PENDLER & DFA

Publikus dokumentáció

2020. JÚLIUS 20.

DONGOTEAM

info@dongoteam.hu

Egy üres oldal…

Pendler 2. TestEngine

Dongo Team

PENDLER 2 & DFA [UI]

ÁLTALÁNOS INFORMÁCIÓK

MI AZ A PENDLER & DFA?

Egy olyan alkalmazás és könyvtárcsomag, mely webes alkalmazásokhoz end-to-end tesztesetek tervezésére, írására, futtatására (mind lokális számítógépen, mind szerveren), kategorizálására és karbantartására ad egy teljes körű megoldást.

Fő komponensei a következőek:

- **Pendler2 szerver**
  Egy keret webalkalmazás.
- **DFA**
  A Pendler2 webalkalmazás számára írt bővítmény, mely lehetővé teszi DFA típusú tesztesetek kezelését és futtatását.
- **DFA környezet**
  DFA tesztek futtatását elvégző környezet, mely lehet lokális, lehet Docker alapú szerver és natív típusú szerver is.
- **Ego**
  API böngésző kezelésére.
- **Madder**
  Dokumentumkezelő bővítmény a Pendler2 webalkalmazás számára.
- **Pendler Cloe**
  Böngészőbe épülő bővítmény a böngészőben történt felhasználói interakciók automatikus rögzítésére.

Jelen dokumentáció ezen komponenseket együtt ismerteti.

JOGOSULTSÁGOK

Az alkalmazás kétfajta kategóriát alkalmaz a jogosultság meghatározására. A Pendler jogosultsági kategóriát és a DFA jogosultsági kategóriát. Az előbbi az alkalmazás körüli műveleteket szabályozza, mint a felhasználó meghívása, új tároló létrehozása; míg utóbbi egy darab tárolóra vonatkozó engedélyeket tartalmazza, mely tárolónként is lehet eltérő.

Felhasználói szintek és jogosultsági csoportok az egyes a kategóriákra bontva a következők:

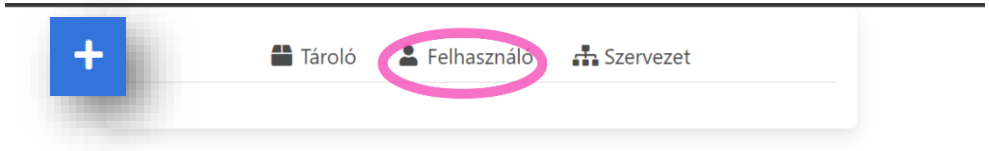
- Pendler jogosultsági kategória
  - Általános felhasználó
  - Adminisztrátor
- DFA jogosultsági kategória
  - Olvasó
  - Szervező
  - Általános
  - Adminisztrátor

FELHASZNÁLÓKEZELÉS

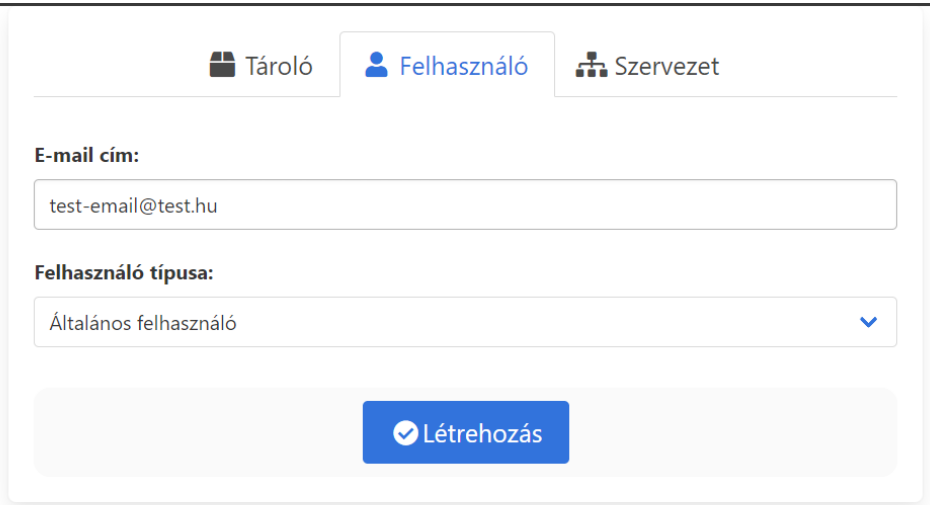
FELHASZNÁLÓ LÉTREHOZÁSA

Új felhasználót regisztrálni a rendszerben a Pendler::Adminisztrátor típusú felhasználók tudnak.

Bejelentkezés után a bal oldali sávon kattintsunk a „+” jelre, majd válasszuk ki a „Felhasználó” menüpontot!



A megjelenő beviteli űrlap a regisztrálandó felhasználó e-mail címét és bejegyzett jogkörét követeli meg tőlünk, mely utóbbi lehet „Általános felhasználó”, illetve „Adminisztrátor”.



A megfelelő adatok megadása után, rányomhatunk a „Létrehozás” feliratú gombra. Ellenben mielőtt megnyomnánk, ellenőrizzük le a megadott e-mail címet gondosan! A rendszer nem kér második ellenőrzést, a gombra való kattintással ki is küldi a regisztrációra való meghívót.

A felhasználó létrehozása, e-mail kiküldése több időt is igénybe vehet! Türelmesen várjuk meg, amíg a betöltő animáció eltűnik a gombról, illetve megjelenik a meghívó kiküldésének sikerességéről tájékoztató üzenet számunkra.

A megadott e-mail címre kiküldött meghívó tartalmazza a további instrukciókat. Tartalmaz egy regisztrációs linket, melyet megnyitva a felhasználó megadhatja adatait és használandó jelszavát. Az itt megadott jelszót a rendszerből nincs lehetőségünk kinyerni, gondosan válasszuk meg!

FELHASZNÁLÓ TÖRLÉSE

Felhasználót a rendszerből törölni az adminisztrátorok tudnak.

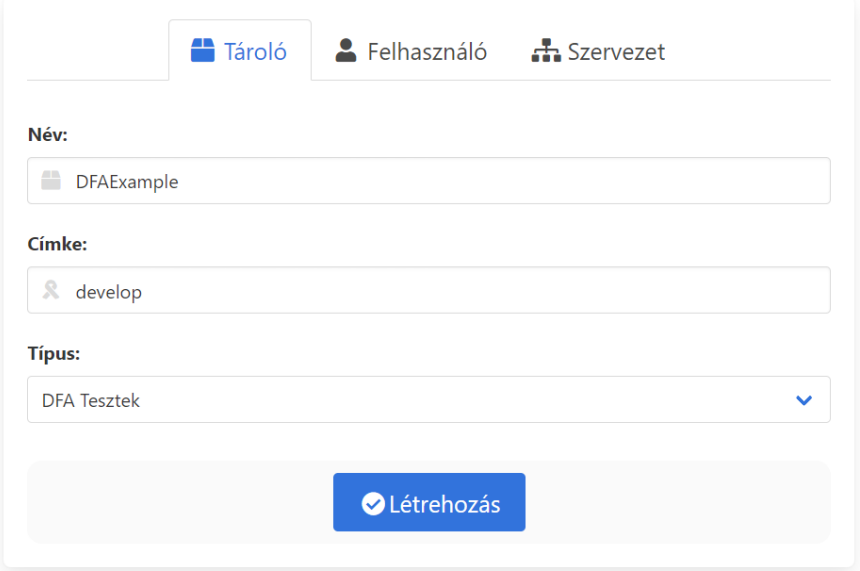
TÁROLÓKEZELÉS

TÁROLÓ LÉTREHOZÁSA

Új tárolót létrehozni mind a Pendler Általános felhasználó, mind a Pendler Adminisztrátor tud.

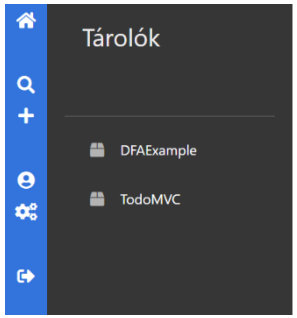
Kattintsunk a bal oldali sávban a „+” gombra, majd a „Tároló” feliratú fülre.

A megjelenő űrlapon meg kell adnunk a létrehozandó tároló nevét, címkéjét és típusát. A tároló egyedi azonosítóját a neve és címkéje teszi ki együtt. Azaz létrehozhatunk különböző címkével ellátott tárolókat azonos névvel is.



TÁROLÓ KERESÉSE

A főoldalon találhatjuk az adott felhasználó által látható tárolókat a „Tárolók” oldalsávon. Ezen oldalsáv felső részében látható vonal egy beviteli mezőt ábrázol, melybe belekattintva beleírhatjuk a keresendő szövegrészletünket.

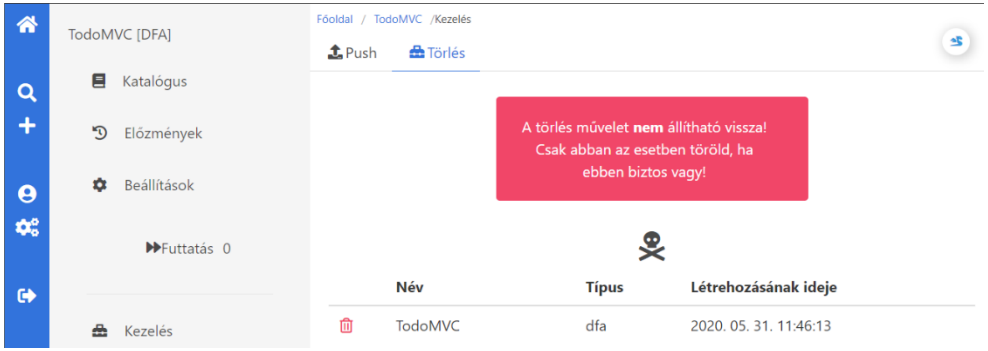


A beírt szöveget a tárolók nevében keresi és találati listát dinamikusan változtatja.

TÁROLÓ TÖRLÉSE

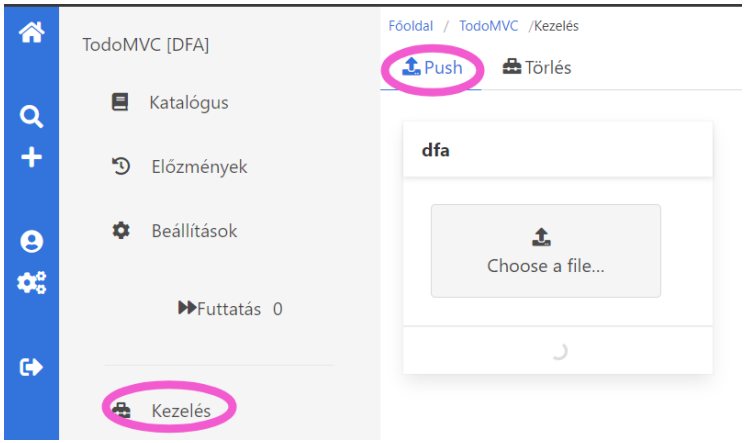
Tároló törléséhez a „Tárolók” oldalsávon megnyitva a „Kezelés” menüpontra kattintsunk. Az oldalon megjelenő „Törlés” fület válasszuk ki!

A figyelmeztető üzenet elfogadása után jelennek az adott csoportban általunk látható tárolók és mely tárolókhoz van törlési jogosultságunk egy kattintható szemetes ikon is megjelenik. Erre a szemetes ikonra kattintva törölhetjük a tárolót.



TÁROLÓ FRISSÍTÉSE

A tárolók frissítését érdemes az automatikus folyamattal megvalósítani script segítségével. Például *githook*, vagy *CI/CD* alkalmazásba bekötött megvalósítással.



Ellenben, ha szükséges, ezt kézzel is meg tudjuk tenni. Első lépésként a projektfájlt csomagoljuk be egy *zip* állományba! Az érintett tárolót nyissuk meg a „Tárolók” oldalsávon, majd Kezelés menüponton belül a „Push” fület válasszuk ki! Ezen fül alatt megjelenik az összes, adott tároló csomagban jelen lévő modul, melyekhez jogosultságunk, illetve frissítési jogosultságunk is van. A *„Choose a file…”* gombra kattintva válasszuk ki a feltölteni kívánt zip állományt, majd nyomjunk rá a feltöltés gombra!

KATALÓGUS

A katalógusban többféle rendezettség alapján is megtekinthetjük a projektet. Alapértelmezett nézet a „Fájlstruktúra” nézet, mely a projekt állományait eredeti fájlstruktúrája alapján jeleníti. Emellett viszont hozzáadhatunk további projektspecifikus nézeteket, mely a teszteseteket más struktúrába rendezi és mutatja számunkra.

További különbség az alapértelmezett „Fájlstruktúra” nézet és az egyedi nézetek között, hogy míg „Fájlstruktúra” alatt minden egyes állomány látható, addig az egyedi nézetek csak a teszteseteket képesek megjeleníteni, illetve ezek közül is csak az alapvető szintaktikának helyes teszteseteket.

Egyedi nézet definícióját a projekt konfigurációs fájljában, új *„$directory”* típusú meta definíció felvételével és ezen metaadat a tesztesetek kitöltésével készíthetünk el.

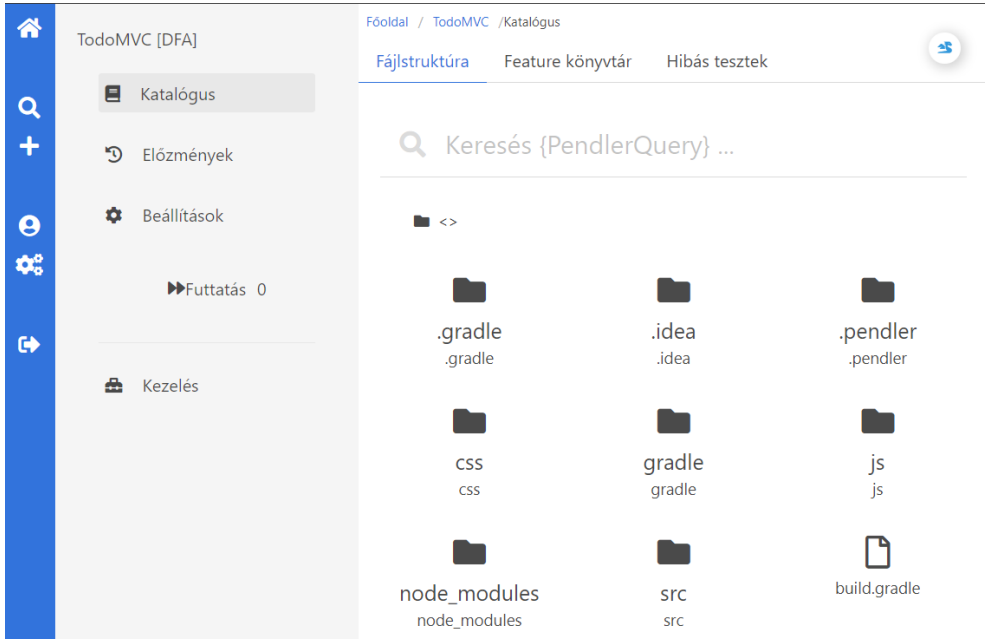
A konfigurációs fájl ezen része példaként a következőképpen nézhet ki:

```
config {
  definitions {
    metas = [
      {
        name = "FeatureDirectory"
        type = "$directory"
        languages = [
          {
            name = "en"
            value = "Feature directory"
          }
          {
            name = "hu"
            value = "Feature könyvtár"
          }
        ]
      }
    ]
  }
}
```

Továbbá a katalógusban láthatunk egy „Hibás tesztek” feliratú fület is. Ezen fület megnyitva tájékozódhatunk, mely tesztesetek nem felelnek meg az alapvető szintaktikai szabályoknak, ezáltal nem is jelennek meg a katalógus egyedi nézeteiben.

TESZTESET A KATALÓGUSBAN

Egy tesztesetet megnyitva a katalógusban láthatjuk azon adatait, meta információit, megtekinthetjük futási előzményeit, továbbá gráf alapú megjelenítését és valós forráskódját.



TESZTESET ADATAI

Egy teszteset adatai között a katalógusban három különböző kategóriába sorolható információval találkozhatunk.

- **Információs metaadatok**, melyeket a felületen egy gombnyomásra törölhetünk, vagy épp újakat vehetünk fel. Ezen metaadatokat felhasználhatjuk a katalógus szűrőmezőjében keresési feltételként.
- **Statikus metaadatok**, melyeket a teszteset fejlesztője adott hozzá a teszthez a fejlesztés során. Módosításához forráskód módosítására is szükség van. Ezen metaadatokat felhasználhatjuk a katalógus szűrőmezőjében keresési feltételként.

- **Teszt leírása**, melyet a fejlesztő a teszteset fejlesztésekor készített el és adott hozzá Markdown formátumban. Módosítása a teszteset forráskódjával lehetséges.

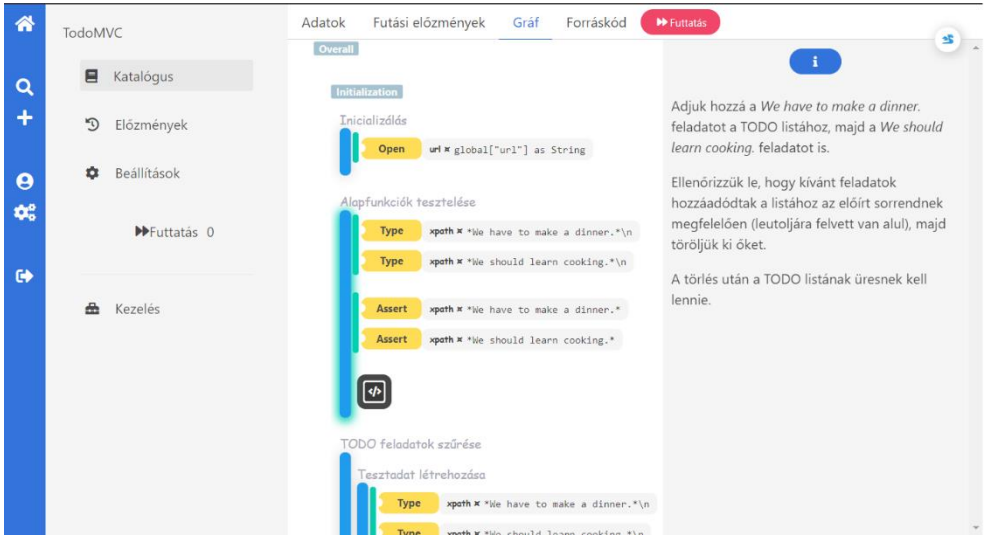
FUTÁSI ELŐZMÉNYEK

A „Futási előzmények” menüpontra kattintva láthatjuk azon előzményeket, mikor az adott, kiválasztott tesztesetet is futtattuk. Továbbá ezen előzmények között szűrhetünk, kereshetünk.

GRÁF

A vizuális gráf (röviden gráf) a teszteset tartalmának egy vizuális megjelenítése, mely kevesebb fejlesztői tapasztalattal rendelkező kollégák számára segít, illetve ad egy gyorsabb áttekintési lehetőséget.

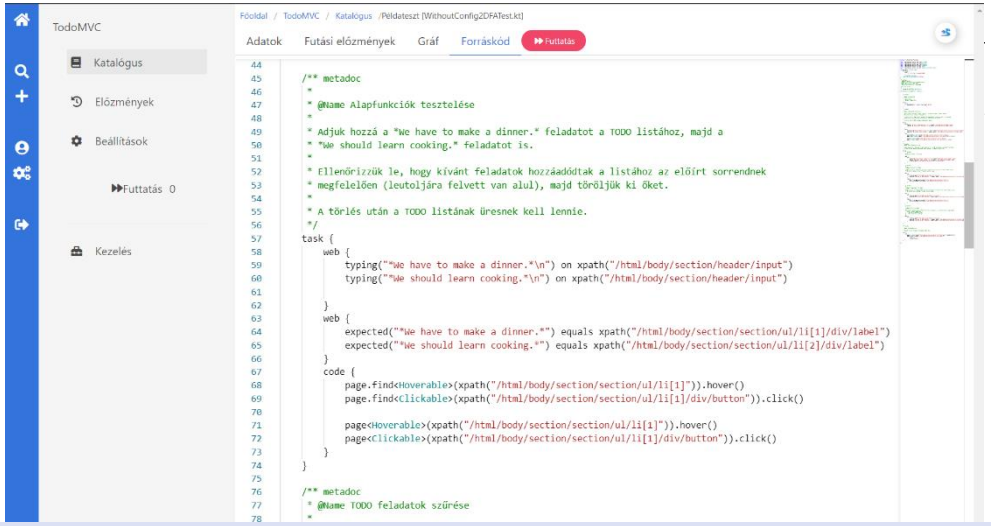
Továbbá ezt a gráfot alkalmazzuk teszteset futási előzményében hibafelderítése.



FORRÁSKÓD

A forráskód menüpontot megnyitva láthatjuk az adott teszteset teljes forráskódját.

Habár kényelmi okokból a forráskódot tudjuk szerkeszteni, ellenben a módosításokat az alkalmazás elmenteni nem képes.



KERESÉS - PENDLER QUERY

A katalógus keresőmezőjében alkalmazhatunk egy speciális erre a célra szánt egyszerű lekérdező nyelvet. Ebben a nyelvben megadhatunk szűrési feltételeket a meta adatokra vonatkozóan, illetve teszteset nevére, tartalmára vonatkozó tartalmazási feltételt.

Szintaktikája a következő:

```
@MetaKey=MetaValue, @MetaKey=MetaValue, TextContent
```

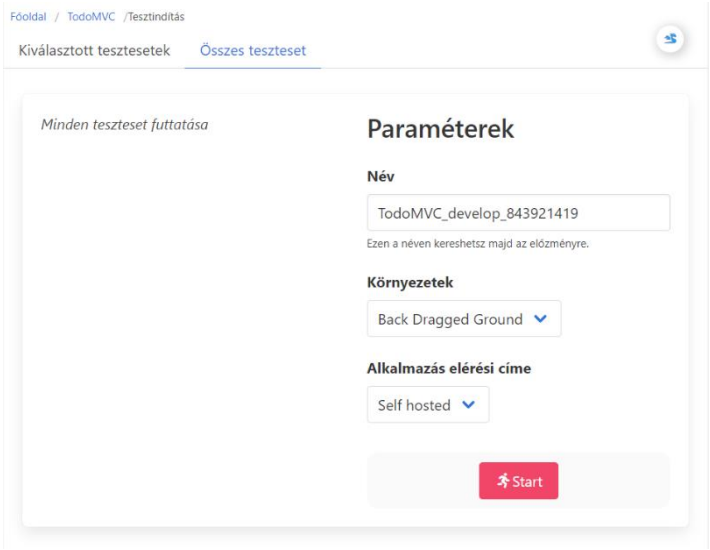
Metaadatokra való feltételeket „@” jellel adhatjuk meg, majd vesszővel elválasztva folytathatjuk a következő feltétellel.

Példa:

```
@author=ekiss, Elemo calculation
```

FUTTATÁS

A futtatás menüpontot kiválasztva elindíthatjuk az összes tesztesetet, vagy a katalógusban futtatásra megjelölt teszteseteket indíthatjuk el.

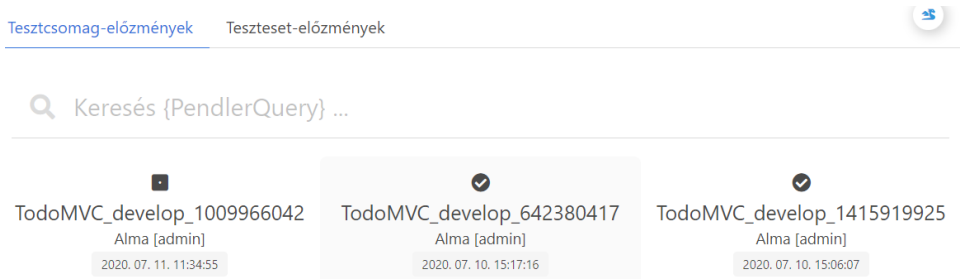


A paraméterek részen található „Név” beviteli mező minden esetben látszik, ezen a néven lesz látható az előzmények között a futási eredmények. A további mezők pedig az adott projektre vonatkozóan egyedileg definiáltak.

ELŐZMÉNYEK

Az előzmények menüpontot kiválasztva kereshetünk a korábban futtatott tesztcsomagok, tesztesetek között, illetve az aktuálisan futó tesztesetek között is. Valamint a keresés mellett, megnyithatjuk őket, megnézhetjük tartalmukat.

TESZTCSOMAG-ELŐZMÉNY



A tesztcsomag-előzmények panelen láthatjuk az egyes tesztcsomag-előzményeket, továbbá kereshetünk, szűrhetünk közöttük.

Egy elem rendelkezik állapotát jelző ikonnak, névvel, a tesztcsomag futtatását kérvényező felhasználó nevével és azonosítójával, valamit azt az időpontot mikor a tesztcsomag futtatása megkezdődött

Az állapotot jelző ikonok a következőek lehetnek:

ikon	*jelentés*
	A tesztcsomag jelenleg még fut, vagy várakozó sorban áll, hogy megkezdhesse futását.
	A tesztcsomag befejezte futását, ellenben egy (blokkoló) hiba történt a futása során.
	A tesztcsomag befejezte futását, ellenben egy figyelmeztető hiba történt a futása során.
	A tesztcsomag sikeresen befejezte futását, hiba nem történt a futtatás során.
	Habár a tesztcsomag befejezte a futását, de semmilyen ellenőrzés nem történt a futás során.

ALAPADATOK

Az „Alapadatok” fülre rákattintva láthatjuk a tesztcsomaghoz kapcsolódó legfontosabb információkat. Mikor lett elindítva a tesztcsomag, ki indította el, illetve pontosan mely teszteseteket foglalja magába. Az egyes tesztesetek előzményét megnyithatjuk, vagy a katalógus oldalára ugorhatunk, de lehetőség van egy a rendszertől függetlenül, önállóan is megjeleníthető jegyzőkönyv generálására és letöltésére is.

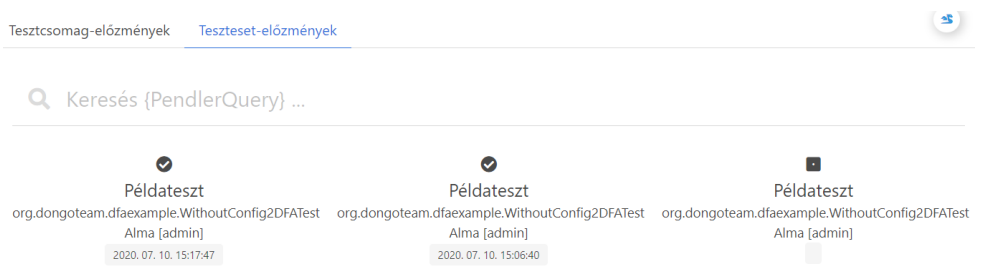
TERMINÁL

A terminál fülre rákattintva megtekinthetjük a tesztfutás során alapértelmezett kimenetre írt úgynevezett log információkat. Ez a lehetőség tapasztaltabb felhasználók számára hasznos abban az esetben, hogy valamilyen hiba miatt a tesztcsomag nem képes elindulni. Például a tesztesetek fordításához nem érhető el valamely külső erőforrás, esetleg maga a fordítást nem sikerült végrehajtani egy fordítási hiba következtében.

TÖRLÉS

A törlés gombra kattintva, értelemszerűen töröljük az adott csomag előzményt a rendszerből. Egy tesztcsomag-előzmény önmagában is nagy méretű, ezért ez a művelet egy valós törlést hajt végre. Végrehajtása után már nem visszaállítható.

TESZTESET-ELŐZMÉNY



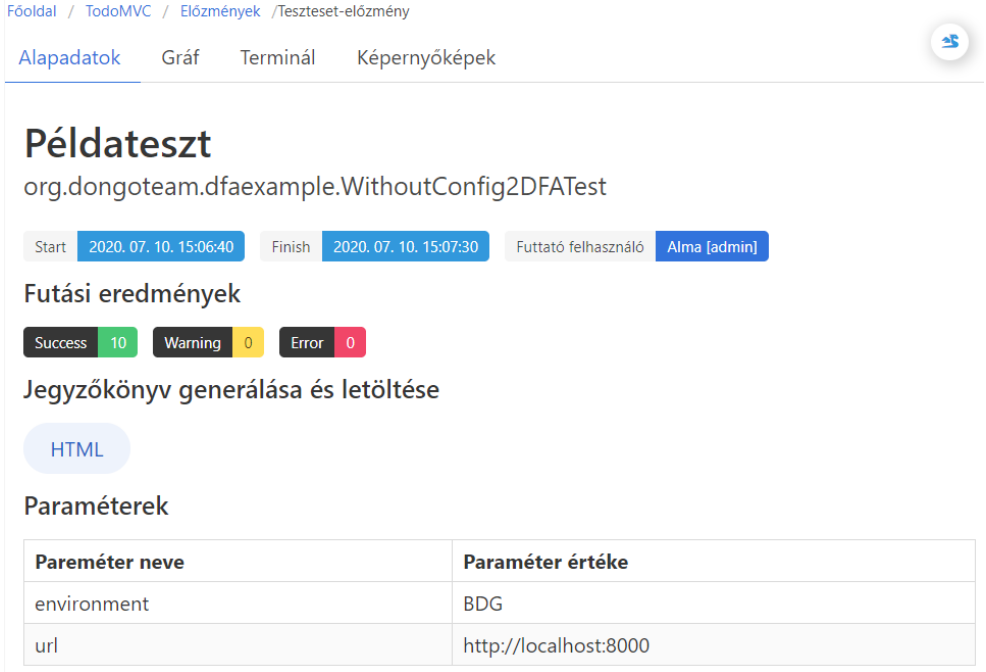
A teszteset-előzmények panelen láthatjuk az egyes teszteset-előzményeket, továbbá kereshetünk, szűrhetünk közöttük.

Egy elem rendelkezik állapotát jelző ikonnak, névvel, a teszteset futtatását kérvényező felhasználó nevével és azonosítójával, valamit azt az időpontot mikor a teszteset futtatása megkezdődött. Amennyiben ez utóbbi üres, a teszteset még nem kezdte meg futását.

Az állapotot jelző ikonok a következőek lehetnek:

ikon	*jelentés*
	A teszteset jelenleg még fut, vagy várakozó sorban áll, hogy megkezdhesse futását.
	A teszteset befejezte futását, ellenben egy (blokkoló) hiba történt a futása során.
	A teszteset befejezte futását, ellenben egy figyelmeztető hiba történt a futása során.
	A teszteset sikeresen befejezte futását, hiba nem történt a futtatás során.
	Habár a teszteset befejezte a futását, de semmilyen ellenőrzés nem történt a futás során.

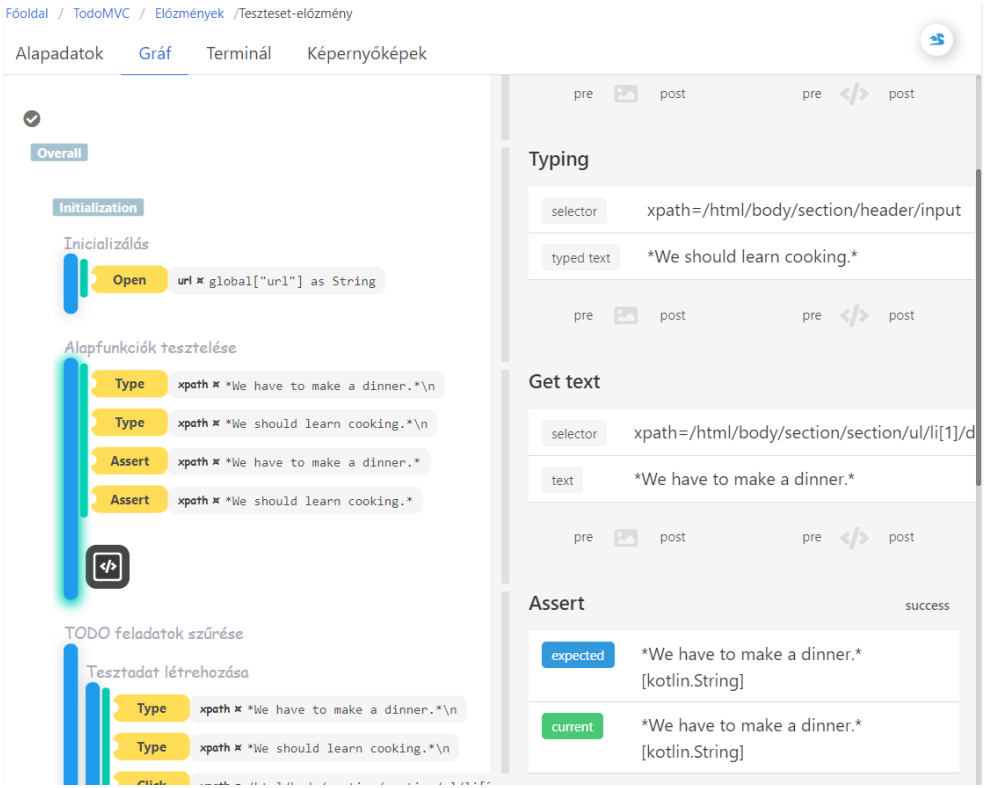
ALAPADATOK



Az alapadatok alatt megtekinthetjük, hogy az adott teszteset mikor kezdett el futni, mikor fejezte be a futását, illetve megy felhasználó indította el a teszteset futtatását. Továbbá láthatjuk a kívülről kapott, a teszteset indításakor megadott paramétereket, futási eredményeit. Illetve letölthetjük az adott teszteset futásának jegyzőkönyvét is.

GRÁF

A „Gráf” fül alatt egy a katalógusban megismert felületet látunk, ellenben jelen oldal egy kicsit bővebb, kiegészített a futási információkkal, a futtatás során történt eseményekkel a segédiformációkkal.



A gráfban egy elemre kattintva megtekinthetjük minden, az adott elemhez kapcsolódó eseményt. Illetve, ahol ennek jelentősége van, az eseményhez fűzött képernyőképeket és összegyűjtött HTML forrásokat is megtekinthetjük.

A képek és forráskódok esetében is a *„pre”* feliratú gombra kattintva az esemény végrehajtása előtti állapotot láthatjuk, míg a *„post”* gombra kattintva az adott esemény végrehajtása utána állapotot tekinthetjük meg.

TERMINÁL

A terminálban az adott teszteset-előzmény futása során keletkező események tekinthetjük meg úgynevezett *„raw”* formában. Ez a funkció tapasztaltabb felhasználók számára van a rendszerben.

KÉPERNYŐKÉPEK

Kizárólag kísérleti opció és folyamatos változás alatt áll. Jelen dokumentáció ezért nem tartalmaz leírást ehhez a funkcióhoz.

JEGYZŐKÖNYV

Jegyzőkönyvet lehet generálni az egyes tesztesetek futási eredményeiről, de akár egy tesztcsomag-előzményből is, mely tartalmazza a tesztcsomag minden tesztesetének jegyzőkönyvét.

JEGYZÉK – JOGOSULTSÁGOK

Jogosultsági kategória	**Beielentkezés**	**Felhasználó létrehozása**	**Felhasználó törlése**	**Tároló létrehozása**
Pendler \ általános felhasználó	✓	⊗	⊗	✓
Pendler \ adminisztrátor	✓	✓	✓	✓

NEVEZÉKTAN

- Teszteset
- Tesztcsomag
- Teszteset-előzmény
- Tesztcsomag-előzmény
- Katalógus
- Esemény
- Jegyzőkönyv

PENDLER 2 & DFA

[ÜZEMELTETÉS]

A Pendler2 alkalmazást többféle csomagolásban is elérhető, csomagolások szerint különböző módon lehet telepíteni. Jelen dokumentáció tartalmazza is legalább említés szintjén ezen különböző csomagokat, mi mégis első sorban a „vps package” elnevezésű csomagot javasoljuk.

VPS PACKAGE

Hardware követelmények:

- legalább 20 GB szabad hely a merevlemezen
- legalább 2 magos, legalább 2.4 GHz sebességű processzor
- HyperVM támogatás
- internetelérés legalább a Docker Hub felé

Szoftver követelmények:

- Linux alapú rendszer valamely friss (támogatott) változata
- Docker
- docker-compose 3-as vagy frissebb változata
- Powershell core 6 vagy frissebb változata
- Egy e-mail szerverhez való elérés

CSOMAG TARTALMA

A csomag a következő állományokból áll:

- extract.sh
- vps-package.tar.gz
- vps-package-repository.tar.gz

TELEPÍTÉS LÉPÉSEI

Kicsomagolás nélkül másoljuk az állományokat a telepítési mappába; majd futtassuk le az *„exract.sh”* állományt! Ez a script létrehoz egy *„vps-package”* nevű mappát, kicsomagolja a tömörített állományokat, illetve a *„vps-package”* mappán belül megfelelő helyre helyezi tartalmát. Továbbá egyéb konfigurációkat is elvégezhet és a már nem szükséges állományokat is törli.

A *„vps-package”* mappában futtassuk a *„pendler”* nevű script állományt *„start”* paraméterrel. Azaz a következő parancsot futtassuk:

```
cd vps-package
./pendler start
```

Ez a parancs miután bekérte a konfiguráláshoz szükséges adatokat, a Docker image-ket lefordítja, letölti függőségeit, majd végezetül elindítja az alkalmazás összetevőit. Első futtatása során az internetsebességtől függően több időt is igénybe vehet egészen 10 perctől 30 percig is terjedően. Későbbiekben a letöltést már nem fogja végrehajtani, a Docker komponenseket egy helyi gyorsítótárból fogja már használni.

A konfiguráláshoz szükséges adatok az alábbiak:

[domain]

A webszerver domain címe http/https prefix nélkül. Például demo73.dongoteam.hu. Ha portot is fog tartalmazni, itt azt is meg adni.

[ssl]

Itt kell megadnunk, hogy a [domain] számára megadott cím rendelkezni fog-e SSL tanúsítvánnyal, vagy sem. Azaz https:// prefixet, vagy a http:// prefixet kell-e majd alkalmaznunk.

Értéke lehet „Y” és „N”. Alapértelmezett értéke „Y”. Jelentésük az alábbi:

- **N** – Nem lesz SSL tanúsítvány a megadott [domain] címhez.
- **Y** – Igen, lesz SSL tanúsítvány a megadott [domain] címhez.

[database.schema]

Az adatbázisban az itt megadott névvel fogja a vps csomag létrehozni a benne található adatbázisban létrehozni a sémát, továbbá ezt fogja a Pendler használni adattárolásra.

Alapértelmezett értéke „pendler”. Ha bizonytalanok vagyunk ennek az értékének a jelentésében, vagy nincsenek egyedi igényeink, hagyjuk az alapértelmezett értéket.

[database.user]

A vps csomag által biztosított adatbázis eléréséhez használni kívánt felhasználónév.

Alapértelmezett értéke egy a script futtatásakor generált véletlen karaktersorozat. Ha bizonytalanok vagyunk ennek az értékének a jelentésében, vagy nincsenek egyedi igényeink, hagyjuk az alapértelmezett értéket.

[database.password]

A vps csomag által biztosított adatbázis eléréséhez használni kívánt jelszó.

Alapértelmezett értéke egy a script futtatásakor generált véletlen karaktersorozat. Ha bizonytalanok vagyunk ennek az értékének a jelentésében, vagy nincsenek egyedi igényeink, hagyjuk az alapértelmezett értéket.

[master.user]

Ez lesz az első felhasználó, mellyel a Pendler webalkalmazásba bejelentkezhetünk. A további felhasználókat ennek a felhasználónak a segítségével hívhatjuk meg.

Alapértelmezett értéke egy a script futtatásakor generált véletlen karaktersorozat. Ha bizonytalanok vagyunk ennek az értékének a jelentésében, vagy nincsenek egyedi igényeink, hagyjuk az alapértelmezett értéket.

[master.password]

Ez lesz az első felhasználó jelszava, mellyel a Pendler webalkalmazásba bejelentkezhetünk.

Alapértelmezett értéke egy a script futtatásakor generált véletlen karaktersorozat. Ha bizonytalanok vagyunk ennek az értékének a jelentésében, vagy nincsenek egyedi igényeink, hagyjuk az alapértelmezett értéket.

[email.ssl]

E-mail szervered rendelkezik SSL tanúsítvánnyal?

Lehetséges értéke „Y” és „N”. Alapértelmezett értéke „Y”. Jelentésük az alábbi:

- N – Nem, az e-mail szerver nem rendelkezik SSL tanúsítvánnyal.
- Y – Igen, e-mail szerver rendelkezik SSL tanúsítvánnyal.

[email.smtp.host]

Az e-mail szerver SMTP elérési címe.

[email.smtp.port]

Az e-mail szerver SMTP portja.

[email.smtp.user]

Az e-mail szerver SMTP felhasználója.

[email.smtp.password]

Az e-mail szerver SMTP jelszava.

[email.smtp.sender-user]

Az e-mail üzenetek küldő felhasználója.

[directories.temporary]

Mely mappát használhatja a Pendler ideiglenes állományok tárolásához?

Alapértelmezett értéke „/temp-exec-working-dir”. Ha bizonytalanok vagyunk ennek az értékének a jelentésében, vagy nincsenek egyedi igényeink, hagyjuk az alapértelmezett értéket.

Az alkalmazás végezetül a 9000-es porton fog elindulni.

Az alkalmazás leállításához az alábbi parancsot futtassuk a *„vps-package”* mappában állva:

```
cd vps-package
./pendler stop
```

FÁJLSTRUKTÚRA

Ebben a fejezetben nem térünk ki minden egyes mappa és tartalma céljára, kizárólag a hibakereséshez szükséges log állományokat helyét és a biztonsági mentéshez szükséges adatállományok helyét nézzük meg.

LOG ÁLLOMÁNYOK

A vps package két alkalmazásból áll. Egy „pendler-app” nevű alkalmazásból, illetve egy „pendler-exec” nevű alkalmazásból. Mindkét alkalmazás a „vps-package”-ben belül található „.log” mappába készíti el a napló állományai a számára megfelelő mappában.

ADATÁLLOMÁNYOK

Minden adatállomány a „vps-package” mappán belüli „.data” mappában helyezkedik el. Ez magába foglalja az adatbázisfájlokat és a pendler-app alkalmazás adatfájlait is.

[OPCIONÁLIS] NGINX KONFIGURÁLÁSA

Terjedelmi okok miatt, kizárólag az Ubuntu-ra való telepítést tartalmazza jelen dokumentáció.

A terminálba az alábbi utasítással telepíthetjük az Nginx nevű alkalmazást.

```
sudo apt update
sudo apt install nginx
```

Telepítés után az */etc/nginx/conf.d/default.conf* állományt szerkesztve irányítsuk át a bejövő kérést a localhost domain 9000-es portjára. Továbbá a maximális feltölthető fájlméretet módosítsuk 90 megabájtra. Ezeket a „proxy_pass” illetve a „client_max_body_size” kulcsok megadásával tehetjük a alapértelmezett „try” utasítás törlése vagy megjegyzésbe helyezés után.

```
location / {
  proxy_pass http://127.0.0.1:8080;
  client_max_body_size 90M;
}
```

A fájl módosítása és mentése után a következő paranccsal indítsuk újra az Nginx alkalmazást!

```
systemctl reload nginx
systemctl restart nginx
```

[OPCIONÁLIS] UFW TŰZFAL KONFIGURÁLÁSA

Habár a Pendler minden komponensében ideiglenes, gyorsan lejáró azonosítókat alkalmaz a kommunikációra és azonosításra, javasolt valamely tűzfal alkalmazása mely kizárólak a 9000-es portot, vagy külső Nginx alkalmazása esetén a 80/443-as portot engedi át a további személyes preferencia mellett.

Ubuntu alatt javasolt például az „ufw” nevű tűzfal alkalmazása.

ARCHIVÁLÁS ÉS BIZTONSÁGI MENTÉS

Biztonsági mentés készítéséhez futtassuk az alábbi parancsot:

```
./pendler backup
```

A parancs a „.backup” mappát létrehozza ha még nem létezik és dátummal ellátva ebbe a mappába készíti el a biztonsági mentést a Pendler alkalmazásban lévő adatokról.

ZOZO PACKAGE

Hardware követelmények:

- legalább 3 GB szabad hely a merevlemezen
- legalább 2 magos, legalább 1.3 GHz sebességű processzor
- HyperVM támogatás
- internetelérés legalább a Docker Hub felé

Szoftver követelmények:

- Linux alapú rendszer valamely friss (támogatott) változata
- Docker
- Java JRE 11 vagy frissebb változat
- PostgreSQL 9 vagy frissebb változat
- Egy e-mail szerverhez való elérés

Jelen dokumentáció írásakor a zozo package publikussá tétele még egyeztetés alatt áll, ezért további információt nem közlünk róla.

CLOUD PACKAGE

A cloud package automatikus skálázású komponenseket is tartalmaz számos további összetevővel együtt. Jelen dokumentáció írásakor nem publikus csomag, ezért egyéb információt nem közlünk róla.

FAQ

A vps csomag minden komponense Docker-ben fut. Miért kötelező mégis Linux alapú rendszer alatta?

Ennek oka a hivatalos Docker képfájlnak számító dockerizált PostgreSQL-ben keresendő. Ennek a képfájlnak vannak problémái Windows alatt, melyek igaz megoldhatóak nem hivatalos képfájl alkalmazásával, vagy egyedileg való elkészítésével. Igény hiányában ezek támogatását, felesleges pluszmunka elkerülésének érdekében nem vállaljuk.

Mit tehetek, ha én mégis Windows rendszeren szeretném futtatni a vps csomagot?

Nyitottak vagyunk ebben a kérdéskörben. Vedd fel velünk a kapcsolatot és megbeszélhetjük a pontos igényeid és igényed részleteit!

Pendler 2. TestEngine

Dongo Team

PENDLER 2 & DFA

[TESZTESET-FEJLESZTÉS]

Jelen dokumentáció írásakor kizárólag a Kotlin nyelven írt tesztesetek és Gradle build system mellett támogatott!

A teljes példaprojekt a következő helyen található:
https://gitlab.com/dongoteam/pedler-todomvc-example-test

INICIALIZÁCIÓ

GRADLE KONFIGURÁCIÓ

A Pendler és DFA projektben való konfigurálásához az alábbi teendőnk van:

- A dfa-environment könyvtárat vegyük fel a függőségek közé!
- A metadoc-gradle-plugin Gradle plugint vegyük fel a plugin függőségek közé és aktiváljuk is a plugint a projektünkön!
- Vegyük fel a dongoteam repository-t, illetve egy paraméteres repository-t, melyet a Pendler futtató környezete fog használni!
- Készítsünk el egy „stub” nevű taszkot, melyet a Pendler futtató környezete fog hívni!

A fentebb leírt teendők elvégzését az alábbiakban részletezzük.

```
buildscript {
  ext.pendler_main_version = '2.3.0'
  ext.kotlin_version = '1.3.72'
  ext.dfaenv_version = pendler_main_version
  repositories {
    mavenLocal()
    if(project.hasProperty('PendlerExecutor')) {
      maven {
        url "${project.property("host")}/static/maven/private-
repository/3e6e12b1-1532-4f24-a916-adb146685f2a"
      }
    }
    maven {
      url "https://public.repository.dongoteam.hu"
    }
    mavenCentral()
    jcenter()
  }
  dependencies {
    classpath "org.jetbrains.kotlin:kotlin-gradle-
plugin:$kotlin_version"
    classpath "org.dongoteam:metadoc-gradle-
plugin:$pendler_main_version"

  }
}

group 'org.dongoteam.pendler.example'
version '2.3.0'

apply plugin: 'kotlin'
apply plugin: 'metadoc'

repositories {
  mavenLocal()
  if(project.hasProperty('PendlerExecutor')) {
    maven {
      url "${project.property("host")}/static/maven/private-
repository/3e6e12b1-1532-4f24-a916-adb146685f2a"
    }
  }
  maven {
    url "https://public.repository.dongoteam.hu"
  }
  mavenCentral()
  jcenter()
}

dependencies {
  compile "org.jetbrains.kotlin:kotlin-stdlib-
jdk8:$kotlin_version"
  testCompile group: 'junit', name: 'junit', version: '4.12'
```

```
  testCompile "org.dongoteam.pendler.dfa:dfa-
environment:$dfaenv_version"
}

task stub(type: JavaExec) {
  main = 'org.dongoteam.pendler.envs.dfa.stub.StubKt'
  args = [
      project.hasProperty('host') ? project.property("host") :
"",
      project.hasProperty('auth') ? project.property("auth") : ""
  ]
  classpath = sourceSets.main.runtimeClasspath +
sourceSets.test.runtimeClasspath
}
```

PROJEKT-PENDLER KONFIGURÁCIÓ

A projektmappa gyökerébe hozzunk létre egy „.pendler” nevű mappát és helyezzünk el egy „Dockerfile” nevű állományt (kiterjesztés nélkül) és egy „config.conf” nevű állományt!

A „config.conf” állomány kezdeti tartalma legyen a következő.

```
config {
  engine {
    type = "dfa"
    testFolder = "/src/test/kotlin"
  }
  definitions {
    metas = [ ]
    userMetas = [ ]
  }
  runner {
    parameters = [ ]
  }
}
```

A tartalmának jelentését és a beállítási lehetőségeket későbbi fejezetekben lesznek kifejtve.

A „Dockerfile” pedig legyen a következő.

```
FROM pendler-dockerbucket-basecontainer
```

Amennyiben a későbbiekben szükségünk van a Java mellett további eszközökre is, telepítésüket ebben a Dockerfile-ban megtehetjük annak figyelembevételével, hogy ennek alapja egy Ubuntu képfájl. Jelen dokumentáció írásakor 18:04-es, *bionic* kódnevű változata.

TESZTESET ÍRÁSA

Az egyes teszteseteket az „src/test/kotlin” mappába helyezhetjük, ezen belül tetszőlegesen rendezhetjük, a Kotlin és Java nyelvi szabályainak megfelelően. Minden egyes teszteset egy osztály, mely a „DFATest” nevű osztályból származik és további plusz követelmény, hogy minden tesztesosztály neve „DFATest” szöveggel kell végződnie.

A tesztosztály alapszerkezete a következőképpen néz ki.

```
class MyExampleDFATest : DFATest({
  …
})
```

Illetve egyedi konfiguráció esetén a következőképp nézhet még ki.

```
class MyExampleDFATest : DFATest({
  …
}) {
  override val config: DFAConfig
    get() = …
}
```

A tesztosztály fölé, forráskód dokumentációban írhatjuk a teszteset rövid leírását, illetve @Name forráskód-annotációval a teszteset nevét adhatjuk meg és @Meta forráskód-annotációval további meta kulcs-érték párokat adhatunk meg. A forráskód dokumentációt ellenben kötelező ellátni „metadesc” jelzővel.

Például tekintsük meg a következő forráskód dokumentációt.

```
/** metadesc
 * @Name Példateszt
 *
 * # Címsor példa 1
 *
 * - Egy példa a működés tesztelésére.
 */
```

```
class MyExampleDFATest : DFATest({
  …
})
```

A forráskód dokumentációban Markdown formázást alkalmazhatunk, mind itt, mind pedig a későbbiekben.

A teszteset törzsét a DFATest absztrakt osztálynak paraméterül átadott „függvény” (valójában ez nem egészen csak egy függvény) tartalmazza. A tesztlépéseket, tesztutasításokat itt írhatjuk meg.

TASK BLOKK

A legfelső, többször is ismételhető komponens a tesztlépések között a „task”. Ez több lépést foglalhat magába, csoportosítja a lépéseket, kiemelhet egy-egy egységes feature-t, továbbá több task egymásba ágyazásával hierarchiát határozhatunk meg.

A task a tesztosztály mellett az a komponens, mely rendelkezhet dokumentációs blokkal. Ellenben a tevékenységének leírása mellett kizárólag a @Name forráskód-annotációval rendelkezhet és „metadesc” helyett „metadoc” jelzőt kell alkalmazni.

```
/** metadoc
 *
 * @Name Alapfunkciók tesztelése
 *
 * Adjuk hozzá a *We have to make a dinner.* feladatot a TODO
 * listához, majd a *We should learn cooking.* feladatot is.
 *
 * Ellenőrizzük le, hogy kívánt feladatok hozzáadódtak a listához
 * az előírt sorrendnek megfelelően (leutoljára felvett van
 * alul), majd töröljük ki őket.
 *
 * A törlés után a TODO listának üresnek skell lennie.
 */
task {
  …
}
```

Következő egy hosszabb példa a taskok egységbe ágyazásáról.

```
/** metadoc
  * @Name TODO feladatok szűrése
  *
  * Ismét vegyük fel a *We have to make a dinner.* és a *We
  * should learn cooking.* felatokat a TODO listára, majd a *We
  * should learn cooking.* feladatot jelöljük ki.
  */
task {
  /** metadoc
    *
    * @Name Tesztadat létrehozása
    */
  task {
    …
  }

  /** metadoc
    *
    * @Name **Active** szűrőmező
    *
    * Válasszuk ki az **Active** feliratú szűrőmezőt és
    * ellenőrizzük, hogy valóban csak a
    * *We have to make a dinner.* feladat jelenik-e meg!
    */
  task {
    …
  }

  /** metadoc
    *
    * @Name **Completed** szűrőmező
    *
    * Válasszuk ki az **Completed** feliratú szűrőmezőt és
    * ellenőrizzük, hogy valóban csak a *We should learn
    * cooking.* feladat jelenik-e meg!
    */
  task {
    …
  }

  /** metadoc
    *
    * @Name **All** szűrőmező
    *
    * Végezetül kattintsunk az **All** feliratú szűrőmezőre és
    * ellenőrizzük le, hogy minden felvett feladat megjelenik-e:
    *  - *We have to make a dinner.*
    *  - *We should learn cooking.*
    */
  task {
    …
```

```
  }
}
```

WEB BLOKK

A „web” blokkot kizárólag task-on belülre helyezhetünk el és tartalma is szigorúan között. Böngésző kezelésével, böngészőben végzett műveletekkel foglalkozhat.

Lehetséges utasítások a web blokkon belül az alábbiak lehetnek.

```
open("<url>")
open("<url>") alias "my-browser"
```

Böngészőablak megnyitása az adott url-re való navigálás. A második esetben egy nevet is adunk a böngészőnek.

```
close()
close("my-browser")
```

Alapértelmezett webböngésző bezárása, míg a második esetben, adott névvel ellátott webböngésző bezárása.

```
click on xpath("<xpath>")
click on id("<id>")
```

Kattintás a megadott elemre.

```
typing("<text>") on xpath("<xpath>")
typing("<text>") on id("<id>”)
```

Írás a megadott beviteli mezőbe.

```
expected("<text>") equals xpath("<xpath>")
expected("<text>") by "<title>" equals xpath("xpath")

expected("<text>") equals id(„<id>”)
expected("<text>") by "<title>" equals id("<id>")
```

Ellenőrzi, hogy a paraméterül megadott szöveg valóban az adott HTML elemben található szöveggel megegyezik-e.

```
val myVariable = get from id("my-little-id")
```

```
val myVariable = get from xpath("my-little-xpath")
```

Adott HTML elem szövegének lekérdezése és egy változóba való helyezése.

CODE BLOKK

Code blokk szintén kizárólag csak task blokkon belül lehet, ellenben tartalma tetszőleges programkódot tartalmazhat. Azon utasításokat érdemes ezen blokkba rakni, melyek leírása túlmutat a web blokk lehetőségein. Továbbá a böngésző kezelését a web blokktól eltérően egy úgynevezett „Ego” nevezetű könyvtár segíti.

Az Ego könyvtár a Selenium Webdriver megoldását rejti magába, melyet közvetlenül is elérhetünk, de szükség esetén ajánlott inkább az EgoItem komponensek saját célú bővítése, melyet későbbi fejezetben részletezünk.

A code blokk biztosít egy „page” objektumot alapértelmezetten, mely egy böngészőablakot takar és egyben egy Ego objektum.

```
task {
  code {
    …
  }
}
```

TASKCODE BLOKK

Ha egy task blokkon belül kizárólag egy darab code blokkot helyezünk el, akkor ezt megtehetjük egy összevont, „taskCode” függvényhívással is. Ebben az esetben a task-ra vonatkozó dokumentációs tulajdonság is megmarad.

A függvény törzsében a code blokkra vonatkozó szabályok érvényesek.

Példa.

```
/** metadoc
 *
 * @Name Alapfunkciók tesztelése
 *
 * Adjuk hozzá a *We have to make a dinner.* feladatot a TODO
```

```
 * listához, majd a *We should learn cooking.* feladatot is.
 *
 * Ellenőrizzük le, hogy kívánt feladatok hozzáadódtak a listához
 * az előírt sorrendnek megfelelően (leutoljára felvett van
 * alul), majd töröljük ki őket.
 *
 * A törlés után a TODO listának üresnek skell lennie.
 */
taskCode {
  …
}
```

EGYEDI KONFIGURÁCIÓ

Ha vannak olyan műveleteink, melyeket minden teszteset előtt, végén, vagy esetleges hiba esetén el szeretnénk végezni; kiemelhetjük őket egy konfigurációs osztályba, melyet hozzáfűzhetünk minden tesztesetünkhöz. Ennek kimondott előnye, hogy a duplikációk mennyiségét csökkentjük.

Egyedi konfigurációt tartalmazó osztály létrehozásához a DFAConfig osztályból kell származtatnunk. A DFAConfig absztrakt osztály felüldefiniálható függvényei az alábbiak.

- **initTestCase**: A tesztesetek futtatása előtt fog lefutni, minden egyes teszteset előtt.
- **endTestCase**: Azután fog megfutni, hogy a teszteset lefutott, függetlenül attól, hogy sikeresen, vagy hibásan futott le a teszteset.
- **onErrorTestCase**: Akkor fut le, ha teszteset futása során valamely hiba keletkezett.

A következő példa egy minta konfigurációt mutat be.

```
import org.dongoteam.pendler.envs.dfa.DFAConfig
import org.dongoteam.pendler.envs.dfa.Global

class  TodoMVCConfig : DFAConfig( ) {
  override fun initTestCase(global: Global): Task =
    Task {
      web {
        open(global["url"] as String) alias "default"
      }
      code { … }
    }

  override fun endTestCase(global: Global): Task =
    Task {
      web {
          close( ) // Csak példa, ilyet nem kell csinálni!
      }
      code { … }
    }

  override fun onErrorTestCase(global: Global): Task =
    Task {
      code { … }
    }
}
```

Fontos! Az itt látható „Task” hivatkozások valóban nagy kezdőbetűvel kezdődnek. Nem elírás!

Konfiguráció alkalmazása egy tesztesetnél a következő példában látható.

```
class MyExampleDFATest : DFATest({
  …
}) {
  override val config: DFAConfig
    get( ) = new TodoMVCConfig( )
}
```

FÜGGŐSÉGEK INJEKTÁLÁSA

……

EGO KÖNYVTÁR

Az Ego könyvtár a Selenium köré épített wrapper könyvtár, mely további kényelmi tulajdonságokkal vértezi fel a körülzárt könyvtárat. Központi objektuma a webböngészőt képező Ego objektum, melyből kiindulva tudunk lekérni az adott weboldal egy elemét reprezentáló EgoItem-et. Az EgoItem-en keresztül pedig az adott weboldal elemen végezhetünk műveleteket.

A könyvtár DI rendszerének köszönhetően tetszőlegesen bővíthető saját EgoItem-ekkel is alapértelemezetten adottak mellett.

Általános szerkezet egy Ego utasításhoz.

```
page<Tulajdonság>(<selector>).<művelet>
```

A „Tulajdonság” egy EgoItem, a „<selector>” lehet xpath, vagy id; míg a „<művelet>” az EgoItem egy függvénye.

Példaként tekintsük meg a következő mintákat.

```
page<Typeable>(id("name-form-input")).typing("Adoniz")
page<Clickable>(xpath("/ul/li[2]/a")).click()

val text = page<Getter>(id("name-head-profil")).text()
```

EGOITEM

EMPTY

Technikai elem.

CLICKABLE

A selector-ban megadott HTML elem rendelkezik kattintási tulajdonsággal.

Lehetséges műveletek az alábbiak lehetnek:

- click(): Unit
  Az elemre való kattintás bal egérgombbal.
- doubleClick(): Unit
  Az elemre való dupla kattintás bal egérgombbal.

TYPEABLE

A selector-ban megadott HTML elem rendelkezik írási tulajdonsággal. Jellemzően valamilyen beviteli input mezőről lehet szó.

Lehetséges műveletek az alábbiak lehetnek:

- clear(): Typeable
  A beviteli mező tartalmának törlése.
- typing(value: String): Typeable
  Megadott „value” tartalom bevitele, majd az ENTER billentyű leütése.
- typing(value: String, afterKeys: List<Keys>): Typeable
  Megadott „value” tartalom bevitele, majd az „afterKeys” listában található billentyűk leütése.

HOVERABLE

A selector-ban megadott HTML elemre rávíve az egérkurzort, valamely változás történik.

Lehetséges műveletek az alábbiak lehetnek:

- hover(): Unit
  Az elemre vigyük rá az egérkurzort.

GETTER

A selector-ban megadott HTML elem tartalma lekérdezhető.

Lehetséges műveletek az alábbiak lehetnek:

- text(): String
  Az elem szöveges tartalmának lekérdezése és visszaadása.

SAJÁT EGOITEM IMPLEMENTÁLÁSA

Az alap EgoItem-ek mellé tetszőlegesen készíthetünk saját EgoItem-eket, akár az adott tesztelendő webalkalmazásra vonatkozó specifikus elemeket is.

```
import org.dongoteam.pendler.ego.*
import org.openqa.selenium.WebElement

class MyEgoItem(
    base: Base,
    private val selector: Selector,
    private val webElement: WebElement
) : EgoPageItem(base) {
  fun myFunction(): Int {
    return 42;
  }
}
```

Példa a használatára a következő kódban látható.

```
page<MyEgoItem>(id("my-element-id")).myFunction()
```

VÁLTOZÓK HASZNÁLATA

A teszteset futása során felhasznált tesztadatokat, illetve azon adatokat melyeket a futás során többször is felhasználunk, érdemes kigyűjteni a teszteset elejére. Erre a DFA biztosít egy egyedi blokkot „Var” néven. Használatához tekintsd meg a következő példát.

```
class MyExampleDFATest : DFATest({
  val texts = Var {
    Texts(
      one = "We have to make a dinner.",
      two = "We should learn cooking."
    )
  }
  task {
    code {
      …
    }
  }
}) {
  data class Texts(
    val one: String,
    val two: String
  )
}
```

Immutable objektumként való használat nem kötelező, „val” helyett a „var” kulcsszót is alkalmazhatjuk és továbbá egy meglévő Var-al létrehozott változó értékét is megváltoztathatjuk. Erre mutat példát a következő kód.

```
class MyExampleDFATest : DFATest({
  var texts = Var {
    Texts(
      one = "We have to make a dinner.",
      two = "We should learn cooking."
    )
  }
  task {
    code {
      …
    }
    texts = Var {
      texts.copy(
        one = "One TEXT"
      )
    }
    web {
      typing(texts.one) on xpath("input[4]")
    }
    code {
      …
      texts = Var {
        texts.copy(
          two = page<Getter>(id("name-field-input")).text()
        )
      }
      …
    }
  }
}) {
  data class Texts(
    val one: String,
    val two: String
  )
}
```

PROJEKT TESTRESZABÁSA

METAADATOK

Minden teszteset tartalmazhat metaadatokat, melyeket a felületen statikus metaadatnak hívunk. Ezeket a forráskódban helyezzük a teszteset fölé „@Meta” forráskód annotációval.

```
/** metadesc
 * @Name Példateszt
 * @Meta MyStaticMetaKey-Example Example-value
 *
 * # Címsor példa 1
 *
 * - Egy példa a működés tesztelésére.
 */
class MyExampleDFATest : DFATest({
  …
})
```

Ezen kulcs-érték párok tartalma tetszőlegesek lehetnek, ellenben van két fontos megkötés:

1. Egy metaadat kizárólag egy sort foglalhat el a forráskódban. Azaz sortörést nem helyezhetünk el benne.
2. A kulcs nem tartalmazhat semmilyen whitespace karakter. Sem szóközt, sem tabulátort.

Ezzel már kész is vagyunk a statikus metaadat hozzáadásával. Miután frissítettük az UI-n a tárolót a változtatással, már meg is kell jelennie az új statikus metaadatnak.

SAJÁT KATALÓGUS NÉZET

A statikus metaadat alapján készíthetünk saját katalógusnézetet. Ehhez a projektünk „.pendler/config.conf” állományban definiálnunk kell a használni kívánt metadatot „$directory” típusként és mindenhol az adott metaadat értékében egy url-t kell megadnunk. Ez utóbbi url fogja megadni, hogy a saját nézetben hol helyezkedjen a teszteset.

Példaként definiáljuk a „FeatureDirectory” nevű metaadatot!

A „.pendler/config.conf” ebben az esetben a következőképpen nézhet ki.

```
config {
  engine {
    type = "dfa"
    testFolder = "/src/test/kotlin"
  }
  definitions {
    metas = [
      {
        name = "FeatureDirectory"
        type = "$directory"
        languages = [ ]
      }
    ]
    userMetas = [ ]
  }
  runner {
    parameters = [ ]
  }
}
```

Ezután adjuk meg a „FeatureDirectory” metaadatot a teszteseteknek!

```
/** metadesc
 * @Name Példateszt
 * @Meta FeatureDirectory /feature-1/example-ffeature
 *
 * # Címsor példa 1
 *
 * - Egy példa a működés tesztelésére.
 */
class MyExampleDFATest : DFATest({
  …
})
```

FUTÁSI PARAMÉTEREK

Tesztesetünk rendelkezhet olyan paraméterrel, melyet meg szeretnénk adni a teszteset indításakor minden alkalommal. Minden futtatás esetén lehet értéke más és más.

Ilyen paraméter lehet például a tesztelendő webalkalamazás url címe. Az url-t azért is érdemes kívülről jövő paraméterként definiálni, mert ebben az esetben egy

paraméter beállításával lefuttathatjuk a teszteket mind fejlesztői, teszt környezeten; mind éles, ügyfélkörnyezeten. Ezáltal hordozhatóvá válik, könnyen átültethető, alkalmazható több környezet tesztelésére is.

Futási paraméter definiálásához a „.pender/config.conf” állományt kell kiegészítenünk a „runner.parameters” tömböt.

Például belerakhatjuk ebbe a tömbbe a következő json értéket:

```
{
  name = "url"
  type = "String"
  languages = [
    {
      name = "hu"
      value = "Alkalmazás elérési címe"
    }
  ]
  userInterface {
    select {
      options = [
        {
          id = "http://localhost:8000"
          title = "Self hosted"
        },
        {
          id = "http://develop.envs.cloud7.12.dongoteam.hu"
          title = "Develop environment"
        },
        {
          id = "http://qa.envs.cloud7.12.dongoteam.hu"
          title = "QA environment"
        },
        {
          id = "http://qa2.envs.cloud7.12.dongoteam.hu"
          title = "QA2 environment"
        }
      ]
    }
  }
}
```

A „type” értéke minden esetben „String”, de megadása kötelező. Felhasználói oldalon látható felülete jelen verzió esetében lehet „select” és „text”. Míg a „select” egy választólistát jelenít meg, addig a „text” egy tetszőleges karaktersorozattal kitölthető szöveges beviteli mezőt fog megjeleníteni.

A következőben nézzünk meg egy „text” típusú változatot.

```
{
  name = "url"
  type = "String"
  languages = [
    {
      name = "hu"
      value = "Alkalmazás elérési címe"
    }
  ]
  userInterface {
    text {
      placeholder = "Elérési cím..."
    }
  }
}
```

Ha a definícióval kész vagyunk, már használhatjuk is a tesztesetünkben. Minden futása paraméter egy „global” nevű map típusú objektumba kerül. Viszont fontos, hogy az ebből az objektumból jövő értékeket cast-olnunk kell.

Például a következőképpen használhatjuk.

```
…
web {
  open(global["url"] as String) alias "default"
}
…
```

FAQ

A Gradle konfiguráció elég macerás. Nem lehetne egyszerűbben megoldani?

Lesz egyszerűbb megoldás! Egy Gradle plugin fogja biztosítani az egyszerűsített konfigurációt. Ellenben viszont alacsony prioritással fut nálunk jelenleg ez a feladat, ezért elvárható határidőt sem tudunk még közölni, mikorra készül el.

Megtehetem, hogy web blokkokat nem használok és kizárólag code blokkokra építem a teszteseteimet?

Természetesen igen.