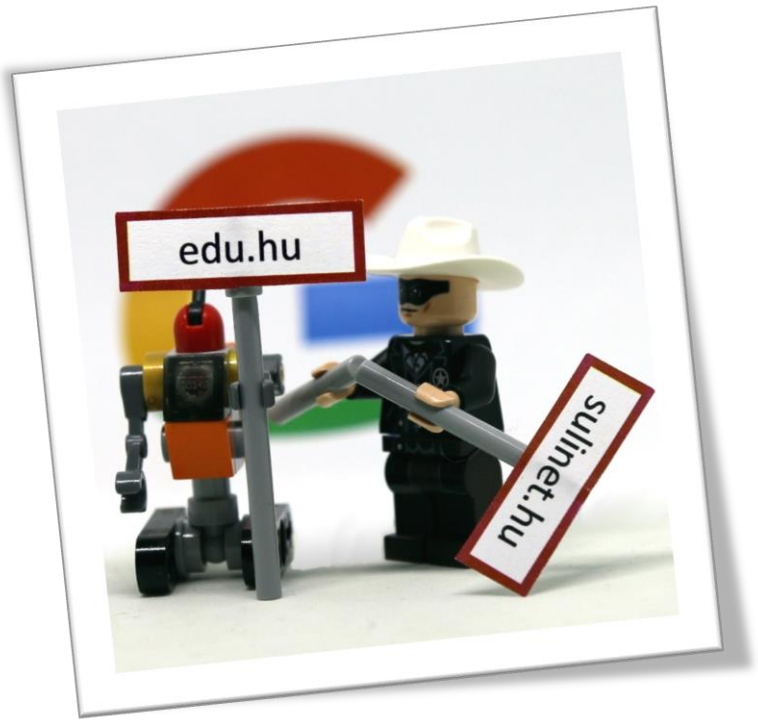
Viszlát Sulinet! Hello Edu!
Elsődleges domain módosítása G Suit-ban



Néhány hónap múlva búcsút mondhatunk a **sulinet.hu** domainnek és helyette bevezetésre kerül az **edu.hu**. Ezt le kell követnünk G Suit-ban (Google Workspace-ben) is, ha ilyen domainnel használjuk. Ennek megoldására mutat példát a következő dokumentum.

Debrecen, 2020. 12. 29.

Venczel József

Tartalomjegyzék

1. Előkonfiguráció

A dokumentum címéből úgy tűnhet, csupán egyetlen probléma megoldására törekszem. Ebben a leírásban azonban több olyan technika is bemutatásra kerül, melyek segítségével nagyobb volumenű, unalmas adminisztrációs feladatokat automatizálhatunk, vagy éppen egyszerűsíthetünk a Google Workspace (G Suit for Education) rendszerünkben.

Az Apps Script a Google saját nyelve, amit a különböző felhős szolgáltatásaikhoz fejlesztettek ki. Ez lényegében a JavaScript-re épül, amit kiegészítettek egy rakás Google specifikus objektummal. Ezek segítségével egyszerűbben hozzáférhetünk a különböző szolgáltatásokhoz, funkciókhoz.

Az irodai programcsomag elemei (mint pl. a Dokumentumok, Táblázatok, Diák) is ezt a nyelvet használják a makrók rögzítéséhez, de további applikációkból is elérhető, mint pl. a levelezés, a Naptár, vagy a Drive.

A következő hivatkozáson bőségesen akad olvasnivaló ezzel kapcsolatban:

https://developers.google.com/apps-script/overview

A címben szereplő alapvető probléma megoldásához, pedig az alábbi linken kaphatunk útmutatást a Google-től:

https://support.google.com/a/answer/7009324?hl=en

Azzal kell kezdenünk, hogy a Felügyeleti konzol Domainek funkciójában fel kell venni a meglévő *sulineve.sulinet.hu* domain mellé a *sulineve.edu.hu* domaint is. Ezután igazolnunk kell, hogy befolyással bírunk fölötte. Ehhez pontosan ugyanazokat a lépéseket kell megtennünk, mint a G Suit for Education regisztrálásakor.

Ha ez sikerült, akkor már csak be kell állítanunk a megfelelő DNS rekordokat, hogy az e-mailek erre az új domain-re is megérkezzenek.

A következő lépés, hogy beállítjuk elsődleges domain-ként az új (*edu.hu*) domaint.

Eddig a pontig (én legalább is) nem tapasztaltam semmi változást. Minden funkció ugyanúgy üzemelt, mint előtte, illetve annyi változott, hogy pl. új felhasználó létrehozásakor az új domaint ajánlotta fel alapértelmezettként az e-mail címhez.

A következő lépés a legfontosabb, de ez már lényegében rajtunk múlik.

Meg kell határoznunk egy előre tervezett időpontot, amikor átállítjuk minden felhasználónak és csoportnak az e-mail címét. Erről mindenképpen tájékoztatni kell a felhasználókat, mert ha ezt megtettük, az eszközeiken ezután már csak az új e-mail címükkel tudnak belépni.

Érdemes azt is fontolóra venni, hogy egyúttal ne állítsuk-e alaphelyzetbe a felhasználók jelszavát? Tapasztalatból mondom, a diákok jelentős része nem tudja a saját maga által megadott jelszót. Ez legtöbbször csak akkor derül ki, amikor másik eszközre váltanak (pl. lecserélik a telefonjukat, tabletjüket) és újra be kellene lépniük a Google Workspace rendszerbe. Ráadásul, többnyire a jelszó visszaállító e-mail címet és/vagy telefonszámot sem szokták beállítani.

A script, amit megírunk, a rendszerből fogja lekérdezni az e-mail címeket és a módosításokat is ennek megfelelően fogja elvégezni, így ehhez nincs szükség semmire. Ha viszont a jelszavakat is alaphelyzetbe szeretnénk állítani, akkor ahhoz némiképpen elő kell készülnünk.

Nálunk pl. létre kellett hozni egy táblázatot, ami tartalmazta a diákok iskolai e-mail címét és a hozzá tartozó OM azonosítót. A G Suit for Education bevezetésekor mindenkinek a saját OM azonosítója volt a kezdő jelszava, amit az első bejelentkezéskor meg kellett változtatnia. Ezt a helyzetet szerettük volna újra előállítani.

A kollégáknak nem hoztuk alaphelyzetbe a jelszavát, nem jelezte senki sem, hogy nem tudja.

Ha mindent átgondoltunk és a felhasználókat is kellőképpen tájékoztattuk, akkor lefuttathatjuk a scriptet. A program nem csak megváltoztatja az e-mail címeket, hanem a régit (*sulinet.hu* végződésűt) beteszi aliasként mindenkinek. Így, amíg él a *sulinet.hu* domain, minden oda érkező levelet is megkapnak.

Más teendőnk nincs, mivel ingyenes oktatási licencel rendelkezünk, a Google dokumentációjában szereplő, számlázásra vonatkozó résszel nem kell foglalkoznunk.

A következő oldalakon a script forráskódját fogjuk megvizsgálni. Az egyes kódrészletek sorról sorra megtalálhatóak a szövegben, de ha valaki egyben kíváncsi a scriptre, akkor az megnézheti a következő linken megosztott táblázatban:

https://docs.google.com/spreadsheets/d/1_NpAR3qFhq4y815ISzmARQflO-oWInTfki_pVN6DmKA/edit?usp=sharing

Kaja a hűtőben, akarom mondani script a Szkripteditorban!

A táblázat olvasásra van megosztva, így annak használata előtt másolatot kell róla készíteni. A Szkripteditor is csak úgy működik.

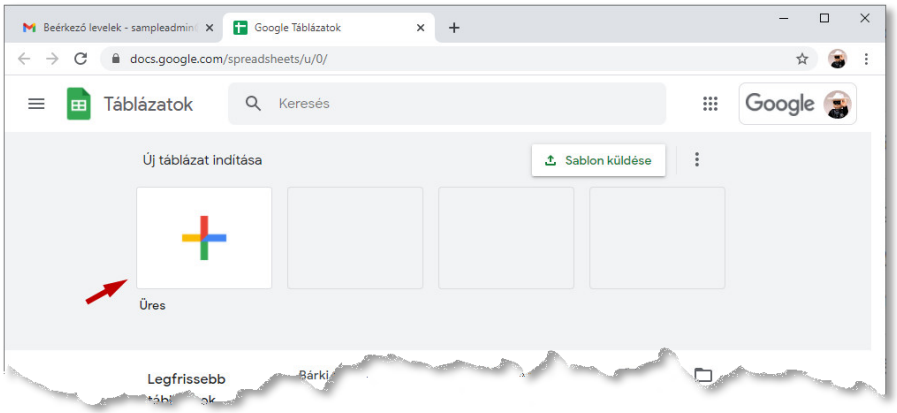
A hozzá tartozó Github repo pedig itt található:

https://github.com/Longeye/G-Suit-for-Education-Adminbot/tree/master/Domainvalto

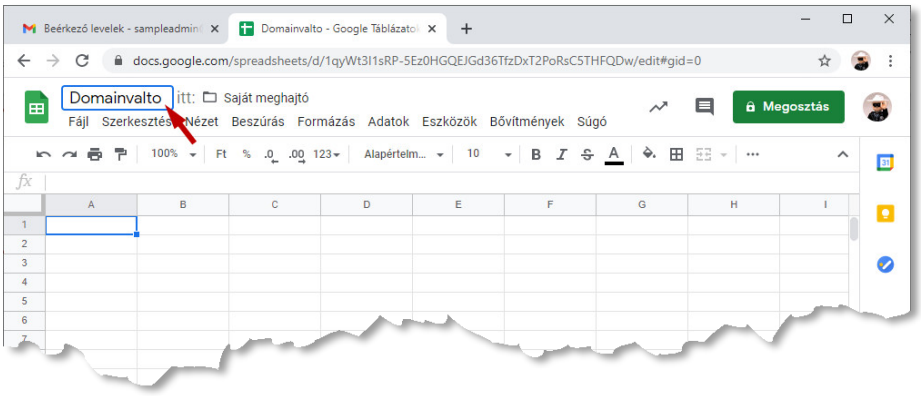
Innen letölthető ennek a dokumentumnak a PDF változata is.

2. Táblázat létrehozása, script konfigurálása

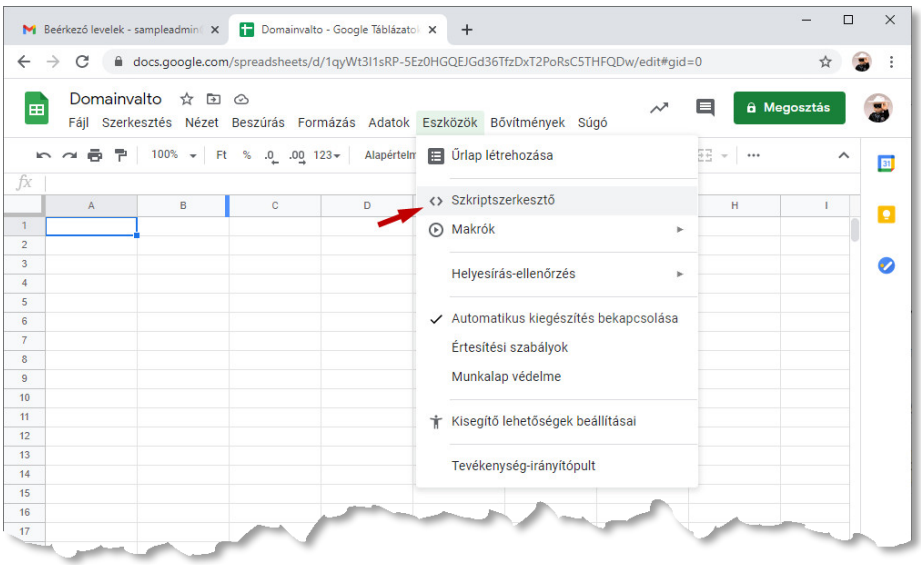
Fejlesztőkörnyezetnek a Google Táblázatokat fogjuk használni. Lépjünk be rendszergazdai jogosultságú felhasználóval a G Suit / Google Workspace rendszerbe és indítsuk el a Táblázatok szolgáltatást!



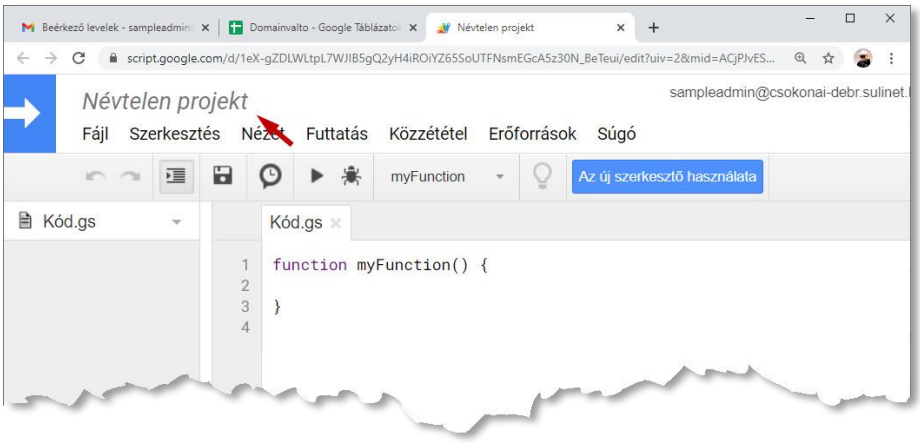
Új táblázat létrehozásához kattintsunk a nagy színes „+” jelre!



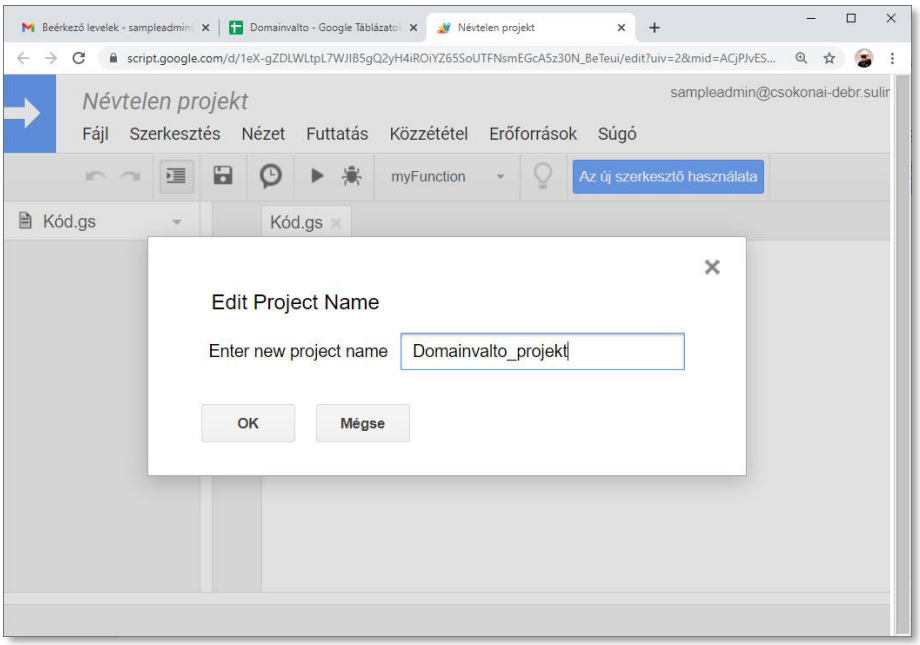
Adjunk nevet a táblázatnak! Én *Domainvalto* –nak neveztem el.



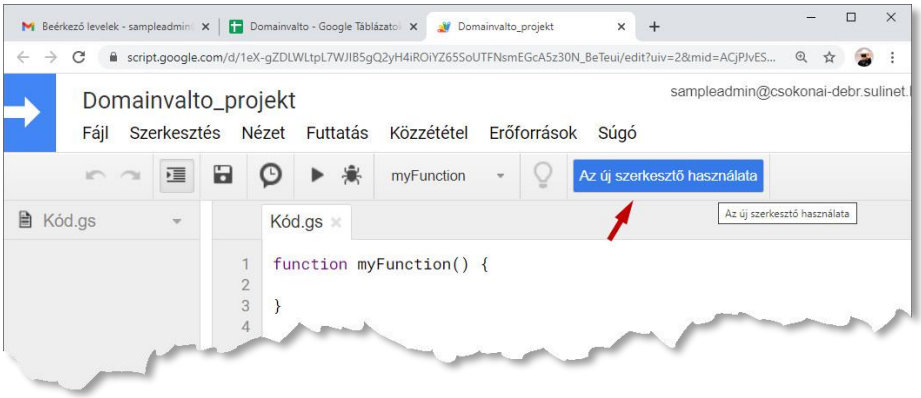
Válasszuk ki az *Eszközök* menüpont *Szkriptszerkesztő* funkcióját!



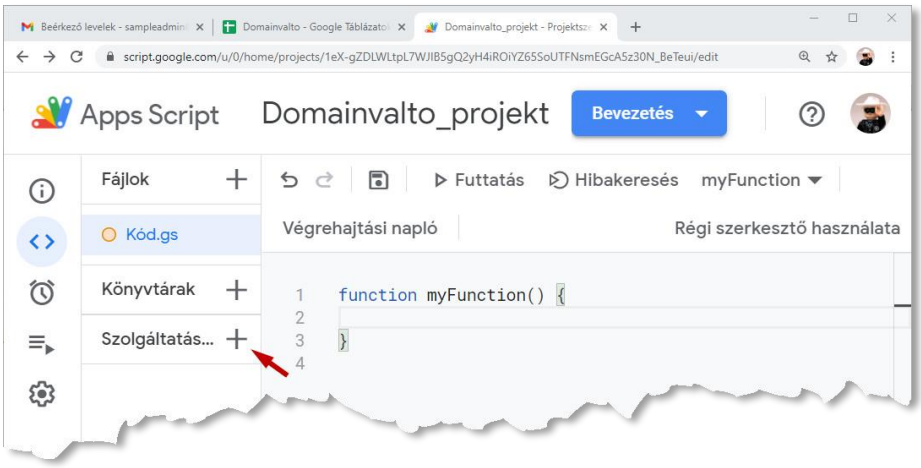
Újabb böngészőfüllel gazdagodunk. A táblázathoz létrejön egy új projekt, amit elnevezhetünk, ha rákattintunk a „Névtelen projekt” feliratra a képernyő bal felső sarkán.



Én a Domainvalto_projekt nevet adtam neki, de ennek sincs semmi jelentősége a jelenlegi feladat szempontjából.



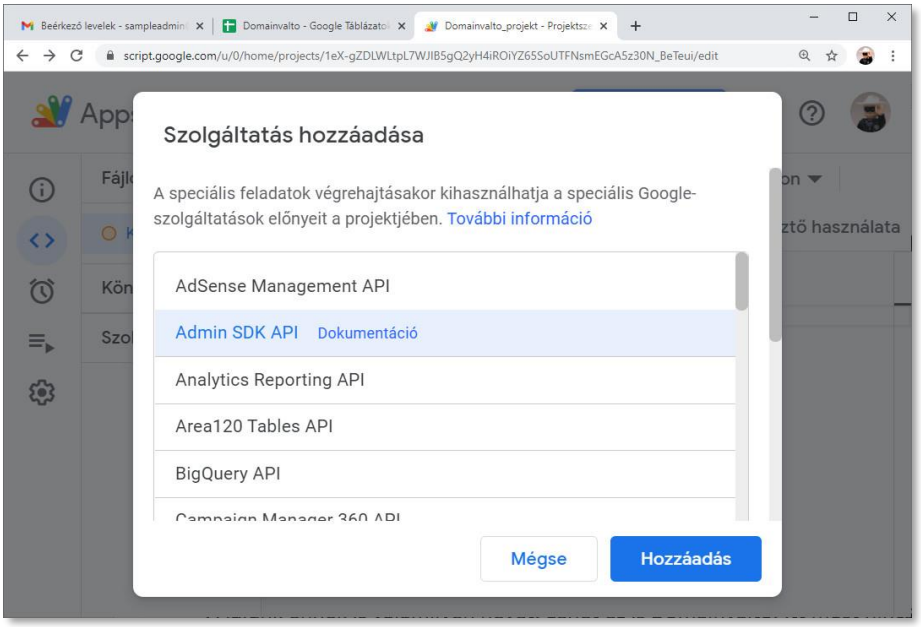
A Google nemrég frissítette a Szkriptszerkesztőjét. Ha egyből nem az új jönne be, akkor váltsunk át rá! Egyrészt, mert előbb vagy utóbb meg kell tenni, másrészt meg mert sokkal letisztultabb és átláthatóbb a felülete.



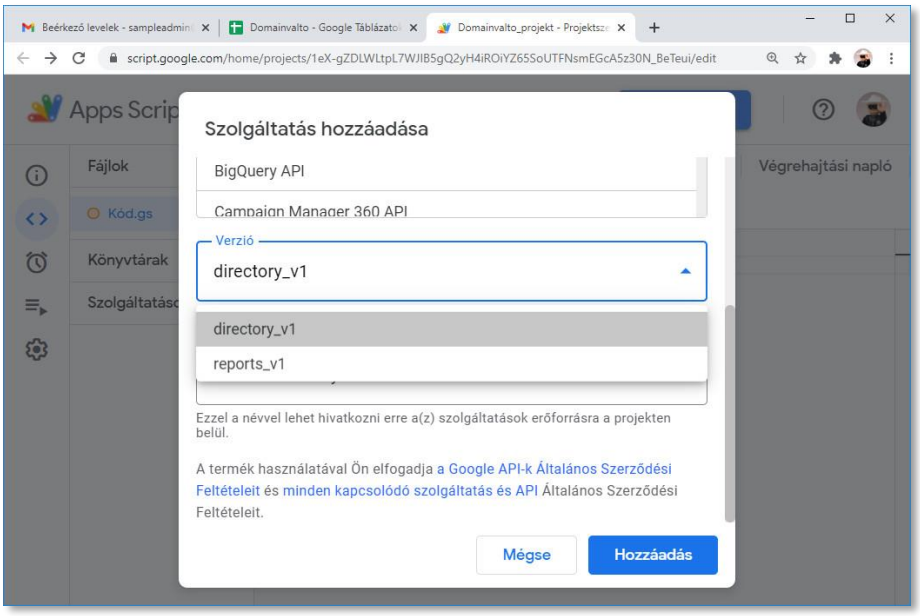
Szemmel láthatóan, nagyon máshogy néz ki, mint a régi.

Mielőtt belefognánk a kódolásba, be kell állítanunk, hogy milyen funkciókat szeretnénk használni a scriptünkben. A Google API-kat fejlesztett ki az egyes szolgáltatásaihoz, melyeken keresztül saját programjainkból (nem csak Apps Scriptből, hanem pl. Pythonból, Javából, C#-ból, PHP-ből, stb.) is elérhetjük azokat és beépíthetjük saját szoftvereinkbe. Először is ki kell választanunk, hogy ezek közül melyekhez szeretnénk hozzáférni.

Ehhez kattintsunk a Szolgáltatás… felírat mögötti + jelre!

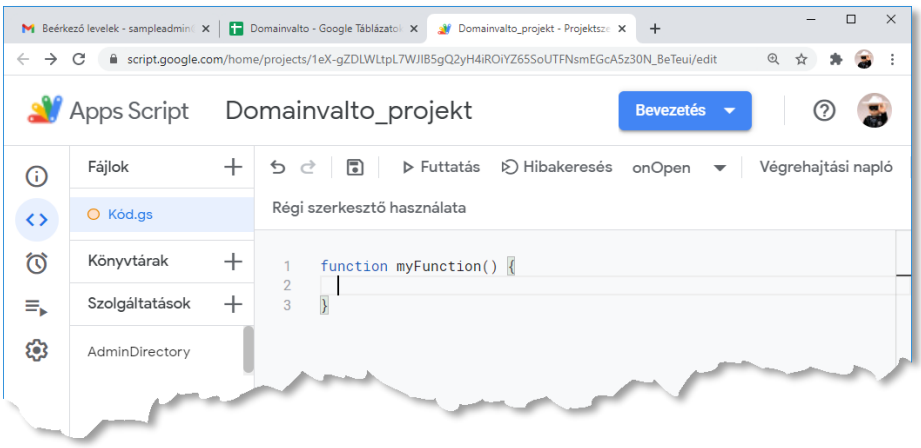


A megjelenő ablakon válasszuk ki az Admin SDK API feliratú sort, majd görgessünk le az ablak legaljára!



Az itt található „*Verzió*” feliratú lenyitható listából válasszuk ki a „*directory_v1*” elemet! Ez az *Admin SDK API Directory* szolgáltatását jelöli, amelynek segítségével a felhasználókkal, csoportokkal és szervezeti egységekkel kapcsolatos adminisztrátori tevékenységek végezhetők el.

Ha megvan, kattanjunk rá a „*Hozzáadás*” gombra!



A szerkesztőbe visszatérve, a bal oldalon megjelenik a „*Szolgáltatások*” között az „*AdminDirectory*” elem.

A szerkesztőben lévő szöveget, amely csak a `myFunction` függvény definícióját tartalmazza, törölhetjük. Itt fogjuk ugyanis elkészíteni a scriptet.

3. Táblázat inicializálása

Az első programrész, amit megírunk, létrehozza és megformázza a működéséhez szükséges munkalapokat és menüket.

```
function onOpen() {
```

A kódot egy speciális függvény definíciójával kezdjük. Amikor megnyitunk egy táblázatot, akkor a hozzá tartozó scriptek közül az `onOpen` nevezetű automatikusan elindul. A cél ugyanis az, hogy ha valaki csak a kódot másolja be egy üres táblázat szkriptszerkesztőjébe, akkor is megfelelően működjön a programunk. Tehát minden szükséges munkalapot és menüpontot automatikusan létre fog hozni.

```
  PropertiesService.getScriptProperties().setProperty("testmode", "true");
  var testmode = (PropertiesService.getScriptProperties().getProperty("testmode") == ”true”);
```

A kódunk több, egymástól függetlenül futó scriptből fog állni, melyek ráadásul nem is egyszerre futnak le. Ezért nem használhatunk klasszikus értelemben vett globális változót, mert az csak egy-egy script futási idejére tartaná meg az értékét. Szerencsére gondoltak erre a Google Apps Script megalkotói is, ezért létrehozták a `PropertiesService` osztályt. Ennek segítségével tudunk átadni adatokat az egyes scriptek között. A `getScriptProperties()` metódus előállít egy olyan objektumot, ami a scripthez tartozó összes tulajdonságot tartalmazza. Ennek `setProperty()` metódusa pedig hozzáad ezekhez egy kulcsszó – érték párost. Itt most a `testmode` kulcsszóhoz rendelünk hozzá egy *igaz* értéket.

A második sorban ezt berakjuk a `testmode` változóba. A `getProperty()` metódus a zárójelben megadott kulcsszóhoz tartozó értéket adja vissza.

A `testmode` változó azt jelzi a programunk számára, hogy **TESZT** üzemmódban fut-e a scriptünk. Ha igen, akkor csak azokkal a felhasználókkal foglalkozik, akiknek az e-mail címe a „*teszt*” szóval kezdődik és csak azokkal a csoportokkal ,amelyeknek a neve szintén a „*teszt*” szóval kezdődik. Így, amikor átírjuk a scripteket, nem kell attól tartanunk, hogy az éles adatokat bántaná. Egyszerűen csak létrehozunk néhány „*teszt*” kezdetű e-mail címmel rendelkező felhasználót és néhány tesztcsoportot, amin kipróbálhatjuk a program hatását. A tesztüzemmódról azonban minden scriptnek tudnia kell, amelyik babrál a G Suit adataival, ezért van szükségünk „globális változóra”.

Tovább ismereteket a `property`-kről a következő linken szerezhetünk:

https://developers.google.com/apps-script/reference/properties

```
var tbl = SpreadsheetApp.getActiveSpreadsheet();
```

A `SpreadsheetApp` osztályon keresztül tudjuk menedzselni a táblázatainkat. A `getActiveSpreadsheet()` metódus az aktuálisan aktív táblázatot reprezentáló objektumot ad vissza. Ezt fogja tartalmazni a `tbl` változó.

Erről bővebben itt tájékozódhatunk:

https://developers.google.com/apps-script/reference/spreadsheet/spreadsheet-app

```
  var ml_status = tbl.getSheetByName("Status");
```

A táblázat `getSheetByName()` metódusával megkapjuk azt a munkalapot, amelyiknek megadtuk paraméterként a nevét. Ha nem talál ilyen munkalapot, akkor `null` értékkel tér vissza.

```
  if(ml_status == null){
      ml_status = tbl.insertSheet();
      ml_status.setName("Status");
      ml_status.setColumnWidths(1, 2, 300);
      ml_status.getRange("A4").setValue("Mit:");
      ml_status.getRange("A5").setValue("Mire:");
      ml_status.getRange("A4:A5").setHorizontalAlignment("right");
```

```
    ml_status.getRange("B4:B5").setBackground("#ffd966");
    ml_status.getRange("A10").setValue("Módosított e-mail címek");
    ml_status.getRange("A10").setFontWeight("bold");
}
```

Ha nincs a megadott feltételnek megfelelő munkalap, akkor létrehozzuk és kicsit megformázzuk. Az `insertSheet()` függvény hozzáad egy új munkalapot a táblázathoz, melyre az `ml_status` nevű változóval hivatkozhatunk majd.

A munkalapot a `setName()` metódusával átnevezzük. Ez a „`Status`” nevet kapja a keresztségben.

A `setColumnWidths()` függvény három paramétert vár. Az első annak az oszlopnak a sorszáma, amelyiknek be akarjuk állítani a szélességét. A második, ahány oszlopnak be akarjuk állítani a szélességét. A harmadik pedig maga a kívánt szélesség értéke pixelekben megadva. Szerintem nem túl jó megoldás ez a mértékegység, de legalább jobban értelmezhető, mint az MS Excel átlagos karakterszélessége.

Itt most az első két oszlop szélességét beállítjuk 300 pixel szélesre.

A `getrange()` metódust gyakran használjuk a táblázatos műveletekben. Általánosságban megfogalmazva, egy tartományt kell megadni paraméterként és egy olyan objektummal tér vissza, ami ezt a tartományt reprezentálja. Amint azt az alábbi linken is láthatjuk, négy féle módon is paraméterezhetjük:

https://developers.google.com/apps-script/reference/spreadsheet/sheet#getRange(Integer,Integer)

A tartomány legegyszerűbb esetben lehet akár egyetlen cella is, mint az a fenti forráskódból látható. Egy cellának a `setValue()` függvénnyel adhatunk értéket. Ha a `getRange()` metódusban cellatartományt adunk meg, akkor a tartomány minden cellája felveszi ezt az értéket.

További metódusai is vannak egy ilyen „range” objektumnak. Ezek leírását itt találhatjuk:

https://developers.google.com/apps-script/reference/spreadsheet/range

A fenti kódrészletben az `A4`-es cellába betesszük a „*Mit:*”, az `A5`-ös cellába pedig a „*Mire:*” szövegeket.

Ezt követően az `A4:A5` cellatartomány tartalmát jobbra igazítjuk a `setHorizontalAlignment()` függvény segítségével.

A mögötte lévő, `B4:B5` cellák hátterét pedig beállítjuk „*világossárgára*” (vagy valami hasonlóra). Erről gondoskodik a `setBackground()` metódus.

Az `A10`-es cellában is elhelyezünk még egy szöveget („*Módosított e-mail címek*”) és megvastagítjuk a `setFontWeight()` segítségével. Ezt a funkciót eredetileg csak a teszteléskor használtam, hogy lássam, ténylegesen megtörténtek a módosítások. A program végén ugyanis itt fognak megjelenni a módosított e-mail címek.

```
var ml_diakok = tbl.getSheetByName("Diákok");
if(ml_diakok == null){
    ml_diakok = tbl.insertSheet();
    ml_diakok.setName("Diákok");
    ml_diakok.getRange("A1").setValue("tesztdiak1234@diak.simplesample.sulinet.hu");
    ml_diakok.getRange("B1").setValue("71233211235");
}
```

Az előzőhöz teljesen hasonló a következő kódrészlet is. Itt a Diákok munkalap meglétét ellenőrizzük, amelyiken a diákok e-mail címeinek és kezdeti jelszavainak kell szerepelnie (lásd a bevezetőben).

Ha még nem létezik, létrehozzuk, elnevezzük és az első sorában elhelyezzük egy „*mintadiák*” adatait.

```
var cella = ml_status.getRange("A7");
cella.setValue("TESZT");
cella.setHorizontalAlignment("center");
cella.setFontWeight("bold");
```

```
  cella.setFontColor("#ffffff");
  if(testmode){
    cella.setBackground("#009600");
  } else {
    cella.setBackground("#960000");
  }
```

A következő kódrészletben a tesztüzemmódot jelezzük a munkalapon. Ha tesztüzemmódban van a programunk, akkor az `A7`-es cellának zöld a háttere, ha pedig éles adatokkal dolgozik a programunk, akkor vörös színű háttérrel jelezzük azt. Ebben a részben nincs semmi újdonság, csak az előbbiekben megismert eszközöket használjuk.

```
  var ui = SpreadsheetApp.getUi();
  var menu = ui.createMenu("Domainváltó");
  menu.addItem("Start", "IntelligentFastDomainChanger");
  menu.addItem("Cserélendő domain", "Mit");
  menu.addItem("Új domain", "Mire");
  menu.addSeparator();
  menu.addItem("Tesztmód KI", "Teszt");
  menu.addToUi();
}
```

Az `onOpen()` függvény utolsó részében létrehozzuk a programunkhoz a kezelőelemeket. Ez lényegében egy menü lesz a táblázat app főmenüjében.

A `SpreadsheetApp` osztály magát a táblázatkezelő aplikációt jelenti. Részletesebb leírása itt található:

https://developers.google.com/apps-script/reference/spreadsheet/spreadsheet-app

A `getUi()` metódusa pedig egy olyan objektumot ad vissza, ami a kezelőfelületre hivatkozik. Ennek segítségével tudunk további funkciókat (jelen esetben pl. egy menüt) hozzáadni a felhasználói felülethez.

Új menüt a `createMenu()` metódussal hozhatunk létre. Paraméterként a menü elnevezését kell megadni, ami megjelenik majd a főmenüben. Fontos tudni, hogy csak egy menüponttal bővíthetjük a főmenüt és ezt, illetve a hozzá tartozó almenüpontokat a későbbiekben nem tudjuk sem lekérdezni, sem módosítani. Ha már van egy menünk és a `createMenu()` metódussal létre szeretnénk hozni egy újabb menüt, akkor az előző törlődik.

Egy menühöz az `addItem()` függvénnyel adhatunk hozzá új menüpontot. Az első paraméterben a nevét kell megadni, a másodikban pedig annak a függvénynek a nevét várja szövegként, amelyiket meg fog hívni, amikor kiválasztják.

Az almenünkben az első menüpont a „Start” nevet fogja viselni és a szerény nevű „IntelligentFastDomainChanger” függvényt fogja meghívni, ami lényegében a script legfőbb tevékenységi körét valósítja meg.

A második és harmadik menüpont a domaincsere legfontosabb paramétereinek beállítására szolgál. Ezeket egyébként megadhatjuk a „Status” munkalapon is a megfelelő cellák kitöltésével.

A fenti menüpontokat egy elválasztójel követ, amit az `addSeparator()` metódussal hozunk létre.

Az utolsó menüpont a tesztüzemmód kikapcsolására szolgál, mint ahogyan azt a neve is jelzi. Mivel ennek különös jelentősége van a program működésének szempontjából, ezért ezt a funkciót nem szerettem volna könnyebben elérhetővé tenni a táblázatban.

Végezetül, az `addToUi()` metódussal hozzáadjuk ténylegesen is a felhasználói felülethez a `menu` változóban definiált menüt.

4. A menüpontokhoz tartozó függvények

A programkód további része lényegében az egyes menüpontokhoz tartozó függvényeket tartalmazza. Ezeken fogunk szépen sorbamenni, kivéve a Start menüponttal összekapcsolt IntelligentFastDomainChanger nevű függvényt, mert az külön fejezetet érdemel.

4.1. Mit

```
function Mit() {
```

A `Mit()` függvény definíciójával kezdünk, ami azt a domaint fogja bekérni, amit ki szeretnénk cserélni.

```
  var ui = SpreadsheetApp.getUi();
```

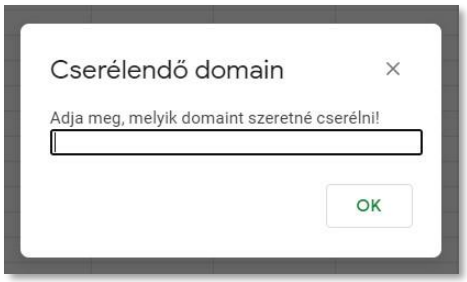
Az első sora már ismerős, kérünk egy objektumot, ami a táblázat felhasználói felületét reprezentálja.

```
  var response = ui.prompt("Cserélendő domain", "Adja meg, melyik domaint szeretné cserélni!", ui.ButtonSet.OK);
```

A következő sor már érdekesebb. A `prompt` metódus létrehoz egy objektumot, ami egy beviteli mezővel rendelkező ablakot reprezentál. Az első paramétere az ablak címét jelöli, a második pedig a bekérendő adat leírására szolgál. A harmadik paraméterben az ablakban megjelenő gombcsoportokat választhatunk ki. A `ButtonSet.OK` olyan gombcsoportot jelöl, amiben csak az `OK` gomb van benne, tehát csak ez fog megjelenni az ablakban. A `Buttonset` további értékei itt találhatóak:

https://developers.google.com/apps-script/reference/base/button-set

A fenti paraméterezéssel ez a függvény a következő ablakot eredményezi:




```
  if(response.getSelectedButton() == ui.Button.OK){
    var ml_status = SpreadsheetApp.getActiveSpreadsheet().getSheetByName("Status");
    ml_status.getRange("B4").setValue(response.getResponseText());
    SpreadsheetApp.flush();
  }
```

Az ablakban bekövetkezett eseményt, vagyis hogy a felhasználó melyik gombra kattintott, a `getSelectedButton()` függvény segítségével kérdezhetjük le. A feltételes elágazásban azt vizsgáljuk, hogy ez az érték megegyezik-e `Button.OK`-vel. A `Button` további lehetőségeit ezen az oldalon találjuk:

https://developers.google.com/apps-script/reference/base/button

Ha igen, akkor az `ml_status` változó segítségével kiválasztjuk a „*Status*” munkalapot és azon a `B4`-es cellába berakjuk az ablak beviteli mezőjébe begépelt szöveget. Ezt az értéket a `getResponseText()` függvénnyel kérdezhetjük le.

Ennek a funkciónak, mint ahogyan a következőnek, nincs sok értelme, hiszen a `B4`-es és a `B5`-ös cellák tartalmát közvetlenül a táblázatban is módosíthatjuk. Példának viszont jól mutatnak.

4.2. Mire

```
function Mire() {
  var ui = SpreadsheetApp.getUi();
  var response = ui.prompt("Új domain", "Adja meg az új domain nevét!", ui.ButtonSet.OK);

  if(response.getSelectedButton() == ui.Button.OK){
    var ml_status = SpreadsheetApp.getActiveSpreadsheet().getSheetByName("Status");
    ml_status.getRange("B5").setValue(response.getResponseText());
    SpreadsheetApp.flush();
  }
}
```

Úgy gondolom, ezen a függvényen nem kell sokat rágódni. A Mit() függvény másolásával és a megfelelő paraméterek módosításával készítettem.

4.3 Teszt

A következő függvény azt a célt szolgálja, hogy a megfelelő menüpont (Tesztmód KI / Tesztmód BE) kiválasztásával beállítja (kikapcsolja / bekapcsolja) a TESZT üzemmódot.

```
function Teszt() {
  var testmode = (PropertiesService.getScriptProperties().getProperty("testmode") == "true");
  var ui = SpreadsheetApp.getUi();
```

A függvény első három sora nem hoz újdonságot, hisz előfordultak már az előző részben.

```
  if(testmode){
```

Ebben az `if` utasításban megvizsgáljuk, hogy TESZT üzemmódban van-e a programunk.

```
    var response = ui.alert("Figyelem!", "Biztosan kikapcsolja a TESZT üzemmódot?", ui.ButtonSet.YES_NO);
    if(response == ui.Button.YES){
      testmode = false;
      PropertiesService.getScriptProperties().setProperty("testmode", "false");
      var menu = ui.createMenu("Domainváltó");
      menu.addItem("Start", "IntelligentFastDomainChanger");
      menu.addItem("Cserélendő domain", "Mit");
      menu.addItem("Új domain", "Mire");
      menu.addSeparator();
      menu.addItem("Tesztmód BE", "Teszt");
      menu.addToUi();
    }
```

Az igaz ágon először is küldünk egy figyelmeztetést a felhasználó felé, hogy éppen a TESZT üzemmód kikapcsolására készül. Ezt az `alert()` metódus segítségével oldjuk meg, melynek használata nagyon hasonlít a `prompt()`-éra:

https://developers.google.com/apps-script/reference/base/ui#alert(String,String,ButtonSet)

Csak akkor kapcsoljuk ki a TESZT üzemmódot, ha a *Yes* gombra kattintott a felhasználó. Ekkor a következő lépéseket tesszük:

A `testmode` változó értékét `false` -ra állítjuk és hogy ezt más scriptek is lássák, elmentjük a `PropertiesService testmode` kulcsába is.

Ezt követően felülírjuk a menüt. Az utolsó menüpontot megváltoztatva „`Tesztmód BE`” feliratúra. Az előzőekben már volt róla szó, hogy a menük elemeit nem lehet külön kezelni. Akkor is teljesen felül kell írni az egész menüt, ha csak egy-egy menüpont tulajdonságait szeretnénk megváltoztatni.

```
} else{
```

```
    testmode = true;
    PropertiesService.getScriptProperties().setProperty("testmode", "true");
    var menu = ui.createMenu("Domainváltó");
    menu.addItem("Start", "IntelligentFastDomainChanger");
    menu.addItem("Cserélendő domain", "Mit");
    menu.addItem("Új domain", "Mire");
    menu.addSeparator();
    menu.addItem("Tesztmód KI", "Teszt");
    menu.addToUi();
}
```

A hamis ágon tulajdonképpen ugyanazt csináljuk, mint a másikon, csak itt nincs szükség figyelmeztetésre, hiszen abból nem származhat semmi baj, ha TESZT üzemmódba kapcsol a program. Illetve a megfelelő értékeket az előzővel ellentétes irányban változtatjuk meg.

```
  var cella = SpreadsheetApp.getActiveSpreadsheet().getSheetByName("Status").getRange("A7");
  if(testmode){
    cella.setBackground("#009600");
    cella.setFontColor("#ffffff");
  } else {
    cella.setBackground("#960000");
    cella.setFontColor("#ffffff");
  }
```

Végezetül, a `testmode` változó értékének megfelelően beállítjuk az `A7`-es cella tartalmát, vagyis inkább csak a háttér színét. Ha valaki azt mondja, hogy teljesen felesleges a két `setFontColor` utasítás, annak igaza van. Eredetileg a betűk színét is állítottam fehérről sárgára és vissza, hogy feltűnőbb legyen a váltás.

A táblázat celláin elvégzett módosításokat az interpreter cacheli és kötegeli a program végrehajtásának optimalizálása érdekében.

```
  SpreadsheetApp.flush();
}
```

A változások megjelenítését a `flush()` metódussal eszközölhetjük ki.

5. Intelligens, Gyors DomainVáltó függvény

Ez a függvény végzi el a tényleges feladatot négy lépésben.

Először lekéri az összes felhasználó adatait és kiválogatja közülük azokat, akiket érint a domain módosítása. Ha pl. TESZT üzemmódban futtatjuk a programot, akkor csak azokkal a felhasználókkal kell foglalkoznunk, akiknek az e-mail címe „teszt” szöveggel kezdődik. A többi e-mail címet eldobjuk. Nem módosítjuk annak a felhasználónak sem az e-mail címét, akinek a nevében futtatjuk a scriptet, hisz ez olyan lenne, mintha saját magunk alatt vágnánk a fát.

A második körben történik meg a kiválogatott e-mail címek módosítása. Lecseréljük az e-mail címben a domaint és a régi e-mail címet berakjuk aliasként. Azoknak a felhasználóknak pedig, akik a „Diákok” nevű munkalapon szerepelnek, beállítjuk az e-mail címük mellett található jelszót.

A harmadik körben a csoportok e-mail címének módosítása következik. Csoportból jóval kevesebb van, mint felhasználóból, így ez hamar megvan.

A negyedik lépés már nem kötelező. Ezt csak a teszteléshez készítettem, hogy lássam a műveletek végeredményét. Visszaolvasom a módosított e-mail címeket és bemásolom azokat a „Status” nevű munkalapra.

5.1. Alapértékek beállítása

```
function IntelligentFastDomainChanger() {

  var testmode = (PropertiesService.getScriptProperties().getProperty("testmode") == "true");
  var admin = Session.getActiveUser().getEmail();
```

A függvény első két sora csak a szokásos. A függvény fejléce után a `testmode` változó értékének beállítása következik a `PropertiesService`-ben tárolt kulcs-értékpárok alapján.

A `Session` osztály segítségével az aktuális munkamenetről kérhetünk le információkat:

https://developers.google.com/apps-script/reference/base/session

A harmadik sorban annak a felhasználónak tesszük bele az e-mail címét az `admin` változóba, akinek a nevében futtatjuk a scriptünket.

```
  var tbl = SpreadsheetApp.getActiveSpreadsheet();
  var ml_status = tbl.getSheetByName("Status");
  var ml_diakok = tbl.getSheetByName("Diákok");
  var emailek = ml_diakok.getRange("A:A").getValues().join().split(',');
```

A következő kódrészlet első három sora megint csak ismerős. A `tbl` változó fogja reprezentálni a táblázatot, az `ml_status` és az `ml_diakok` pedig a nevüknek megfelelő munkalapokat.

Az utolsó sor némi magyarázatra szorul. A bevezetőben már említettem, hogy úgy döntöttünk, a diákok jelszavát alaphelyzetre állítjuk. Ehhez a *Diákok* táblába fel kell tölteni a tanulók Google Workspace-beli felhasználónevét és az alapértelmezett jelszavát, ami nálunk mindenkinek a saját OM azonosítója. A program, mielőtt lecserélné valakinek az e-mail címét, megnézi benne van-e ebben a táblázatban. Ha benne van, akkor a `password` tulajdonságát is beállítja a hozzá tartozó OM azonosítóra.

Az `emailek` változóba átmásoljuk a Diákok munkafüzet első oszlopában lévő e-mail címeket. A `getRange("A:A")` kifejezés jelöli ki az első oszlopot. A `getValues()` metódus pedig a `getRange()`-vel kijelölt cellatartományból adja vissza az értékeket.

A táblázat tulajdonképpen micsoda? Kétdimenziós tömb. Lehet, hogy nem teljesen logikus, de mindenképpen következetes, hogy a `getValues()` mindig kétdimenziós tömbként adja vissza a cellatartomány értékeit. Akkor is, ha csak egyetlen oszlopot jelöltünk ki neki. Ha itt megállnánk, akkor az `emailek` változó valahogy így nézne ki:

```
[[lajoska@simplesample.sulinet.hu], [pistike@simplesample.sulinet.hu], [sarika@simplesample.sulinet.hu] … ]
```

Az elemeire pedig valahogy így hivatkozhatnánk:

```
emailek[0][0] // == lajoska@simplesample.sulinet.hu
emailek[1][0] // == pistike@simplesample.sulinet.hu
emailek[2][0] // == sarika@simplesample.sulinet.hu
stb…
```

Ahhoz, hogy az `indexOf()` függvényt tudjuk használni ezeken az adatokon, át kell alakítanunk ezt az adatstruktúrát.

A `join()` metódus, a `getValues()` által visszadott tömb elemeit összefűzi egyetlen hosszú stringgé úgy, hogy vesszővel választja el egymástól az értékeket.

Na, erre rápakolunk még egy `split()` metódust, ami meg szétbontja az előbb előállított stringet (egy dimenziós) tömbbé, a vesszők figyelembevételével.

Így kapjuk meg a kívánt egydimenziós tömböt, ami a tanulók e-mailcímeit tartalmazza. Ezen pedig már tudjuk alkalmazni majd az `indexOf()` függvényt a későbbiekben.

Kicsit bonyolultnak tűnhet, de a másik két lehetőség sem csábítóbb. Az egyik, hogy a `getValues()` által visszaadott tömbön használjuk a `filter()` függvényt (https://www.w3schools.com/jsref/jsref_filter.asp). Ez viszont megint csak tömböt ad vissza, így kb. nem nyerünk semmit.

A másik lehetőség, hogy saját keresőfüggvényt írunk, ami lekezeli a `getValues()` által visszaadott kétdimenziós tömböt. Nem akartam tovább bonyolítani a programstruktúrát egy felesleges függvénydefinícióval. Egy ilyen függvény legalább három sorból állt volna (a fejléccel együtt), így viszont csak két metódus nevét kellett hozzáadni a sorhoz.

Számomra ez a „mega-metódus-halmozó” megoldás használata tűnt a legegyszerűbbnek mind közül.

```
var mit = ml_status.getRange("B4").getValue();
var mire = ml_status.getRange("B5").getValue();
var felhasznalok = new Array();
var userMAX = 2000;


tbl.setActiveSheet(ml_status);
```

Akinek egy kicsit összekócolta az idegeit az előbbi eszmefuttatás, most megnyugodhat. Egyszerűbb utasítások jönnek. A `mit` változóba betesszük a lecserélendő domain nevét, a `mire` változóba pedig azt a domaint, amire le szeretnénk cserélni (a lecserélendőt :-).

A felhasználók e-mail címeinek tárolására létrehozunk egy új tömböt, `felhasznalok` néven.

A `userMAX`-ba pedig beletesszük a felhasználók becsült számát, akik benne vannak a rendszerünkben. Érdekes, de a Google Workspace rendszer nem ad lehetőséget a felhasználók számának lekérdezésére. Sem a jelentések között, sem a Google API hívások között nem találtam ilyen funkciót. Valószínűleg nincs is jelentősége, nekünk is csak azért van rá szükségünk, hogy követni tudjuk a képernyőn a történéseket az első perctől fogva.

Végül aktivizáljuk a Status munkalapot, mert ezen fogjuk megjeleníteni, hogy épp hol tart a programunk.

5.2. TESZT üzemmód ellenőrzése

```
if (!testmode){
```

```
    var ui = SpreadsheetApp.getUi();
    var response = ui.alert("Figyelem!!!", "A program az éles adatokon fog lefutni! Folytathatjuk?",
                                                                                ui.ButtonSet.YES_NO);
    if (response == ui.Button.YES) {
      Logger.log("A script futása folytatódik...");
    } else {
      ml_status.getRange("A1").setValue("A SCRIPT leállt.");
      SpreadsheetApp.flush();
      return;
    }
  }
```

A következő kódrészlet is ismerős elemeket tartalmaz. Ha a `testmode` változó értéke `false`, akkor figyelmeztetjük a felhasználót, hogy éles adatokon fog dolgozni a script.

Ha a megjelenített figyelmeztető ablakban a *Yes* gombra kattint, akkor naplózzuk, hogy a script futása folytatódik és kész.

Ha viszont bármi másra kattint, akár csak „*kixeli*” az ablakot, leállunk. Az `A1`-es cellban jelezzük ezt a „`A SCRIPT leállt.`” felirattal és kiszállunk egy `return` utasítással a függvényből, mielőtt baj történne.

5.3. A folyamatok nyomonkövetésének előkészítése

```
  ml_status.getRange("A1").setValue("Felhasználók lekérdezése");
  ml_status.getRange("B1").setFormula(
    '=SPARKLINE(C1;{"charttype"\\"bar";"max"\\'+userMAX+';"color1"\\"#960000"})');
  ml_status.getRange("C1").setValue(0);
  SpreadsheetApp.flush();
```

Az első sor világos és egyszerű. Az `A1`-es cellában elhelyezzük a „`Felhasználók lekérdezése`” feliratot.

A második sor is egyszerű és világos, csak még nem használtuk korábban ezeket az utasításokat, függvényeket. A `B1`-es cellában egy képletet teszünk be. Ilyenkor nem a `setValue()` metódust, hanem a `setFormula()` metódust használjuk. Paraméterként egyszerűen csak a képletet kell átadni neki, string formában. Mivel a képlet idézőjeleket is tartalmaz, ezért az egészet aposztrófok közé tettem.

A képlet egyetlen függvényből áll, a `SPARKLINE`-ból, ami egycellás grafikonokat jelenít meg:

https://support.google.com/docs/answer/3093289?hl=en

Magyarul is megtalálható ám a függvény leírása, a LOGOUT-on:

https://logout.hu/bejegyzes/sonar/google_sheet_sparklines.html

Első paraméterként az adatot/adatokat kell megadni, amiből grafikont kell készíteni. Most csak egyetlen cellát adunk meg, a `C1`-et. Ide fogjuk majd bepakolni, hányadik adatelem feldolgozásánál tart a programunk.

A második paraméter kapcsos zárójelek közé tett kulcs-értékpár halmaz, amivel a megjelenítendő grafikont tudjuk befolyásolni. Az eredeti leírás szerint a kulcsokat az értéküktől vesszővel kell elválasztani, míg a kulcs-értékpárokat pontosvesszők tagolják.

Ez viszont nyelvterülettől is függ. Mivel az én rendszerem magyarra van állítva, ezért backslash-t (fordított per jelet) kell használnom vessző helyett. Ebből pedig sosem elég, ezért tettem a biztonság kedvéért rögtön kettőt is mindenhova ;o) (Természetesen a stringkezelés miatt áll ott dupla backslash. Az első escape karakter.)

A kulcs-érték párok segítségével beállítjuk a grafikon típusát oszlopdiagramra, maximális (viszonyítási) értéknek pedig megadjuk `userMax`-ot. A végén pedig a színét is beállítjuk sötétvörösre.

Érdemes itt megjegyezni, hogy a `setFormula()` metódus paraméterében mindig csak a függvények eredeti, angol nevét használhatjuk, a helyi beállításoktól függetlenül. (Sosem értettem, miért kellett egyáltalán a függvények neveit is magyarosítani a táblázatkezelőkben. Ilyen esetekben kifejezetten problémás lehet.)

Ezután a `C1`-es cella értékét lenullázzuk, majd minden módosítást megjelenítünk a képernyőn a `flush()` metódussal.

5.4. Felhasználók beolvasása, számlálása, válogatása

```
userMAX=0;
db=0;
```

A `userMax` egyelőre megtette kötelességét. Lenullázzuk, mert a következő lépésben úgyis összeszámoljuk pontosan hány felhasználóval kell dolgoznunk. Ez nem feltétlenül egyezik meg a rendszerben lévő felhasználók számával. Ha ugyanis TESZT üzemmódban működik a program, akkor csak azokkal a felhasználókkal foglalkozik, akiknek az e-mail címe „`teszt`”-tel kezdődik. A többit figyelmen kívül hagyja.

Éppen ezért használunk egy `db` nevű változót is, ami meg az összes felhasználó számát összesíti.

```
01:   var p = { customer: "my_customer",
02:             maxResults: 30,
03:             pageToken: "" };
04:   do {
05:     var lista = AdminDirectory.Users.list(p);
06:     p.pageToken = lista.nextPageToken;
07:     if(lista.users)
08:       for(var i = 0; i<lista.users.length; i++){
09:         if((lista.users[i].primaryEmail != admin) &&
10:                           ((testmode && lista.users[i].primaryEmail.indexOf("teszt")>=0) || !testmode)){
11:           felhasznalok.push([lista.users[i].primaryEmail]);
12:           userMAX++;
13:         } // if vége
14:         db++;
15:       } // for vége
16:     ml_status.getRange("C1").setValue(db);
17:     SpreadsheetApp.flush();
18:   } while (p.pageToken);
```

A következő programrészletet besorszámoztam, mert így könnyebb lesz hivatkozin az egyes részeire.

A Google rendszeréből az `AdminDirectory.Users.list()` utasítással kérdezhetjük le a felhasználókat. Az `AdminDirectory` jelenti azt, hogy az adminisztrációs feladatok elvégzéséhez kifejlesztett Google API egyik szolgáltatását vesszük igénybe (ehhez adtunk engedélyt a scriptnek még az elején). A `Users` a felhasználókezelésre vonatkozó függvényeket fogja csokorba. A `list()` segítségével lehet lekérdezni a felhasználók adatait. Meg kell adni neki egy objektum típusú paramétert, amiben a listázáshoz szükséges adatok szerepelnek.

https://developers.google.com/admin-sdk/directory/reference/rest/v1/users/list

A kódrészlet azzal kezdődik, hogy egy `p` nevű objektumot definiálunk, aminek három tulajdonsága van. A `customer` annak a felhasználónak az ún. Customer ID-je, akinek a nevében lekérjük az adatokat. A Customer ID nem azonos a „sima” ID-vel, így az e-mail címünket sem használhatjuk helyette. A „`my_customer`” kifejezés jelöli azt a felhasználót akinek a nevében fut a program. Ehelyett lehetne használni a `domain` tulajdonságot is, de az én esetemben ez nem célravezető, mert a diákok külön aldomainba tartoznak, és az aldomainekkel nem foglalkozik a Google API. Tehát őket külön kellene kezelni.

A Google API-k a lekérdezések eredményét lapokra bontva adják vissza. Azt pedig, hogy hány bejegyzés legyen egy lapon, a `maxResults` tulajdonság határozza meg. Ha túl nagy értéket választunk, akkor sokáig tart letölteni az eredményt és úgy tűnhet, hogy a program lefagyott. A 30 ebben az esetben egy optimálisnak tűnő szám.

A harmadik tulajdonság a lapozásban segít: `pageToken`. Egyfajta könyvjelző, ami mindig megmutatja honnan kell folytatni a következő lap lekérdezését. PHP-ban írtam már olyan programot, ami hasonlóan működött. Én ott egy paraméterben mindig a már beolvasott sorok számát tároltam, így annyit mindig eldobtam a következő lekérdezésből. Gondolom ez is valami hasonló lehet.

Ezt a `p` objektumot adjuk tehát át a `list()` metódusnak, ami visszaad egy objektumot, melynek egyik property-je a felhasználók 30 fős listája.

Ezt az objektumot belerakjuk az 5. sorban, a `lista` nevű változóba.

Ugyanebben az objektumban kapjuk vissza a következő „listadarabra”, azaz lapra mutató `pageToken`-t, `nextPageToken` néven. Ezt gyorsan el is mentjük a következő lekérdezéshez a `p` változó megfelelő tulajdonságába a 6. sorban.

A 7. sorban ellenőrizzük, hogy nem üres listát kaptunk-e vissza, mert az azt jelenti, hogy a felhasználóink listájának a végére értünk.

Ha nem üres, akkor a 8. sortól a 15. sorig tartó `for` ciklusban végigmegyünk az aktuális lap elemein.

A 9.-10. sorokban lévő feltételben azt vizsgáljuk, hogy az éppen következő e-mail cím a feldolgozandó e-mail címek közé tartozik-e? Ehhez a következő feltételeknek kell teljesülniük:

`(lista.users[i].primaryEmail != admin)`
: Az aktuális listaelem nem egyezhet meg annak a felhasználónak az e-mail címével, akinek a nevében futtatjuk a scriptet.

`(testmode && lista.users[i].primaryEmail.indexOf("teszt")>=0) || !testmode)`
: Két eset lehetséges. Vagy TESZT üzemmódban vagyunk és a vizsgált felhasználói e-mail cím „teszt” szöveggel kezdődik. Vagy pedig nem vagyunk TESZT üzemmódban. Ez utóbbi esetben minden e-mail címet módosítani kell.

Ha az előbbi feltételek teljesülnek, akkor a 11. sorban a `felhasznalok` tömbbe betesszük az aktuális e-mail címet és a 12. sorban növelhetjük a userMAX változó értékét is eggyel.

Az `if` feltételes elágazás eredményétől függetlenül, megnöveljük a `db` értékét is, még a `for` cikluson belül.

Ha végigszaladtunk az aktuális 30 darabos listaszakaszon, akkor a `C1` cellába berakjuk a `db` értékét, ami mindig az összes felhasználót számlálja és frissítjük a képernyő tartalmát.

A 18. sorban ellenőrizzük, hogy befejezhetjük-e a `do – while` ciklust. Ez akkor következik be, ha a „lapozó jelölőben” (`pageToken`) üres értéket kaptunk vissza. Vagyis nincs több felhasználókat tartalmazó oldal.

Huhh! Mennyire egyszerűbb lenne egyszerre lekérni az összes felhasználót tartalmazó listát! Sajnos a `userMAX` változóba nem tehetünk tetszőlegesen nagy számot. Bár nem minden API esetében van ledokumentálva a maximális érték, de általában 1000-nél több rekordot nem nagyon hajlandóak visszaadni a Google lekérdezései. Úgyhogy, még ha nem is akarnánk kijelezni éppen hol tart a folyamat, akkor is lapozgatnunk kellene.

5.5. Felhasználók e-mail címének módosítása

```
01:   ml_status.getRange("A1").setValue("E-mail címek módosítása folyamatban...");
02:   ml_status.getRange("B1").setFormula('=SPARKLINE(C1;{"charttype"\\"bar";"max"\\'+
03:                                                                   userMAX+';"color1"\\"#960000"})');
04:   ml_status.getRange("C1").setValue(0);
05:   SpreadsheetApp.flush();
06:
07:   felhasznalok.forEach((email, i) => {
08:     var indexDiak = emailek.indexOf(email);
09:     if(indexDiak >= 0)
10:       var felh = { primaryEmail: email.replace(mit, mire),
11:                    password: ml_diakok.getRange(indexDiak+1,2).getValue(),
12:                    changePasswordAtNextLogin: true,
13:                    aliases: email };
14:     else
15:       var felh = { primaryEmail: email.replace(mit, mire),
16:                    aliases: email };
17:
18:     AdminDirectory.Users.update(felh, email);
19:
20:     if(i%5 == 0){
21:       ml_status.getRange(1,3).setValue(i);
22:       SpreadsheetApp.flush();
23:     }
24:   });
```

Az első öt sor már nem szorul magyarázatra. Csak emlékeztetőül jegyzem meg, hogy a `userMAX` változóban most már a feldolgozandó felhasználói e-mail címek tényleges száma szerepel, tehát a `SPARKLINE` függvény korrekt módon mutatja majd a feldolgozás előrehaladtát.

A `felhasznalok` tömbben meg van már mindenkinek az e-mail címe, „aki számít”, úgyhogy akár egy `forEach()` metódus segítségével is végigbattyoghatunk rajta, a 7. sortól kezdődően.

A 8. sorban megnézzük hányadik a soron következő e-mail cím a diákok között. Az eredményt eltesszük egy `indexDiak` nevű változóban.

A 9. sorban el kell döntenünk mit tegyünk. Ha az `indexDiak` változó értéke `0`, vagy annál nagyobb, akkor az email egy diákhoz tartozik, így ennek megfelelően kell felépítenünk azt az adatstruktúrát, ami a felhasználó adatainak módosításához szükséges.

Ebben az esetben a `felh` objektum `primaryEmail` mezőjébe betesszük a megfelelően átalakított e-mail címet, a `password` mezőjébe pedig a „Diákok” munkalapon megadott jelszót. A `changePasswordAtNextLogin` tulajdonság értékét igazra állítjuk, mert azt szeretnénk, hogy az első bejelentkezés alkalmával a felhasználó megváltoztassa a jelszavát. Végül az aliasok közé berakjuk a jelenlegi e-mail címet, hogy a később még erre a címre érkező levelek is betaláljanak az új névvel ellátott postaládájába.

Ha 9. sorban nem teljesül a feltétel, akkor olyan felhasználóról van szó, akinek nem kell apaértelmezés szerinti értékre állítani a jelszavát, csak az e-mail címét kell cserélni. A hamis ágon tehát csak a primaryEmail és az aliases mezők értékét állítjuk be.

A változásokat a 18. sorban érvényesítjük. Az `update()` metódus segítségével frissítjük az adott felhasználó adatait. Ez két paraméter vár, a felhasználó frissítendő adatait tartalmazó adatstruktúrát (a `felh` objektum) és a felhasználó `id`-jét, ami helyett az e-mail címét is használhatjuk.

Ezután a ciklusmag minden ötödik futásának alkalmával frissítjük a táblázat első sorának harmadik celláját. Tehát a C1 cella tartalmát, az indexváltozó aktuális értékével. Azért csak minden ötödik alkalommal, mert a képernyőfrissítés viszonylag lassú és fölösleges rá több időt pazarolni. Lehet, hogy még kevesebbet kellene.

5.6. Csoportok e-mail címének módosítása

Természetesen nem csak a felhasználók, de a csoportok e-mail címe is módosul. Ezt oldja meg ez a programrész. Csoportból jóval kevesebb van, mint felhasználóból, így ezekkel hamarabb végzünk. Éppen ezért nem készítettem olyan programrészt, ami meghatározná a csoportok számát. Merthogy egyszerűen lekérdezni ezt sem lehet, természetesen. Lekérdezzük és azonnal módosítjuk is a csoportok e-mail címét.

```
01:   userMAX = db;
02:
03:   ml_status.getRange("A1").setValue("Csoport e-mail címek cseréje");
04:   ml_status.getRange("B1").setFormula(
05:                               '=SPARKLINE(C1;{"charttype"\\"bar";"max"\\'+100+';"color1"\\"#960000"})');
06:   ml_status.getRange("C1").setValue(0);
07:   SpreadsheetApp.flush();
08:
09:   db=0;
10:   var p = { customer: "my_customer",
11:             maxResults: 30,
12:             pageToken: "" };
13:
14:   do {
15:     var lista = AdminDirectory.Groups.list(p);
16:     p.pageToken = lista.nextPageToken;
17:     if(lista.groups)
18:       for(var i = 0; i<lista.groups.length; i++){
19:         if((testmode && lista.groups[i].name.indexOf("teszt")>=0) || !testmode){
20:           var oldEmail = lista.groups[i].email;
21:           var csop = { email: lista.groups[i].email.replace(mit, mire),
22:                        aliases: oldEmail };
23:           AdminDirectory.Groups.update(csop, oldEmail);
24:         } // if vége
25:         db++;
26:         ml_status.getRange("C1").setValue(db);
27:         SpreadsheetApp.flush();
28:       } // for vége
29:   } while (p.pageToken);
```

A `userMAX` segédváltozóvá fokozódik vissza. A `db` változóban már benne van hány felhasználónk van a rendszerben összesen. Erre szükségünk lesz még a végén, amikor újra végigszaladunk az összes felhasználón és beolvassuk az adataikat az ellenőrzéshez. Szóval most elmentjük a `db` tartalmát a `userMAX`-ba. A `db`-ben pedig a csoportok számát fogjuk számlálni.

Erre a célra persze fel lehetett volna venni egy új változót, de ne „pazarékoljuk” a drága tárhelyet!

A 3.-7. sorok tartalma megint nem nagyon igényel már külön magyarázatot. Funkciója ugyanaz, mint korábban. A csoportok számára egy 100-as becslést tettem a `SPARKLINE` függvényben. Nálunk jelenleg nincs ennél több csoport.

A 9. sorban lenullázzuk a `db`-t, amit tehát a csoportok számlálásához fogunk használni.

A 10.-12. sorokban található `p` változó definíciójával is találkozhattunk már. A Google lekérdezésekhez fogjuk használni paraméter gyanánt.

A 14.-29. sorok `do - while` ciklusa is ismerős lehet. Addig-addig ismételgeti önmagát, amíg elfogynak a listázáskor lekért lapok. Lássuk a ciklusmagot!

A csoportokat is kötegelve, azaz lapokra osztva lehet lekérdezni. A 15. sorban kérjük le az első adagot, amit be is rakunk a `lista` változóba. A `Groups` jelzi az utasításban, hogy most csoportok adataival dolgozunk.

A 16. sorban, az előzőhöz hasonlóan, elmentjük a `pageToken` értékét.

A `lista groups` mezőjében lévő tömbben kapjuk vissza a lekérdezett csoportokat egy tömbben. Ha ez a tömb nem üres, akkor kezdhetjük a feldolgozást. Ezt ellenőrzi a 17. sor `if` utasítása.

A 18. sorban definiált for ciklussal végigmegyünk a lekér csoportok adatain és a testmode értékétől függően vagy módosítjuk az e-mail címét egy-egy csoportnak, vagy nem.

A 20. sortól kezdődik a módosítás. A jelenlegi címet eltesszük egy `oldEmail` változóba.

Majd a 21.sorban létrehozunk egy csop nevű objektumot, amiben a módosítandó adatokat tároljuk. Az email mezőben a megfelelően módosított e-mail címet, az aliases-ben pedig a jelenlegi e-mail címet mentjük el.

A 23. sorban, az `update()` metódussal juttatjuk érvényre a módosításokat. A felhasználók e-mail címének módosításakor látottakhoz hasonlóan ez az `update` is két paramétert vár. Az első, az adatokat tartalmazó objektum (`csop`), a második a csoport azonosítója, amit most az e-mail címével helyettesítünk.

A 25. sorban növeljük a db értékét eggyel.

A 26. és 27. sorban pedig a megszokott módszerrel jelezzük, hol járunk a folyamatban.

Minden ciklusnak vége szakad egyszer, még a végtelen ciklusnak is. A 29. sorban mi is elhagyjuk a `do - while` szerkezetet, ha nincs már több lap, amit lekérhetnénk.

5.7. Ellenőrzés

Úgy érzem, ez a programrész már nem igényel további magyarázatot, hiszen úgy jött létre, hogy lemásoltam a felhasználók adatainak beolvasását a függvény elejéről és csak a feliratokat módosítottam benne.

```
ml_status.getRange("A1").setValue("Ellenőrzés...");
ml_status.getRange("B1").setFormula(
                          '=SPARKLINE(C1;{"charttype"\\"bar";"max"\\'+userMAX+';"color1"\\"#960000"})');
ml_status.getRange("C1").setValue(0);
SpreadsheetApp.flush();

var felhasznalok = new Array();
userMAX=0;
db=0;
var p = { customer: "my_customer",
          maxResults: 30,
          pageToken: "" };

do {
  var lista = AdminDirectory.Users.list(p);
  p.pageToken = lista.nextPageToken;
  if(lista.users)
    for(var i = 0; i<lista.users.length; i++){
      if((lista.users[i].primaryEmail != admin) &&
                          ((testmode && lista.users[i].primaryEmail.indexOf("teszt")>=0) || !testmode)){
        felhasznalok.push(lista.users[i].primaryEmail);
        userMAX++;
      }
```

```
        db++;
      }
    ml_status.getRange("C1").setValue(db);
    SpreadsheetApp.flush();
  } while (p.pageToken);
```

Nézzük meg inkább a függvény utolsó néhány sorát:

```
  ml_status.getRange("A1").setValue("A program futása befejeződött.");
  felhasznalok.forEach((email,i,felhasznalok) => felhasznalok[i] = [email]);
  ml_status.getRange(11, 1, userMAX, 1).setValues(felhasznalok);
  SpreadsheetApp.flush();
}
```

Vigyázó szemeiteket a második sorra vessétek!

Az elején, amikor a „*Diákok*” munkalapról húztuk be egy tömbbe a diákok e-mail címét (`emailek`), már volt róla szó, hogy a `getValues()` metódus mindig kétdimenziós tömböt ad vissza.

Most hasonló problémával állunk szemben, csak fordítva. Van egy egydimenziós tömbünk (`felhasznalok`), aminek a tartalmát be kellene raknunk egy táblázat tartományába. Ehhez pedig ezt az egydimenziós tömböt kétdimenzióssá kellene alakítanunk, mert a következő sorban lévő `setValues()` függvény csak ilyet fogad el.

Ehhez azt csináljuk, hogy egy `forEach` metódussal végigszaladunk a tömb minden elemén és berakjuk azokat egyelemű tömbként saját magába.

Ha nem akarunk bajlódni ezzel az átalakítással, akkor meg lehet oldani úgy is a problémát, hogy az előző kódrészletben szögletess zárójelek közé tesszük a push metódis paraméterét. Valahogy így:

```
        felhasznalok.push( [lista.users[i].primaryEmail] );
```

Így eleve egyelemű tömbök kerülnek bele a `felhasznalok` tömmbe és ettől kétdimenziós lesz.

Akárhogy is, a végén meg fognak jelenni a módosítot e-mail címek a „`Status`” munkalapon.

6. Tapasztalatok

Nagyon ritkán, de előfordul, hogy egy sokszor kipróbált és jól működő Apps Script megáll és megjelenik egy olyan hibaüzenet, ami arról tájékoztat, hogy a script leállt, mert a szolgáltatás nem elérhető. Természetesen az egyik ilyen alkalom az volt, amikor élesben futtattam a scriptet ;o)

Ilyenkor elég egyszerűen újra elindítani a végrehajtást, mert csupán arról van szó, hogy a Google szerverei épp túlterheltek és nem tudnak foglalkozni a mi scriptünkkel.

Olyan is előfordulhat, hogy egy adatbeviteli (prompt) mezőjének kitöltése közben otthagyjuk a gépet, vagy épp csak átváltunk egy másik alkalmazásra, mert más dologgal kell foglalkoznunk. Ha 30 percnél tovább nem térünk vissza hozzá, akkor már egy hibaüzenet vár bennünket, mert egy script legfeljebb fél óráig futhat. Ha ennél tovább megy, akkor leállítja azt a Google. Természetesen ez is megvolt ;o)

Ettől sem kell hátast dobni, nyugodtan folytathatjuk a munkát.

Mi lenne, ha a program mindenféle ellenőrzés és gondolkodás nélkül végigmenne a felhasználók e-mail címein és módosítaná azokat? Vajon mi történne akkor, amikor ahhoz a rendszergazdához ér, aki éppen futtatja a scriptet?

Semmi. ;o)

Simán módosítja az ő e-mail címét is, mindenféle következmény nélkül.

A Google ugyanis nem az e-mail címek alapján kezeli a felhasználókat belülről, hanem az egyedi és megismételhetetlen `id`-jük (illetve `customerId`-jük) segítségével. Minden más, csak egyszerű adatmezőnek számít, amit bármikor át lehet írni. (Ez nem teljesen igaz, mert van még néhány mező, aminek nem lehet és nem is szabad megváltoztatni a tartalmát. Pl.: `kind`, `etag`, stb. A `primaryEmail` viszont, amivel dolgozunk, pont nem ilyen.)

https://developers.google.com/admin-sdk/directory/reference/rest/v1/users

Tehát tulajdonképpen szükségtelen az a programrész, ami a scriptet futtató admin e-mail címének átírását akadályozza meg. Azért tettem bele, mert úgy gondoltam enélkül olyan, mintha saját magunk alatt vágnánk a fát.

Ez azonban nem így van. Sőt! „*Más alól sem tudjuk kihúzni a szőnyeget*.” Ha egy felhasználó be van bárhol, bármilyen eszközön jelentkezve, és módosítjuk az e-mail címét, akkor sem történik semmi. A rendszer egy idő után érzékeli, hogy ez az adat megváltozott és frissíti magának.

Én attól tartottam, kidobja majd a felhasználókat és újra be kell jelentkezniük. Mivel tapasztalatból tudtam, hogy sok diák már rég elfelejtette a jelszavát, ezért írtam meg úgy a scriptet, hogy a „Diákok” munkalapon szereplő felhasználóknak a jelszavát is írja vissza az alapértelmezettre. Nos, erre sincs igatából szükség.

A kódban bennehagytam ugyan ezt a funkciót, de a „Diákok” munkalap feltöltése opcionális.

Azért van néhány eset, ami indokolhatja a használatát. Ha valaki ki van jelentkezve a böngészőjében a Google fiókjából, amikor a váltás megtörténik. Ekkor ugyanis nem módosul automatikusan az e-mail címe a böngészőben és újra be kell jelentkeznie (az új e-mail címmel). Ha már elfelejtette a jelszavát, mert pl. megjegyeztette a böngészőjével, akkor gondjai lesznek. A legtöbben ugyanis azt sem tudják, hol lehet megnézni az elmentett jelszavakat.

Egy másik példa, amikor új eszközt kap valaki, azon még eleve nem volt bejelentkezve sohasem…

Maga az Apps Script egy nagyon hasznos kiegészítője a Google applikációknak. Egyetlen hátránya a sebessége és a korlátai. Amikor elkezdtem kialakítani a rendszeremet és megismertem az Apps Scriptet, úgy gondoltam, írok egy scriptet, ami automatikusan feltölti adatokkal az egész rendszert a KRÉTA napló adatai alakpján. Felhasználók, szervezeti egységek, csoportok, kurzusok, minden.

Úgy gondoltam, ez a tökéletes megoldás, hiszen nem függ a kliens gép adottságaitól, mivel a Google saját környezetében fut. Sőt, mivel otthonos környezetben mozog, gyors és hatékony eszköz.

Csak amikor majdnem teljesen elkészültem, akkor szembesültem vele, hogy rengeteg korlátozás vonatkozik rá:

https://developers.google.com/apps-script/guides/services/quotas

Ugyanez elmondható a külső API hívásokkal kapcsolatban is, de érdekes módon, ott kedvezőbbek a kvótaelőírások. Nagyobb, összetettebb feladatokat tehát inkább úgy érdemes megoldani.

Egyelőre ennyit szerettem volna elmondani erről a technikáról.

Köszönöm, ha végigolvastad!

Jó játékot Mindenkinek!