Neprekinjena integracija in postavitev MAUI mobilnih aplikacij

Alen Granda

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

S prihodom ogrodja MAUI je v svet mobilnih informacijskih rešitev razvitih v tehnologiji .NET prispela možnost avtomatizacije postopka gradnje in izdaje novih verzij programske opreme s pomočjo vsebnikov, ki se izvajajo pod operacijskim sistemom Linux. Posledično razvoj poteka bolj produktivno, hitreje zaznamo morebitne napake v programski kodi ter poskrbimo za boljšo uporabniško izkušnjo. V prispevku predstavimo poglavitne lastnosti ogrodja MAUI, opišemo primerjavo z njegovim predhodnikom Xamarin ter izpostavimo učinkovitostne izboljšave. Prikažemo način, kako avtomatizirati gradnjo in izdajo novih verzij produkcijskih mobilnih aplikacij razvitih v ogrodju MAUI s pomočjo metode neprekinjene integracije in postavitve v platformi GitLab. Metoda se v celoti izvaja v vsebniku pod operacijskim sistemom Linux. Predstavimo tudi praktični primer samodejnega posodabljanja demonstracijske mobilne aplikacije ob izdaji nove verzije z opisanim postopkom. Dodani so še koristni napotki ob implementaciji predstavljenega demonstracijskega primera.

Ključne besede:
MAUI
CI/CD
GitLab
mobilne aplikacije
.NET

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-286-772-0
DOI https://doi.org/10.18690/um.feri.8.2023.10

1 Uvod

Razvoj nativnih mobilnih aplikacij je še danes pogost način implementacije rešitev poslovnih problemov. Z nativnostjo poskrbimo za učinkovitost, imamo večji nadzor nad napravo ter izkoristimo vse njene prednosti. Zaradi uporabe vse bolj popularnega agilnega pristopa k razvoju programske opreme se nove funkcionalnosti kot tudi popravki zelo pogosto integrirajo k obstoječi mobilni aplikaciji. Hkrati s hitrimi nadgradnjami je potrebno poskrbeti tudi za dostavo novih različic programske opreme h končnemu uporabniku. Žal je lahko v primeru razvoja nativnih mobilnih aplikacij na podlagi ogrodja Xamarin izvajanje posodobitev zamudno. Nadgradnje najpogosteje potekajo ročno, od grajenja projekta, generacije paketa aplikacije ter vse do nalaganja h končnemu uporabniku. Razvojne smernice iz sorodnih področij v ta namen koristijo metodo neprekinjene integracije in postavitve (ang. CI/CD) programske opreme, ki poskrbi za avtomatizacijo izvajanja posodobitev in razbremenitev razvijalca od tega procesa.

V prispevku predstavimo način, kako integrirati metodo neprekinjene integracije in postavitve mobilnih informacijskih rešitev na osnovi ogrodja .NET MAUI. Pripravili smo demonstracijski projekt v ogrodju .NET MAUI, ki je namenjen operacijskemu sistemu Android. Dodali smo primer konfiguracije cevovoda CI/CD na platformi GitLab. Ob izdaji nove verzije programske opreme se paket mobilne aplikacije samodejno ustvari in shrani na register paketov na repozitoriju platforme GitLab s pomočjo zaganjalnika GitLab Runner, ki zažene registrirane naloge v cevovodu CI/CD. Mobilna aplikacija ob vsakem zagonu s pomočjo poizvedb na aplikacijski programski vmesnik GitLab REST API preveri, ali je na voljo nova verzija ter v pritrdilnem primeru ponudi uporabniku avtomatsko posodobitev. Z opisanim načinom smo poskrbeli za samodejno posodabljanje mobilne aplikacije brez uporabe tretjih ponudnikov kot je npr. Google Play. Postopek je uporaben predvsem za poslovne rešitve, ki so namenjene privatnim podjetjem. Dodatno v prispevku omenimo še primerjavo z ogrodjem Xamarin, učinkovitostne primerjave ter prednosti oziroma slabosti tehnologije.

2 Ogrodje MAUI

2.1 Predstavitev

Ogrodje .NET MAUI (ang. Multi-platform App UI) je multi-platformno usmerjeno ogrodje za razvoj nativnih mobilnih ter namiznih aplikacij z uporabo programskih jezikov C# ter XAML [1]. Namenjeno je razreševanju več izzivov, s katerimi se razvijalci med razvojem programske opreme soočajo, kot so:

- konstantno spreminjajoče se zahteve aplikacije,
- nove poslovne priložnosti in izzivi,
- kontinuirane povratne informacije med razvojem, ki pogosto vplivajo na obseg in zahteve aplikacije.

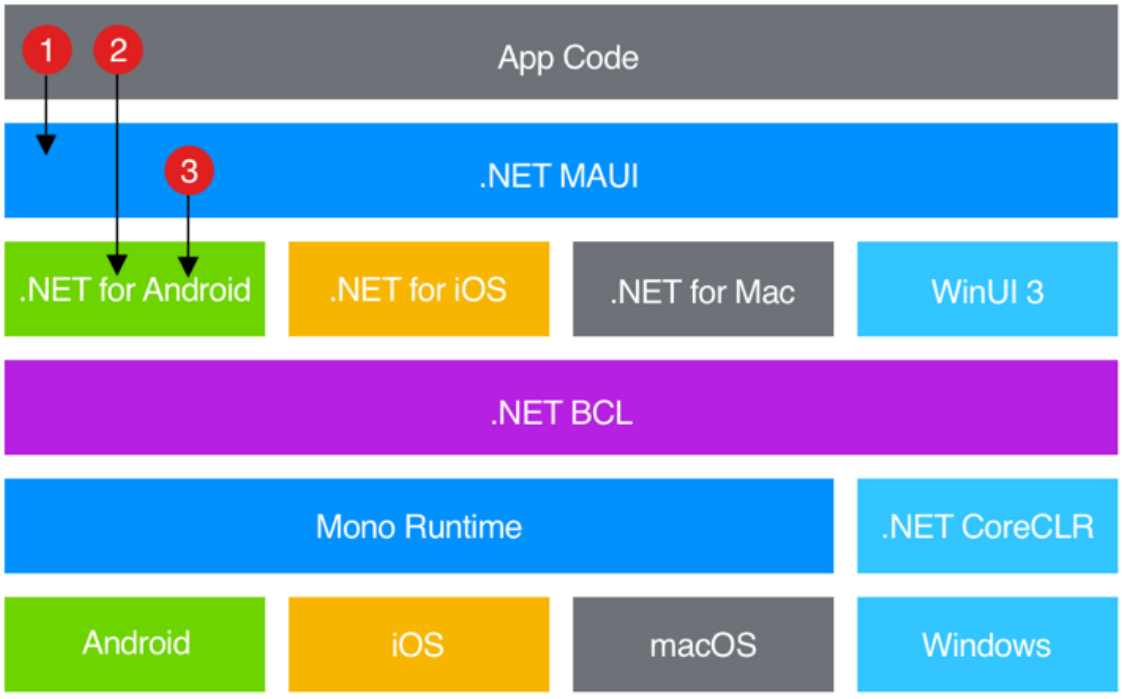
S tem v mislih je pomembno razvijati aplikacije, ki jih je mogoče sčasoma preprosto spremeniti ali razširiti. Oblikovanje takšne prilagodljivosti je lahko težavno, saj zahteva arhitekturo, ki omogoča neodvisen razvoj in testiranje posameznih delov aplikacije v izolaciji, ne da bi to vplivalo na preostali del aplikacije. Takšna arhitektura razvoja programske opreme zahteva razbitje aplikacije na več posameznih, šibko povezanih komponent, katere je možno enostavno integrirati v celoto. S tem pridobimo več prednosti [2]:

- preprost razvoj, testiranje in razširjanje funkcionalnosti aplikacije s pomočjo več ekip razvijalcev,
- delegacija opravil razvijalcem, ki imajo na posameznem področju največ izkušenj,
- zanesljiv nadzor nad posameznimi deli aplikacije zaradi ločitve odvisnosti med gradniki aplikacije, kot so poslovna logika, dostop do podatkov, avtentikacija ipd.

Ogrodje MAUI nam omenjene prednosti omogoča z uporabo več abstrakcij, ki združijo aplikacijske programske vmesnike operacijskih sistemov Android, iOS, macOS ter Windows v enoten vmesnik. Posledično je podprt razvoj deljene programske kode kot tudi razvoj enotnega uporabniškega vmesnika, ki deluje na vseh omenjenih platformah. Razvoj temelji na skupni bazi programske kode, ki je enaka za vse platforme in posledično pripomore k hitrejšemu in cenejšemu razvoju programske opreme. Ogrodje je odprtokodno in razvito na podlagi popularnega ogrodja za razvoj nativnih mobilnih aplikacij Xamarin.

Iz diagrama poteka interpretiranja programske kode na Slika opazimo, da nam ogrodje zraven abstrakcijskega nivoja, ki samodejno poskrbi za klic ustreznih metod v izhodiščnem operacijskem sistemu (označeno s številkama 1 ter 3 na sliki), ponuja tudi direkten dostop do aplikacijskih programskih vmesnikov naloženega operacijskega sistema, v kolikor bi to potrebovali (označeno s številko 2 na sliki 1). Natančneje, gradnja paketov aplikacije za željene operacijske sisteme poteka na sledeč način [1]:

- Android: izvorna programska koda se iz programskega jezika C# prevede v vmesni jezik (ang. intermediate language), ki se ob zagonu aplikacije na napravi prevaja instantno (ang. just-in-time),
- iOS: celoten projekt se prevede vnaprej (ang. ahead-of-time) v zbirno kodo ARM,
- macOS: ogrodje v tem primeru koristi tehnologijo Mac Catalyst s par dodatnimi posegi,
- Windows: v ta namen ogrodje MAUI koristi knjižnico Windows UI 3.



Slika 1: Diagram poteka interpretiranja programske kode, ki komunicira s programskim aplikacijskim vmesnikom ogrodja MAUI
Vir: [1].

2.2 Primerjava z ogrodjem Xamarin

Tako kot ogrodje MAUI, je tudi ogrodje Xamarin namenjeno razvoju nativnih aplikacij z deljeno programsko kodo, razvito v programskih jezikih C# ter XAML [3]. Omogoča grajenje aplikacij, ki se lahko izvajajo na več platformah. Uradno podprte platforme so Android, iOS ter Windows. Razvoj ogrodja Xamarin se je pričel že leta

2011 [4], nakar ga je leta 2016 kupilo podjetje Microsoft [5] in ga zelo kmalu integriralo v razvojno okolje Visual Studio za brezplačno uporabo [6].

Ogrodje Xamarin je projekt, ki je izoliran od celotnega sistema tehnologij .NET, saj deluje na podlagi izvajalnega okolja Mono. Ker je želja podjetja Microsoft unifikacija vseh njihovih orodij, izvajalnih okolij in programskih knjižnic, so se hitro začeli fokusirati v nadgradnjo ogrodja Xamarin ter tako prejšnje leto maja [9] širši javnosti predstavili ogrodje .NET MAUI kot njegovega naslednika. Prihod nove generacije razvoja multi-platformnih poslovnih rešitev je poleg integracije v celotno okolje družine .NET in posledično uporabe abstrakcijskega nivoja .NET BCL (ang. .NET Base class library) prinesel več zanimivih sprememb, ki so povzete v tabeli 1 [1, 7, 8, 10].

Tabela 1: Primerjava ogrodij MAUI ter Xamarin.

	MAUI	Xamarin
Podprte platforme	Android 5.0+, iOS 10+, macOS 10.15+, WinUI, Tizen[1], Linux[2].	Android 4.4+, iOS 9+, UWP, Tizen[1], macOS, GTK#[2], WPF[2].
Struktura projekta	Enotna.	Projekt za posamezno platformo.
Ukazna vrstica	.NET CLI	.NET Framework
Osnovna tehnologija	.NET 6, .NET 7	.NET Framework
Upravljanje virov	Enotni viri za vse platforme.	Viri za posamezno platformo.
Vroče posodabljanje (ang. hot reload)	Da.	Ne, vendar je eksperimentalno podprto.
Razvojne paradigme	MVVM, RxUI, MVU.	MVVM, RxUI.
Arhitektura nadzornih gradnikov	Arhitektura upodabljalnika (ang. renderer architecture).	Arhitektura upravljalnika (ang. handler architecture).

Vir: [1, 7, 8, 10]

Posledica tovrstne integracije je med drugim tudi enovita uporaba grafičnih programskih knjižnic za razvoj uporabniškega vmesnika ter uporaba multi-platformnih programskih vmesnikov za dostop do nativnih funkcionalnosti naprav, kot so signali GPS, pospeškometer, stanje baterije, dostop do interneta ipd. [1].

Ena izmed prednosti napram ogrodju Xamarin je prav tako podpora tehnologiji Blazor Hybrid [21]. Tehnologija omogoča izvajanje gradnikov spletnega vmesnika nativno na končni napravi. To stori tako, da gradnike upodablja v vgrajenem nadzorniku spletnega ogleda (ang. embedded Web View), ki se izvaja neodvisno od spletnega brskalnika ali tehnologije WebAssembly [21]. Posledično lahko združimo programsko kodo spletnega vmesnika s programsko kodo nativnih aplikacij in poenostavimo razvoj poslovnih rešitev.

Za namen avtomatizacije posodobitev aplikacij je najpomembnejša novost podpora uporabi vmesnika ukazne vrstice .NET (ang. .NET CLI). Orodje omogoča razvoj, gradnjo, zagon ter izdajo aplikacij .NET na več platformah. Posledično ga lahko uporabimo v sklopu neprekinjene integracije in postavitve mobilnih aplikacij v vsebnikih, ki se izvajajo pod operacijskim sistemom Linux. Bolj podrobno je cevovod obrazložen v poglavju priprave praktičnega primera.

2.3 Učinkovitostni vidiki

Eden izmed ključnih motivacij za razvoj naslednika ogrodja Xamarin je tudi izboljšava učinkovitosti delovanja aplikacije ter zmanjšanje njene velikosti. Ob javni naznanitvi ogrodja MAUI so v ta namen objavili tudi izčrpno poročilo primerjave učinkovitosti aplikacij, razvitih na podlagi ogrodja Xamarin napram aplikacijam, zgrajenih na podlagi njegovega naslednika. Če povzamemo nekaj ključnih izboljšav ogrodja MAUI, ki temelji na tehnologiji .NET 6 [11]:

- pohitritev zagona zaradi kombinacije uporabe profiliranega grajenja strojne kode vnaprej (ang. profiled AOT),

[1] https://www.defect
[2] seznam sestavnih

- izboljšave v interoperabilnosti s programskim jezikom Java:
    - optimizirano klicanje konstruktorjev in metod programskega jezika C# iz programskega jezika Java z minimizacijo klica sistemske metode System.Reflection.Emit(),
    - reduciranje klicev programske kode v jeziku Java,
    - menjava klicev metode LayoutInflater.Inflate() s programskimi metodami,
- menjava splošnega gostitelja .NET (ang. .NET general host) za konfiguracijo, upravljanje dnevnika ter vključitve odvisnosti s specifično implementacijo za mobilne naprave,
- izogib uporabi knjižnice Newtonsoft.Json zaradi koriščenja interne knjižnice za serializacijo in deserializacijo System.Text.Json.

S prihodom tehnologije .NET 7 so v podjetju Microsoft šli še nekoliko dalje in predstavili še nekaj dodatnih izboljšav, kot so [12]:

- učinkovitostne izboljšave zaradi zmanjšanja nepotrebnih klicev metode View.Context(), minimizacije preskokov med programskima jezikoma C# in Javo, uporaba nativne implementacije kalkulacije orientacije smeri teksta,
- izboljšane metode za primerjavo nizov, pohitritev izvajanja regularnih izrazov z uporabo atributa [RegexGenerator],
- izboljšano prevajanje gradnikov XAML v vmesni jezik,
- ignoriranje metapodatkovnih datotek, namenjenih programskemu jeziku Kotlin, ki so posledica vse bolj pogoste implementacije Google-ovih programskih knjižnic družine AndroidX v programskem jeziku Kotlin.

Kljub omenjenim izboljšavam in analizam je tehnologija še mlada in jo je potrebno obravnavati z rezervo. Čeprav je že dobro leto dni na voljo širši javnosti, se še vedno razvija in izboljšuje zahvaljujoč skupnosti, ki tehnologijo uporablja in sproti obvešča razvijalce o težavah. Napake se sproti z razvojem tehnologije .NET 8 odpravljajo in dokumentirajo na blogu razvijalcev tehnologije .NET [13], [14].

3 Neprekinjena integracija in postavitev aplikacij (CI/CD)

3.1 Predstavitev

Neprekinjena integracija in postavitev aplikacij (CI/CD) avtomatizira večino potrebnega ročnega dela za prenos sprememb iz programske kode v produkcijo [16]. Ročno grajenje, nalaganje in postavitev aplikacij je zamudno, še posebej kadar pozabimo postopek cevovoda. Z definiranim avtomatskim cevovodom CI/CD razbremenimo razvijalca od procesa integracije novo razvitih funkcionalnosti ali popravkov v produkcijske aplikacije in posledično poskrbimo, da se osredotoči le na razvoj programske opreme.

V kategorijo neprekinjene integracije spadajo avtomatsko grajenje, testiranje in vključitev sprememb v glavne veje repozitorija. Tovrsten način razvoja programske opreme spodbuja pogosto združevanje vej repozitorija, kar zmanjša možnost konfliktov v programski kodi ter poveča njeno zanesljivost zaradi pogostega testiranja programske kode. Posledično nam neprekinjena integracija prinaša zgodnje razreševanje morebitnih napak programske kode ali varnostnih ranljivosti.

Za procesom neprekinjene integracije sledi še avtomatizacija postavitve aplikacij h končnim uporabnikom. Proces iz zgrajene in testirane programske kode pripravi končne artefakte, ki so pripravljeni za prenos v produkcijsko okolje. Poleg tega v to kategorijo spada še zagotavljanje potrebne infrastrukture za zagon aplikacije, npr. nastavitev in namestitev paketov v vsebniku (ang. container).

V praksi se je uporaba CI/CD uveljavila predvsem zaradi več ključnih prednosti, kot so [15]:

- hitrejši razvoj novih funkcionalnosti in posledično obvladovanje konkurenčnosti na trgu,
- pomaga pri pridobivanju novega kadra, zadrževanju talentiranih razvijalcev ter zmanjšanju izgorelosti, saj se lahko razvijalci fokusirajo le na razvoj programske opreme,
- izboljšana kvaliteta programske kode zaradi specializacije razvijalcev,
- zadovoljni uporabniki zaradi manjšega števila napak v produkciji in hitre predstavitve novih funkcionalnosti.

3.2 CI/CD v ogrodju MAUI

Ena izmed slabosti poslovnih rešitev razvitih v ogrodju Xamarin je slaba podpora grajenju projektov s pomočjo ukazne vrstice, še posebej v smislu vsebnikov, ki tečejo na operacijskem sistemu Linux. Posledično ni mogoče avtomatizirati procesa integracije in postavitve aplikacij. Vsaka sprememba v programski kodi, nova funkcionalnost ali predstavitev popravka se mora ročno prevesti, zagnati, testirati in šele ob potrditvi primernosti delovanja oddati uporabniku v testiranje. Postopek je zamuden, izpostavljen napakam in sili razvijalca k pogostemu spreminjanju konteksta. Koncentracija in produktivnost se zmanjšata, razvijalec postane nezadovoljen. Posledično se lahko hitro zgodi, da se v celotnem postopku kakšna izmed nalog pozabi ali naredi napačno, kar poveča možnost napak v programski opremi in nezadovoljstvo uporabnika.

Izpostavljena pomanjkljivost je s prihodom ogrodja MAUI ublažena, saj so v podjetju Microsoft s popolno integracijo ogrodja v ekosistem okolja .NET poskrbeli tudi za integracijo z vmesnikom ukazne vrstice .NET (ang. .NET CLI). Z njegovo pomočjo je mogoče na podlagi ukazne vrstice graditi, izvajati in objavljati aplikacije na več platformah. V naslednjem poglavju prikažemo praktični primer implementacije cevovoda CI/CD na podlagi orodja .NET CLI v platformi GitLab za demonstracijski projekt, ki je razvit v ogrodju MAUI.

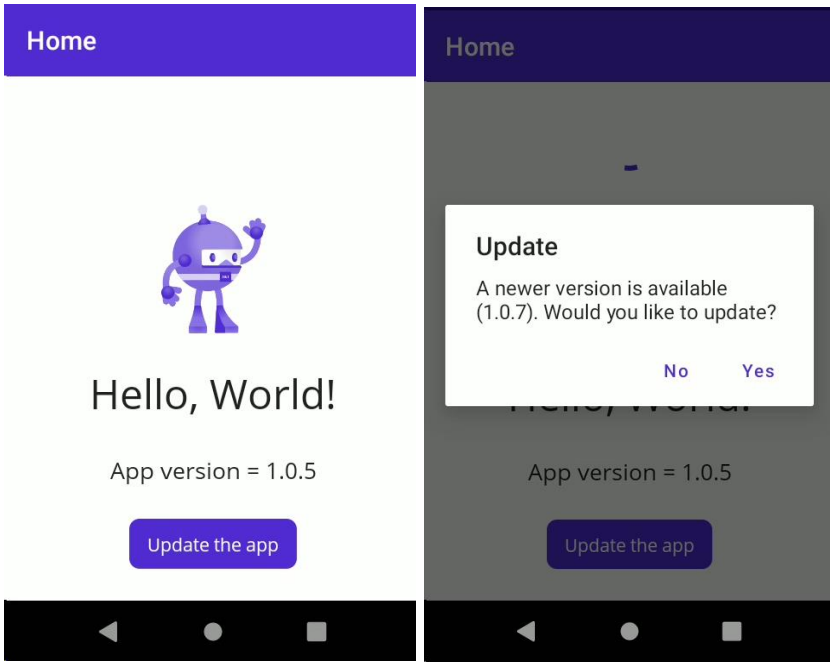
4 Praktični primer cevovoda CI/CD na vsebniku Linux ter platformi GitLab

Čeprav je implementacija neprekinjene integracije in postavitve aplikacij v ogrodju MAUI mogoča, v literaturi še ni moč zaznati veliko primerov uporabe. V [17] so prikazali način, kako sestaviti cevovod CI/CD v oblačnem sistemu Azure ter v orodju verzioniranja programske kode GitHub za aplikacije pripravljene v ogrodju MAUI. Oba primera izvedeta cevovod na virtualnem računalniku z operacijskim sistemom macOS ali Windows. Ker so dandanes vse bolj uporabni in popularni vsebniki, ki delujejo na osnovi poljubne verzije operacijskega sistema Linux, nas je zanimalo, ali je mogoče pripraviti cevovod CI/CD, ki bi se zagnal na tovrstnem vsebniku.

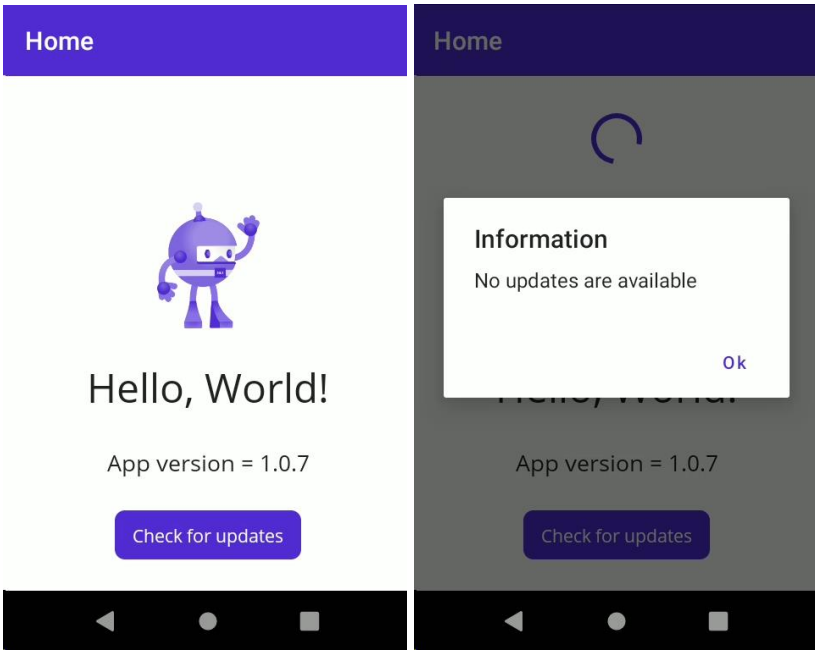
V nadaljevanju predstavimo praktični primer neprekinjene integracije in postavitve demonstracijske mobilne aplikacije, razvite v ogrodju MAUI. Mobilna aplikacija je namenjena operacijskemu sistemu Android in skrbi za samodejno posodabljanje, v kolikor zazna novo verzijo v registru paketov. Cevovod CI/CD je pripravljen za platformo GitLab, ampak ga je mogoče enostavno reproducirati na njemu podobne platforme. Ob izdaji nove verzije produkcijske aplikacije se cevovod zažene, ustvari paket mobilne aplikacije in ga objavi v registru paketov. Gradnja paketa aplikacije Android poteka v celoti na vsebniku pod operacijskim sistemom Linux. Dodatno še prikažemo kako nastaviti ukaz za generiranje podpisanih paketov mobilne aplikacije Android.

4.1 Android mobilna aplikacija

Za demonstracijski primer smo pripravili kar preprosto mobilno aplikacijo, ki je zasnovana po primeru iz [18] (Slika). Aplikacija temelji na tehnologiji .NET 7 in je namenjena operacijskemu sistemu Android. Na začetno stran smo dodali gumb, ki ob kliku preveri, ali so na voljo posodobitve (slika 2). To preverimo s pomočjo poizvedbe na končno točko registra paketov, ki jo ponuja platforma GitLab [22]. V kolikor obstaja paket z novejšo verzijo mobilne aplikacije, uporabniku ponudimo možnost avtomatske posodobitve. V pritrdilnem primeru se paket prenese na mobilno napravo in mobilna aplikacija se samodejno posodobi (slika 3).



Slika 2: Začetna stran mobilne aplikacije in predlog posodobitve.



Slika 3: Posodobljena aplikacija in ponovna preverba novih posodobitev.

4.2 Predloga vsebnika

Vhodna predloga vsebnika (ang. Docker image) naše nastavitvene datoteke *Dockerfile* je predloga uradnega kompleta za razvoj programske opreme na osnovi tehnologije .NET [19]. Ta nam omogoča uporabo vmesnika ukazne vrstice .NET. Nato s pomočjo upravljalca paketov naložimo razvojni komplet programskega jezika Java verzije 11, komplet za razvoj programske opreme Android, sprejmemo licence uporabe ter nazadnje naložimo še izbirno razširitev tehnologije .NET za gradnjo mobilnih aplikacij Android imenovano „maui-android". Opisan postopek grafično prikazuje slika 4.

```
FROM mcr.microsoft.com/dotnet/sdk:7.0

ENV JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64/
ENV ANDROID_SDK_ROOT=/usr/lib/android-sdk
ENV PATH=$ANDROID_SDK_ROOT/cmdline-tools/latest/bin:$PATH

# JAVA
RUN apt-get update && \
    apt-get install -y openjdk-11-jdk && \
    rm -rf /var/lib/apt/lists/*

# Utils
RUN apt-get update && apt-get install -y \
    jq && \
    rm -rf /var/lib/apt/lists/*

RUN apt-get update && apt-get install -y \
    unzip && \
    rm -rf /var/lib/apt/lists/*

# Install Android SDK
RUN mkdir -p /usr/lib/android-sdk/cmdline-tools/latest && \
    curl -k "https://dl.google.com/android/repository/commandlinetools-linux-9123335_latest.zip" -o commandlinetools-linux.zip && \
    unzip -q commandlinetools-linux.zip -d /usr/lib/android-sdk/tmp && \
    mv  /usr/lib/android-sdk/tmp/cmdline-tools/* /usr/lib/android-sdk/cmdline-tools/latest && \
    rm -rf /usr/lib/android-sdk/tmp/ && \
    rm commandlinetools-linux.zip

# Accept licenses
RUN yes | sdkmanager --licenses && \
    sdkmanager "platform-tools" && \
    sdkmanager "build-tools;33.0.0" "platforms;android-33" \

# Install workload maui
RUN dotnet workload install maui-android --ignore-failed-sources
```

Slika 4: Nastavitev predloge vsebnika.

4.3 Nastavitev cevovoda v platformi GitLab

Za demonstracijske namene smo si cevovod zamislili na sledeč način. Najprej kot bazno predlogo vsebnika uporabimo predlogo, predstavljeno v 4.2. Nato zgradimo podpisan paket mobilne aplikacije Android, pri čemer je certifikat za podpisovanje predhodno pripravljen s pomočjo ukaza *keytool* [23]. Certifikat nato podamo kot parameter ukazu *dotnet publish* [23] skupaj z geslom in vzdevkom. Ukaz izda paket mobilne aplikacije Android. Paket shranimo v register paketov na repozitoriju platforme GitLab tako, da ga dodamo v telo končne točke za shranjevanje datotek v registru paketov. Platforma GitLab namreč omogoča hrambo, verzioniranje in nalaganje poljubnih datotek za vsak projekt posebej [20]. Pripravljen cevovod se zažene ob vsaki stvaritvi značke iz produkcijske veje. Značka je oblike „prod_<verzija>", na podlagi česa lahko na enostaven način identificiramo novo različico programske opreme, ki jo prenašamo uporabniku v produkcijo. Implementacijo opisanega postopka prikazuje slika 5.

4.4 Koristni predlogi

Ob inicializaciji projekta mobilne aplikacije so kot privzete platforme nastavljene vse možne, ki jih ogrodje MAUI podpira. Žal grajenje projektov MAUI pod operacijskim sistemom Linux še ne vključuje podpore za vse platforme. Če v vsebniku, ki se izvaja nad predstavljeno vsebniško predlogo, zaženemo ukaz *dotnet workload search*, opazimo, da so trenutno dostopne razširitve vmesnika ukazne vrstice .NET razširitev za operacijski sistem Android, Windows ter Tizen. Zato moramo v datoteki projekta .csproj eksplicitno nastaviti oziroma odstraniti platforme, ki trenutno še niso podprte za gradnjo. V nasprotnem primeru gradnja ne bo uspešna. Ker smo demonstracijski primer predstavili za platformo Android, smo torej morali prilagoditi značko XML, ki opredeli ciljne platforme (slika 6). V kolikor bi želeli dodati še kakšno platformo, bi ustrezno posodobili XML značko.

```
image: registry.cicd.alcad.si/dotnet-maui

variables:
  APP_NAME: maui-test

stages:
  - publish

publish_android:
  stage: publish
  tags:
    - build
  script:
    - tags=(${CI_COMMIT_TAG//_/ })
    - app_version=${tags[1]}
    - echo "Publishing the project $APP_NAME with version $app_version"
    - dotnet publish
      -f:net7.0-android
      -c:Release
      -o $PUBLISH_OUTPUT_DIR
      -p:AndroidSdkDirectory="$ANDROID_SDK_ROOT"
      -p:AndroidPackageFormats=apk
      -p:AndroidKeyStore=true
      -p:AndroidSigningKeyStore="$(pwd)/${ANDROID_SIGNING_KEYSTORE_FILE}"
      -p:AndroidSigningKeyAlias="$ANDROID_SIGNING_ALIAS"
      -p:AndroidSigningKeyPass=$ANDROID_SIGNING_KEY_PASS
      -p:AndroidSigningStorePass=$ANDROID_SIGNING_KEY_PASS
      -p:Version=$app_version
    - ls -la $PUBLISH_OUTPUT_DIR
    - >
      for apk in $(find $PUBLISH_OUTPUT_DIR/ -type f -name "*.apk" -printf "%f\n"); do

      echo "Deploying the file = $apk"

      curl --header "JOB-TOKEN: $CI_JOB_TOKEN" --upload-file ${PUBLISH_OUTPUT_DIR}/${apk}
      "${CI_API_V4_URL}/projects/${CI_PROJECT_ID}/packages/generic/${APP_NAME}/${app_version}/${apk}"

      echo "File $apk deployed!"

      done
  allow_failure: false
  rules:
    - if: '$CI_COMMIT_TAG =~ /^prod_/'
```

Slika 5: Nastavitev cevovoda CI/CD v platformi GitLab.

```
<PropertyGroup>
    <TargetFrameworks>;net7.0-android</TargetFrameworks>
    <!-- continuing content of Project.csproj -->
    <!-- ... -->
```

Slika 6: Nastavitev značke XML v datoteki projekta .csproj za opredelitev ciljne platforme.

Za uspešno izvedbo samodejnega posodabljanja mobilne aplikacije moramo v datoteki AndroidManifest.xml nastaviti dovoljenji za nalaganje ter za pojavitev zahteve o nalaganju novih paketov (slika 7). Dodatno moramo še programsko preveriti, ali je uporabnik že potrdil dovoljenje aplikaciji za zahtevek o nalaganju aplikacij. To lahko storimo s pomočjo izrezka programske kode na slika 8, kjer v primeru nepotrjenega dovoljenja uporabnika preusmerimo k dialogu za odobritev.

```
<uses-permission android:name="android.permission.INSTALL_PACKAGES" />
<uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES" />
<!-- other permissions -->
<!-- ... -->
```

Slika 7: Nastavitev dovoljenj mobilni aplikaciji Android za nalaganje novih paketov na mobilno napravo.

```
if (!PackageManager.CanRequestPackageInstalls())
{
    StartActivity(new Intent(
        Android.Provider.Settings.ActionManageUnknownAppSources,
        Android.Net.Uri.Parse("package:" + Android.App.Application.Context.PackageName)));
    return;
}
```

Slika 8: Validacija dovoljenja za nalaganje aplikacij.

5 Zaključek

V primerjavi s svojim predhodnikom nam ogrodje MAUI prinaša mnogo nadgradenj in znatno olajša delo razvijalca. Je popolnoma integrirano v ekosistem tehnologij .NET in ponuja enakovredne aplikacijske programske vmesnike za razvoj aplikacij na različnih platformah. Izboljšave so zaznane tudi v učinkovitosti ter v platformni neodvisnosti. Za namen prispevka je najpomembnejša nadgradnja integracije z vmesnikom ukazne vrstice .NET, kar nam omogoča uporabo neprekinjene integracije in postavitve mobilnih poslovnih rešitev na več platformah. Slednje smo utemeljili s praktičnim primerom na platformi GitLab in mobilno aplikacijo, ki omogoča samodejno posodabljanje na podlagi samodejnih izdaj novih verzij na registru paketov.

Kljub vsem prednostim ne smemo pozabiti, da je tehnologija še mlada in se še razvija. Pomanjkljivosti in napake se še vedno pojavljajo ter se s prihodom vsake nove verzije tehnologije .NET popravijo, kar morebiti spremeni tudi aplikacijski programski vmesnik. Omenjeno moramo vzeti v obzir, če imamo namen nadgraditi obstoječe mobilne rešitve, razvite v ogrodju Xamarin, vsaj dokler ima le-to še uradno podporo. Nedvomno ima ogrodje MAUI potencial biti učinkovit naslednik ogrodja Xamarin, katerega razvoj se splača spremljati.

Literatura

[1] https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-7.0, What is .NET MAUI?, obiskano 20. 06. 2023.

[6] STONIS Michael, »Enterprise Application Patterns using .NET MAUI«. Redmond, Washington, 2022.

[7] https://learn.microsoft.com/en-us/xamarin/get-started/what-is-xamarin-forms, What is Xamarin.Forms?, obiskano 20. 06. 2023.

[8] https://tirania.org/blog/archive/2011/May-16.html, Miguel de Icaza, Announcing Xamarin, obiskano 20. 06. 2023.

[9] https://blogs.microsoft.com/blog/2016/02/24/microsoft-to-acquire-xamarin-and-empower-more-developers-to-build-apps-on-any-device/, GUTHRIE Scott, Microsoft to acquire Xamarin and empower more developers to build apps on any device, obiskano 20. 06. 2023.

[10] https://techreport.com/news/xamarin-now-comes-free-with-visual-studio/, FERREIRA Bruno, Xamarin now comes free with Visual Studio, obiskano 20. 06. 2023.

[11] https://learn.microsoft.com/en-us/xamarin/get-started/supported-platforms?tabs=windows, Xamarin.Forms supported platforms, obiskano 20. 06. 2023.

[12] https://learn.microsoft.com/en-us/dotnet/maui/supported-platforms, Supported platforms for .NET MAUI apps, obiskano 20. 06. 2023.

[13] https://devblogs.microsoft.com/dotnet/learn-dotnet-maui/, SOUCOUP Matt, New Resources to Get Started with .NET MAUI, obiskano 20. 06. 2023.

[14] https://www.syncfusion.com/blogs/post/xamarin-versus-net-maui.aspx, KATHIRESAN G. Selva, Xamarin Versus .NET MAUI, obiskano 20. 06. 2023.

[15] https://devblogs.microsoft.com/dotnet/performance-improvements-in-dotnet-maui/, PEPPERS Jonathan, Performance Improvements in .NET MAUI, obiskano 20. 06. 2023.

[16] https://devblogs.microsoft.com/dotnet/dotnet-7-performance-improvements-in-dotnet-maui/, PEPPERS Jonathan, .NET 7 Performance Improvements in .NET MAUI, obiskano 20. 06. 2023.

[17] https://devblogs.microsoft.com/dotnet/announcing-dotnet-maui-in-dotnet-8-preview-3/, ORTINAU David, Announcing .NET MAUI in .NET 8 Preview 3, obiskano 20. 06. 2023.

[18] https://devblogs.microsoft.com/dotnet/announcing-dotnet-maui-in-dotnet-8-preview-5/, ORTINAU David, Announcing .NET MAUI in .NET 8 Preview 5, obiskano 20. 06. 2023.

[19] https://about.gitlab.com/resources/ebook-fuel-growth-cicd/, Modernize your CI/CD, obiskano 21. 06. 2023.

[20] https://about.gitlab.com/topics/ci-cd/, What is CI/CD, obiskano 21. 06. 2023.

[21] https://devblogs.microsoft.com/dotnet/devops-for-dotnet-maui/, SATPATHY Sweeky, Getting Started with DevOps and .NET MAUI, obiskano 21. 06. 2023.

[22] https://learn.microsoft.com/en-us/dotnet/maui/get-started/first-app, Build your first app, obiskano 21. 06. 2023.

[23] https://hub.docker.com/_/microsoft-dotnet-sdk/, .NET SDK | Docker Hub, obiskano 21. 06. 2023.

[24] https://docs.gitlab.com/ee/user/packages/generic_packages/#publish-a-generic-package-by-using-cicd, Publish a generic package by using CI/CD, obiskano 21. 06. 2023.

[25] https://learn.microsoft.com/en-us/aspnet/core/blazor/hybrid/?view=aspnetcore-7.0, ASP.NET Core Blazor Hybrid, obiskano 22. 06. 2023.

[26] https://docs.gitlab.com/ee/api/packages.html, Packages API, obiskano 23. 06. 2023.

[27] https://learn.microsoft.com/en-us/dotnet/maui/android/deployment/publish-cli?view=net-maui-7.0, Publish an Android app using the command line, obiskano 23. 06. 2023.