Avtomatizirani testi uporabniških vmesnikov ogrodja .NET MAUI

Alen Granda

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

V razvoju mobilnih aplikacij postaja zagotavljanje stabilnosti in pravilnega delovanja uporabniškega vmesnika vse pomembnejše, še posebej pri večplatformskih rešitvah, kot to ponuja ogrodje .NET MAUI. Ročno preverjanje delovanja aplikacij je zamudno, ponovljivost testov je omejena. Zato predstavlja avtomatizacija testiranja uporabniškega vmesnika ključen korak k večji zanesljivosti in učinkovitemu razvojnemu procesu. Prispevek obravnava celovit postopek vzpostavitve avtomatskih testov uporabniških vmesnikov za aplikacijo, razvito znotraj ogrodja .NET MAUI. Prikazan je razvoj testov z uporabo knjižnice Appium, priprava platformno specifične kode za operacijski sistem Android ter vključitev celotnega postopka v cevovod neprekinjene integracije in postavitve na platformi GitLab. Poseben poudarek je na izvajanju testov znotraj vsebnika, ki teče v operacijskem sistemu Linux. Podan je tudi opis vseh korakov, vključno z zagonom virtualne naprave, vzpostavitvijo strežnika Appium in izvedbe testov. Prispevek se zaključi s pregledom ključnih prednosti in slabosti pristopa ter ponudi smernice za nadaljnjo optimizacijo tovrstnih sistemov testiranja.

Ključne besede:
.NET MAUI,
CI/CD,
zagotavljanje kakovosti,
UI testi,
Appium.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.20

1 Uvod

Zaradi naraščajočih zahtev po večplatformskih aplikacijah, ki morajo delovati zanesljivo tako na mobilnih napravah kot na namiznih računalnikih, se razvojni procesi vse bolj osredotočajo na avtomatizacijo in integracijo kakovostnih mehanizmov testiranja. Ogrodje .NET MAUI [2] razvijalcem omogoča razvoj enotne programske kode za različne platforme, kot so Android, iOS, Windows in macOS. Vendar ta večplatformskost predstavlja izziv tudi pri testiranju uporabniških vmesnikov UI (ang. User interface), ki so pogosto podvrženi napakam, povezanih z vizualnimi elementi, postavitvami in interakcijami [3].

Ročno testiranje aplikacij hitro postane neučinkovito in nezanesljivo, še posebej pri hitrem razvojnem ciklu in pogostih spremembah uporabniškega vmesnika [4]. Pogoste spremembe, ki vplivajo na logiko ali izgled vmesnika, lahko hitro povzročijo napake, ki jih je brez avtomatiziranih testov težko pravočasno zaznati. Zato je avtomatizacija testiranja ključna za zagotavljanje kakovosti in stabilnosti končnih rešitev. V prispevku predstavimo, kako smo v projekt ogrodja .NET MAUI vpeljali avtomatizirane teste uporabniškega vmesnika z uporabo knjižnice Appium [6] ter jih vključili v proces neprekinjene integracije CI (ang. Continuous integration) s pomočjo platforme GitLab [7]. Testi so zasnovani tako, da omogočajo ponovljivo preverjanje funkcionalnosti aplikacije v različnih razvojnih fazah, kar dolgoročno izboljšuje zanesljivost produkta [5].

Poseben poudarek namenjamo zasnovi in vzpostavitvi cevovoda CI, ki izvaja teste znotraj vsebnika (ang. Container). Takšna zasnova omogoča ponovljivost, razširljivost in enostavnejše upravljanje testnega okolja ter zmanjšuje odvisnost od ročnega nastavljanja okolij na različnih napravah ali agentih. V našem primeru smo se za namen izvedbe testov osredotočili izključno na platformo Android, saj je to trenutno edina podprta mobilna platforma, ki jo podjetje uporablja v produkciji. Testiranje drugih platform, kot sta iOS ali Windows, v tej fazi razvoja ni prednostno, kar je vplivalo tudi na tehnično poenostavitev celotnega procesa.

Prispevek je strukturiran tako, da bralca najprej seznani z uporabljenimi tehnologijami, kot so .NET MAUI, Appium, Docker in GitLab. V nadaljevanju postopoma predstavimo pripravo osnovnega uporabniškega testa v okolju .NET MAUI, vključno z uporabo platformno specifične kode za Android. Sledi podroben opis, kako testno okolje prenesemo v vsebnik, ki omogoča izolirano izvajanje testov na operacijskem sistemu Linux. Sledi integracija avtomatiziranega zagona testov v cevovod CI, kjer so opisani vsi potrebni koraki za zagon virtualne naprave Android, strežnika Appium, namestitve aplikacije in izvajanja testov. Prispevek se zaključi z analizo prednosti in slabosti predstavljenega pristopa ter s kratkim povzetkom ključnih ugotovitev.

2 Uporabljene tehnologije

Za vzpostavitev avtomatiziranega testiranja uporabniškega vmesnika v okolju .NET MAUI smo uporabili tri ključne tehnologije: ogrodje .NET MAUI [2] za razvoj aplikacije, knjižnico Appium [6] za avtomatizacijo testov uporabniškega vmesnika ter platformo GitLab [7] za upravljanje z različicami kode in izvajanje procesov neprekinjene integracije in postavitve CI/CD. Vsaka izmed teh tehnologij ima pomembno vlogo pri zagotavljanju stabilnega in ponovljivega testnega okolja.

.NET MAUI

Ogrodje .NET MAUI [2] je ogrodje podjetja Microsoft za razvoj večplatformskih aplikacij z enotno programsko kodo, katerih zgrajena koda teče na operacijskih sistemih Android, iOS, Windows in macOS. Predstavlja naslednika ogrodja Xamarin.Forms in temelji na tehnologiji .NET 6+.

Njegova glavna prednost je možnost deljenja večino izvorne kode med vsemi podprtimi platformami, kar zmanjšuje kompleksnost razvoja in omogoča hitrejšo dostavo funkcionalnosti. Poleg tega se promovira uporaba arhitekture MVVM [13] (ang. Model-View-ViewModel), ki olajša ločevanje predstavitvene plasti od poslovne logike ter posledično izboljša berljivost, vzdržljivost in testabilnost programske kode. Arhitektura MVVM omogoča strukturirano gradnjo aplikacije, kar je ključnega pomena za dolgoročno vzdrževanje in širitev projekta.

V okviru prispevka smo pripravili projekt za testiranje programske kode znotraj rešitve, ki služi kot predloga za vse projekte v podjetju. Rešitev vključuje osnovno strukturo aplikacije z definiranimi sloji uporabniškega vmesnika in poslovne logike, kar omogoča lažjo ponovljivost in standardizacijo razvoja. Nadgradili smo jo z avtomatiziranimi testi uporabniškega vmesnika in jih integrirali v proces neprekinjene integracije in postavitve CI/CD, kar omogoča avtomatsko potrjevanje funkcionalnosti ob vsaki spremembi izvorne kode. V našem primeru smo se osredotočili izključno na platformo Android, saj je fokus podjetja osredotočen le na izbran operacijski sistem.

Appium

Appium [6] je odprtokodno orodje za avtomatizacijo testiranja uporabniških vmesnikov mobilnih aplikacij. Temelji na standardu WebDriver [8] in omogoča izvajanje testov brez potrebe po spremembah izvorne programske kode aplikacije.

Appium podpira več programskih jezikov (npr. C#, Java, Python, JavaScript) in omogoča testiranje aplikacij na platformah Android in iOS, kar ga naredi izjemno prilagodljivega za uporabo v večplatformskih okoljih. V ozadju Appium deluje kot strežnik, ki posluša zahteve testnega odjemalca z uporabo protokola WebDriver in jih usmerja na ustrezen gonilnik – npr. gonilnik UiAutomator2 za Android ali gonilnik XCUITest za operacijski sistem iOS [8]. Gonilnik nato komunicira z mobilno napravo ali virtualno napravo in izvaja dejanske ukaze, kot so:

- pritisk na gumb,
- vnos besedila v polja,
- preverjanje vidnosti elementov,
- pomikanje po zaslonu,
- čakanje na prisotnost določenih elementov ipd.

Ta arhitektura omogoča, da Appium deluje kot posrednik med testno logiko, ki jo pišemo v domenskem programskem jeziku, kot je C#, in napravo ali virtualno napravo, na katerem teče aplikacija. Appium torej simulira resničnega uporabnika – klikne gumbe, vnese besedilo, preverja postavitve in odzive elementov uporabniškega vmesnika. To omogoča preverjanje funkcionalnosti na način, ki je najbližji dejanskemu uporabniškemu scenariju, kar je ključnega pomena za zagotavljanje kakovostne uporabniške izkušnje.

V našem primeru Appium deluje v povezavi z virtualno napravo Android, kjer izvaja avtomatizirane teste UI za aplikacijo, razvito znotraj ogrodja .NET MAUI. V kombinaciji s funkcionalnostjo platforme GitLab smo Appium integrirali v avtomatiziran testni cevovod, kjer se testi UI sprožijo samodejno po vsaki združitvi vej programske kode ali pred izdajo nove različice aplikacije.

GitLab

GitLab [7] je celovita platforma, ki združuje nadzor nad izvorno programsko kodo, sledenje napakam in močan sistem za izvajanje procesov CI/CD. Ključna prednost platforme je integrirani cevovod CI/CD, ki omogoča samodejno izvajanje testov, gradnjo in nameščanje aplikacij ob vsaki spremembi kode.

V našem primeru smo GitLab uporabili za avtomatizacijo postopka testiranja, kjer se ob vsaki združitvi glavnih vej programske kode sproži cevovod, ki v vsebniku zažene testni strežnik Appium in izvede vnaprej definirane teste UI. S tem zagotovimo, da se morebitne napake odkrijejo zgodaj in da testno okolje ostaja ponovljivo in neodvisno od posameznih razvijalcev. Z uporabo funkcionalnosti GitLab CI/CD lahko tako dosežemo višjo stopnjo avtomatizacije, hitrejšo povratno informacijo po spremembah ter boljšo sledljivost napak znotraj razvojnega procesa.

3 Zasnova zagona testov znotraj vsebnika

Za namen avtomatizacije testiranja uporabniških vmesnikov, zgrajenih v ogrodju .NET MAUI, smo pripravili lastno sliko Docker (ang. Docker image), zasnovano kot nadgradnja slike, opisane v [9]. Slika temelji na osnovni sliki podjetja Microsoft, ki vsebuje vse potrebne gradnike za gradnjo, testiranje in objavo projektov .NET. Osnovna slika vključuje orodja, kot so komplet za razvoj programske opreme .NET (ang. .NET SDK), komplet za razvoj programske opreme operacijskega sistema Android (ang. Android SDK) ter gradbene komponente za aplikacije Android, kar omogoča celovit razvoj znotraj vsebniškega okolja.

V našem primeru smo sliko nadgradili z dodatnimi komponentami, ki so ključne za izvajanje testov UI:

- Tehnologija NodeJS [10] – nameščena je bila za delovanje strežnika Appium. Naložena je bila trenutno zadnja stabilna verzija 23.
- Knjižnica Appium – nameščena je bila z uporabo upravitelja paketov npm, saj Appium deluje kot aplikacija NodeJS. Dodatno smo namestili še gonilnik appium-uiautomator2-driver [12], ki je potreben za testiranje naprav Android.
- Virtualna naprava operacijskega sistema Android AVD (ang. Android virtual device), ki omogoča izvajanje aplikacij v virtualnem okolju. Uporabili smo vmesnik za programiranje aplikacij operacijskega sistema Android verzije 34, naprava je bila Pixel 5 [11].

Za izvajanje testov smo pripravili vsebnik na podlagi opisane slike in zasnovali postopek, ki se izvede znotraj tega okolja. Potrebno je bilo izvesti več korakov, ki so opisani v nadaljevanju.

3.1. Priprava testov v ogrodju .NET MAUI

Prvi korak v procesu avtomatiziranega testiranja je priprava same testne kode, ki se izvaja znotraj vsebnika. Za ta namen smo uporabili ogrodje za pisanje testov v jeziku C# v kombinaciji s knjižnico Appium. Naša osnovna testna logika je zapisana v razredu UITest1, ki vsebuje metodo AppLaunches. Ta metoda poskrbi, da se ob zagonu aplikacije ustvari zaslonski posnetek zaslona naprave (Slika 1). Test uporabi metodo GetScreenshot() za zajem trenutnega stanja zaslona naprave in ga shrani kot datoteko s formatom PNG, kar omogoča vizualno potrditev, da je aplikacija uspešno zagnana.

Za uspešno izvajanje testov na virtualni napravi operacijskega sistema Android smo pripravili platformno-specifično konfiguracijo, ki inicializira okolje Appium in vzpostavi povezavo z virtualno napravo. To konfiguracijo smo zapisali v razred AppiumSetup, ki vsebuje metodi, ki se izvedeta

```
12  public class UITest1 : BaseTest
13  {
14      [Test]
        1 reference
15      public void AppLaunches()
16      {
17          App.GetScreenshot().SaveAsFile($"{nameof(AppLaunches)}.png");
18      }
19  }
```

Slika 1: Demonstracijska testna metoda ob zagonu aplikacije .NET MAUI.

enkrat pred začetkom vseh testov in enkrat po koncu (Slika 2).

Ta nastavitev omogoča Appiumu, da:

- vzpostavi povezavo z lokalnim strežnikom Appium,
- uporablja gonilnik UIAutomator2, ki je priporočeni gonilnik za operacijski sistem Android,
- prepozna aplikacijo po paketu (AppPackage) in začetni aktivnosti (AppActivity),
- samodejno zažene virtualno napravo z vnaprej definiranim profilom AVD,
- obdrži stanje aplikacije s parametrom NoReset, da se izogne odstranitvi knjižnic za hitro objavo paketa.

Kot lahko opazimo, smo v nastavitvah povečali časovno omejitev zagona aplikacije iz privzetih 60 sekund na kar 6 minut. Do te spremembe je prišlo, ker je postopek inicializacije okolja Appium, zagona virtualne naprave in vzpostavitve povezave z aplikacijo izredno dolgotrajen – še posebej, kadar se izvaja na računalnikih z omejenimi strojno-programskimi zmogljivostmi. V praksi se je izkazalo, da lahko celoten proces od zagona vsebnika do zaključka testov traja tudi 30 minut. To predstavlja eno ključnih slabosti poganjanja testov UI znotraj vsebnikov, saj je postopek še vedno neoptimiziran, strojno zahteven in časovno potraten. Zaradi teh razlogov tovrstno testiranje trenutno ni primerno za hitre in pogoste iteracije, kot jih zahtevajo moderne prakse CI/CD.

```
9   [SetUpFixture]
    1 reference
10  public class AppiumSetup
11  {
        3 references
12      private static AppiumDriver? driver;
13
        1 reference
14      public static AppiumDriver App => driver ?? throw new NullReferenceException("AppiumDriver is null");
15
16      [OneTimeSetUp]
        0 references
17      public void RunBeforeAnyTests()
18      {
19          var androidOptions = new AppiumOptions
20          {
21              AutomationName = "UIAutomator2",
22              PlatformName = "Android",
23          };
24
25          androidOptions.AddAdditionalAppiumOption(MobileCapabilityType.NoReset, "true");
26          androidOptions.AddAdditionalAppiumOption(AndroidMobileCapabilityType.AppPackage, "com.impol.mauitemplate");
27          androidOptions.AddAdditionalAppiumOption(AndroidMobileCapabilityType.AppActivity, $"com.impol.mauitemplate.MainActivity");
28          androidOptions.AddAdditionalAppiumOption("uiautomator2ServerLaunchTimeout", 360000);
29          androidOptions.AddAdditionalAppiumOption("uiautomator2ServerInstallTimeout", 360000);
30          androidOptions.AddAdditionalAppiumOption("avd", "Pixel_5_API_34");
31
32          driver = new AndroidDriver(new Uri("http://127.0.0.1:4723/"), androidOptions, TimeSpan.FromMinutes(10));
33      }
34
35      [OneTimeTearDown]
        0 references
36      public void RunAfterAnyTests()
37      {
38          driver?.Quit();
39      }
40  }
```

Slika 2: Domensko specifična konfiguracija sistema Appium.

Poleg zgornjih nastavitev smo v glavni aktivnosti aplikacije Android MainActivity.cs dodali naslednji atribut:

[Register("com.app.mauitemplate.MainActivity")]

Ta atribut zagotovi, da je razred MainActivity pravilno registriran znotraj zagonske datoteke aplikacije in da ga Appium lahko najde ter zažene. Brez tega atributa Appium izpiše napako, da zagonska aktivnost ne obstaja ali je ne more zagnati, kar bi preprečilo uspešno inicializacijo testnega okolja.

3.2. Zagon strežnika Appium

Programska koda je pripravljena, sedaj sledi opis postopka zagona testov znotraj vsebnika. Najprej smo zagnali strežnik Appium z uporabo ukaza:

appium --log-level error

Appium je poslušal na privzetih vratih protokola TCP 4723 in čakal na povezavo z odjemalcem (t.j. testnim projektom .NET MAUI). Le-ta mu ob zagonu testov pošilja ukaze za izvajanje testov nad virtualno napravo.

3.3. Zagon virtualne naprave Android

Da bi bilo okolje primerno za avtomatsko izvajanje v cevovodu CI/CD, kjer ni grafičnega uporabniškega vmesnika, smo virtualno napravo zagnali v t. i. načinu brez glave (ang. Headless). Za ta namen smo uporabili naslednji ukaz:

emulator -avd "Pixel_5_API_34" -gpu off -noaudio -no-boot-anim -netdelay none -no-window -no-snapshot -memory 4096 -partition-size 4096 -no-accel

Pojasnilo parametrov:

- -avd "Pixel_5_API_34" – ime profila virtualne naprave AVD (Pixel 5 z API 34), ki je bil prej konfiguriran in registriran.
- -gpu off – izklop grafičnega pospeševanja, saj ni potrebe po tej funkcionalnosti v načinu brez glave.
- -noaudio – onemogoči zvok v virtualni napravi.
- -no-boot-anim – preskoči animacijo ob zagonu naprave, kar pohitri zagon.
- -netdelay none – izklopi simulacijo mrežne zakasnitve.
- -no-window – zažene virtualno napravo brez okna, v ozadju.
- -no-snapshot – onemogoči uporabo posnetkov virtualne naprave za večjo stabilnost.
- -memory 4096 – dodeli 4 GB pomnilnika za boljšo zmogljivost.
- -partition-size 4096 – dodeli 4 GB prostora za virtualni disk, da se preprečijo napake ob namestitvi aplikacij.
- -no-accel – onemogoči strojno pospeševanje, saj v nekaterih vsebnikih to ni na voljo.

Izbrana konfiguracija omogoča zanesljiv in hitrejši zagon virtualne naprave znotraj vsebnika, kar je ključno za ponovljivo testno okolje.

3.4. Namestitev in zagon aplikacije na virtualni napravi

Preden smo lahko zagnali teste UI z ukazom dotnet test, smo morali najprej zagnati aplikacijo na virtualni napravi. Če tega koraka ne izvedemo, se pri zagonu testov pojavi napaka:

Activity name '.com.app.mauitemplate.MainActivity' used to start the app doesn't exist or cannot be launched! Make sure it exists and is a launchable activity.

Do te napake pride, ker virtualna naprava ne vsebuje nobene aplikacije, ki bi jo Appium lahko zagnal ali nadziral. Zato smo aplikacijo ročno namestili in zagnali s pomočjo ukaza:

```
dotnet build -t:Run -f net8.0-android /p:AndroidDevice=Pixel_5_API_34
App.MauiTemplate/App.MauiTemplate.csproj
```

S tem smo zagotovili, da je aplikacija dejansko nameščena in zagnana na virtualni napravi, kar je pogoj za uspešen zagon testov UI.

3.5. Zagon testov

Po zagonu virtualne naprave, strežnika Appium in uspešnem zagonu aplikacije na napravi, smo lahko iz konzole sprožili zagon testov z ukazom:

```
dotnet test
```

Testi so se z uporabo knjižnice Appium povezali z virtualno napravo in izvedli vse definirane korake. Rezultati testov so se prikazali v konzoli.

4 Avtomatizacija testov uporabniških vmesnikov

Postopek zagona testov uporabniškega vmesnika, kot je bil opisan v prejšnjem poglavju, smo integrirali v cevovod CI platforme GitLab z namenom doseči popolno avtomatizacijo preverjanja delovanja aplikacije. V okviru obstoječega cevovoda smo dodali namenski postopek (ang. job), ki skrbi za pripravo okolja in izvedbo testov znotraj vsebnika. Cevovod sledi naslednjim korakom (Slika 3):

- Zažene se virtualna naprava Android v načinu brez glave.
- Počaka se na popoln zagon virtualne naprave z večkratnim preverjanjem lastnosti sys.boot_completed.
- Ko je virtualna naprava pripravljena, se v ozadju zažene strežnik Appium.
- Aplikacija .NET MAUI se zgradi in zažene na virtualno napravo.
- V zadnjem koraku se izvedejo avtomatizirani testi UI z uporabo ukaza dotnet test.

Testi so zasnovani tako, da se po potrebi poženejo do trikrat zapored. Razlog za to odločitev je v tem, da se v praksi pogosto zgodi, da prva iteracija testov odpove zaradi prehitrega zagona aplikacije ali strežnika Appium, preden se virtualna naprava v celoti vzpostavi in stabilizira. V takem primeru ni smiselno takoj označiti testov kot neuspešne, saj je lahko razlog zgolj v časovni nedoslednosti priprave okolja. Z dodatnimi poskusi zmanjšamo število lažno negativnih rezultatov in s tem povečamo zanesljivost testnega procesa.
Pomembno je poudariti, da gre za precej kompleksen in časovno zahteven proces. Četudi se vsi koraki izvedejo uspešno, lahko en sam zagon testov traja tudi od 20 do 30 minut. Glavni razlogi za dolgotrajnost so:

- počasna vzpostavitev virtualne naprave v virtualiziranem okolju brez strojne pospešitve,
- zagon strežnika Appium in vzpostavitev povezave z napravo,
- grajenje aplikacije in njen prenos na virtualno napravo,
- morebitni vmesni izpadi ali napake, ki sprožijo dodatne poskuse izvajanja testov.

Takšna arhitektura za avtomatizacijo testov UI sicer zagotavlja zanesljivo preverjanje aplikacije v kontroliranem okolju, vendar ni optimalna za hitro iterativno razvijanje. Zaradi njene počasnosti in porabe virov predstavlja ta pristop predvsem osnovo za občasno preverjanje stabilnosti uporabniškega vmesnika, ne pa za vsakodnevno uporabo znotraj intenzivnega cikla cevovoda CI/CD.

5 Zaključek

V prispevku smo predstavili celoten postopek priprave avtomatiziranega testiranja uporabniškega vmesnika za mobilno aplikacijo, razvito na podlagi ogrodja .NET MAUI. Najprej smo pripravili osnovne teste UI s pomočjo knjižnice Appium, kjer smo ob zagonu aplikacije posneli zaslonski posnetek naprave. V nadaljevanju smo opisali, kako je bilo potrebno za delovanje testov implementirati tudi platformno specifično kodo za operacijski sistem Android, vključno z nastavitvijo gonilnika ter ustrezno označiti glavno aktivnost v aplikaciji. Poseben poudarek smo namenili selitvi testnega okolja v vsebnik, ki teče na operacijskem sistemu Linux. Vse skupaj smo vključili v cevovod platforme GitLab, kjer se virtualna naprava, strežnik Appium in sama aplikacija samodejno vzpostavijo ter se poskrbi za izvedbo testov z možnostjo ponovnega zagona v primeru neuspeha. S tem pristopom smo dosegli popolno avtomatizacijo testiranja v nadzorovanem okolju.

Med ključne prednosti pripravljene rešitve štejemo predvsem njeno avtomatizacijo, saj se celoten postopek odvije brez ročnega posredovanja. Zagon testov v vsebniku omogoča neodvisnost od operacijskega sistema, kar pomeni, da lahko teste izvajamo kjerkoli, kjer imamo na voljo podporo k tehnologiji Docker. Poleg tega takšna rešitev nudi ponovljivost rezultatov, saj je vsaka testna seja zagnana v svežem okolju, kar zmanjšuje vpliv zunanjih dejavnikov. Postopek predstavlja tudi dobro osnovo za nadaljnje razširitve, kot so testiranje različnih scenarijev, platform ali vključitev poročanja o pokritosti.

Po drugi strani ima takšen pristop tudi svoje slabosti. Izvajanje testov na virtualni napravi Android v glavi cevovoda CI je zelo časovno zahtevno – posamezna seja lahko traja tudi 30 minut, kar je za hitro iteracijo v razvojnem ciklu pogosto predolgo. Dodatno kompleksnost prinaša konfiguracija okolja. Pravilna nastavitev virtualne naprave, strežnika Appium in aplikacije zahteva precej tehničnega znanja in natančnosti. Delovanje virtualne naprave brez strojnega pospeševanja je počasno, kar še dodatno zmanjšuje učinkovitost in odzivnost testov.

Kljub navedenim izzivom predstavlja pripravljena rešitev učinkovit temelj za avtomatizirano testiranje aplikacij .NET MAUI. Omogoča zgodnje zaznavanje napak v uporabniškem vmesniku in pripomore k višji kakovosti končnega izdelka. V prihodnosti bi bilo smiselno nadalje optimizirati hitrost zagona, razdeliti teste glede na prioriteto ter poiskati možnosti za vzporedno izvajanje testov na različnih napravah.

```
.ui_test:
  script:

  - echo "Waiting for emulator to boot..."
  - |
    for i in {1..30}; do
      if adb shell getprop sys.boot_completed | grep -m 1 "1"; then
        echo "Emulator booted."
        break
      else
        echo "Waiting for emulator... ($i)"
        sleep 10
      fi
    done

  - echo "Waiting extra time for emulator to stabilize..."
  - sleep 10

  - echo "Starting Appium..."
  - nohup appium > appium.log 2>&1 &

  - sleep 10

  - echo "Building and running .NET MAUI app..."
  - dotnet build -t:Run -f net8.0-android /p:AndroidDevice=Pixel_5_API_34 MauiTemplate/MauiTemplate.csproj

  - echo "Running tests with up to 3 retries if needed..."
  - |
    attempt=1
    max_attempts=3      Alen Granda, yesterday • SAP-19590 -Try with demo pipeline
    while [ $attempt -le $max_attempts ]; do
      echo "Test attempt #$attempt"
      dotnet test --logger "trx;LogFileName=test_results_$attempt.trx"
      exit_code=$?

      if [ $exit_code -eq 0 ]; then
        echo "Tests passed on attempt #$attempt"
        break
      else
        echo "Tests failed on attempt #$attempt"
      fi

      if [ $attempt -eq $max_attempts ]; then
        echo "All $max_attempts attempts failed. Exiting with failure."
        exit 1
      fi

      attempt=$((attempt + 1))
      echo "Retrying tests in 15 seconds..."
      sleep 15
    done
```

Slika 3: Postopek cevovoda v platformi GitLab, namenjen izvedbi testov uporabniškega vmesnika.

Literatura

[1] https://medium.com/innovies-club/running-android-emulator-in-a-docker-container-19ecb68e1909, Running Android Emulator in a Docker Container, Salem Amr, obiskano 3. 7. 2025.

[2] https://dotnet.microsoft.com/en-us/apps/maui, .NET Multi-platform App UI, obiskano 3. 7. 2025.

[3] https://www.lambdatest.com/blog/visual-testing-challenges, The Challenges of Visual Testing [With Solutions], Gazala Saniya, obiskano 3. 7. 2025.

[4] https://muuktest.com/blog/manual-software-testing, Manual Software Testing: A Practical Guide, The MuukTest Team, obiskano 3. 7. 2025.

[5] https://www.testim.io/blog/test-automation-benefits/, Benefits of Test Automation: A Complete Guide, obiskano 3.7. 2025.

[6] https://appium.io/docs/en/latest/, Appium, obiskano 3. 7. 2025.

[7] https://about.gitlab.com/, About GitLab, obiskano 3. 7. 2025.

[8] https://appium.io/docs/en/2.1/intro/drivers/, Intro to Appium drivers, obiskano 3. 7. 2025.

[9] 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.

[10] https://nodejs.org/en, Node.js – Run JavaScript everywhere, obiskano 4. 7. 2025.

[11] https://developers.google.com/android/images, Factory images for Nexus and Pixel devices, obiskano 4. 7. 2025.

[12] https://github.com/appium/appium-uiautomator2-driver, Appium UiAutomator2 Driver, obiskano 4. 7. 2025.

[13] https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm, Model-View-ViewModel (MVVM), obiskano 7. 7. 2025.