION_VERSION=$(xmlstarlet sel -t -v "//ApplicationVersion" $PROJECT_PATH)
    APPLICATION_NAME=$(xmlstarlet sel -t -v "//ApplicationTitle" $PROJECT_PATH)
    APP_SETTINGS_FILE="./Impol.${APPLICATION_NAME}/Resources/Raw/appSettings.json"

  # replace everything except for tenant id in appSettings.json
  - |
    sed -i -e 's#\$MDM_API_URL#'"${MDM_API_URL}"'#' ${APP_SETTINGS_FILE}
    # + others that are skipped for brevity

    cp ${APP_SETTINGS_FILE} ${APP_SETTINGS_FILE_COPY}

  # nuget config file
  - |
    echo '<?xml version="1.0" encoding="utf-8"?>' > ${NUGET_CONFIG_FILE}
    echo '<configuration>' >> ${NUGET_CONFIG_FILE}
    echo '    <packageSources>' >> ${NUGET_CONFIG_FILE}
    echo '        <add key="nuget.org" value="https://api.nuget.org/v3/index.json" protocolVersion="3" />' >> ${NUGET_CONFIG_FILE}
    echo '        <add key="Devexpress" value="'"${MOBILE_DEVEXPRESS_NUGET_REPO}"'" />' >> ${NUGET_CONFIG_FILE}
    echo '        <add key="gitlab-alcad" value="'"${GITLAB_NUGET_REPO}"'" />' >> ${NUGET_CONFIG_FILE}
    echo '    </packageSources>' >> ${NUGET_CONFIG_FILE}
    echo '</configuration>' >> ${NUGET_CONFIG_FILE}

  # copy nuget configs
  - |
    projectNugetConfigFile="./Impol.${APPLICATION_NAME}/nuget.config"
    cp ${NUGET_CONFIG_FILE} ${projectNugetConfigFile}
    # + others that are skipped for brevity

  # setup the base API URL based on commit branch
  - MDM_BASE_URL="https://${CI_COMMIT_BRANCH}.${DOCKER_API_PROD}/${MDM_API_URL}/api/v1"
  - apiBaseUrl="https://${CI_COMMIT_BRANCH}.${DOCKER_API_PROD}/"
```

Slika 6: Predprocesiranje metapodatkov aplikacije v postopku cevovoda.

```
.publish_android:
  tags:
    - build
  script:
    # publish it for each tenant
    - |
      ##################################
      # replace API url of the appsetting.json
      sed -i -e 's#\$DOCKER_API_PROD#'"${apiBaseUrl}"'#' ${APP_SETTINGS_FILE_COPY}

      ##################################
      # receive app id from MDM API

      urlRequest="${MDM_BASE_URL}/applications?\$filter=code%20eq%20'${APPLICATION_NAME}'"
      response=$(curl -X GET --location "$urlRequest" --header 'Tenant-Id: 11' --header 'Tenant-Type: O' --header 'Content-Type: application/json')

      num_elements=$(echo "$response" | jq length)
      if [ $num_elements -ne 1 ]; then
        echo "Error: App ID could not be retrieved. Ensure APPLICATION_NAME is properly set. APPLICATION_NAME = ${APPLICATION_NAME}"
        echo "Response:"
        echo "$response"
        exit 1
      fi

      MDM_APP_ID=$(echo "$response" | jq -r '.[0].id')

      ##################################
      # publish it for each tenant

      for tenantId in ${TENANT_IDS}; do
        ##################################
        appId="${APPLICATION_ID}_${CI_COMMIT_BRANCH}_${tenantId}"

        echo "Publishing the project ${APPLICATION_NAME} with version ${APPLICATION_DISPLAY_VERSION} for tenant ${tenantId}"
        echo "Environment: $CI_COMMIT_BRANCH"
        echo "App ID: $appId"

        cp ${APP_SETTINGS_FILE_COPY} ${APP_SETTINGS_FILE}
        sed -i -e 's#\$TENANT_ID#'"${tenantId}"'#' ${APP_SETTINGS_FILE}

        # use appSettings.json and not appSettings.Development.json
        sed -i -e 's#appSettings.Development.json#appSettings.json#g' ./Impol.${APPLICATION_NAME}/Impol.${APPLICATION_NAME}.csproj
        sed -i -e 's#<ApplicationId>.*</ApplicationId>#<ApplicationId>'"${appId}"'</ApplicationId>#' ./Impol.${APPLICATION_NAME}/Impol.${APPLICATION_NAME}.csproj
        sed -i -e 's#RegisterConfigurations<AppSettings>(true)#RegisterConfigurations<AppSettings>(false)#g' ./Impol.${APPLICATION_NAME}/MauiProgram.cs

        ##################################
        # check if this version can even be published...

        # e.g.: applications/1/versions?$filter=versionNumber eq 1 and version/name eq '0.9.0'
        urlRequest="${MDM_BASE_URL}/applications/${MDM_APP_ID}/versions?%24filter=versionNumber%20eq%20${APPLICATION_VERSION}%20and%20version%2Fname%20eq%20%27${APPLICATIO
        response=$(curl -X GET --location "$urlRequest" --header 'Tenant-Id: '"${tenantId}"'' --header 'Tenant-Type: O' --header 'Content-Type: application/json')

        # Check if the response array is not empty using jq
        if [[ $(echo "$response" | jq '. | length') -gt 0 ]]; then
          echo "App with such ID and version name already exists! Version name = ${APPLICATION_DISPLAY_VERSION}, version number = ${APPLICATION_VERSION}"
          exit 1
        fi

        ##################################
        # Publish
        dotnet publish \
          -f net8.0-android \
          -c Release \
          -o ${OUTPUT_DIR} \
          -p:AndroidSdkDirectory="${ANDROID_SDK_ROOT}" \
          -p:AndroidPackageFormats=apk \
          -p:AndroidKeyStore=true \
          -p:AndroidSigningKeyStore="$(pwd)/${ANDROID_SIGNING_KEYSTORE_FILE}" \
          -p:AndroidSigningKeyPass=${ANDROID_SIGNING_KEY_PASS} \
          -p:AndroidSigningStorePass=${ANDROID_SIGNING_KEY_PASS} \
          -p:ApplicationDisplayVersion=${APPLICATION_DISPLAY_VERSION} \
          -p:ApplicationVersion=${APPLICATION_VERSION} \
          -p:ApplicationId=${appId} \
          -p:ApplicationTitle=${APPLICATION_NAME}

        ##################################
        # deploy
        for apk in $(find $OUTPUT_DIR/ -type f -name "*-Signed.apk" -printf "%f\n"); do

          echo "Deploying the file = $apk"

          # deploy to apk project
          DEPLOY_URL=${CI_API_V4_URL}/projects/910/packages/generic/${appId}/${APPLICATION_DISPLAY_VERSION}/${apk}

          # push the package to package registry
          response=$(curl --header "JOB-TOKEN: $CI_JOB_TOKEN" --upload-file ${OUTPUT_DIR}/${apk} "${DEPLOY_URL}")
          status_code=$(echo "$response" | sed 's/[^0-9]*//g')
          if [ "${status_code}" != "201" ]; then
            echo "Error: Response status is not 201. It is ${status_code}"
            echo "Response: $response"
            exit 1
          fi

          response=$(curl -w "%{http_code}" --location "${MDM_BASE_URL}/applications/${MDM_APP_ID}/versions" \
            -X POST \
            --header 'Tenant-Id: '"${tenantId}"'' \
            --header 'Tenant-Type: O' \
            --header 'Content-Type: multipart/form-data' \
            --header 'Authorization: Bearer '"${MDM_AUTH_TOKEN}"'' \
            --form 'PackageName='"${appId}"'' \
            --form 'VersionName='"${APPLICATION_DISPLAY_VERSION}"'' \
            --form 'VersionNumber='"${APPLICATION_VERSION}"'')

          status_code=$(echo "$response" | tail -c 4 | sed 's/[^0-9]*//g')
          if [ "${status_code}" != "200" ]; then
            echo "Error: Response status is not 200. It is ${status_code}"
            echo "Response: $response"
            exit 1
          fi

          echo "File ${apk} deployed to ${DEPLOY_URL}!"

        done
      done
```

Slika 7: Grajenje in objava nove različice aplikacije znotraj cevovoda.

5 Koristni napotki

5.1. Zagon in namestitev aplikacij

Znotraj zaganjalnika uporabljamo občutljive akcije, kot so zagon, namestitev in brisanje aplikacij. To je lahko velik poseg v varnost same naprave, zato je razumljivo, da je potrebno za tovrstne akcije potrebno eksplicitno dovoljenje uporabnika. V ta namen smo morali v nastavitveni datoteki AndroidManifest.xml zahtevati pravice za iskanje paketov, namestitev in brisanje (Slika). Dodatno bi omenili, da moramo poleg zahteve za iskanje paketov podati tudi značko za iskanje "<queries>". V nasprotnem primeru se paket ne najde.

```
<!-- For automatically installing apps -->
<uses-permission android:name="android.permission.INSTALL_PACKAGES" />
<uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES" />
<uses-permission android:name="android.permission.REQUEST_DELETE_PACKAGES" />

<!-- used to launch apps -->
<uses-permission android:name="android.permission.QUERY_ALL_PACKAGES" />
<queries>
    <intent>
        <action android:name="android.intent.action.MAIN" />
    </intent>
</queries>
```

Slika 8: Zahteve za iskanje, nameščanje in brisanje paketov.

Dodatno moramo paziti, da nastavimo način zagona glavne aktivnosti (ang. Main activity) na enojni vrh (ang. Single top). V kolikor pustimo privzeto vrednost, se zaganjalnik ob zagonu druge aplikacije zapre, najverjetneje zaradi hrošča (ang. Bug) tehnologije .NET 8 [10].

```
[Activity(Theme = "@style/Maui.SplashTheme",
    LaunchMode = LaunchMode.SingleTop, // need to set to SingleTask / SingleTop! See: https://github.com/dotnet/maui/issues/18692
    Exported = true, MainLauncher = true, ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation | ConfigChanges
[IntentFilter([Intent.ActionMain], Categories = [Intent.CategoryLauncher, Intent.CategoryHome, Intent.CategoryDefault])]
0 references
public class MainActivity : MauiAppCompatActivity
{
```

Slika 9: Nastavitev zagona glavne aplikacije na enojni vrh.

Opozorili bi tudi, da ogrodje MAUI samo po sebi že ponuja zagon drugih aplikacij z lastnimi abstrakcijami [11]. Edina slabost zaganjalnika ogrodja je, da kličoči aplikaciji ne moremo posredovati parametrov, kot je na primer identifikator naprave. Posledično smo se odločili za lastno implementacijo z uporabo platformno specifične programske kode.

5.2. Cevovod v platformi GitLab

Nikjer nismo omenili kakšna predloga vsebnika se uporablja za grajenje aplikacij ogrodja MAUI znotraj cevovoda v platformi GitLab. Predloga izhaja iz [12] in je prilagojena za grajenje v tehnologiji .NET 8. Opozoriti moramo, da za grajenje aplikacij za operacijski sistem Android potrebujemo komplet razvojnih orodij programskega jezika Java verzije 11. To velja vse do različice 14 operacijskega sistema Android [14]. Ker uradna predloga podjetja Microsoft za grajenje paketov v tehnologiji .NET 8 [13] temelji na operacijskem sistemu Debian 12, ki uradno ne podpira več Jave 11, imamo 2 možnosti. Ena je, da ročno poskrbimo za instalacijo v tovrstnem operacijskem sistemu. Po drugi strani lahko uporabimo predlogo, ki temelji na operacijskem sistemu Ubuntu [15]. Tovrstna predloga še namreč ponuja uraden način instalacije s pomočjo upravljalca paketov.

6 Zaključek

V zaključku lahko povzamemo, da uvedba lastnega sistema za upravljanje in nadzor mobilnih naprav prinaša številne prednosti za organizacije z več poslovnimi aplikacijami. Z avtomatizacijo posodobitev, izboljšano varnostjo in centraliziranim nadzorom se zmanjša časovna in administrativna obremenitev razvijalcev ter poveča učinkovitost poslovnih procesov. Sistem omogoča tudi boljše sledenje različicam aplikacij in prilagoditve glede na specifične potrebe okolja. Kljub izzivom, ki jih prinaša integracija z ogrodjem MAUI, so koristi jasne in pripomorejo k večjemu zadovoljstvu končnih uporabnikov ter poenostavitvi razvoja in vzdrževanja aplikacij. V prihodnje bi nadaljnje izboljšave sistema lahko še dodatno optimizirale procese in funkcionalnosti.

Kljub številnim prednostim sistema za upravljanje mobilnih naprav ostaja še veliko prostora za izboljšave. Ena izmed pomanjkljivosti je pomanjkanje povratne informacije pri komunikaciji čez tehnologijo SignalR, kjer lahko na strani odjemalca dodamo logiko za potrditveni odgovor zalednemu sistemu. Prav tako nismo vpeljali enotne prijave (ang. Single sign-on) ob zagonu izbranih aplikacij znotraj zaganjalnika, kar bi olajšalo uporabniško izkušnjo. Predlagamo uporabo šifriranih metapodatkov za izboljšano varnost pri prijavnih procesih. Šifrirani metapodatki o uporabniku se lahko pošiljajo hkrati s podatkom o identifikatorju naprave. Z implementacijo izboljšav lahko sistem postane še bolj učinkovit in varen, kar pripomore k optimizaciji poslovnih procesov in k zadovoljstvu uporabnikov.

Literatura

[1] https://dotnet.microsoft.com/en-us/apps/aspnet/signalr, Real-time ASP.NET with SignalR, obiskano 16. 7. 2024.

[2] https://learn.microsoft.com/en-us/mem/intune/fundamentals/what-is-intune, Microsoft Intune overview, 16. 7. 2024.

[3] https://soti.net/products/soti-mobicontrol/, SOTI MobiControl, obiskano 16. 7. 2024.

[4] https://www.zebra.com/us/en/software/mobile-computer-software/mobility-dna.html, Mobility DNA - Enterprise Mobility Software, obiskano 16. 7. 2024.

[5] https://about.gitlab.com/, GitLab, obiskano 17. 7. 2024.

[6] https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui, What is .NET MAUI?, obiskano 17. 7. 2024.

[7] https://devblogs.microsoft.com/dotnet/announcing-dotnet-maui-in-dotnet-8, Announcing .NET MAUI in .NET 8, obiskano 17. 7. 2024.

[8] https://learn.microsoft.com/en-us/previous-versions/xamarin/get-started/what-is-xamarin, What is Xamarin?, obiskano 17. 7. 2024.

[9] https://www.nuget.org/packages/Microsoft.AspNetCore.SignalR.Client, Microsoft.AspNetCore.SignalR.Client, obiskano 17. 7. 2024.

[10] https://github.com/dotnet/maui/issues/18692, MAUI Android build crashes when app is reopened from background. It throws the exception: 'Window was already created.', obiskano 18. 7. 2024.

[11] https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/appmodel/launcher?view=net-maui-8.0&tabs=android, Launcher, obiskano 18. 7. 2024.

[12] GRANDA Alen "Neprekinjena integracija in postavitev MAUI mobilnih aplikacij", OTS 2023 Sodobne informacijske tehnologije in storitve: Zbornik 26. konference, Maribor, Fakulteta za elektrotehniko, računalništvo in informatiko, Inštitut za informatiko, 115-126.

[13] https://hub.docker.com/r/microsoft/dotnet-sdk, microsoft/dotnet-sdk – Docker Image, obiskano 18. 7. 2024.

[14] https://developer.android.com/build/jdks, Java versions in Android builds, obiskano 18. 7. 2024.

[15] https://github.com/dotnet/dotnet-docker/blob/main/src/sdk/8.0/jammy/amd64/Dockerfile, dotnet-docker/src /sdk/8.0/jammy/amd64/Dockerfile, obiskano 18. 7. 2024.

[16] SCHWABER Ken "SCRUM Development Process", Business Object Design and Implementation, Springer London, London, 1997, str. 117-134.

[17] https://learn.microsoft.com/en-us/dotnet/communitytoolkit/maui/, .NET Multi-platform App UI (.NET MAUI) Community Toolkit documentation, obiskano 18. 7. 2024.

[18] https://www.devexpress.com/maui/, .NET MAUI Controls, obiskano 18. 7. 2024.