[Külső fekete borítólap formátuma]

Széchenyi István Egyetem

Gépészmérnöki, Informatikai és Villamosmérnöki Kar

Informatika Tanszék

**SZAKDOLGOZAT**

**Szűcs Henrik**

**Gazdasági Informatikus BSc szak**

2022

| [Gerincen:] Hallgató Neve, Évszám {Titkosított} |
| --- |



**SZAKDOLGOZAT**

**Webalkalmazás fejlesztése visual scripting segítségével**

**Szűcs Henrik**

**Gazdasági Informatikus BSc szak**

**2022**

**[Ennek a bekezdésnek a helyére szerkessze be az aláírásokkal ellátott feladatkiírási lap szkennelt változatát.]**

1. **Nyilatokzat**

Alulírott, *Szűcs Henrik (G28AZI)* Gazdasági Informatikus BSc szakos hallgató kijelentem, hogy a *Webalkalmazás fejlesztése visual scripting segítségével* című szakdolgozat feladat kidolgozása a saját munkám, abban csak a feltüntetett forrásokat, és a megjelölt mértékben használtam fel, az idézés szabályainak megfelelően, a hivatkozások pontos megjelölésével.

Eredményeim saját munkán, számításokon, kutatáson, valós méréseken alapulnak, és a legjobb tudásom szerint hitelesek.

Győr, 2022.09.30 …………………………….

hallgató

**Kivonat**

Webalkalmazás fejlesztése visual scripting segítségével

A Construct 3 webes technológiákon alapuló visual scripting keretrendszer, melyet elsősorban játékok fejlesztéséhez terveztek. Mivel a legtöbb mai webalkalmazás a felhasználói interakciók, válaszidők és grafikai megjelenítés tekintetében is hasonló egy számítógépes játékhoz, felmerül a kérdés, hogy ezt a keretrendszert fel lehet-e használni általános célú webalkalmazások készítésére, esetleg különféle bővítmények elkészítése árán. Ez a kezdeti befektetés megtérülhet, ha később kevésbé magasan képzett fejlesztők is bevonhatók egy összetett szoftver fejlesztésének folyamataiba.

**Abstract**

Web application development with visual scripting language

Construct 3 is a web based visual scripting framework which is developed for creating games. Nowadays most web applications are similar to computer games in graphics, response time and user input. The question is whether this framework can be used to build general purpose web applications even with custom made plugins. The initial extra work may pay off if fewer professional workers are enough to work on complex projects.

**Tartalomjegyzék**

[**1. Bevezetés 1**](#_cgzg9wnxn5k)

[1.1. Témaválasztás indoklása 1](#_ht5770aingsc)

[1.2. A megrendelő problémája 1](#_iw62k5azmmp0)

[1.3. A fejlesztő problémája 1](#_dnaq38lyhwip)

[1.4. Környezet kiválasztása 2](#_qjxporec7zkh)

[1.5. Példa probléma 6](#_97qb6khpvas)

[**2. Szerkesztő bemutatása 7**](#_m4n5yksiwj3g)

[2.1. Telepítés 7](#_gwp0mgjvyx21)

[2.2. Beállítások és eszközök: 8](#_ln7d56ja7m61)

[2.3. Új Projekt 9](#_a57087y5d6mn)

[2.4. Projekt szerkezete 9](#_ya3xefsku10z)

[2.5. Kezelőfelület beállítása 11](#_cfw0esac7rv3)

[2.6. Példa elkészítése 12](#_eacerrrqxfp)

[2.6.1. Projekt beállítása 12](#_vu88l3gzef4)

[2.6.2. Elrendezés 13](#_9g6o739kim8u)

[2.6.3. Esemény lap 16](#_47ntf6nv40t1)

[2.6.4. Exportálás 22](#_intp6c9mo38z)

[**3. Tömeges email küldő 23**](#_9j711pr2wm6p)

[3.1. Működés 24](#_p0ultmilryc9)

[3.2. Felület 25](#_800k7l2wr4os)

[3.3. Hiányzó elemek 26](#_68kyeh8ztt0l)

[**4. Kiegészítők elkészítése 26**](#_5oz68eun4o0t)

[4.1. Fejlesztői mód 26](#_ak0snspfdk8s)

[4.2. JavaScript környezet 28](#_84hjieh06xqr)

[4.3. Kiegészítők felépítése 29](#_xlytzhtjpzio)

[4.3.1. Szerkesztői fájlok 30](#_hugb5xbviiw)

[4.3.2. Futási környezeti fájlok 30](#_aiozv9wgecmf)

[4.4. Lokalizáció bővítmény 31](#_3esxvomlrnns)

[4.4.1. Szerkesztő 32](#_crpm9b76no2h)

[4.4.2. Futási környezet 32](#_s0c2b3o4nmzs)

[4.5. Lokalizáció viselkedés 32](#_5xonxqfked6m)

[4.5.1. Szerkesztő 32](#_ylze9vn9i6b9)

[4.5.2. Futási környezet 33](#_vv9ozp8l1uj5)

[4.6. Titkosítás 33](#_a0zoeb1lzdj2)

[4.6.1. Szerkesztő 33](#_mvvyk5cu9hem)

[4.6.2. Futási környezet 33](#_49sfdkctgj5t)

[4.7. Email küldés 33](#_umr99ok43hf1)

[4.7.1. Szerkesztő 34](#_1lokqbbba9x)

[4.7.2. Futási környezet 34](#_kciyv1ssp4rk)

[4.8. Szövegszerkesztés 34](#_gd5krzs9ncvv)

[4.8.1. Szerkesztő 35](#_x0rnptvo50xz)

[4.8.2. Futási környezet 35](#_ceud3ix0549d)

[4.9. CSV formátum kezelés 35](#_1ah9equ3yywy)

[4.9.1. Szerkesztő 35](#_28ydzf56g0q6)

[4.9.2. Futási környezet 35](#_7xn4qtar0gbs)

[**5. Tömeges email küldő elkészítése 36**](#_19ggyqyhmpj6)

[5.1. Alap beállítások 36](#_53lfcc9o17u0)

[5.2. Adatszerkezet 37](#_785mig2jzqry)

[5.2.1. Ideiglenes adatok 37](#_r7w0ys24rudl)

[5.2.2. Helyi beállítások 37](#_x6ximykvee02)

[5.2.2. Felhasználók 37](#_k8b8m3yt9mrh)

[5.2.3. Szerverek 39](#_ou3trdrra2d7)

[5.2.4. Feladatok 39](#_5rvc8adrneov)

[5.3. Betöltés és mentés 41](#_883poq4n1t9y)

[5.4. Bejelentkezés 43](#_yp3oryyjutg2)

[5.5. Főmenü 44](#_49wyqbwb4se5)

[5.6. Beállítások 45](#_5gfiu7b4xust)

[5.7. Szerverek kezelése 45](#_e0jbocb50es7)

[5.8. Feladat kezelése 46](#_qjvzctxwxymu)

[5.8.1. Küldő szerverek kezelése 46](#_sk31o61dwvmv)

[5.8.2. Címzettek beállítása 47](#_31lrtryrrk2d)

[5.8.3. Levél szerkesztés 47](#_3kel2wp4utb9)

[5.8.4. Feladat állapota 47](#_ga49hz48btkd)

[5.9. Küldés 47](#_mq1829wx6mdb)

[**6. Konklúzió 47**](#_gsszlbn8111u)

[6.1. Program 47](#_j3o2xb8eqymc)

[6.2. Szerkesztő 47](#_5qwqc88bnvfi)

[**Irodalomjegyzék 49**](#_wsikuiwoz7qw)

[**Mellékletek 54**](#_4cdc7r4caisp)

# 1. Bevezetés

## 1.1. Témaválasztás indoklása

Munkám során a megrendelők egyedi programokat rendelnek, azonban azok átadását követően általában felmerül a program bővítésének igénye. A hagyományos módszerekkel elkészített programokat csak egy speciális szaktudású programozó tudja bővíteni. Személyes célom, hogy egy megírt programot könnyen lehessen bővíteni és a programkészítés folyamata bárki számára elérhető legyen még akkor is, ha nem jártas az adott programozási nyelvben. A szakdolgozat egy problémán keresztül mutatja be az elképzelésem megvalósítását. A cél, hogy a kialakított szerkesztő környezet támogatni tudja a felhasználók kreativitását. Eszközt szeretnék adni az embereknek ahhoz, hogy saját programokat tudjanak létrehozni és a megírt programot hatékonyan tudják csapatban, másokkal együtt továbbfejleszteni.

## 1.2. A megrendelő problémája

A szoftverfejlesztés egyik legnagyobb nehézsége, hogy a megrendelőnek egyedi igényei lesznek. A szoftvert az adott feladathoz testre kell szabni és a megfelelő informatikai környezetbe kell illeszteni, ami nagy költségekkel jár. Az egyedi szoftverfejlesztés költségei nagyon magasak már a munka kezdetekor. Külön szakembert/csapatot kell bérelni és az esetleges további módosításokhoz is a szakemberhez/csapathoz kell fordulnunk. Ha a prototípus elkészítése valamilyen okból nem sikerül, az nagy veszteségeket okozhat, mert ennek kifejlesztése során is szakembereket kellett fizetnünk és a program készült el.

## 1.3. A fejlesztő problémája

Problémát jelent a fejlesztések költségének féken tartása, illetve a szükséges szakemberek keresése, betanítása. A költségek nagy részét a munkaerő bére teszi ki. Gazdaságosabb a megrendelt programot nem teljesen alapjaitól megírni, érdemes kiválasztani egy keretrendszert (sablont), hogy akár kevésbé szakképzett fejlesztő, kevesebb idő alatt készítsen el egy programot. Továbbá a fenntartási költségeken is érdemes spórolni, ha az alkalmazás funkcióit önállóan, külső szolgáltatás igénybevétele, licencek nélkül is tudjuk üzemeltetni.

## 1.4. Környezet kiválasztása

Fejlesztés megkezdése előtt keretrendszert fogok választani. Egy keretrendszer lényege, hogy a különböző alkalmazásokban leggyakrabban használt elemeket egyetlen csomagba foglalja, ez megkönnyíti a fejlesztést mert az alapfunkciókat nem kell megírnom. További előny, hogy a keretrendszert használó fejlesztők segítséget tudnak nyújtani egymásnak, illetve példamegoldásokat oszthatnak meg. Keretrendszer kiválasztásakor a legfontosabb szempontok lehetnek az alábbiak:

* **platformfüggetlen**: Az elkészített programok működjenek ugyanúgy a különféle rendszereken. Lehetőleg csak megkerülhetetlen platformspecifikus megkötések legyenek (Pl.: Asztali környezeti tálca ikon, áruház integráció stb.). További előny, ha a fejlesztőkörnyezet is platformfüggetlenül használható.
* **bővíthető**: A keretrendszer funkcióit ki lehessen egészíteni saját tartalmakkal, kódokkal. Előny, ha a kiegészítés teljesen vagy részlegesen kompatibilis egyéb programnyelvvel, illetve programokban használt kiegészítőkkel/tartalmakkal, ezen felül támogatást biztosít ezek importálásához.
* **könnyen kezelhető**: A keretrendszerhez legyen támogatás, dokumentáció és frissítés, ami a későbbi munkát segíti. A fejlesztői program kezelése könnyen elsajátítható legyen (pl. hasonló programoktól kevés eltérés). A fejlesztői vagy az elkészített program lehetőleg „Out of the Box” működjön, vagyis beüzemelése ne igényeljen más külső, nem mellékelt keretrendszer(eke)t, extra beállításokat, illetve összetett konfigurációt (Pl.: önállóan futtatható legyen, hordozható legyen a program, ne igényeljen külső SDK letöltést, ne kelljen környezeti változókat, regisztereket beállítani stb.).

A piacon rengeteg rendelkezésre álló programnyelv és szoftver van. Keresésnél elsődleges szempontom, hogy asztali, mobilos és webes környezetben is működjön az elkészített program, mert személyes tapasztalatom, hogy a megrendelők sokféle eszközön akarják majd futtatni a megrendelt applikációt. Másodlagos szempontom, hogy lehetőség legyen önállóan futtatható (monolit) vagy self-hosted program készítésére, így készíthető külső szolgáltatás-független alkalmazás, ami további költségmegtakarítást jelent a fejlesztő számára, mert a megrendelő saját rendszerén futtathatja a programot, így nem kell szolgáltatást fenntartani, karbantartani a fejlesztőnek.

Számos programozási nyelven készítenek programfejlesztési eszközöket, ezért nincs programnyelvbeli megkötésem, de számomra a webes technológiák ismerete miatt választáskor a JavaScript-re épülő rendszerek élveznek előnyt.

* **Flutter** [1]: Modern és reszponzív felhasználói felület építésére összpontosító keretrendszer, azonban itt a közös kódot a fordító platformspecifikus natív kóddá alakítja, így a platformok között eltérés tapasztalható.
* **React** [2]: Ingyenes és nyílt forráskódú JavaScript keretrendszer, kliens oldali alkalmazásokhoz. A Meta cég és a felhasználói közössége tartja karban. A keretrendszer elsősorban a felhasználói felület készítését segíti. Ehhez programozói tudás és fejlesztői környezet telepítése, beállítása szükséges.
* **Vue** [3]: Ingyenes és nyílt forráskódú JavaScript keretrendszer, kliens oldali „egylapos” alkalmazásokhoz. Funkcióját tekintve hasonló a React-hez. Megjelenítés HTML elemeken keresztül történik, a különböző elemeket interaktívan kezeli.

Számtalan további keretrendszer létezik, de ezek a keretrendszerek speciális programozói tudást igényelnek, telepítésük, a kód karbantartása, továbbadása és verziókezelése nehézkes lehet egy kezdő számára, így egyszerűbb keretrendszereket kerestem. CMS (Content Management System) rendszerek a tartalom kezelésére összpontosítanak, általában kényelmes grafikus felületet biztosítanak a működés és tartalom kezelésére.

* **Wordpress** [4]: Eredetileg blogger keretrendszernek készítették. Webszerverre történő telepítés után a felhasználó webes böngészőn keresztül testre szabhatja weboldalát grafikus kezelőfelületen. Kiegészítőkkel bővíthető, azonban a legnagyobb hátránya, hogy kevés jól működő alapfunkciója van. A keretrendszert használó pedig eltévedhet a rengeteg félkész, fizetős, kis funkcionalitást biztosító kiegészítők tengerében.

Számtalan további tartalomkezelő rendszer létezik különböző célokra optimalizálva, pl. Joomla [5], Drupal [6], Wix [7]. A bővíthetőség, könnyen kezelhetőség adottak, de kizárólag webszerverre telepítve működik, webes környezetben fut, offline futtatható alkalmazás készítésére nem alkalmasak és a kiegészítők összességében drágává tehetik.

Programok készítésére „no code”/„low code” keretrendszerek is rendelkezésre állnak. Ezek a rendszerek a programkészítés során a hagyományos kódolási feladatokat csökkentik és áthelyezik egy kezdők számára könnyen átlátható grafikus felületre. Hasonló grafikus felületet biztosítanak, mint a CMS rendszerek, de ezeknél maga a program készítés nagyobb hangsúlyt kap.

* **Appsheet** [8]: Google üzleti alkalmazás készítő szolgáltatása. Többi Google szolgáltatás összekötésével egyedi munkamenetünkhöz igazodó alkalmazást hozhatunk létre.
  + Előnyök: Jól támogatott, egyszerű a használata
  + Hátrányok: Csak szolgáltatás formában elérhető, nem lehet teljes mértékben testre szabni.
* **Appery.io** [9]: Webes és platformspecifikus applikáció készítő szolgáltatás. Vizuális szerkesztő mellett backend szolgáltatásokat is biztosít. Különböző bővítményekkel bővíthető az alkalmazás.
  + Előnyök: Sokféle funkció és kiegészítés, nagyfokú testre szabási lehetőség
  + Hátrányok: Csak szolgáltatás formában elérhető, kisebb projektekhez drága

Legtöbb ilyen rendszer csak szolgáltatásként érhető el és a teljeskörű bővítése nem lehetséges. Rengeteg hasonló további „no code” [10] és „low code” [11] alkalmazás létezik a piacon.

Tovább bővítettem a keresést játékmotorokra is, mivel ezekkel a rendszerekkel olyan programokat lehet írni, amelyek külső szolgáltatás nélkül, gyorsan tudnak reagálni a felhasználói bevitelre, animációkat, rendszer funkciókat tudnak kezelni és önállóan is futtathatók.

* **Unity** [12], **Unreal Engine** [13], **Godot** [14]: Ezeknek a motoroknak az alapelve, hogy stabil, jól bevált nyelveken, (C, C++, C#, Python stb.) fejlett funkciókat valósítanak meg, amelyeket a programozó felhasználhat. Mivel a felhasznált nyelvek legtöbb funkcióját kihasználja a motor, a fejlesztőnek is haladó tudással kell rendelkeznie az adott programozási nyelven és a motorról.
  + Előnyök: Nagy közösség, kiterjedt funkcionalitás, fejlett szerkesztő
  + Hátrányok: Használata speciális tudást igényel (programozási nyelv, függvények/motor ismerete)
* **Construct 3** [15], **Game Maker: Studio** [16], **Gdevelop** [17]: Ezen programok a fentiekben bemutatott programok ötvözete. Gyors tanulási lehetőséget biztosítanak vizuális programozási lehetőséggel (visual scripting), azonban lehetőség van a motort bővíteni hagyományos programozási nyelveken írt kódokkal, így bonyolultabb projektek készítésére is alkalmasak.
  + Előnyök: bárki számára könnyen elsajátíthatók, bővíthetők
  + Hátrányok: kevésbé professzionális megoldások, esetleges korlátozott futási sebesség

A kész program költségcsökkentése céljából olyan keretrendszerek közül választok, amelyek nem igényelnek különleges képzést és tudást, így a munkaerő költsége, illetve a kész program elkészítési ideje is csökkenthet. Végül a **Construct 3**, **Game Maker: Studio**, **Gdevelop** programokra szűkítem a keretrendszer választásomat, de az alábbiakban a lehető legjobb választási eredmény érdekében a programokat egymással is összehasonlítom.

**Game Maker: Studio:** C++-ban írt fizetős asztali játékprogram szerkesztő. Úgy a vizuális programozást (GML Visual), mint a szerkesztő saját programozási nyelvét is támogatja (GML Code). A szerkesztő saját GML fordítót használ a kész program C++ kódjának elkészítéséhez. Ennek fizetős modellje változatos funkciókat támogat.

**Gdevelop**: JavaScriptben írt webes szerkesztő. Az exportált program is JavaScriptben készül egy környezeti motorba beágyazva. A program teljesen ingyenes. Csak vizuális szerkesztés lehetséges. Kiegészítők készítésére is lehetőség van, azonban haladóbb, programozói szinten nem lehet bővíteni ezt a programot. A program felépítése kissé összefüggéstelen (nehéz átlátni, hogy mely objektumokból hogyan épül fel a program).

**Construct 3**: JavaScriptben írt fizetős webes szerkesztő program. Az exportált program is JavaScriptben készül egy környezeti motorba beágyazva. Vizuális programkódot és a JavaScript programozási nyelvet is támogatja. Kiegészítők készítésére is lehetőség van JavaScript nyelven. Nagy hangsúly van a generált kód optimalizálásán, modern API-okat, illetve technológiákat használ.

**Game Maker: Studio** – **Gdevelop** – **Construct 3** összehasonlítása: A három motor közül a Construct 3 dolgozik a legújabb technológiákkal [18]. A Gdevelop ingyenessége miatt a három keretrendszer közül a legkevésbé fejlett. Bővíthetőség tekintetében mind a Gdevelop mind Construct 3 bővíthető, azonban Game Maker: Studio-nál ilyen lehetőségek nincsenek. Használni a Game Maker: Studio-t a legnehezebb, grafikus programozási felületének képessége eléggé korlátozott. Gdevelop és Construct 3 kezelése egyszerű, azonban Construct 3 felülete számomra könnyebben használható. Beüzemelés tekintetében a Construct 3 a legegyszerűbb mert nem igényel semmilyen telepítést és betöltés után minden funkció azonnal elérhető.

*1. táblázat: Game Maker: Studio, Gdevelop, Construct összehasonlító táblázat*

|  | Game Maker: Studio | Gdevelop | Construct 3 |
| --- | --- | --- | --- |
| Funkciók | Fejlettebb | Kevésbé fejlett | Fejlett |
| Használat | Nehezebb | Egyszerű | Egyszerű |
| Ára | Fizetős (3400 Ft/hó) | Ingyenes (opcionális szolgáltatás 1700 Ft/hó) | Fizetős (3400 Ft/hó) |

Személyes választásom a Construct 3-ra esett. Fizetős a program, de az általam támasztott követelményeknek ez felel meg a legjobban. Kiegészítőkkel bővíthető, visual scripting segítségével könnyen használható, a generált JavaScript kód platformfüggetlenül, önállóan futtatható és elég gyors.

## 1.5. Példa probléma

Small/home office, kisvállalati, oktatási környezetben jellemző, hogy évente egy-két alkalommal rengeteg felhasználónak kell köremail-t küldeni. Valós példák: Országos Magyar Vadász Kamaránál az éves jegy hosszabbítás február végén történik és az esetleges tudnivalókról több ezer tagot kell értesíteni egyidőben. Az oktatásban a tanároknak évkezdéskor több száz diákot és szülőt kell értesíteniük a változásokról köremail formájában.

A megrendelők egy olyan szoftveres megoldást szeretnének, amivel akár 1000-2000 email is elküldhető. Külön szervert és informatikust nem tudnak biztosítani a beüzemelésre. Lehetséges megoldások a problémára:

* Szolgáltatás igénybevétele: Egy tömeges email küldő szolgáltatásra történő előfizetés (Mailchimp [19], Mailgun [20], stb.).
  + Előnyök: azonnal rendelkezésre áll, nagy kínálat a szolgáltatások között
  + Hátrányok: szolgáltatás mivolta miatt az egyszeri használat drága lehet (általában a cél a nagyobb csomag megvásárlása), céges adatvédelmi szabályzatokkal való ütközés miatt nem használható minden cégnél.
* Program használata (Outlook [21], Thunderbird [22], stb.):
  + Előnyök: teljesen független az előfizetéstől, bárki számára hozzáférhető, használható. Hosszú távon olcsóbb.
  + Hátrányok: meglévő email fiók szükséges

A megrendelők inkább egy számítógépen futó programot képzelnek el, mint például az Outlook, Office, Chrome stb. Jelenleg a következő programokat találtam alkalmasnak a tömeges email küldésre:

* Thunderbird: Mozilla alkotóinak levelező kliense. Nyílt forráskódú, ingyenes, rengeteg platformra letölthető. Ingyenessége miatt nagyrészt közössége fejleszti. Nincsenek prémium funkciók és szolgáltatások a programhoz mellékelve. Használata kicsit archaikus, gyakorlást igényel, ha másik levelező rendszerről térünk át használatára.
  + Előnyök: ingyenes
  + Hátrányok: tömeges levélküldés csak „3rd party” (külsős) kiegészítővel lehetséges, ennek nehézkes a beüzemelése (kiegészítők megbízhatósága kérdéses)
* Sendblaster [23]: Delivery Tech Corp. által fejlesztett Windows-os szoftver (megjelenéséből ítélve valószínűleg .NET, WPF applikáció), a fejlesztő által „befejezett” szoftvernek látszik. Ára 99 euró + a frissítések és kiegészítők további költséggel járnak.
  + Előnyök: egyszerű használat, kész alkalmazás
  + Hátrányok: drága (a fő verziókat külön kell megvenni), csak Windows rendszerre elérhető, jövőbeli frissítései kérdésesek (utolsó frissítés: 2020.04.10.)

Megrendelő a „Sendblaster” szoftverhez hasonló programot szeretne. Cél, hogy a szerkesztővel készített program a megrendelő egyedi igényeit megvalósítsa és a fejlesztő gazdaságosan, fenntarthatóan tudja elkészíteni a programot.

# 2. Szerkesztő bemutatása

A szerkesztő felépítését és szerkesztőben történő programszerkesztést egy alap példa programon keresztül fogom bemutatni. A példa feladatban egy adott téglalap területét kell kiszámítani. A program a téglalap két oldalának hosszát fogja kérni és kiírja a kiszámolt téglalap területét.

## 2.1. Telepítés

A szerkesztő bármely verziójának megnyitásakor a böngésző elkezdi letölteni és frissíteni a különböző forrásokat. A letöltés végeztével a program internet nélkül is használható, azonban a felhőbe való mentéshez, felhőbe történő exportáláshoz és a licenckulcs frissítéséhez internetkapcsolat szükséges.

Frissítések heti rendszerességgel érkeznek. Szakdolgozat írásakor az r336 volt a legfrissebb stabil verzió. A szerkesztőt különböző verziókban, módokon lehet elindítani:

* Normál / stabil verzió: <https://editor.construct.net/>. Stabil kiadás az azt megelőző béta ciklus után (8-10 verzió) érkezik. A stabil verziók a béta frissítéseket is tartalmazzák.
* Béta verzió: <https://editor.construct.net/beta>. Új funkciók, komplexebb javítások első megjelenése. Kevésbé tesztelt, néha hibásan működhet.
* Megadott verzió betöltése: https://editor.construct.net/rXYZ pl.: 308.2-es verzió <https://editor.construct.net/r308-2>. Az adott verzió betöltése hosszabb távú projekteknél lehetővé teszi a frissítések kihagyását, így hibamentesebb lesz a fejlesztés folyamata és maga a végeredmény is.
* Biztonságos mód: <https://editor.construct.net/?safe-mode>. Minden „3rd party” (külső fél) kiegészítőjét kikapcsolja, hogy elindulhasson a szerkesztő, ha valamelyik kiegészítő hibásan működött.

## 2.2. Beállítások és eszközök:

Bármely verzió indításakor egy előre beállított kezdőképernyő fogadja a felhasználót, ahol hasznos linkeket, ajánlott minta projekteket és a legutóbb megnyitott projekteket érheti el.



*1. ábra: Construct szerkesztő kezdőoldala*

A felső sávon a balra lévő *Menu* (*Menü*) gomb állandó elem. Ennek segítségével hívhatók elő a szerkesztő fontosabb beállításai.

*Menu* (*Menü*) > *Settings* (*Beállítások*) menüpont tartalmazza a szerkesztő beállításait. Itt beállíthatók a nyelv, felületi beállítások, mentési helyek és szerkesztői konfigurációk.

A szerkesztőt kiegészítőkkel lehet bővíteni a *Menu* (*Menü*) > *View* (*Nézet*) > *Addon manager* (*Kiegészítő kezelő*) menüpontban. Itt lehet a telepített kiegészítőket megtekinteni, törölni, illetve új kiegészítőket telepíteni.

## 2.3. Új Projekt

Új projekt létrehozásához a *Menu* (*Menü*) > *Project* (*Projekt*) > *New* (*Új*) elemet kell használni. Projekt létrehozásakor egy felugró ablakban kezdeti konfigurációkat adhatunk meg, ezek a beállítások később is módosíthatók. Jelenleg semmilyen beállítást nem adunk meg, csak a *Create* (*Létrehozás*) gombra kattintunk.

Létrehozás után plusz ablakok jelennek meg a kezelőfelületen. Ezek testre szabásáról a későbbiekben lesz szó.

## 2.4. Projekt szerkezete

Projekt létrehozása után alapértelmezett beállítások használatakor a jobb oldalon a *Project* (*Projekt*) ablakban látható a projekt szerkezete:



*2. ábra: Szerkesztő projekt ablaka*

Projekt nevével ellátott mappa a főkönyvtár. Ezen belül almappák, az almappák-on belül elemek és/vagy további almappák találhatók. A mappák és elemek átnevezését, törlését, másolását és egyéb funkciókat tartalmazó menüt a jobb egérgombbal lehet előhívni. A mappák funkció szerint vannak kialakítva:

* **Layouts (Elrendezések)** mappa tartalmazza az elrendezéseket, vagyis azokat a felületeket, melyeket a felhasználó a kész programban látni fog. Elrendezésen belül egymás alatt elhelyezkedő rétegek találhatók. Egyszerre egy elrendezés jeleníthető meg, de a rétegeket egy másik elrendezés módosíthatja. Elrendezésekhez egy-egy eseménylap is társítható.
* **Event sheets (Esemény lapok)** mappa tartalmazza az esemény lapokat, amelyek az elrendezések vezérlését végzik. Az eseménylap fogja tartalmazni a program konkrét működési logikáját. Az eseménylapok egymásba is beágyazhatók.
* **Scripts (Szkriptek)** mappa JavaScript kódok tárolására szolgál. Az itt tárolt programkódoknak ugyanaz a feladata, mint az esemény lapoknak, csak nem vizuálisan történik a programlogika elkészítése, hanem JavaScript nyelven egy szöveges fájlban.
* **Object types (Objektum típusok)** mappa tartalmazza a projekt építő elemeit, az objektumokat. Minden látható vagy nem látható elemet, funkciót (gomb, hang, videó, adattárolás) objektumok valósítanak meg, amiket a projekthez kell majd hozzáadni.
* **Families (Családok)** mappában családok szerepelnek. Ezek olyan csoportosítások, amelyekbe az azonos típusú objektumokat (pl.: gombokat) egy csoportba (családba) rakhatjuk, ezzel egyszerűbben, egy egészként tudjuk őket kezelni (pl.: a családban lévő minden szöveg objektumot egyszerre piros színre tudunk állítani)
* **Timelines (Idővonalak)** mappában idővonal elemek találhatók. Egy idővonal elemmel a különböző objektumok értékeinek időbeli vezérlését adhatjuk meg. *Eases* (*Átmenetek*) almappában az egyedi átmenetek találhatók. Az átmenet két értékváltozás közötti interpolációt [24] (pl.: lineáris, köbös stb.) adja meg.
* **Sounds (Hangok)** mappa olyan hangfájlok tárolására szolgál, amelyeknek az időben történő lejátszása kritikusan fontos (pl. kattintás hangja), ezért a mappába másolt hangokat a program indítás előtt be fogja tölteni.
* **Music (Zenék)** hasonlóan a hang mappához itt is hangfájlokat lehet tárolni, de az itt található audió fájlok nem töltődnek be a program indulásával, így lehetnek kisebb akadozások a lejátszás során.
* **Videos (Videók)** mappa a videó fájlok tárhelye.
* **Fonts (Betűtípusok)** különböző betűtípus fájlok (.ttf, .woff, stb.) tárhelye.
* **Icons (Ikonok)** a program által használt ikon fájlok (.png) találhatóak meg itt. Beállítható, hogy a fájlok melyik ikon típushoz legyenek társítva.
* **Files (Fájlok)** mappában bármilyen típusú fájl tárolható, akár az előzőekben kategorizált fájltípusok is. Csak az egyéb, nem kategorizálható fájlokat érdemes ide másolni, mert az ebben a mappában lévő fájlokat a szerkesztő nem fogja tudni optimalizálni (pl.: hang esetében nem végez tömörítést).

## 2.5. Kezelőfelület beállítása

A *Menu* (*Menü*) > *View* (*Nézet*) > *Bars* (*Sávok*) menüből nyolc különböző ablakot lehet előhívni, ezek az ablakok különböző funkciókat biztosítanak [25]. A projekt elrendezéseit, esemény lapjait, egyéb fájljait is külön ablakban lehet megjeleníteni.



*3. ábra: Szerkesztő alapértelmezett felülete*

Minden ablak törlése után csak a felső menü elem marad.



*4. ábra: Szerkesztő üres felülete*

Az ablak megnyitása után, az ablak szabadon mozgatható az adott felületen az ablak felső szélének egérrel történő húzásával.

Lehetőség van az ablakok illesztésére is, ez az ablak kék négyzetekre történő ráhelyezésével tehető meg. Illeszteni a képernyő vagy egy ablak négy oldalára lehet. Ha ráillesztjük az ablakot egy másik ablakra, akkor a megjelenő fülekkel lehet váltani az egymást fedő ablakok között. Az illesztés helyét jelzi a program.



*5. ábra: Szerkesztő felülete átrendezés közben*

Mobilon történő megjelenítés esetén az idővonal ablak nem elérhető, illetve az ablakok nem változtathatók, csak oldalról húzhatók be.

A továbbiakban a példa készítésénél az alapértelmezett elrendezést fogom használni, vagyis bal oldalon a *Properties* (*Tulajdonságok*) ablak, a jobb oldal felső részén a *Project* (*Projekt*) ablak, a jobb alsó részén pedig a *Layers* (*Rétegek*) ablak jelenik meg.

## 2.6. Példa elkészítése

### 2.6.1. Projekt beállítása

Egész projektre vonatkozó beállítások eléréséhez kattintsunk bal egérgombbal a projekt nevére a *Project* (*Projekt*) ablakban. Ezután a tulajdonság ablak a projekt tulajdonságait fogja megmutatni. *Properties* (*Tulajdonságok*) ablakban szerkesszük a következő értékeket:

* *Name* (*Név*) mezőbe írjuk be a projekt nevét: „Terület számoló”
* *Description* (*Leírás*) mezőben adjunk meg: „Téglalap terület számoló program.”
* *ID* mezőbe írjuk be: „com.teglalap.terulet”
* *Author* (*Szerző*) mezőbe írjuk be saját nevünket.
* *First layout* (*Első elrendezés*) listából válasszuk ki az egyetlen elrendezésünket.



*6. ábra: Projekt tulajdonságainak beállítása*

*Viewport size* (*Nézetablak méret*) mezőben a programunk felbontását állíthatjuk be. A többi beállítással jelenleg nem foglalkozok, a projekt létrehozásakor a sablonok ezeket a projekt tulajdonságokat előre beállítják.

### 2.6.2. Elrendezés

*Project* (*Projekt*) ablak > *Layouts* (*Elrendezések*) mappa > *Layout 1* (*Elrendezés 1*) elemet dupla bal egér kattintással vagy jobb egérgomb és *Open* (*Megnyitás*) gombbal nyithatjuk meg (ha még nem lenne megnyitva). Ilyenkor az elrendezés nevével elnevezett ablakban lesz látható az elrendezés.

Ha kiválasztjuk az elrendezés elemet egyszeri bal egér kattintással, akkor a *Properties* (*Tulajdonságok*) ablakban megjelennek az elrendezés tulajdonságai. Szerkesszük a *Size* (*Méret*) mezőket 854x480-as értékre, ezzel az elrendezésünk méretét a projektünk virtuális méretéhez igazítottuk.



*7. ábra: Elrendezés tulajdonságainak beállítása*

Projektünk különböző részeit objektumok fogják alkotni. Ezeket az *Object types* (*Objektum típusok*) mappában tudjuk kezelni. Új objektum hozzáadásához először a *Project* (*Projekt*) ablakban > *Object types* (*Objektum típusok*) mappára > jobb egérgombbal kattintva > *Add new object type* (*Új objektum típus hozzáadása*) gombbal kell megnyitni a *Plugins* (*Bővítmények*) listáját.



*8. ábra: Objektum hozzáadása*

A példa programunkhoz két beviteli (hosszúság és szélesség) mező és egy megjelenítési (kiszámolt terület) mező szükséges. A beviteli mezők *Text Input* (*Beviteli mező*) típusú objektumok lesznek, míg az eredmény megjelenítéséhez egy *Text* (*Szöveg*) típusú objektumot használunk.



*9. ábra: Objektum lista*

A létrehozott három objektumot egérrel történő húzással elhelyezhetjük az elrendezésen. Rendezzük el tetszés szerint a szövegdobozokat.

A bal egérgombra kattintással kijelölt objektumok tulajdonságai meg fognak jelenni a *Properties* (*Tulajdonságok*) ablakban. Állítsuk be a három objektum tulajdonságát:

* Eredmény szöveg:
  + *Name* (*Név*) legyen: „Eredmeny”
  + *Text* (*Szöveg*) legyen: „Eredmény:”
* Hosszúság beviteli mező:
  + *Name* legyen: „Hosszusag”
  + *Placeholder* (*Helyőrző*) legyen „Hosszúság”
* Szélesség beviteli mező:
  + *Name* legyen: „Szelesseg”
  + *Placeholder* legyen: „Szélesség”



*10. ábra: Elrendezésen lévő objektumok szerkesztése*

Ezzel a felhasználói felület elkészült. *Menu* (*Menü*) > *Project* (*Projekt*) > *Preview* (*Előnézet*) gombbal vagy fenti lejátszás gombbal kipróbálható a projekt. A különböző eszközökön egyformán fog megjelenni az elkészített elrendezésünk (jelenleg két beviteli mező és egy szöveg).

### 2.6.3. Esemény lap

A programot esemény lapokban lévő utasításokkal vezérelhetjük. Az eseménylap ugyanolyan funkciót tölt be, mint egy programkód, csak a megírt vezérlés emberközelibb formában jelenik meg pl.: leíró jellegű, külön részekre bontható, nincs szigorú szintaxis, beépített „varázslók” segítenek (végig vezetnek a beállításokon).

*Project* (*Projekt*) ablak > *Event sheets* (*Esemény lapok*) mappa > *Event Sheet 1* (*Esemény lap 1*) elemet dupla bal egér kattintással vagy jobb egérgomb és *Open* (*Megnyitás*) gombbal nyithatjuk meg (ha nem lenne még megnyitva). Ilyenkor az esemény lap nevével elnevezett ablakban lesz látható az esemény lap.



*11. ábra: Üres eseménylap szerkesztése*

Egy esemény lap Action (Művelet), Condition (Feltétel), Expression (Függvény, angol rövidítés: „ACE”) típusú blokkokból épül fel. A blokkok hierarchikusan épülnek fel, egy blokk tartalmazhat feltételt, műveletet, illetve további alblokkokat. Típusai az alábbiak lehetnek:

* Művelet a program „cselekvése”. Valamilyen konkrét műveletet végez pl.: írás, olvasás, áthelyezés, megjelenítés stb. Két fajta művelet létezik:
  + **Szinkron művelet**, aminek a befelyezését meg kell várni, további művelet nem futtatható közben.
  + **Aszinkron művelet**, ahol a művelet kezdete és befejezése között folytatható a további műveletek futtatása. Ezt olyan funkcióknál alkalmazzák, ahol a kezdés és befejezés között hosszabb idő telik el pl.: fájl beolvasása, webről történő adatkérés stb. A szerkesztő egy óra szimbólummal jelöli.
* Feltételek feladata: a műveletek lefutásának kezelése. A feltételben lévő blokkok akkor aktiválódnak, ha a feltétel igaz. A feltételeknek három fajtája van:
  + **Normál feltétel**, akkor aktiválja az alatta elhelyezkedő blokkokat, ha a feltétel igaz.
  + **Ciklus feltétel,** hasonló a normál feltételhez, de a feltétel újra ismétli a benne lévő blokkokat, amíg a feltétel igaz. A szerkesztő egy ismétlés ikonnal jelzi ezt.
  + **Esemény** olyan feltétel, ami egy történés hatására (pl.: egérkattintás, betöltés befejezése stb.) lefuttatja a blokkot. A szerkesztő jobbra mutató zöld nyíllal jelöli ezt.
* **Függvény** segíti a műveletek és feltételek paramétereinek megadását. A függvényeknek szöveg vagy szám formájában paramétereket adhatunk és szöveget vagy szám értéket kapunk eredményként.

Az esemény lap logikája (kivéve az eseményeket) minden képernyő frissítésekor lefut (általában 60-szor másodpercenként).

Most a konkrét példát készítjük el vagyis, ha a felhasználó megadja a hosszúságot és szélességet, akkor a program beolvassa a két oldalt és összeszorozza, az eredményt pedig kiírja a képernyőre. Példa programunkban egy eseménnyel figyeljük azt, hogy a beviteli (hosszúság és szélesség) mező tartalma megváltozott-e, vagyis a felhasználó beírt-e valamit.

1. *Add event* (*Esemény hozzáadása*) gombbal megnyitjuk a varázslót. Kiválasztjuk a hosszúság megadására szolgáló korábban megadott szöveg beviteli objektumot (nálam „Hosszusag” névvel rendelkezik).
2. A felsorolt feltételek közül kiválasztjuk az *On text changed* (*Szöveg változásakor*) esemény feltételt. Ez további paramétert nem kér, így hozzá is adja a szerkesztő az esemény lapunkhoz.

Mivel a másik beviteli mezőt (szélességet) is figyelni kell, ehhez is létrehozunk a fenti módszerrel egy eseményt. A két esemény összefűzhető egy feltétel blokkba, azonban a feltétel blokk alapértelmezetten *AND* (*ÉS*) kapcsolatot akar kialakítani a két feltétel között. Ezt meg kell változtatnunk *OR* (*VAGY*) kapcsolatra. A blokk szélére jobb egér kattintással előhívható menüből keressük ki a *Make OR block* (*VAGY blokk létrehozása*) elemet.



*12. ábra: Feltétel blokk VAGY kapcsolatra állítása*

A VAGY blokkban már összekapcsolható a két feltétel. Ebben a blokkban lévő további blokkok akkor fognak aktiválódni, ha a hosszúság vagy szélesség mezőt megváltoztatja a felhasználó.

Most a meglévő feltételek alá szeretnék új üres blokkot létrehozni. Ezt úgy tudjuk megtenni, hogy a létrehozott feltétel bal szélére (így az egész feltétel blokkot kijelöli, nem csak a feltétel részeket) jobb egér gombbal kattintunk, majd az előhívott menüből az *Add blank sub-event* (*További üres alesemény hozzáadása*) elemre kattintunk.



*13. ábra: Üres blokk hozzáadássa*

A létrehozott üres blokkhoz adjunk egy változót is, amiben a kiszámolt területet tároljuk. Ezt az üres blokkra történő jobb egérkattintással, majd az *Add* (*Hozzáadás*) > *Add local variable* (*Lokális változó hozzáadása*) gombbal tehetjük meg. A felugró ablakban adjuk meg a változó nevét („Terulet” nevet fogom használni).



*14. ábra: Változó hozzáadása*

Most kiszámoljuk a területet az oldalak összeszorzásával majd eltároljuk azt az előbb létrehozott változóban. Az üres blokk mellett lévő *Add action* (*Művelet hozzáadása*) gomb megnyomásával előugró ablakban válasszuk ki a *System* (*Rendszer*) elemet, mert a változókkal történő műveletek rendszer objektumban találhatók. *Set value* (*Érték beállítása*) művelet kiválasztása után a következő párbeszédablakban a művelet paramétereit állíthatjuk be. Nem kell más változót megadni (összesen egy változónk van a projektben), az érték részhez pedig a területet kell megadni.

A függvények segítségével dinamikusan tudjuk számolni az oldalakból a területet. A *Find Expressions* (*Kifejezések keresése*) gomb segítségével előhívható, hogy az adott objektumoknak milyen függvényei vannak, illetve milyen és hány paramétert kell átadni a függvényeknek. A következő függvényeket fogjuk használni:

* Beviteli mező *Text* tulajdonságát, ami a beviteli mezőbe írt aktuális szöveget adja vissza. Hivatkozni a beviteli mező nevével lehet, majd ponttal elválasztva tudjuk a megfelelő tulajdonságát használni pl.: Hosszusag.Text, Szelesseg.Text
* Rendszer *float* függvényét, amely az első paraméterben megadott szöveget konvertálja számmá. Ha az nem konvertálható, nullát ad vissza.

A területszámítási összefüggés a következőképpen alakul: float(Hosszusag.Text) \* float(Szelesseg.Text)



*15. ábra: Területszámítás elkészítése*

A területszámítás elkészítése után ellenőrizzük, hogy a terület pozitív-e. Ha igen, akkor kiírjuk az eredményt a képernyőre, egyébként kiírjuk, hogy nem jók az adatok.

Adjunk a területszámítás blokkjának alsó szomszédságába egy üres blokkot (fenti üres blokk hozzáadása rész ismétlése). Az újonnan létrehozott üres blokkra duplán bal egér gombbal kattintva feltételt adhatunk meg. *System* (*Rendszer*) objektumon belül *Compare variable* (*Változó hasonlítása*) feltételt válasszuk ki. A paramétereket úgy állítsuk be, hogy ha a változó nagyobb, mint nulla, akkor legyen igaz a feltétel.



*17. ábra: Területszámítás ellenőrzése*

A feltételhez adjunk meg kiíratási műveletet. *Add action* (*Művelet hozzáadása*) gomb után válasszuk ki az eredményt megjelenítő szöveg objektumot. A szöveg objektum *Set text* (*Szöveg beállítása*) műveletének segítségével kiírható az eredmény. Statikus szöveget “-jelek között lehet megadni, & jellel lehet összefűzni a szövegeket.

****

*18. ábra: Eredmény kiírása*

A különben feltételt az egész feltételre történő bal egérkattintással, majd *Add* (*Hozzáadás*) > *Add 'else'* (*'Különben' hozzáadása*) gombbal tehetjük meg. A létrejött feltételhez készítsünk egy műveletet, ami az eredmény szövegbe kiírja, hogy „hibás eredmény” (Lokalizációt nem támogat a szerkesztő, ezért ezt hasonlóan kell megtenni, mint az eredmény kiírást, csak statikus „hibás eredmény” szöveggel). Az elkészült programunkat tesztelhetjük is a fenti lejátszás ikonnal.



*19. ábra: Végső programlogika*

### 2.6.4. Exportálás

Exportálás során az elkészült projektünket önállóan futtatható programmá tesszük. Asztali exportáláshoz menjünk a: *Menu* (*Menü*) > *Project* (*Projekt*) > *Export* (*Exportálás*) > *NW.js* elemre.



*20. ábra: Projekt és végső program kapcsolata*



*21. ábra: Exportálás elkezdése*

A felugró ablakon lévő beállítások maradhatnak alapértelmezett értékeken, de a *Minify mode* (*Tömörítési mód*) elemet *advance* (*haladó*) értékre állíthatjuk gyorsabb, kisebb méretű programhoz, azonban ebben a módban lassúbb lesz az exportálás.

A következő ablakban az operációs rendszert és egyéb NW.js-re vonatkozó paramétereket választhatjuk ki (alapértelmezett beállítások megfelelőek). Ha valamely rendszerre nem akarunk exportálni, akkor az operációs rendszer neve melletti jelölőnégyzet ne legyen kijelölve a felsorolásban.

Ezután elkezdődik a program exportálása. Exportálás után egy csomagolt fájlt fog letölteni a böngészőnk. Ez a fájl a programunk futtatható változatát tartalmazza.

# 3. Tömeges email küldő

A szerkesztő általános bemutatása után rátérek az elkészítendő program megtervezésére. Ez a program már egy valós igényt fog lefedni. A terveket a megrendelő kérései és elképzelései alapján készítem el, majd a tervek alapján a szerkesztőben állítom össze a programot.

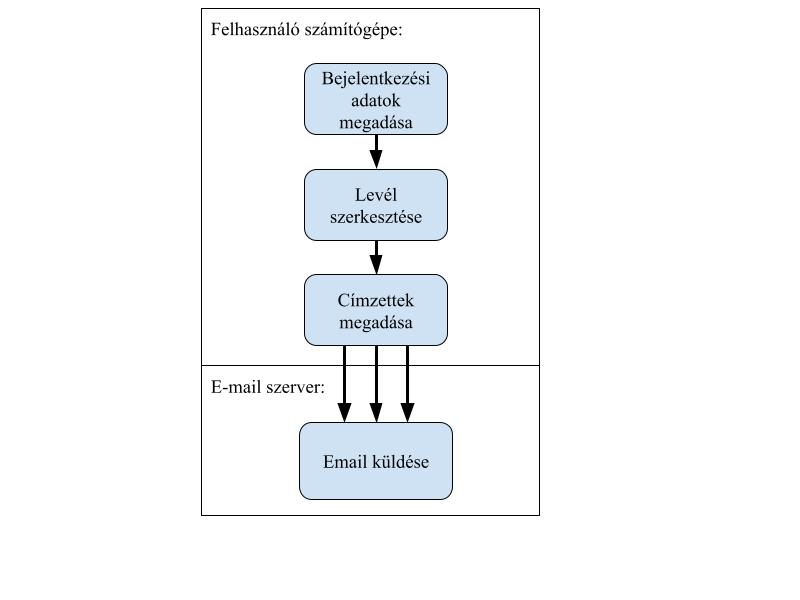
## 3.1. Működés

Az email küldő applikáció hasonlítani fog egy hagyományos levelező kliensre. A felhasználónak a meglévő e-mail címének elérését és bejelentkezési adatait meg kell adni, majd a program a levelező szerveren SMTP [26] protokollon keresztül az e-maileket kiküldi.

Tömeges e-mail küldés esetén a levelek egyszerre történő küldése nem járható út a címzettek nagy száma miatt. Száznál több címzett esetén SMTP szerver gyorsítótárazását külön konfigurálni kell, illetve különböző DDOS védelmi szabályok is korlátozzák a nagy számú küldést. A stabilabb működés érdekében leveleket egyesével vagy kisebb csoportokban kell elküldeni a levelező szerver felé. Gyorsabb e-mail küldés érdekében több e-címről, levelező szerverről is küldhetők lesznek a levelek. A programban beállítható lesz a küldés sebessége is.

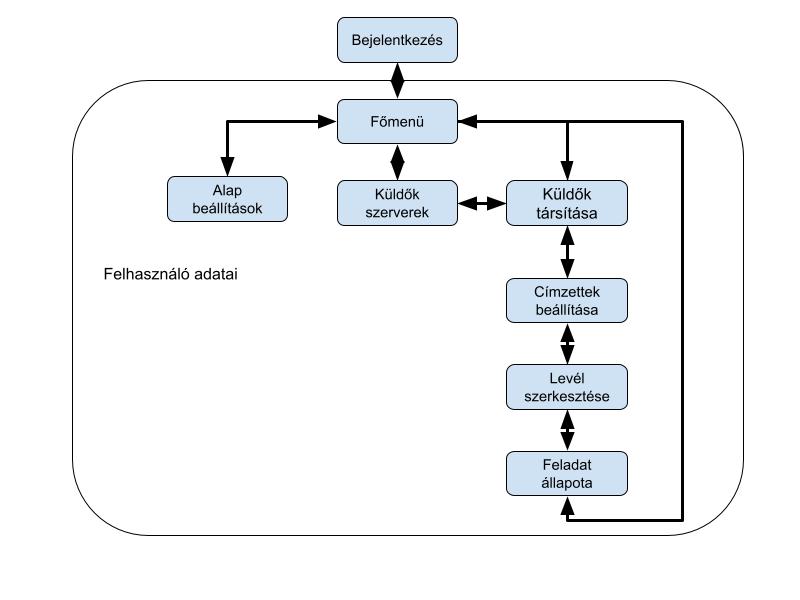
A különösen fontos a sikeres küldés kezelése, a programot az alábbi hibalehetőségekre mindenképpen fel kell készíteni:

* kapcsolat megszakadása az SMTP szerverrel
* program hirtelen bezáródása



*22. ábra: Tömeges email küldés folyamata*

## 3.2. Felület



*23. ábra: UI folyamatábra*

* **Bejelentkezés**: Ezen a felületen láthatjuk a felhasználókat, illetve újakat hozhatunk létre. Ha felhasználó fiókja jelszóval védett akkor, meg kell adni a jelszót az adatok feloldásához és bejelentkezéshez. Elfelejtett jelszó esetén csak törölhető a fiók.
* **Főmenü**: A fiók kezdőlapján az e-mail küldési feladatok és állapotaik láthatók. Ezen a felületen kiválasztható egy e-mail küldési feladat vagy készíthető egy új is. Továbbá erről a felületről navigálhatunk vissza a kezdőképernyőre, illetve elérhetjük a profil és szerver beállításokat. A feladat kiválasztása után menüpontokon keresztül állíthatjuk be a különböző beállításokat.
* **Felhasználói beállítások**: Ezen a felületen megváltoztatható a felhasználó neve, illetve módosítható a titkosítás jelszava.
* **Küldő szerverek**: A küldést végző e-mail címek listáját lehet itt kezelni. Meg kell adni az SMTP szerver helyét, bejelentkezés felhasználónevét és jelszavát, küldési gyakoriságot, illetve opcionálisan egy feladó e-mail-t is.
* **Küldők társítása**: A feladathoz kapcsolat küldő email címet lehet kezelni, lehetőség van átnavigálni a szerverek kezeléséhez is.
* **Címzettek beállítása**: Címzettek listája vesszővel elválasztott értékekkel (CSV) adható meg. Egy fejléc opció is bekapcsolható lesz, így a levélben személyre szabott paramétereket is be lehet állítani pl.: név, titulus stb.
* **Levél szerkesztése**: Ebben a felületben lehetőség van a levél tartalmának szerkesztésére. Levelet egy egyszerűsített szövegszerkesztőben lehet szerkeszteni. Lehetőség lesz a szöveg formázására képek vagy csatolmányok beszúrására.
* **Feladat állapota**: Ezen a felületen lesz látható a feladat állapota (jelenleg fut-e vagy befejezett-e), naplózása, továbbá ezen a felületen indítható és állítható le a folyamat.

## 3.3. Hiányzó elemek

A program elkészítéséhez a szerkesztő nem biztosít minden funkciót:

* lokalizáció,
* titkosítás,
* e-mail küldés,
* szövegszerkesztés,
* CSV formátumkezelés

funkciók teljesen hiányoznak a szerkesztőből, ezen funkciókat fogom megvalósítani a szakdolgozat keretein belül.

# 4. Kiegészítők elkészítése

A szerkesztőhöz különböző kiegészítőket telepíthetünk. A jelenlegi projekt hiányzó funkcióit nem közvetlen a programra szabva fejlesztem ki, hanem általánosabb funkciókat készítek kiegészítők formájában. Ily módon ezek a funkciók más programban is használhatók lesznek. A Construct 3 kiegészítőket JavaScript [27] nyelven kell írni (természetesen további programozási nyelvek és technológiák használata lehetséges).

## 4.1. Fejlesztői mód

Ahhoz, hogy fejlesztés alatt álló kiegészítőket tudjunk telepíteni a szerkesztőhöz, be kell kapcsolnunk a Construct fejlesztői módját. A *Menu* (*Menü*) > *Settings* (*Beállítások*) ablak felső szélére történő négy darab bal egér kattintással megjeleníthetjük az *Enable developer mode* (*Fejlesztői mód engedélyezése*) jelölőnégyzetet, ezt bejelölve a fejlesztői módot bekapcsolásra kerül. A fejlesztői módban telepített kiegészítők minden indításkor újra betöltődnek, így a legutolsó módosítás is azonnal tesztelhető.



*24. Ábra: Fejlesztői mód bekapcsolása*

A biztonsági szabályok miatt fejlesztői bővítményeket csak a „https://localhost/” url címekről lehet betölteni. Jelenleg egy saját Electron környezetben megírt és előre konfigurált webszervert (EDS - Easy Development Server) használok a fejlesztői bővítmények könnyebb betöltéséhez, de használható Xampp [28] vagy egyéb más webszerver is.

Az Easy Development Server egy titkosított HTTP [29] szervert futtat gyorsítótárazás nélkül. A programban a következő négy mező állítható be:

* Bővítmény útvonala: kiválasztható a bővítményt tartalmazó mappa, ezt a mappát fogja a program elérhetővé tenni a „https://localhost/” url-en.
* Port: megadja azt, hogy a localhost cím melyik portján lehessen elérni a mappát
* Privát kulcs: titkosításhoz használt privát kulcs
* Tanúsítvány: titkosítás hitelesítéshez használt tanúsítvány. Ezt az operációs rendszerre külön is fel kell telepíteni, hogy a titkosítást biztonságosnak ítélje meg a böngésző.

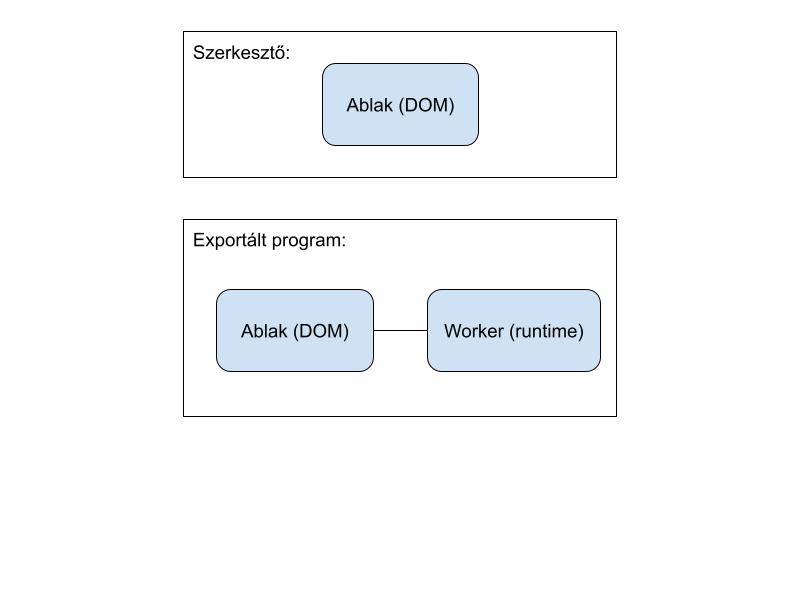
A Start gomb megnyomása után a program kiír egy URL-t, ahol a kiegészítő elérhető.



*25. Ábra: Easy Development Server működés közben*

## 4.2. JavaScript környezet

JavaScript kód különböző környezetekben fut. A szerkesztőben futtatott JavaScript kód hozzáfér a „window” objektumhoz, viszont az elkészített programban a kiegészítő kódja külön szálon fog futni. Az exportált programban a fő szál és a megnyitott szálak egy csatornán kommunikálnak egymással.



*26. Ábra: JavaScript kontextus különbözik a szerkesztőben és az exportált programban*

Az elkészített JavaScript kódnak továbbá támogatnia kell a Google Closure Compiler által biztosított egyszerű és összetett tömörítést is [30]. Vannak egyéb Construct direktívák a JavaScript használatra és formázására, amikhez a fejlesztés során tartom magam:

* Szigorú mód használata („use strict”).
* Hatáskör (scope) korlátozása kapcsos zárójelekkel: {}
* Deklaráláskor const, let használata a hatáskör jól behatárolhatóságához
* Prototípusok és tulajdonságok elnevezése szóközök nélkül, szavak nagy kezdőbetűvel: PrototipusMetodus
* Belső, saját használatú objektumok prototípusa alsó kötőjellel és nagybetűs kezdéssel: \_PrototipusMetodus
* Javascript változók neve első betűje kicsi, egybeírandó és a szavak nagybetűvel kezdődnek: peldaValtozo
* Küldésre szánt objektumok kulcs nevei kisbetűvel, kötőjellel elválasztottak legyenek: pelda-valtozo
* Nem tisztázott esetekben a Google Stílus Útmutatója [31] az irányadó

## 4.3. Kiegészítők felépítése

Construct 3 kiegészítő egy tömörített zip fájl .c3addon kiterjesztés elnevezéssel. Következő hat fajta kiegészítőt lehet készíteni a szerkesztőhöz:

* Bővítmény: Ezek a projekt építőelemei. Ezek a modulok bármilyen funkciót megvalósíthatnak. A bővítményeknek háromféle típusa létezik:
  + **Látható és példányosítható bővítmény**: Ezek a bővítmények az elrendezésen láthatóvá válnak és bármennyi példány készíthető belőlük. Például: Szöveg, Gomb, Videó
  + **Nem látható és példányosítható bővítmény**: Olyan bővítmények, melyek az elrendezésen nem láthatóak (mert nincs értelmük), de különböző példányban létrehozhatjuk őket. A szerkesztő alapértelmezetten globális objektumként kezeli. Például: Adatszerkezetek (szótár, tömb, bináris tömb, json objektum, xml objektum)
  + **Nem látható és nem példányosítható bővítmény**: ilyen típusú bővítmények nem kerülnek megjelenítésre és nem is hozhatunk belőlük létre több másolatot. Ezeket olyan funkciók implementálására használják, ahol több példány létrehozása nem megoldható, nem vezet konzisztens eredményhez. Például: egér bevitel, érintés bevitel, asztali platform funkciók kezelése, tárhelykezelés. Ezeket a bővítményeket globálisan hozzáférhetőek minden eseménylap számára.
* **Viselkedés**: Ezek bővítményekhez társítható további funkciók. Olyan modulok, amelyek önmagukban nem használhatóak (pl.: mozgás), de egy bővítmény példányára helyezve azt kezelni tudja.
* **Effetek**: A viselkedésekhez hasonlóan önmagukban az effektek sem használhatóak. Ezek kizárólag látható példányokra, rétegekre, elrendezésre helyezhetők és vizuális módosításokat hajtanak végre.
* **Téma**: A szerkesztő kinézetét módosítja.

A kiegészítőnek előre meghatározott fájlokat kell tartalmaznia [32]. A fájlok rendeltetés szerint két részre osztható: szerkesztői és futási környezeti fájlok.

### 4.3.1. Szerkesztői fájlok

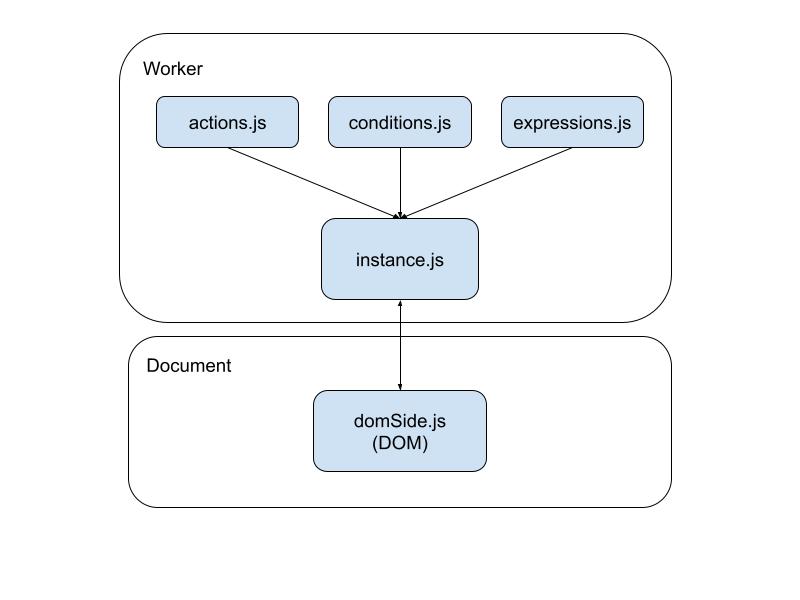
A szerkesztő számára tartalmaznak adatokat, az exportált programba nem fognak szerepelni ezek a funkciók.

* **./addon.json**: Bővítmény meta adatait tartalmazza pl. név, szerző, fájlok, leírást stb.
* **./icon.svg**: Bővítmény ikonja
* .**/aces.json**: Bővítmény műveleteit, feltételeit és kifejezését leíró fájl.
* **./lang/en-US.json**: Bővítmény szövegeit tartalmazza a megadott nyelven pl. magyar hu-HU.json.
* **./plugin.js** vagy **./behavior.js** - Szerkesztőben történő létrehozást végző szkript.
* **./instance.js**: Bővítményből létrehozott példányra vonatkozó szkript.
* **./type.js**: Bővítmény típusát beállító szkript.

### 4.3.2. Futási környezeti fájlok

Az exportált programban a funkciókat megvalósító szkripteket tartalmazza.

* **./c3runtime/actions.js**: A művelet szkripteket tartalmazó fájl.
* **./c3runtime/conditions.js**: A feltétel szkripteket tartalmazó fájl.
* **./c3runtime/expressions.js**: A kifejezés szkripteket tartalmazó fájl.
* **./c3runtime/plugin.js** vagy **./c3runtime/behavior.js**: A bővítmény létrehozását tartalmazó szkript.
* **./c3runtime/type.js**: A bővítmény típusazonosítását tartalmazó szkript.
* **./c3runtime/instance.js**: A bővítmény példányára vonatkozó kódok.
* **./c3runtime/domSide.js**: Az fő szálon futtatandó szkript kódját tartalmazza.



*27. Ábra: Bővítmény fájlok viszonya és JavaScript kontextusa*

## 4.4. Lokalizáció bővítmény

A programok különböző nyelveken történő megjelenítése manapság alapkövetelmény. Különböző nyelvek betöltéséhez a legegyszerűbb módszer az, ha a szövegeket egy kulcs-érték párban tároljuk és megjelenítéskor a megfelelő értékeket kiolvassuk. Ez azonban sok szöveggel rendelkező program esetén nehézkesen kezelhető, a különböző programrészek nehezen kapcsolhatók össze. A Constructban részleges megoldást kínálnak a külső felek által készített [33] kiegészítők. Ezek kevés rugalmasságot biztosítanak, nyelv felismerése és kezelése sincsen implementálva (kizárólag a betöltést könnyítik meg). A saját kiegészítőm szöveget, nyelvet és paraméterezett szövegeket is fog kezelni, továbbá különböző tárolási módszereket is támogat.

A kiegészítő a következő tárolási formátumokat fogja támogatni:

* JSON (single): JSON formátum, ahol az adatszerkezet értéke a fordítás szövege. Betöltéskor meg kell adni a forrás nyelvét.
* JSON (multiple): JSON formátum, ahol a adatszerkezet értéke egy olyan objektum, ahol nyelvenként fel vannak sorolva a különböző fordítások.
* CSV: szabványos vesszővel elválasztott értékek, ahol az első sor (fejléc) azonosítja a különböző nyelvek fordításait.
* Construct 3 szótár: szerkesztőben létrehozott szótár formátum. Betöltéskor meg kell adni a forrás nyelvét.
* Construct 3 tömb: szerkesztőben létrehozott tömb formátum, ahol a fejléc (első sor) azonosítja a különböző nyelvek fordításait.

### 4.4.1. Szerkesztő

A kiegészítő egy nem látható és nem példányosítható bővítmény lesz, vagyis a projekten belül bárhol elérhető. A bővítmény tulajdonságai között beállítható az alapértelmezett nyelv és a elérhető nyelvek listája de ezek a program futása során is módosíthatók. A szerkesztő felület biztosít egy konvertálási funkciót is a különböző nyelvi formátumok között.

Alapértelmezett SDK példa [34] felépítését követtem. További konfigurációt a bemenetek ellenőrzése és a konvertálási funkciók igényeltek.

A szerkesztőben megadható kezdeti beállításokat az „OnPropertyChanged” prototípus kezeli. Feladata, hogy a felhasználó csak érvényes nyelvi beállításokat tudjon megadni a tulajdonságok beállításánál pl. új sorba rakja a nyelveket, nem enged olyan kezdeti nyelvet beállítani, ami nincs benne projekt által támogatott nyelvek listájába.

A másik fontos funkciót a konvertálást a „\_Convert” prototípus végzi. A konvertálás szintén a tulajdonságok között elérhető. A konvertálás lefutásakor a függvény kiolvassa a bemeneti adatokat és a kimeneti beállításokat, majd a kimenet mezőbe beírja az átkonvertált szöveg adatot.

### 4.4.2. Futási környezet

Alapértelmezett felépítést használom. Ezt bővítem egy szkripttel ami a DOM oldalon fog futni és az eszköz nyelvi beállításait teszi elérhetővé. A bővítmény továbbá biztosít egy globális interfészt, ahol a lokalizáció viselkedés kommunikálni tud a kiegészítővel. A nyelvi értékek (szövegek) változtatása a felület indulásakor, fordítás megváltozásakor, illetve nyelvváltáskor történik. A különböző nyelv

## 4.5. Lokalizáció viselkedés

A különböző objektumok kényelmes fordításához (szöveg, alakzat, gomb stb) a bővítménynek hozzá kell férni az objektumokhoz. Ezt a hozzáférést viselkedésen keresztül valósítom meg. A lokalizáció bővítmény kezeli a lokalizációt, ha hiányzik a projektből a lokalizáció viselkedés se fog működni.

### 4.5.1. Szerkesztő

A szerkesztőben a viselkedést minden objektumhoz egyszer lehet társítani (több különböző szöveg megjelenítése egy objektumon belül értelmetlen lenne). Ki és be lehet kapcsolni a bővítmény működését, ez hasznos, ha nem akarunk fordítást végezni ideiglenesen az adott objektumon. Lehetőség van alapértelmezett fordítási kulcs beállítására, így nem kell külön esemény lapon beállítani a kulcsokat az objektumra. Továbbá a megjelenítendő nyelvet is be lehet állítani, a beállítandó nyelv kódjával. Ez felülírja a globális nyelvbeállítást az objektum fordításakor.

### 4.5.2. Futási környezet

Egy objektumba tárolja a különböző társított elemeket. A társított objektumok létrehozásakor lemásolja az objektum mutatóját, az objektum törlésekor, pedig eltávolítja a mutatót az adatszerkezetből.

## 4.6. Titkosítás

A kiegészítő feladata különböző titkosítási algoritmusok alapján adatokat titkosítani, továbbá a titkosítást kiegészítő műveletek elvégzése (ellenőrző összeg számítása, kulcsgenerálás stb.). Elsősorban a JavaScript kriptográfiai könyvtárát illesztem a Construct 3-hoz [35].

### 4.6.1. Szerkesztő

A kiegészítő egy nem látható és nem példányosítható bővítmény lesz, így a projekten belül bárhol elérhető. Nincs további módosítási lehetőség a szerkesztőben.

### 4.6.2. Futási környezet

A bővítményt két fő részre lehet osztani: kulcskezelés és titkosítás.

A kulcsokat biztonsági okokból a felhasználó nem tudja közvetlen kezelni, illetve nem minden kulcs olvasható ki. Kulcsokat generálni vagy importálni lehet a memóriába, majd az importáláskor megadott névvel lehet hivatkozni a titkosítás végrehajtásakor.

Titkosítások három kategóriába tartoznak: ellenőrző összeg generálás, aláírás-ellenőrzés, titkosítás-feloldás. Ellenőrző összeg generáláshoz nem kell kulcs, így közvetlen használható. Aláírás és ellenőrzés algoritmusok kizárólag aszinkron titkosítást használnak. Az adat titkosító algoritmusok pedig aszinkron vagy szinkron titkosítást alkalmaznak.

## 4.7. Email küldés

Az e-mailek küldéséhez egy olyan bővítményt kell készíteni, ami egy SMTP protokollon keresztül csatlakozni tud a levelező szerverhez és leveleket tud küldeni megadott címekre. A csatlakozás böngészőben nem valósítható meg, mert nincs lehetőség DNS címfordításra, közvetlen hálózati protokoll és port használatára. Különböző platformokon külön könyvtárakat használok a kommunikáció megvalósítására. Asztali operációs rendszerekre a nodemailer nevű nodejs könyvtárat használom, mobilokon pedig a mailcore2 könyvtár Java és Objective-C alapú könyvtárát integrálom a bővítménybe.

### 4.7.1. Szerkesztő

Szerkesztőben egy nem példányosítható globális bővítmény formájában adható hozzá. A bővítmény belépési adatokat nem tárol, a csatolmányokat pedig minden küldés előtt hozzá kell adni a küldés előtt, így a szerkesztőben nem kell semmit konfigurálni.

### 4.7.2. Futási környezet

A bővítmény a különböző platformokat „Duck typing” [36\_1] módszerrel ellenőrizni, például az asztali környezetben lévő függvények elérhetőek, akkor a program feltételezi, hogy asztali környezethez tartozó függvényeket kell használni.

Asztali exportáláskor a nodemailer könyvtárat közvetlen be tudom illeszteni a kiegészítőbe, azonban a sok különálló részegységet és hivatkozást a Rollup.js [36] eszköz segítségével tömörítettem kell egy fájlba, hogy könnyen kezelhető és hivatkozható lesz.

Mobil eszközökön Cordova kiegészítő formájában tudom integrálni a Mailcore2 [36\_2] könyvtárat, amit npm könyvtár megosztóra feltöltök, majd a kiegészítőben hivatkozok az elérési útvonalára. Androidon Java nyelven, iOS platformon Objective-C nyelven készítettem egy függvényt, aminek szám, logikai vagy szöveg típusú paramétereket lehet átadni, az email küldés pedig a platform natív nyelvén hajtódik végre.

Az email küldés során a felhasználó általában HTML üzenetet fog küldeni, ami tartalmazhat nagyobb beágyazott adatot például képet Base64 [36] kódolásban. Ezeket a beágyazásokat a modern rendszerek biztonsági okokból automatikusan tiltják. Megoldásként a bővítmény az üzenet megkeresi reguláris kifejezéssel az üzenetben szereplő „src” attribútumokat, majd az adatokat külön Content-ID-ként [38] csatolja az email-hez és az src ben lévő adatot kicseréli a létrehozott Content-ID hivatkozásra. Csatolmányok csatolásakor is lehetőség van Content-ID létrehozására, így a levélben közvetlen hivatkozni a csatolmányra például egy képre.

## 4.8. Szövegszerkesztés

Az üzenet szövegének megszerkesztéséhez szükséges egy olyan bemeneti szövegmező, ami egyszerű formázási lehetőségeket kínál a felhasználó számára, mint pl. betűméret állítás, szöveg igazítás, félkövér és dőlt betű beállítás, indexelés stb. Ezen funkciók implementálásához egy bővítményt fogok készíteni, ami egy ingyenes szövegszerkesztő könyvtárat fogok használni a Quill-t [39].

### 4.8.1. Szerkesztő

A kiegészítő egy látható és példányosítható bővítmény lesz. A felhasználó az elrendezésen el tudja helyezni és méretezni is tudja a szövegmezőt. Szerkesztőben csak egy dobozként lehet megjeleníteni a mezőt, mivel a szerkesztő WebGL alapú felülete HTML elemek megjelenítésére nem alkalmas. Beállítható továbbá a szövegmező kezdeti szövege, helyőrző szövege, szerkeszthetősége és láthatósága is. A vázlatszerű megjelenítést a Draw függvény végzi el.

### 4.8.2. Futási környezet

A kiegészítőben kulcsfontosságú szerepet kap a plugin.js fájlban található SDKDOMPluginBase osztály. Feladata, hogy minden létrehozott példányhoz tartozó Worker és DOM oldali objektumot összekapcsolja.

Létrehozáskor a DOM oldalon egy div elemben hozza létre a Quill könyvtár a szerkesztési felületet. A DOM oldal csak a szöveg változtatáskor küldi el a szövegmező értékét a Runtime oldalnak. Ha a Runtime oldalon történik változás akkor GetElementState függvény által generált állapot értékeket kapja meg a DOM UpdateState függvénye.

## 4.9. CSV formátum kezelés

Táblázatos adatok legáltalánosabb tárolási formátuma a CSV formátum (RFC4180 szabvány [40]). A táblázatos adatok betöltésére legkézenfekvőbb ez a formátum, mert szinte minden platform és program támogatja. A bővítmény lehetővé tesz az táblázatos adatok betöltését és mentését CSV szövegként, JSON szövegként és tömb objektumként.

### 4.9.1. Szerkesztő

Szerkesztőben egy nem példányosítható globális bővítmény formájában adható hozzá. Példányosítás azért van letiltva, mert nem adattárolásra, hanem konvertálásra van optimalizálva a kiegészítő és az adatok kezelésére már rendelkezésre állnak beépített kiegészítők is.

### 4.9.2. Futási környezet

Az betöltött adatok 2 dimenziós tömbben tárolódnak. A CSV feldolgozás reguláris kifejezésekkel történik. A felhasználó betöltéskor bármilyen elválasztó karaktert megadhat azonban az üres karaktert, dupla idézőjelet és újsor karaktert a rendszer automatikusan vesszőre cseréli. A betöltött adatok már csak tényleges adatokat tartalmazza feloldójelek nélkül és transzponálni, törölni és olvasni lehet őket.

# 5. Tömeges email küldő elkészítése

Ebben a fejezetben tömeges email küldő program főbb felépítését és elkészítését mutatom be. A szerkesztés során a szerkesztő által biztosított visual scripting eszközöket próbáltam kihasználni. Cél a saját és beépített kiegészítők visual scripting használhatóságának vizsgálata, szoftverfejlesztésben használt „dogfooding” („kutya etetés”) [41] módszer alkalmazása, vagyis a külön elkészített bővítmények egy gyakorlatban való alkalmazása a hibák és bővíthetőségek feltárása érdekében.

## 5.1. Alap beállítások

A program megjelenítésének beállítását adom meg el elsőnek. A cél platformok között álló és fekvő módban működő eszközök is találhatóak (asztali rendszerek általában fekvő 16:9, mobil rendszerek álló 16:9 aránnyal rendelkeznek) ezért álló elrendezést fogom alkalmazni a programra, a fekvő nézettel való kompatibilitást pedig automatikus helykitöltéssel oldom meg.

Elsőként egy betöltést szolgáló eseménylapot és elrendezés hozok létre „Loading” néven. A keretrendszer betöltése után ez felület fog lefutni elsőként. Feladata, hogy a rendszer alap változóit és beállításait létrehozza. A fő elrendezést és eseménylapot „Program” néven hozom létre. A felhasználó interakcióit ez a felület fogja kezelni, ez lesz maga a program. A különböző felületek rétegként adom az elrendezéshez és váltáskor a megfelelő rétegeket jeleníti és rejti el a program. Ezeken felül létrehozok egy „Global” nevű esemény lapot elrendezés nélkül. Ez az eseménylap a „Program” felületnek fog különböző funkciókat biztosítani, így a felépítés hasonlít egy klasszikus frontend és backend felosztásra, ahol a „Program” eseménylap és elrendezés a frontend rész, míg a „Global” eseménylap a backend rész. A „Loading” eseménylap és elrendezés backend és frontend feladatot is ellát, ezt a betöltés speciális helyzete miatt készítettem így. Az említett eseménylapokon felül létrehoztam egy „main.js” szkriptet is, ami rendszer azon korlátozását kerüli meg, hogy a visual szkriptingben írt időnkénti futtatás szünetel, ha nincs fókuszban az ablak.

## 5.2. Adatszerkezet

Az program futása során a fontosabb adatokat egy JSON objektumban tárolja a program. Az adatszerkezet hierarchikusan épül fel a fő elemek tartalmazhatnak al-elemeket. A legfelső (gyökér) részen található a helyi beállítások („settings”), ideiglenes adatok („temp”) és a felhasználók („users”).

### 5.2.1. Ideiglenes adatok

Az ideiglenes adatok nem kerülnek mentésre, az adatok futás közben jönnek létre vagy törlődnek az adatszerkezetből. Az ideiglenes adatok objektumban a következő tulajdonságok találhatóak:

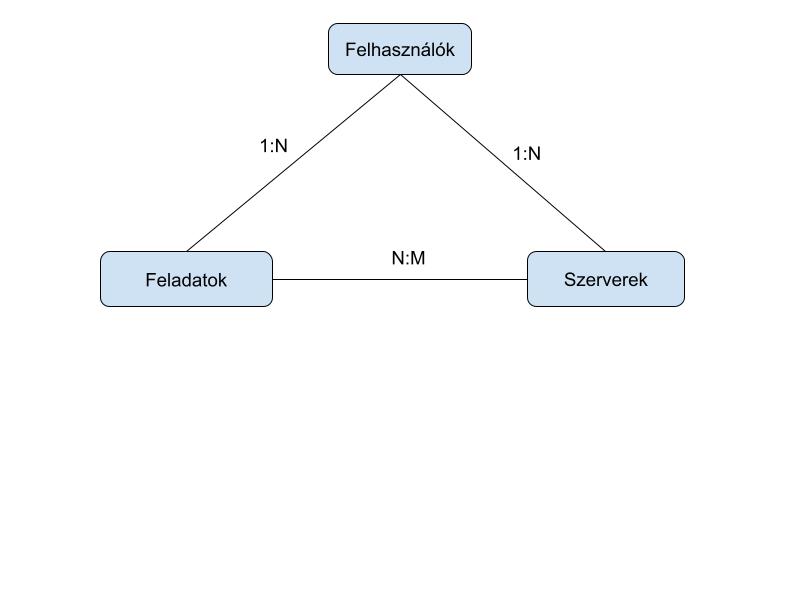
* zárolt egyedi azonosítók („lockedUIDs”) tömb: a már generált, de fel nem használt egyedi azonosítók tömbje. Ebben a tömbben lévő azonosítókat már nem fogja generálni az azonosító generátor függvény, így több azonosítható is generálható ideiglenesen.
* küldés csoportosítása („sending“): aktív és küldési feladatok listáját tartalmazza, a felépítéséről a későbbiekben lesz szó.

### 5.2.2. Helyi beállítások

A helyi beállítások rész három értéket tartalmaz, az utolsónak kiválasztott felhasználó („lastUserUID”), szerver („lastServerUID”) és feladat azonosítóját („lastTaskUID”). Ez a három adat közvetlenül kerül mentésre és betöltésre „settings” néven.

### 5.2.2. Felhasználók

A „users” tulajdonság egy tömb, ami tárolja a felhasználókat, valamint a hozzájuk tartozó szervereket és feladatokat. A felhasználók és feladatok, valamint a felhasználók és szerverek között „egy a többhöz” kapcsolat áll fenn (egy felhasználóhoz több szerver és feladat is tartozhat). Feladatok és szerverek között viszont „több a többhöz” kapcsolatban állnak egymással mert több feladathoz több szerver is tartozhat. A programban azonban a felhasználók között nincs szerver és feladat megosztás, vagyis egy felhasználó feladata nem használhatja a másik felhasználó szervereit.



*28. Ábra: Felhasználók, szerverek és feladatok viszonya*

Egy felhasználó objektum a fejléc („header”) ideiglenes („temp”) és adat („data”) részre osztható. A fejléc („header” objektum) tartalmazza:

* egyedi azonosítót („UID”): ez egy globálisan egyedi azonosító szövegként tárolva. A generált azonosítót a program nem változtatja meg a későbbiekben.
* nevet („name”): a felhasználó neve szövegként van eltárolva, ez a felhasználó számára teszi könnyen megkülönböztethetővé a fiókokat bejelentkezéskor.
* titkosítás meglétét („isCrypted”): nulla vagy egy számként tárolva. Nulla jelzi, ha a felhasználó nem használ titkosítást, az egy jelöli ha a felhasználó titkosította az adatait.
* van-e futtatandó feladata („isNeedRun”): nulla vagy egy számként tárolva. Egy jelzi, ha a felhasználónak futtatandó folyamata van. Ez szükséges a feloldási kérelm jelzéséhez, mert ha az adatok titkosítottak nem állapítható meg hogy a folyamatnak futnia kellene-e.
* adat azonosító („dataUID”): globálisan egyedi azonosító, ami a felhasználó adat részének mentési helyét jelzi.
* lefoglalt azonosítók („storageUIDs”): egy tömb melyben felhasználó által lefoglalt helyek azonosítói találhatóak. Törlés esetén a titkosított adatokból nem lehet megállapítani, hogy mely tárolt adatokat kellene törölni, így titkosítatlanul fel van sorolva az összes felhasználó hoz köthető adat tárolási helye.
* adat („data”): egy objektum, amely a felhasználói adatok feloldása és betöltése után jön létre. A szervereket „servers” nevű tömb tárolja, a feladatokat pedig a „tasks” nevű tömb.

A felhasználóra vonatkozó ideiglenes adat („temp”) objektum tartalmazza az ideiglenesen tárolt adatokat, ez a program bezárásakor nem kerül mentésre. A felhasználóra vonatkozó ideiglenes adat tartalma:

* titkosítás kulcsa („key”): ha a felhasználó feloldotta a titkosítást akkor a kulcs ideiglenesen eltárolódik, ez gyorsítja az ellenőrzést. Mivel a memóriába már be vannak töltve az adatok így plusz kockázatot nem jelent a jelszó tárolása.
* zárolás („isLocked”): nulla vagy egy számként tárolva. Az egyes érték jelzi a felhasználó adatai zárolva vannak-e egy hosszabb művelet például mentés miatt.
* utolsó művelet ideje („lastAction”): időbélyeg [42] pozitív egész szám ként tárolva, ez a memóriakezelőnek jelzi az utolsó olvasás vagy írás idejét.

Az felhasználó adat objektum tartalmazza a szervereket és a feladatokat. Ezek jelszó használata esetén titkosítva vannak a merevlemezen. Ezek tartalmát csak feloldás után tudja olvasni a program.

### 5.2.3. Szerverek

A szerverek adatai a felhasználó adat objektumán belül található „servers” tömbön belül találhatók. Egy szervernek az alábbi tulajdonságai vannak:

* egyedi azonosító („UID”): a felhasználón belül a szerver egyedi azonosítója. Mivel kívülről nem elérhető mások számára a szerver így csak a felhasználón belül egyedi az azonosítója.
* szerver elérése („host”): az SMTP szerver URL-je, amelyen segítségével a program csatlakozni tud a szerverhez.
* felhasználó („user”): a felhasználónév (alapértelmezetten email cím) a szerverhez történő belépéshez. Ez az email cím alapértelmezetten a küldő email cím is.
* jelszó („pass”): az email fiókhoz tartozó jelszó, amivel a program azonosítja magát a szerveren.
* küldő („from”): ha a küldő email címnek hozzáférése van egy mások email címhez a szerveren akkor a küldő email cím nevében fogja kiküldeni a program a leveleket
* küldési gyakoriság („duration”): nullánál nagyobb egész szám, ami két levél küldése közötti várakozási időt tárolja ezred másodpercben. Ez a beállítás megakadályozza, hogy a program túl sok levelet küldjön ki a szerverrel.

### 5.2.4. Feladatok

A feladatok adatai a felhasználó adat objektumán belül található tömbben („tasks”) találhatók. Egy feladat rendelkezik egy fejléc („header”) objektummal, egy ideiglenes adatokat tároló objektummal („temp”) és egy adat objektummal („data”).

A feladat fejléce az alábbi tulajdonságokkal rendelkezik:

* egyedi azonosító („UID”): a felhasználón belül a feladet egyedi azonosítója. Mivel kívülről nem elérhető mások számára a feladat így csak a felhasználón belül egyedi az azonosítója.
* név („name”): a feladat neve, amely a felhasználó számára könnyen megkülönböztethetővé teszi a feladatokat.
* feladat állapota („state”): a feladat állapotát jelző szöveg. Ez lehet befejezett („finished”), futás alatt álló („running”) vagy szüneteltetett („paused”). Újonnan létrehozott szerver automatikus befejezett állapotú lesz.
* csatolt szerverek („serverUIDs”): egy tömb, amely tartalmazza a feladathoz csatolt szervereket, amelyekről az emaileket ki kell küldeni.
* levél adatainak azonosítója („letterUID”, „attachmentsUID”, „toUID”, „toSuccessUID”, „toFailedUID”): az feladat tartalmi adatainak globális azonosítói. A feladathoz tartozó a adatokat külön, kisebb részekben menti a rendszer, így módosításkor nem kell mindent adatot újra kiírni. Az egyedi azonosító alapján betölthetőek az feladat adatai a háttértárról.
* levél adatai („data”): egy objektum, amely a feladat további adatainak feloldása és betöltése után jön létre. Ez a email tartalmi részeit, csatolmányokat és címzetteket tartalmazza.

A feladatokra vonatkozó ideiglenes adat („temp”) objektum tartalmazza, ez a program bezárásakor nem kerül mentésre:

* zárolás („isLocked”): nulla vagy egy számként tárolva. Az egyes érték jelzi, hogy a feladat adatai zárolva vannak-e egy hosszabb művelet például mentés miatt.
* utolsó művelet ideje („lastAction”): időbélyeg pozitív egész szám ként tárolva, ez a memóriakezelőnek jelzi az utolsó olvasás vagy írás idejét.
* jelenleg futó műveletek száma („runCount”): ez a tulajdonság jelöli, hogy jelenleg mennyi megkezdett, le nem zárt küldés van folyamatban.

A feladat adat („data”) objektuma nagy mérete miatt külön kerül betöltésre és a következő tulajdonságokat tartalmazza:

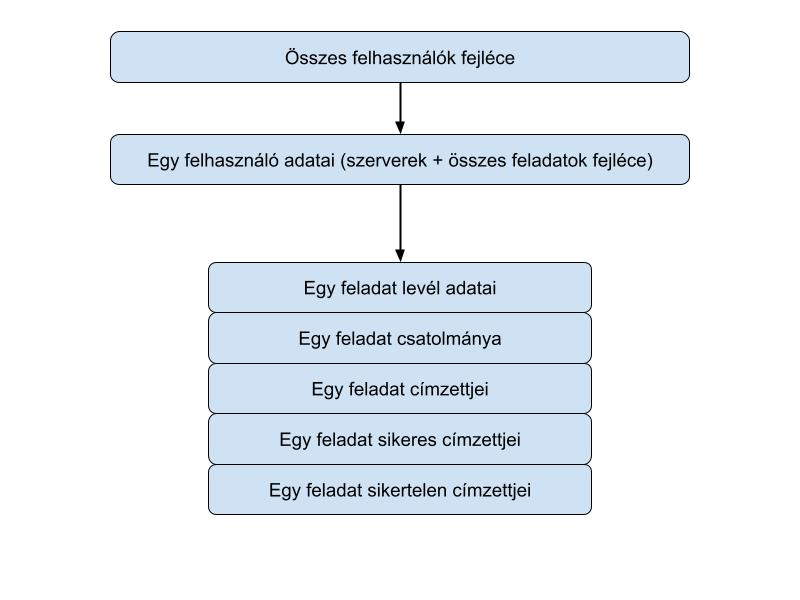
* levél objektum („letter”): email tartalmi adatait tartalmazó objektum. A következő tulajdonságokat tartalmazza az objektum:
  + prioritás („priority”): email fejlécében szereplő prioritás szintje, értéke „0”, „1” vagy „2” érték lehet.
  + tárgy („subject”): email tárgyának szövege.
  + fejlécek használata („useHeaders”): címzettek listája tartalmaz-e fejléceket. A levélben és a tárgy mezőben is lehet kapcsos zárójelek között hivatkozni a címzettek oszlopaira, fejléc használata esetén oszlop neveire, így paraméterezhető lesz az elküldött üzenek például: „{név}” küldéskor a „név” oszlopban szereplő értéket helyettesíti be.
  + elválasztó („separator”): a betöltött címzett adatok határoló karaktere.
  + szöveg („text”): küldendő email szövege.
* csatolmányok tömbje („attachments”): tömb melyben a küldendő csatolmányok találhatóak. Egy csatolmánynak a következő tulajdonságai vannak:
  + név („name”): a csatolmány neve kiterjesztéssel együtt.
  + típus („type”): a csatomány Multipurpose Internet Mail Extensions (MIME) [43] típusa.
  + adat („data”): a csatolmány tartalma Base64 kódolású szövegként.
* címzettek tömbje („to”): a címzetteket tartalmazó tömb, azonban minden címzett egy másik tömbként van letárolva, így a címzettek tömb első dimenziói a címzettek, második dimenziója a címzettek paraméterei.
* sikeresen kiküldött címzettek tömbje („toSuccess”): a sikeresen kiküldött címzettek kerülnek tárolásra.
* sikertelenül kiküldött címzettek tömbje („toFailed”): a sikertelenül küldések címeit tartalmazza a tömb.

## 5.3. Betöltés és mentés

Az memóriába betöltött adatszerkezetet nem lehet egy az egyben lementeni a háttértárra. A adatszerkezet túl nagy ahhoz hogy minden változtatáskor az egészet újra mentsük és bizonyos részei ideiglenesek, vagy titkosítani kell őket. A beépített bővítmény kulcs-érték párokban tud szöveges adatokat menteni a háttértárra ezért a elkészített program külön részegységenként menti a változtatásokat.

Elsőként a titkosítatlanul tárolt helyi beállítások töltődnek be a „settings” helyről, majd az összes felhasználó fejléc adatai szintén titkosítatlanul „all-user-header” helyről. A beállítások és a felhasználók fejléc adatai a futás során soha nem kerülnek törlésre a memóriából. Ez a program kiindulópontja, ezekből az adatokból tudja betölteni az összes többi adatot a program.

Ha egy felhasználót be akarunk tölteni akkor a program a fejlécből kiolvasott adat egyedi azonosítója („dataUID”) alapján keresi ki a megfelelő kulcs érték párt a háttértáron „storage-” és azonosító összefűzésével keletkező azonosítóval. Ha a háttértáron lévő adat titkosított, akkor fel kell oldania a programnak a titkosítást. A program titkosításhoz AES-CBC [44] algoritmust használ 256 bit hosszúsággal és 10000 iterációval, a kulcsot pedig PBKDF2 [45] használatával képzi, így a rövid jelszavakból is megfelelő hosszúságú kulcsot generál a program AES-CBC algoritmus számára. A felhasználói adat a szerverek adatait, illetve a felhasználó összes feladatának fejlécét tartalmazza, a feladat tartalmi adatai a nagy méret miatt külön kerülnek betöltésre és mentésre.



*29. Ábra: Betöltési folyamat részletei*

A program memória kezelő része automatikusan törli a ritkán használt adatokat a memóriából. Egy feladat akkor üríthető a memóriából, ha nem fut, nincs zárolva és megadott idő alatt nem volt használva (jelenleg húsz másodperc). Egy felhasználó adatai pedig akkor kerülnek ürítésre, ha egy feladatára sincs szükség és a megadott idő alatt nem volt használva (jelenleg húsz másodperc).

Az összes felhasználó fejléce a program elején egyszer kerül betöltésre és nem is törlődik a futás során így arra nem hoztam létre külön metódust. Egy felhasználóhoz tartozó adatok betöltését „BLoginUser” metódus végzi, paraméterként a felhasználó egyedi azonosítóját és jelszavát kell megadni. A metódus „not-found-user” hibát dob, ha nem található a felhasználó, illetve „auth” hibát, ha nem megfelelő a jelszó. Egy feladat adatainak betöltést „BLoginTask” metódus végzi, paraméterként a felhasználó egyedi azonosítóját, jelszavát és a feladat egyedi azonosítóját kell megadni. A metódus „not-found-user” hibát dob, ha nem található a felhasználó, „auth” hibát, ha nem megfelelő a jelszó, illetve „not-found-task” hibát ad ha nem található a feladat. A feladat betöltés a felhasználó betöltését is elvégzi így nem szükséges a felhasználó adatait előtte betölteni.

A összes felhasználó fejlécét egybe a „BSave” metódus menti. Nincsen bemeneti paramétere és kimeneti eredménye sem. A felhasználó fejléc részeit egy tömbe másolja majd JSON szövegként elmenti „all-user-header” helyre. Egy felhasználóhoz tartozó adatokat (szervereket és feladatok fejléceit) a „BSaveUser” metódus menti. Bemeneti paraméterként a felhasználó egyedi azonosítóját és jelszavát kell megadni. A metódus „not-found-user” hibát dob, ha nem található a felhasználó, „auth” hibát, ha nem megfelelő a jelszó, illetve „not-loaded” hibát, ha felhasználó adatai még nem töltődtek be (nem található az adat objektum). Feladat metését a „BSaveTask” metódus végzi, bemeneti paraméterként a felhasználó egyedi azonosítóját, jelszavát, feladata azonosítóját és a feladat mentendő részét kell megadni. A mentendő rész lehet az összes rész („all”), vagy a feladathoz tartozó részegységek: levél („letter”), csatolmányok („attachments”), levél („all”), címzettek („to”), sikeres címzettek („toSuccess”), sikertelen címzettek („toFailed”). A metódus „not-found-user” hibát dob, ha nem található a felhasználó, „auth” hibát, ha nem megfelelő a jelszó, illetve „not-loaded” hibát, ha feladat adatai még nem töltődtek be (nem található az adat objektum).

## 5.4. Bejelentkezés

A felület a „ListUsers” metódust használja, ezzel le lehet kérni a felhasználók listáját egy JSON tömbben. A tömbben a felhasználók következő tulajdonságokkal rendelkeznek:

* egyedi azonosító („UID”) szövegként
* felhasználó neve („name”)
* titkosításának megléte („isCrypted”), nulla, ha nincs titkosítva, különben egy.
* bejelentkezés szükségessége („isNeedLogin”), nulla, ha nincs szükség bejelentkezésre, különben egy.

Felhasználó létrehozásához „CreateUser” metódust használja a felület. Paraméterként felhasználónév és jelszót kell átadni. A metódus „exist” hibát ad vissza, ha létezik ilyen nevű felhasználó vagy üres felhasználónév esetén „empty” hibát ad vissza a metódus. Ha sikeres a létrehozás akkor a felhasználó azonosítóját adja vissza.

Felhasználó eltávolításához „DeleteUser” metódust használja a felület. Paraméterként felhasználó azonosítóját kell megadni. A metódus „not-found-user” hibát ad vissza, ha nem létezik ilyen felhasználó. A felhasználó törlése jelszó nélkül is lehetséges, hogy jelszó elfelejtése esetén ne legyen foglalja a helyet a felhasználó.

Rögtön a program betöltése után a felhasználó a bejelentkezési felületet látja. A felület a „LoginRefreshUI” metódussal frissíthető. A frissítés során a középen elhelyezkedő legördülő listába vannak kilistázva a felhasználók. Az lista első elem fix, ezen elem kiválasztáskor megjelenik az új felhasználó létrehozásához szükséges űrlapelemek, ami a név, jelszó és jelszó ismétlő mezők. Konkrét felhasználó kiválasztásakor, ha titkosított a felhasználó, akkor egy jelszó mező jelenik meg, különben csak egy belépés gomb. A felhasználók lekért adatait egy JSON objektumba tárolja a program az későbbi kiolvasás lehetősége miatt.

Felhasználó létrehozása után automatikusan továbblép a program a főmenüben, jelszavas belépés esetén a rendszer a feladatok listázás metódus segítségével hitelesíti a felhasználót, ami a következő fejezetben lesz kifejtve. Sikeres belépés után globális változóba eltárolásra kerül a felhasználó azonosítója és a használt jelszó.

Lehetőség van nyelvváltásra is, ezután a módosítás után a le kell frissíteni a listát az első elem miatt, a többi elem szövegét a bővítmény automatikusan frissíti.

## 5.5. Főmenü

Sikeres bejelentkezés után főmenü jelenik meg. A felület a „ListTasks” metódust használja, ezzel le lehet kérni a felhasználóhoz tartozó feladatok listáját egy JSON tömbben. Paraméterben meg kell adni a felhasználó azonosítóját és jelszavát, hiányzó felhasználó esetén „not-found-user”, hibás jelszó esetén „auth” hibaértéket adja vissza. A tömbben a listázott feladatok következő tulajdonságokkal rendelkeznek:

* egyedi azonosító („UID”) szövegként
* feladat neve („name”)
* feladat állapota („state”)

Feladat létrehozása „CreateTask” metódussal lehetséget. Paraméterként felhasználó azonosítóját, jelszavát és a feladat nevét kell átadni. A metódus „exist” hibát ad vissza, ha létezik ilyen nevű feladat létezik vagy hibás feladatnév esetén „wrong-data” hibát ad vissza a metódus. Ha sikeres a létrehozás akkor a feladat egyedi azonosítóját adja vissza.

Feladat eltávolításához „DeleteTask” metódust használja a felület. Paraméterként felhasználó azonosítóját, jelszavát és a feladat azonosítóját kell megadni.

A felületet a „LoginRefreshUI” metódus frissíti. A bejelentkezési felülethez hasonlóan a feladatok itt is egy legördülő listában vannak elhelyezve. Az első elem kiválasztása a feladat létrehozásához szükséges név mezőt jeleníti meg. A főmenüből a feladat kiválasztáson túl tovább lehet navigálni a felhasználói beállításokra vagy a szerverek kezelésére is. Feladat kiválasztása esetén, ha szüneteltetett vagy elindított a feladat akkor az állapotot kezelő felület jelenik meg, ha befejezett a feladat, akkor küldő szervereket kezelő oldal nyílik meg.

## 5.6. Beállítások

A felhasználó ezen a felületen tudja módosítani a nevét és jelszavát. A bejelentkezéskor létrehozott felhasználói adatokból illeszti be a jelenlegi felhasználó nevét és jelszavát. Külön gombok vannak a jelszó és felhasználónév változtatáshoz.

A nevet a „SetUsername” metódus változtatja. Paraméterben a felhasználói adatok mellett az új felhasználónevet kell megadni. Ha nem foglalt és helyes a felhasználónév akkor megváltoztatja és tájékoztatja erről a felhasználót a felület.

Jelszóváltoztatást a „SetPassword” metódus hajtja végre. Paraméterben a felhasználói azonosító és jelszó mellett az új jelszót kell átadni. Jelszóváltoztatás során az összes felhasználóhoz köthető adatot újra kell menteni a titkosítás miatt. A metódus ilyenkor betölti az összes felhasználói feladatot a memóriába majd mindegyiken végrehajtja a mentés folyamatát. Az eredmény sikerességéről a felület tájékoztatja a felhasználót.

## 5.7. Szerverek kezelése

Ezen felületen lehet kezelni a felhasználóhoz tartozó szervereket. Az eddigiekhez hasonló módon legördülő listából kell kiválasztani egy meglévő szervert vagy az első elem kiválasztásakor újat létrehozni. Szerver kiválasztásakor a lista alatti mezőkbe kerülnek beírásra a szerver adatai. A küldési gyakoriság nincs közvetlen megadva csak a levelek közötti időkihagyás ezredmásodpercben. Az időkihagyást a program átalakítja a programot használó számára könnyebben olvasható formába, vagyis az óránkénti vagy naponkénti küldések számát mutatja a program nem pedig a várakozás idejét.

A szervereket adatait a „ListServers” függvény adja vissza. Létrehozni szervert a „CreateServer” függvénnyel lehet, sikeres létrehozáskor visszaadja a létrehozott szerver azonosítóját. A törlést a „DeleteServer” metódus hajtja végre. A módosításhoz „SetServer” metódus használható. Paraméterként a szerver adatai kell átadni JSON formátumba szövegként. Kizárólag olyan szerverek törölhetőek és módosíthatóak, amelyek nincsenek futó vagy szüneteltetett feladathoz társítva.

Ha a felületet a főmenüből nyitotta meg a felhasználó, akkor a visszafelé történő navigáció is a főmenüre történik azonban, ha egy megnyitott feladat felületéről navigált a szerverek szerkesztéséhez a felhasználó, akkor feladat felületéhez navigál vissza a program.

## 5.8. Feladat kezelése

A feladathoz tartozó adatokat a „GetTask” függvénnyel lehet olvasni és „SetTask” metódussal módosítani. Minden íráshoz és módosításkor paraméteben meg kell adni a felhasználó azonosítóját, jelszavát, a feladat azonosítóját és a feladathoz tartozó adat kategóriáját, valamint írás esetén az új adatot. Kategóriák a következők lehetnek:

* név („name”): a feladathoz megadott nevet lehet lekérni vagy módosítani.
* társított szerverek („servers”): a feladathoz társított szerverek azonosítóit adja vissza egy JSON tömbben, szöveg formátumba. Módosításkor szerver azonosítóit tartalmazó tömböt kell átadni a metódunak.
* prioritás („priority”): a küldendő levél prioritás fejlécének olvasására vagy módosítására van lehetőség.
* tárgy („subject”): a küldendő levél tárgy szövegének olvasására vagy módosítására van lehetőség.
* fejléc használata („useHeaders”): fejléc használati beállítás értékét lehet olvasni vagy módosítani.
* fejléc használata („separator”): a címzett adatok elválasztó karakterének beállított értékét lehet olvasni vagy módosítani.
* szöveg („text”): feladathoz beállított küldendő szöveget lehet olvasni vagy módosítani.
* csatolmányok („attachments”): a küldés csatolmányait tartalmazó JSON tömb. Szerkesztéshez is egy tömböt kell átadni a metódusnak, ami lecseréli az összes korábbi csatolmányt.
* címzettek („to”): címzettek listáját kétdimenziós JSON tömbként lehet olvasni vagy módosítani. Ha nincsenek címzettek egy üres tömböt ad vissza a függvény.

A következő kategóriák csak a „GetTask” függvényben érhetőek el, mivel értékük nem közvetlenül nem módosítható csak olvasható:

* állapot („state”): visszaadja a feladat állapotát
* sikeres címzettek („toSuccess”): a sikeresen kiküldött címeket adja vissza.
* sikertelen címzettek („toFailed”): a sikertelenül küldött címeket adja vissza.
* hátralévő idő („remainTime”): a feladat végrehajtásához szükséges időt adja vissza percben. Ha nincs elindítva a feladat akkor a várható időtartamot adja vissza.
* hátralévő százalék („remainPercent”): nulla és száz közötti egész szám, ami a előrehaladás mértékét adja meg.
* összes („all”): minden olvasható értéket egy JSON objektumba lemásol és szövegként visszaad.

### 5.8.1. Küldő szerverek kezelése

A felület egy feladathoz társított küldő szerverek kezelésére szolgál. Át lehet navigálni a főmenühöz, a címzettek beállításához vagy küldő szerverek szerkesztéséhez. Két lista is található a felületen, a legördülő lista a felhasználó által megadott csatlakozásokat tartalmazza, az alatta lévő lista pedig a feladathoz társított kapcsolatokat mutatja. A két lista közötti hozzáadás gombokkal lehet a legördülő listában kiválasztott szervert társítani a feladathoz. Az eltávolítás gombbal az alsó listában kijelölt szerver kerül eltávolításra, vagyis a kijelölt szerver leválasztásra kerül a feladatról. Egy szerver csak egyszer társítható a feladathoz.

A szerverek listázásához használt „ListServers” függvény a szerverek kezelése részben van kifejtve. A feladathoz társított szervereket „GetTask” függvénnyel listázza ki a majd kiegészíti a „ListServers” korábban lekért szerver adatokkal. Módosítás a „SetTask” metódussal történik.

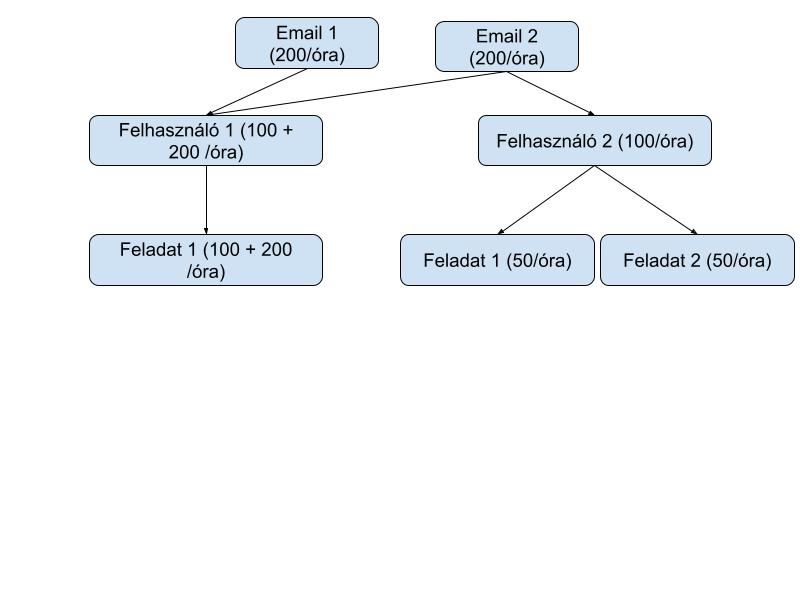
### 5.8.2. Címzettek beállítása

### 5.8.3. Levél szerkesztés

### 5.8.4. Feladat állapota

## 5.9. Küldés

Minden futó feladat elsődlegesen szerverenként lesz csoportosítva, erre azért van szükséges, mert a szerver korlátai globálisan minden azt használó felhasználóra vonatkozik. Másodlagos csoportosítás felhasználónként történik, így közösen használt szerver erőforrásai felhasználók között egyenlően oszlanak el. Harmadik csoportosítás pedig a felhasználó feladatai szerint történik így a felhasználón belüli feladatok egyforma mértékben tudják használni az email szervert.



*30. Ábra: Példa a küldési mennyiség elosztására*

# 6. Konklúzió

## 6.1. Program

Programmal sikerült költséghatékony és biztonságos megoldást kínálni kisebb vállalatok, irodák részére a tömeges e-mail küldés terén.

A készített kiegészítők közül a szövegszerkesztő és CSV kezelő bővítmény publikálásra került a bővítményeket megosztó oldalon. Összesen 40 alkalommal töltötték le szöveg szerkesztő és CSV kiegészítőket, ezen felül a CSV kiegészítőre érkezett egy pozitív visszajelzés is, így szinte biztos, hogy más is hasznát tudja venni a kiegészítőknek a munkája során.

## 6.2. Szerkesztő

Construct 3 Visual Scripting része rengeteg előnnyel rendelkezik, azonban hátrányok is kiütköztek a fejlesztés során. Előnyök közé sorolható a funkciók jól tagolhatósága, tapasztalataim szerint a meglévő komponensek jól illeszthetőek egymáshoz. UI felület könnyen összeépíthető, események könnyen kezelhetőek, logika központú, kevesebb fókuszt vesz el a szintaxis elsajátítása. Fő erősség a platformfüggetlen fejlesztés, szerkesztő beépített funkciói és a meglévő könyvtárak is nagy segítséget nyújtanak. Hátrányok között a legnagyobb az aszinkron műveletek kezelésének limitáltsága. Nem lehet halmozni, majd kivárni az aszinkron függvényeket iterálás közben. Ezt a korlátot a várakoztató műveletekkel vagy JavaScript kóddal (for await módszerrel) lehetett feloldani. További nagy hiány a UI felület építését segítő kiegészítők, jelenleg csak alap űrlap elemek érhetőek el.

# Irodalomjegyzék

[1] Flutter

<https://docs.flutter.dev/resources/architectural-overview>, 2022.11.18 16:22

[2] React

<https://reactjs.org/>, 2022.11.18 16:22

[3] Vue

<https://vuejs.org/>, 2022.11.18 16:22

[4] Wordpress

<https://hu.wordpress.org/about/>, 2022.11.18 16:22

[5] Joomola

<https://www.joomla.org/about-joomla.html>, 2022.10.25 22:03

[6] Drupal

<https://www.drupal.org/about>, 2022.10.25 22:03

[7] Wix

<https://www.wix.com/about/us>, 2022.10.25 22:03

[8] Appsheet

<https://about.appsheet.com/home/>, 2022.11.18 16:22

[9] Appery.io

<https://appery.io/>, 2022.11.18 16:22

[10] List of low-code development platforms

<https://en.wikipedia.org/wiki/List_of_low-code_development_platforms>,

2022.11.18 16:22

[11] List of no-code development platforms

<https://en.wikipedia.org/wiki/No-code_development_platform>, 2022.11.18 16:22

[12] Unity

<https://unity.com/>, 2022.11.18 16:22

[13] Unreal Engine

<https://www.unrealengine.com/en-US/>, 2022.11.18 16:22

[14] Godot

<https://godotengine.org/>, 2022.11.18 16:22

[15] Construct 3

<https://www.construct.net/en>, 2022.11.18 16:22

[16] Game Maker: Studio

<https://gamemaker.io/en/gamemaker>, 2022.11.18 16:22

[17] Gdevelop

<https://gdevelop.io/features>, 2022.11.18 16:22

[18] How JavaScript beats GameMaker Language (GML)

<https://www.construct.net/en/blogs/construct-official-blog-1/javascript-beats-gamemaker-1570>, 2022.10.18 17:22

[19] Mailchimp

<https://mailchimp.com/about/>, 2022.11.18 16:22

[20] Mailgun

<https://www.mailgun.com/>, 2022.11.18 16:22

[21] Outlook

<https://www.microsoft.com/hu-hu/microsoft-365/outlook/email-and-calendar-software-microsoft-outlook/>, 2022.11.18 16:22

[22] Thunderbird

<https://www.thunderbird.net/hu/about/>, 2022.11.18 16:22

[23] Sendblaster

<https://www.sendblaster.com/about-us/>, 2022.11.18 16:22

[24] Interpoláció

<https://hu.wikipedia.org/wiki/Interpol%C3%A1ci%C3%B3>, 2022.11.18 16:22

[25] Construct’s interface

<https://www.construct.net/en/make-games/manuals/construct-3/overview/the-interface>, 2022.11.18 16:22

[26] Simple Mail Transfer Protocol

<https://www.rfc-editor.org/rfc/rfc5321>, 2022.12.12. 16:44

[27] ECMAScript

<https://tc39.es/ecma262/>, 2022.11.18 16:22

[28] Xampp

<https://www.apachefriends.org/hu/index.html>, 2022.11.18 16:22

[29] HTTP Over TLS

<https://www.rfc-editor.org/rfc/rfc2818>, 2023.04.18 20:01

[30] Google Closure Compiler

<https://developers.google.com/closure/compiler>, 2022.11.18 16:22

[31] Google JavaScript Styling

<https://google.github.io/styleguide/jsguide.html>, 2022.11.18 16:22

[32] Construct 3 SDK templates

<https://www.construct.net/en/make-games/manuals/addon-sdk>, 2022.11.18 16:22

[33] Construct 3 Addons

<https://www.construct.net/en/make-games/addons>, 2023.04.08 20:13

[34] Construct 3 Plugin SDK template

<https://construct-static.com/articles/downloads/76675/c3-plugin-sdk-v1-10.zip>, 2022.11.18 16:22

[35] Crypto - Web API

<https://developer.mozilla.org/en-US/docs/Web/API/Crypto>, 2023.04.08 20:23

[36] Rollup

<https://rollupjs.org/introduction/>, 2023.04.18 20:42

[36\_1] Duck typing

<https://en.wikipedia.org/wiki/Duck_typing>, 2023.04.18 20:42

[36\_2] Mailcore 2

<https://github.com/MailCore/mailcore2>, 2023.04.18 20:42

[37] Base64

<https://www.rfc-editor.org/rfc/rfc4648>, 2023.04.18 20:45

[38] Content-ID

<https://www.rfc-editor.org/rfc/rfc2392>, 2023.04.18 20:47

[39] Quill

<https://quilljs.com/>, 2023.04.18 20:47

[40] RFC4180 - CSV

<https://www.rfc-editor.org/rfc/rfc4180>, 2023.04.18 20:47

[41] Dogfooding

<https://en.wikipedia.org/wiki/Eating_your_own_dog_food>, 2023.04.19 20:47

[42] Date and Time

<https://www.rfc-editor.org/rfc/rfc3339>, 2023.04.19 20:47

[43] Multipurpose Internet Mail Extensions

<https://www.rfc-editor.org/rfc/rfc2045>, 2023.04.20 20:47

[44] The AES-CBC Cipher Algorithm and Its Use with IPsec

<https://www.rfc-editor.org/rfc/rfc3602>, 2023.04.20 20:47

[45] PKCS #5: Password-Based Key Derivation Function 2 (PBKDF2)

<https://www.rfc-editor.org/rfc/rfc6070>, 2023.04.20 20:47

[A felhasznált szakirodalom megadása]

[A szerzők nevét mindenütt “Családnév, X.” formában kell megadni, ahol X. a szerző keresztnevének (keresztneveinek) kezdőbetűje. Magyar cikk esetén a vessző a családnév és a keresztnév kezdőbetűje közt elhagyható. Ha az egyértelműség megkívánja, a keresztnév kiírható teljesen is.

A dolgozat szerzője szabadon választhat, az A vagy a B típust használja.]

**[A-típus:**

A cikkekre való hivatkozás egy []-be írt sorszámmal történik. A sorszámozást folytonosan kell megtenni, a sorba rendezés alapja az első szerző családneve.]

[1] Szerző1 (, Szerző2 ...): *Cikk címe*

Folyóirat neve, sorszám, kezdőoldal-végoldal, év.

[2] Szerző1 (, Szerző2 ...): *Konferencia-kiadvány-beli cikk címe*

„Konferenciakiadvány:” Konferencia neve, hely, kezdőoldal-végoldal, év.

[3] Szerző1 (, Szerző2 ...): *Könyvcím*

„Könyv:” Kiadó, hely, oldalszám, év.

[4] Szerző1 (, Szerző2 ...): *Kutatási jelentés címe (csak publikus elérhető jelentés!)*

„Kutatási jelentés”: Kutatási projekt neve, intézet, oldal, év.

[5] Szerző: *Disszertáció címe*

„PhD/kandidátusi/stb. disszertáció”: Egyetem, kar neve, év.

[6] *Internetes oldal elnevezése:*

*URL, letöltés ideje (csak konkrét dokumentumra mutató URL adható meg!)*

**[B-típus:**

A hivatkozás a szerzők családi nevéből és a kiadás évéből képezett azonosítóval történik. Kettőnél több szerző esetén az „et.al.” rövidítés használható. Pl. „[Vijayasundaram, 1986.]”, „[Meister and Sonar, 1998.]”, „[Felcman et.al., 1994.]”

Abban a ritka esetben, amikor több cikknek is azonos azonosító jutna (megegyeznek a szerzők és a kiadás éve) az év után az „a”, „b”, „c”, stb. betűk csatolandók, pl. „[Stone and Norman, 1993a.]”.

A sorba rendezés alapja a szerzők családneve, végül az év.

Ezt leszámítva a formátum ugyanaz, mint az A-típus esetén, de ekkor a hivatkozási sorszám lehagyható.]

# Mellékletek

1. [A dolgozat mellékletei, ha vannak]