# Szakdolgozat


Miskolci Egyetem

# Webalkalmazás fejlesztése szervezeti folyamatok kezeléséhez

**Készítette:**

Utry Máté Attila

Programtervező informatikus

**Témavezető:**

Piller Imre

**Miskolc, 2020**

**Miskolci Egyetem**
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Intézeti Tanszék **Szám:**

# Szakdolgozat Feladat

Utry Máté Attila (YR10UU) programtervező informatikus jelölt részére.

**A szakdolgozat tárgyköre:** webalkalmazás, folyamatmodellezés

**A szakdolgozat címe:** Webalkalmazás fejlesztése szervezeti folyamatok kezeléséhez

**A feladat részletezése:**

*A különféle szervezetek (például intézmények, vállalatok) ügymenete nagyon bonyolult, és egyedi. A dolgozat egy olyan alkalmazás elkészítésének bemutatását célozza, amely konkrét szervezeti formától függetlenül képes azok folyamatainak modellezéséhez eszközt biztosítani.*

*A folyamatok alatt itt elsősorban olyan, véges állapotú automatával leírható műveletsorokat értünk, amely esetenként további, összetettebb (például szerepkörökhöz) kötött feltételek, külső állapotok teljesülését is szükségessé teszik. A dolgozatban ezen állapotgép és feltételrendszer leírási módjához szükséges alkalmazást kell megtervezni és implementálni.*

*Az alkalmazásnak webes környezetben, szerver-kliens architektúrának megfelelően kell elkészülnie. Ehhez szerver oldalon a Falcon nevű, Python alapú mikrokeretrendszer ad alapot, míg a kliens megvalósításához egy elterjedt JavaScript alapú keretrendszert kell használni. Az adatokat az alkalmazásnak relációs adatbázisban kell tárolnia.*

**Témavezető:** Piller Imre (egyetemi tanársegéd)

**A feladat kiadásának ideje:**

. . . . . .
szakfelelős

# Eredetiségi Nyilatkozat

Alulírott **Utry Máté Attila**; Neptun-kód: `YR10UU` a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Programtervező informatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy *Webalkalmazás fejlesztése szervezeti folyamatok kezeléséhez* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, ..... .év ..... .hó ..... .nap

.....

Hallgató

1.

szükséges (módosítás külön lapon)

A szakdolgozat feladat módosítása

nem szükséges

..... .....

dátum témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás): konzulens (dátum, aláírás):

..... .....

..... .....

..... .....

3. A szakdolgozat beadható:

..... .....

dátum témavezető(k)

4. A szakdolgozat ..... szövegoldalt

..... program protokollt (listát, felhasználói leírást)

..... elektronikus adathordozót (részletezve)

.....

..... egyéb mellékletet (részletezve)

.....

tartalmaz.

..... .....

dátum témavezető(k)

5.

bocsátható

A szakdolgozat bírálatra

nem bocsátható

A bíráló neve: .....

..... .....

dátum szakfelelős

6. A szakdolgozat osztályzata

a témavezető javaslata: .....

a bíráló javaslata: .....

a szakdolgozat végleges eredménye: .....

Miskolc, ..... .....

a Záróvizsga Bizottság Elnöke

# Tartalomjegyzék

# 1. fejezet

# Bevezetés

A szakdolgozat egy szervezeti folyamatok modellezését elősegítő gráf szerkesztő alkalmazást mutat be. Egy szervezet életében nagyon sok folyamattal lehet találkozni napi szinten, melyek sokasága és hatékony végrehajtása érdekében célszerű lehet azokat valamilyen folyamatkezelő program segítségével ábrázolni úgy, hogy az ábrákat későbbi használat céljából el is lehessen menteni, majd visszatölteni megtekintés, esetleg módosítás okán.

Több folyamatmodellezésre használható, készen elérhető alkalmazás létezik már. Akkor miért volt szükség még egy elkészítésére a szakdolgozat keretein belül? Többek között erre a kérdésre is választ fogunk kapni a szakdolgozat elolvasása után.

Az első tartalmi fejezet általános áttekintést ad az üzleti (szervezeti) folyamatokról, illetve azok modellezéséről. Ismertetem a témával kapcsolatos fogalmakat, és meg is magyarázom őket. Az ezt követő fejezetek az alkalmazás tervezési folyamatait (alkalmazott technológiáit), megvalósítását (programkódokkal szemléltetve), valamint használatát írja le. Végül ez utóbbi könnyebb megértése érdekében néhány gyakorlati példa segítségével szemlélteti, hogy hogyan történik a folyamatmodellezés az alkalmazás segítségével.

# 2. fejezet

# Üzleti folyamatok és modellezésük

Dolgozatom első tartalmi fejezetének alapjául a(z) [**?**] forrás blogbejegyzéseit használtam, ezeket dolgoztam fel és egészítettem ki önálló gondolatokkal.

## 2.1. Mik azok az üzleti folyamatok?

Elsőként nézzük meg, mit nevezhetünk egy folyamatnak. A folyamat szó más-más jelentéssel bír annak függvényében, hogy hol használjuk: például mást jelent a hétköznapi életben, mint az informatikában. A mi esetünkben a folyamat előre meghatározott vagy tetszőleges sorrendben elvégezhető tevékenységek kapcsolatrendszerét jelenti.

Tehát gyakorlatilag a folyamat olyan tevékenységek halmaza, amelyek egymással kölcsönhatásba lépnek egy adott cél elérése érdekében. Ezt a jelentését használhatjuk az üzleti folyamatokra is, hiszen ha egy üzleti célt tűzünk ki magunk elé, akkor annak megvalósulása is több tevékenység egymásutániságából, több lépésből fog adódni. Ezeket a lépéseket együttesen nevezzük *üzleti folyamat*nak.

Alapvetően kétféle megközelítése van a lépések végrehajtása sorrendjének:

- Tevékenységek tetszőleges sorrendben elvégezhetőek.
- Csak egymás utáni, meghatározott sorrendben követhetik egymást.

Az üzleti folyamatok minden vállalat életében napi szinten jelen vannak és egy áttekinthető láncolatot alkotnak a termék előállításához szükséges nyersanyagok beszerzésétől kezdve a munkafolyamatokon keresztül egészen a piacra kerülésig, és a piacon való sikeres vagy sikertelen szereplésig.

Üzleti folyamatokról általában akkor beszélünk, ha egy adott vállalat termékét, szolgáltatását, vagy menedzsmentjét akarjuk elhelyezni a gazdasági életben. Belátható, hogy a termék minősége és mennyisége összefügg a gyártó menedzsmentjének arculatával, piaci ismertségével, és a termék iránti kereslet erejével. Ezeket az összefüggéseket jeleníti meg egy vállalat reklám, PR, és marketing tevékenysége. Közismert példa, hogy az MGM stúdió világviszonylatban piacvezető termékeket (filmeket) állít elő, de reklámköltsége meghaladja a filmgyártás és forgalmazás költségeit.

Hogy érthetőbb legyen mit is jelent a gyakorlatban az üzleti folyamat, nézzünk rá egy egyszerű példát:

Adott egy startup (vagy magyarosabban *korai fázisú vállalkozás*[**?**]), ahol szükségessé válik bizonyos irodaszerek beszerzése. Magát a beszerzés menetét egy kizárólag kötött sorrendben elvégezhető folyamat fogja leírni. Nézzük a folyamat lépéseit:

1. Első lépésben konkrétan meghatározzuk, hogy melyek azok az eszközök, amelyekre szükség van, és miből hány darab szükségeltetik. Ezeknek az összeírása az első feladat.

2. Miután összegyűjtöttük a megrendelendő tárgyakat, meg kell vizsgálnunk, hogy a vállalkozás mekkora anyagi kerettel rendelkezik, és ebből mennyit tudunk a megrendelésre költeni. Ha rendelkezünk megfelelő pénzösszeggel, akkor minden rendben van, tovább lehet lépni. Ha viszont nem, akkor valamilyen szempont alapján ki kell húzni a listáról bizonyos termékeket (például a legszükségtelenebbeket). Ez utóbbi termékek megrendelése vagy elvetése a következő lépéstől is függ.

3. Ha már rendelkezésünkre áll a lista és a pénzügyi fedezet, utána kell néznünk, hogy az adott eszközök hol szerezhetőek be a legalacsonyabb áron (ehhez segítséget tud nyújtani például az Árukereső[**?**] weboldala, ami ár szerinti növekvő sorrendben kilistázza, hogy egy adott termék mely online webáruházakban és milyen áron érhetők el). Természetesen nem csak az alacsony ár, hanem más szempont is szóba jöhet (például a garancia időtartama az adott termékre, a kiszállítás díja, stb.).

   Ennél a lépésnél derül ki, hogy pontosan mennyi összeget kell a meglévő pénzügyi keretből az új eszközök megrendelésére fordítani. Lehetséges, hogy marad még pénz egy korábban az anyagi keret szűkösségére való hivatkozás miatt lehúzott termék megvásárlására.

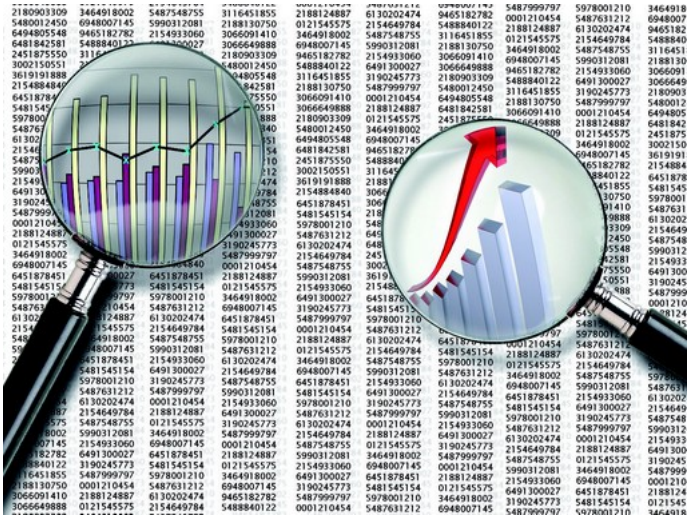
4. Mindezek után elindulhat a megrendelés folyamata. Ez többféleképpen is történhet:

   - Az eszközöket online rendeljük meg. Ennek előnye, hogy egyszerre több helyen is megfigyelhetjük a termékeket, utána járhatunk az azokat árusító weboldalak hitelességének, megbízhatóságának, véleményeket olvashatunk róla, és nem kell személyesen megjelennünk az adott áruházban. Hátránya, hogy időbe telik a termékek kiszállítása, és adott esetben számolnunk kell a szállítási költséggel is. Ez utóbbinak mértéke áruházanként eltérő. Ugyanakkor ma már léteznek olyan webáruházak is, ahol a megrendelt terméket személyesen is át lehet venni egy előre meghatározott üzlethelységben, így azokhoz gyorsabban juthatunk hozzá, viszont ebben az esetben a megrendelőre hárul a szállítási költség.
   - Megtehetjük, hogy nem rendeljük meg előzetesen a termékeket, hanem azok beszerzésére személyesen megyünk be az áruházakba. Ezt akkor célszerű alkalmaznunk, mikor egy nagyobb áruházat fogunk meglátogatni, ahol nagy eséllyel az összes termék rendelkezésünkre fog állni, így azokat egy helyről azonnal meg is tudjuk venni, és elvinni. Nagy előny ebben az esetben (szemben az online rendeléssel), hogy ki is tudjuk próbálni az adott termékeket, hogy hogyan működnek, van-e valamilyen hibájuk.

5. Ezek után, hogy a megrendelt termékek eljutottak az irodába, utolsó lépésként már csak szét kell osztani azokat aszerint, hogy mely terméket ki igényelte.

Ezzel a végére is értünk egy egyszerűbb üzleti folyamatnak.

Azért ezt a fenti példát hoztam szemléltetésként, mert ez valamilyen formában minden vállalkozásnál jelen van. Ha jobban belegondolunk, láthatjuk, hogy gyakorlatilag szinte minden tevékenységet (így adott termék vagy termékek beszerzését is) folyamatban végzünk. Fontos megjegyezni, hogy ennél a példánál, és az ehhez hasonlóknál a lépéseket csak adott sorrendben hajthatjuk végre. Például nem vihetünk el egy terméket az áruházból úgy, hogy csak majd később fizetünk érte.



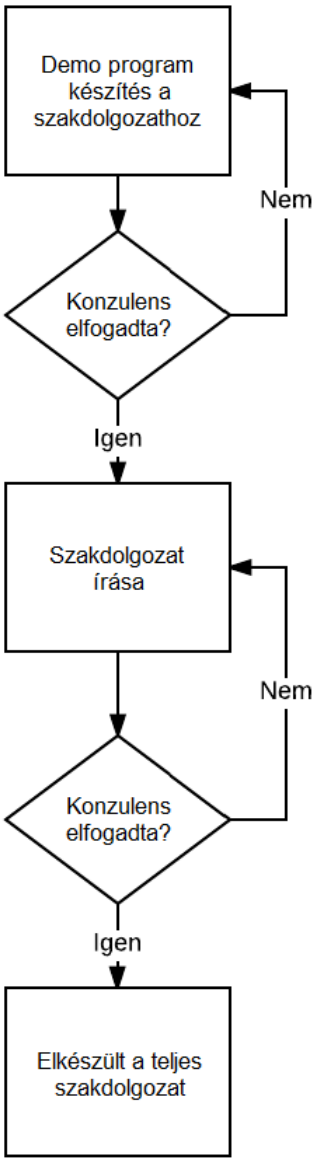
2.1. ábra. Üzleti intelligencia[?]

## 2.2. Üzleti folyamatok modellezése

Az előző részben olvashattuk, hogy nagyon sok mindent folyamatban végzünk. Akár egy legegyszerűbb hétköznapi tevékenységet is. Egy vállalat esetében pedig az ilyen tevékenységekből jóval több van, melyek lényeges szerepet játszanak annak működésében. Emiatt fontos az, hogy a vállalat bizonyos folyamatai rögzítésre kerüljenek. Ezért van szükség a folyamatok *modellezésére*. Ennek egy egyszerű, átlátható formája a folyamatábra használata.

### 2.2.1. Modellezés folyamatábrával

A folyamatábra gyakorlatilag egy olyan eszköz, amely segítségével grafikusan ábrázolhatóak a tevékenységek, folyamatok. A különböző folyamatokat különböző szimbólumokkal jelölhetjük rajta, közöttük sorrendiséget állíthatunk fel meghatározva ezáltal azt, hogy időben melyik esemény melyik után vagy előtt következik be. Hogy érthetőbb legyen, miről is van szó, a **??** ábra egy nagyon egyszerű folyamatábrát szemléltet, amin a szakdolgozatom készítésének legfőbb lépései helyezkednek el.




2.2. ábra. Egy egyszerű folyamatábra

A fent látható folyamatábra egyszerűségét az is adja, hogy nem teljes. Nincsen kiindulópontja, sem végpontja. Olyan, mintha egy nagyobb folyamatból csak néhány tevékenységet ábrázolnánk. Ahhoz, hogy teljes legyen a folyamatábra, szükség van kezdő- és végállapotra.

Nézzük tehát, hogy mitől lesz teljes egy folyamatábra, illetve hogy általánosságban milyen részeket tartalmaz. Az alábbi (leggyakoribb, szinte minden folyamatábrán megtalálható) szimbólumok a következő jelentéssel bírnak:

- *Téglalap* jelöli a tevékenységet. Ebből van a legtöbb a folyamatábrán, hiszen ezek szemléltetik magát a folyamatot, az események egymás utáni történését.
- *Rombusz* alak jelenti az átjárókat. Ezek útválasztóként funkcionálnak azáltal, hogy egy feltételt szabnak meg, melynek kiértékelésétől függ, hogy melyik tevékenység fog következni. Általában két másik tevékenység követi, amelyek közül

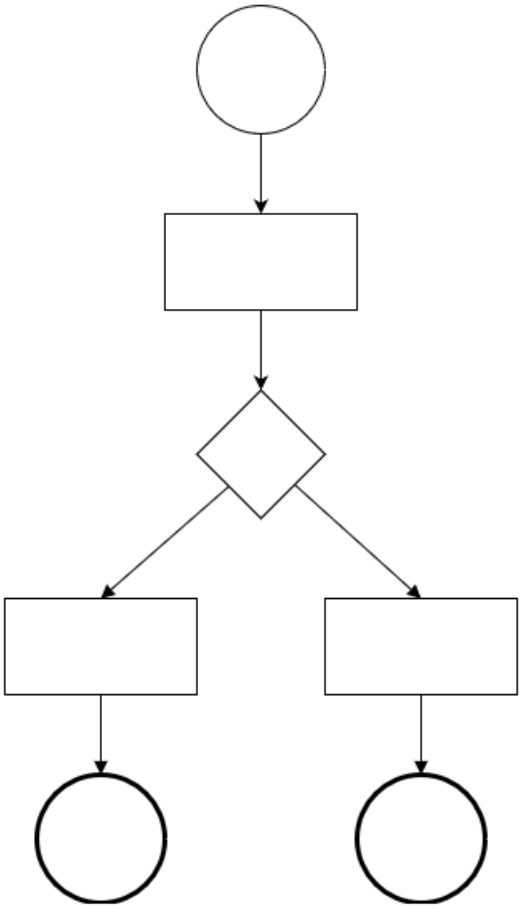
az lesz a soron következő, amelyikre az útválasztó kiértékelése a megfelelő (igaz vagy hamis) értéket adja.

- *Üres fehér kör vékony fekete körívvel* fejezi ki a "START" állapotot *(kezdőállapot)*. Itt kezdődik el a reprezentált folyamat. Legalább egy "START" szimbólumot tartalmaznia kell minden teljes folyamatábrának.

- A folyamat befejezését az *üres fehér kör vastag fekete körívvel* jelöli. Ezt "END" állapotnak *(végállapotnak)* is szokták nevezni. Ebből is legalább egynek lennie kell ahhoz, hogy teljes legyen a folyamatábra, hiszen ahogy elkezdődik egy folyamat, úgy az be is fejeződik egy bizonyos idő után. Előfordul olyan folyamatábra, ami több végállapotot tartalmaz, mint kezdőállapotot. Ez nem hiba, hiszen a való életben is gyakran fejeződik be egy elkezdett folyamat többféleképpen.

- A különböző folyamatokat vonalak kötik össze, melyeket *szekvenciafolyam*nak neveznek. Fontos, hogy ezek nem lehetnek csak vonalak nyíl nélkül, hiszen egyértelműen meg kell határozniuk, hogy melyik tevékenységből melyik másik következik.

Példaként tekintsük a **??** ábrát, ami egy teljes folyamatábrát mutat, mely a fent említett szimbólumok közül mindegyikből tartalmaz legalább egyet (beleértve a kezdő- és végállapotokat is).



2.3. ábra. Egy üres, de teljes folyamatábra (Forrás: [**?**]

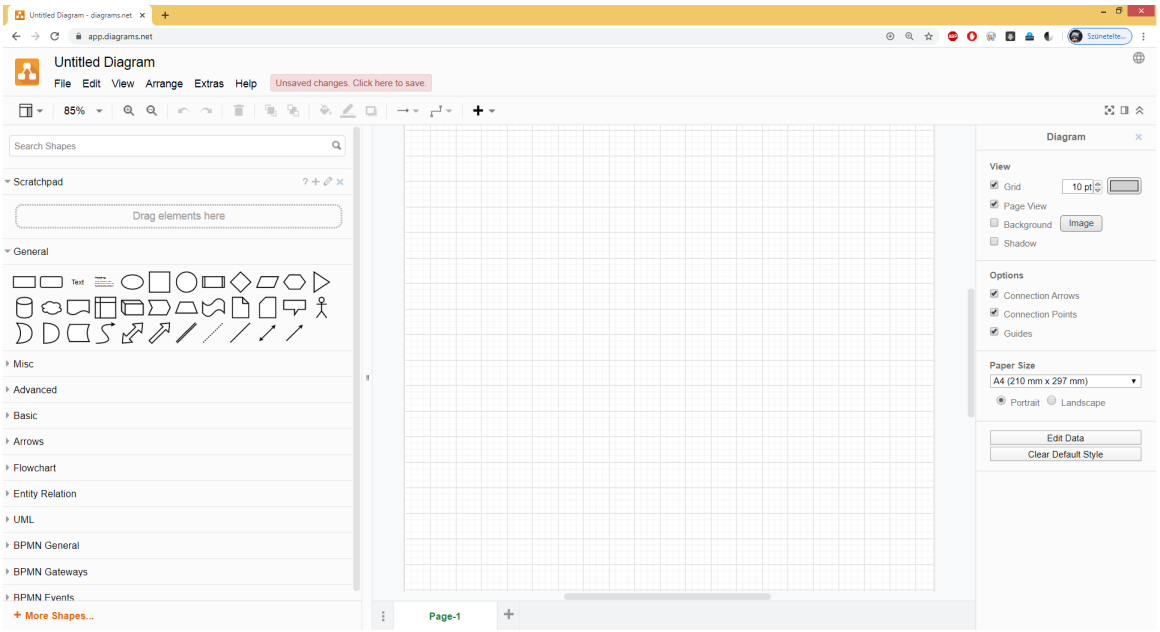
Természetesen léteznek ennél sokkal összetettebb, többféle szimbólumot tartalmazó

folyamatábrák is, azonban már egy ekkora ábrán is meg lehet jeleníteni egyszerűbb üzleti folyamatokat.

## 2.3. Egy folyamatokat szemléltető szoftveres eszköz

### 2.3.1. Draw.io

Folyamatok ábrázolására már nagyon sok kész program létezik. Például a **??** ábrát is egy ilyen folyamatábrázoló szoftver segítségével készítettem el, melynek neve *Draw.io*[**?**]. A Draw.io a legismertebb és legsokoldalúbb ingyenes folyamatábrázoló alkalmazás. Az alkalmazás nem igényel letöltést, a böngésző segítségével el tudjuk érni akár a rövidebb `www.draw.io` URL címen keresztül. További előnye még, hogy regisztrációt sem igényel, hanem azonnal az oldal betöltése után már kezdhetjük is a folyamataink modellezését.




2.4. ábra. A Draw.io nyitóoldala (Forrás:[**?**])

Láthatjuk, hogy rengeteg lehetőség tárul elénk. A modellezéshez nagyon sok féle szimbólum közül választhatunk számunkra megfelelőt. Természetesen mindegyik más jelentéssel bír, azonban a program teljes egészében a felhasználóra bízza azok használatát: nem határoz meg integritási feltételeket, így a felhasználó szabályoktól függetlenül alakíthat ki számára kedvező ábrákat.

Használata viszonylag egyszerű, felhasználóbarát. A felhasználó kiválaszt egy elemet a bal oldali eszköztárból, és azt kattintással, vagy akár behúzással megjeleníti a képernyő közepén található négyzetrácsos részen. Az így megjelenített szimbólumokat nyomva tartott bal egérgomb mellett az egér mozgatásával lehet áthelyezni.

Az általános szimbólumokon kívül (amik a program bal oldali menüjében le vannak nyitva *General* néven) különböző nyilakat, egyéb formákat használhatunk, UML diagramokat készíthetünk, illetve az üzleti folyamatmodell és jelölés (*Business Process Model and Notation*, BPMN) sajátos alakzatait is igénybe vehetjük. De ha még ez se

lenne elég, akkor lehetőség van az eszköztárat kibővíteni további alakzatok hozzáadásával, amit a bal alsó sarokban található **"More Shapes. . . "** (további alakzatok) gomb megnyomásával érhetünk el.

Elkészített folyamatábráink / diagramjaink mentésére is többféle módot kínál az alkalmazás. Lehetőség van PNG, SVG, HTML, és XML formátumba is menteni a munkánkat, amiket a saját eszközünkre, vagy akár Google Drive-ra, GitHub-ra, és még sokféle felületre azonnal exportálhatunk is.

Összegezve tehát a Draw.io egy rendkívül hasznos és változatos alkalmazás, amely sokféle megvalósítási lehetőséget kínál a felhasználók számára. Egyszerű használatának köszönhetően népszerű a felhasználók körében. Üzleti folyamatok modellezésére kiválóan használható. Egyetlen hátránya lehet az, hogy (mivel böngészőből indítható el) internetkapcsolat szükséges hozzá, de ma már szinte bárhol vagyunk, könnyedén csatlakozni tudunk a világhálóhoz.

# 3. fejezet

# Specifikáció

## 3.1. Áttekintés

Az előző fejezetben láthattuk, hogy már léteznek készen elérhető alkalmazások üzleti folyamatok modellezésére. A *Draw.io* összetettségével, lehetőségeinek sokaságával, ugyanakkor egyszerű használatával egy kiváló alkalmazás, azonban azáltal, hogy teljes szabadságot nyújt a diagramok, folyamatok ábrázolásának használatához, könnyen előfordulhat, hogy az elkészült ábrák (főleg nagy elemhalmazzal való dolgozás esetén) bizonyos integritási feltételeknek nem tesznek eleget, és ezáltal hibás folyamatmodellezési ábrák készülhetnek. Ez különösen akkor jelenthet gondot, hogyha formális célból készül az adott ábra.

Ebben a fejezetben fogjuk áttekinteni, hogyan is épül fel a gráf szerkesztő alkalmazásunk, melynek segítségével az üzleti folyamatok modellezése történik, illetve hogy miben tér el a már elkészült, hivatalos gráf szerkesztő programoktól, azoknak a hibáit hogyan próbálja meg elkerülni.

## 3.2. A program felépítése

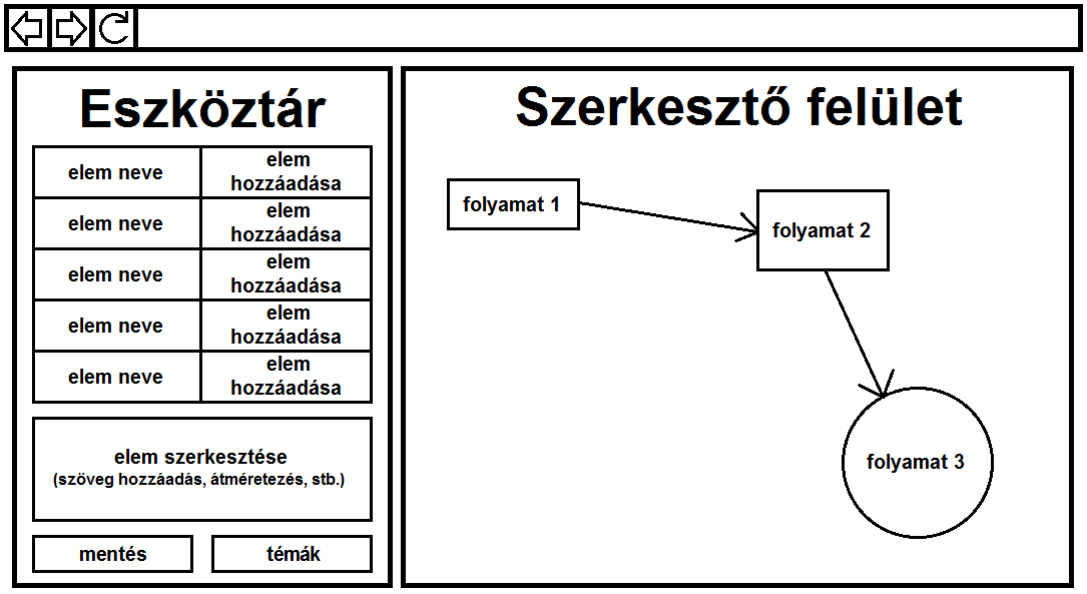
Az elkészített programnak alapvetően sok apró alkotórészt kell tartalmaznia. Lássunk egy egyszerű, sematikus ábrát arról, hogy hogyan is nézzen ki majd a gráf szerkesztőnk. Ezt a **??** ábra szemlélteti.

Az ábra alapján máris van egy elképzelésünk a program szerkezetéről. Nézzük is meg annak konkrét felépítését.

Bal oldalon található az **eszköztár**, amelynek segítségével különböző funkciók érhetőek el. Innen tudjuk irányítani (létrehozni, szerkeszteni) a gráf felépítéséhez szükséges elemeket. Az eszköztárból az alábbi funkciók érhetőek el:

1. Az eszköztár nagy részét egy táblázat teszi ki, amely két oszloppal, és a felvihető elemek számával megegyező sorral rendelkezik. A két oszlop a következő jelentéssel bír:

    - Az **elem neve** rész helyére kerül az adott elem (csomópont) megnevezése. Nevük alapján különböztethetőek meg az egyes elemek. Az elemeket egymás alatt célszerű lehet például használati gyakoriság alapján rendezni (a téglalap, mint tevékenység, az egyik leggyakrabban alkalmazott csomópont típus, ez lehetne legfelül).




3.1. ábra. A gráf szerkesztő sematikus rajza

- Az **elem hozzáadása** felirat helyén az adott elem formája jelenik meg. Ez gyakorlatilag egy gomb, melyre rákattintva hozható létre a mellette feltüntetett nevű elem a szerkesztő felületen egy előre meghatározott helyen. Természetesen ez csak egy alapértelmezett hely, ahol megjelenik kattintás után az adott elem, melyet a későbbiekben lehetőségünk van áthelyezni.

2. A táblázat alatt található egy kisebb rész az elemek szerkesztésére. Ez akkor jelenik meg, amikor a szerkesztő felületen rákattintunk egy csomópontra, melyet szerkeszteni szeretnénk. Itt az alábbi szerkesztési lehetőségek állnak rendelkezésünkre:

   - **Szöveget** adhatunk hozzá a kijelölt csomóponthoz. Elegendő csak beírni a felvinni kívánt szöveget, és az azonnal megjelenik az adott csomópont közepén. Ez egységes minden csomópontra (tehát mindegyik típushoz tudunk írni szöveget).
   - **Átméretezhetjük** a kijelölt elemet. Ez csomópontonként eltérő, hogy milyen paramétereket kíván (például téglalap csomópontnál a szélesség és a hosszúság együttes megadása eredményezi az új méretet, kör alakú csomópont esetében viszont elegendő csak egy paramétert, a sugárt megadni).

3. Az eszköztár alatt található egy **mentés** gomb, melynek segítségével elmenthetjük az elkészített gráfot egy adatbázisba az esetleges későbbi módosítások céljából.

4. Végül a mentés gomb mellett a **témák** feliratú szöveg után egy legördülő menü helyezkedik el, melyre rákattintva választhatjuk ki az elemek különböző témáját, stílusát. Ez egy apróbb felhasználóbarát élményt nyújt azáltal, hogy különböző színű formák használatát is lehetővé teszi az alapértelmezetteken kívül. Természetesen ez nem befolyásolja a program használatát, csupán a kinézetét változtatja meg a felhasználó által kívántra.

A **szerkesztő felület** (vagy más szóval vászon (*canvas*)) a létrehozott elemek megjelenítéséért felelős. Kezdetben üres, egyszínű az egész. Ahogy viszont rákattintunk az eszköztárban egy általunk kiválasztott elemre, úgy az megjelenik rajta, és különböző műveleteket tudunk azon végezni.

1. Minden létrehozott csomópont egy egyedi azonosítószámot (*ID*-t) kap. Ez alapján lehet őket megkülönböztetni egymástól.
    - Nem kaphat két különböző csomópont ugyanolyan azonosítószámot, annak egyedinek kell lennie.
    - Ha bármilyen attribútuma módosul egy csomópontnak (pozíciója, színe, rajta lévő szövege, stb.), akkor is az ID-jának állandónak kell lennie.
2. A csomópontokat tetszés szerint mozgatni lehet a szerkesztő felület teljes területén.
3. A csomópontok között vonalak húzhatóak az egyik végén egy nyíllal.
    - Csak két, általunk kiválasztott csomópontot lehet összekötni vonallal. Megkülönböztetünk *cél-* és *forrás csomópont*ot. Az összekötő vonal végén a nyíl arra az elemre fog mutatni, amelyet később jelölünk ki (tehát a célcsomópontra).
    - Egy csomóponthoz több vonal is tartozhat, amiből a következő pont is következik:
    - Egy csomópont cél- és forrás csomópontként is funkcionálhat.
    - Tehát gyakorlatilag annyi vonal indulhat ki egy csomópontból / érkezhet be egy csomópontba, amennyit csak szeretne a felhasználó.
4. Lehetőség van adott csomópont törlésére.
    - Ha már a továbbiakban nincs szükség egy csomópontra, akkor az letörölhető a szerkesztő felületről.
    - Ilyenkor, ha van a csomóponthoz tartozó vonal, akkor az is törlésre kerül (több vonal esetén az összes).
    - A programnak ügyelnie kell arra, hogy egy csomópont törlése után se legyenek a későbbiekben létrehozott csomópontok ID-jai között azonosak.
5. Továbbá lehetőség van csak a vonal törlésére két adott csomópont között.

## 3.3. Miben különbözik az elkészített program a már használatban lévő gráf szerkesztőktől?

Korábban említettem, hogy a *Draw.io* alkalmazásnak is vannak hiányosságai annak ellenére, hogy rendkívül sok lehetőséget kínál. Most nézzünk néhány példát, amik előfordulhatnak akkor, hogyha a felhasználó nem elég körültekintő egy ilyen vagy ehhez hasonló gráf szerkesztő használata közben, és elkövet olyan hibákat a szerkesztés folyamán, amik miatt nem várt eredmény születhet, illetve hogy ezek a hibák miért jelenthetnek problémát:

1. Nem pontosan köt össze a vonallal két folyamatot.

    - Mivel a Draw.io a vonalakat is a folyamatokhoz hasonló módon kezeli, ezért azt is a felhasználó tetszés szerinti helyre mozgathatja (tehát nem muszáj összekötnie vele két csomópontot, magában is állhat egy vonal). Ezáltal előfordulhat olyan ábra is, aminél a felhasználó a vonal vége és az összekötendő csomópont között figyelmetlenségből egy kis helyet kihagy, így valójában nem kötött össze két folyamatot, amit össze szeretett volna.

2. Ha letöröl egy csomópontot, amely össze van kötve egy másikkal, akkor a hozzá tartozó vonal a "levegőben marad", nem fog két meghatározott csomópontot összekötni, ezáltal az alapvető funkciója válik feleslegessé.

    - Az előző ponthoz hasonló probléma merül fel itt is, tehát a vonal nem két csomópont közötti összeköttetést fog megvalósítani.

3. Takarásba kerülhet két vagy több csomópont.

    - Előfordulhat ezáltal, hogy nem olvasható el a csomóponton található szöveg teljes egészében, vagy akár az is, hogy egy nagyobb méretű csomópont teljesen eltakar egy kisebb méretűt, ezáltal megfeledkezünk róla. Ha nincs rá szükség, célszerűbb lenne letörölni, ha pedig van, mert például valamilyen információt tartalmaz a rajta található szöveg, akkor a takarás által elveszettnek hihetjük, így megtörténhet az is, hogy még egyszer létrehozzuk, ami felesleges munkát eredményez.

4. Nincs legalább egy kezdő- és legalább egy végállapot.

    - Ahhoz, hogy teljes folyamatábrát modellezzünk, szükséges definiálni egy-egy kezdő- és végállapotot.

Ezekkel a fent felsorolt, esetlegesen felmerülő hibákkal szemben nézzük meg minden egyes pontnál, hogy az elkészített program hogyan igyekszik azokat kiküszöbölni:

1. Összekötő vonalat csak adott 2 csomópont között lehetséges húzni.

    - Az összekötést a program úgy oldja meg, hogy a felhasználó jelöli ki azt a két csomópontot, amelyet össze szeretne kötni. Ahogy korábban is említettem, lesz egy forrás csomópont és egy célcsomópont, e kettő közt fogja kirajzolni a program a vonalat úgy, hogy a célcsomópont felé mutat a vonal végén lévő nyíl.

2. Csomópont törlése esetén a hozzákapcsolt vonal is törlésre kerül.

    - Több nyíl esetén az összes törlődik, elkerülve ezáltal a vonalak "levegőben maradását".

3. A program jelez, hogyha egymásra kerülnek csomópontok.

    - A felhasználót a canvas más színűre változtatásával figyelmezteti a program amiatt, hogy fedésbe került két vagy több csomópont.

- Valamint a canvas-on kívülre sem engedi mozgatni a csomópontot a program.

4. Figyelmeztet az alkalmazás, ha hiányzik kezdő- vagy végállapot.

   - Az előző ponthoz hasonlóan itt is arról tájékoztatja a felhasználót a program, hogy nem teljes az általa modellezett folyamatábra.

A fent említett utóbbi két példa egészen addig fog hibajelzést generálni a felhasználó számára, amíg meg nem szűnteti azokat.

## 3.4. Egyszerűbb folyamatábra létrehozása lépésről lépésre

Nézzük meg, hogy az elkészített alkalmazásban a felhasználó hogyan tud létrehozni egy egyszerűbb folyamatábrát. Fontos, hogy ez az alfejezet csak elméletben írja le, hogy hogyan valósítható meg a folyamatmodellezés a program segítségével. Folyamatábrák konkrét létrehozását, használatát, elmentését az 5. fejezet tárgyalja.

A program, ahogy azt a felépítésénél olvashattuk, egy teljesen üres canvas-szal (vászonnal) fogadja a felhasználót kezdetben. Ez maga a szerkesztő felület, ahol a folyamatmodellezés történik, és itt fogjuk látni az elkészített folyamatábránkat végeredményként. Egy egyszerű folyamatábrát az alábbi lépéseket követve tudunk létrehozni (természetesen ezek a lépések csak iránymutatóak, nem kötött sorrendet határoznak meg):

1. Első lépésben kiválasztjuk azokat az elemeket az eszköztárból, amelyek a modellezéshez szükségesek. Ha teljes folyamatábrát szeretnénk készíteni, akkor célszerű először a kezdő- és a végállapotokat létrehozni (ezek hiányában figyelmeztet minket a program).

2. Ha már tudjuk, hogy nagyjából hány tevékenységet szeretnénk ábrázolni, akkor annak megfelelő számú csomópontot hozzáadhatunk a vászonhoz.

   - Természetesen a későbbiekben módosíthatóak a csomópontok száma újabbak hozzáadásával vagy meglévők törlésével.

3. Miután felvittük ezeket az elemeket, elmozgathatjuk azokat egymástól különálló helyekre azért, hogy lássuk mindegyik csomópontot egyszerre az ábrán, és ezáltal be tudjuk határolni, hogy mekkora helyre lesz szükségünk a végső folyamatábránál.

   - Ha nem mozgatjuk el egymásról a csomópontokat, és takarni fogja legalább kettő egymást, akkor az előző alfejezetben a 4. példaként említett figyelmeztetést fogjuk kapni. Ezt elkerülve célszerű minden létrehozott csomópontot azonnal elmozdítani az alapértelmezett helyéről.

4. Ezt követően megírhatjuk az egyes csomópontok szövegeit, hogy azok pontosan milyen tevékenységet jelölnek.

- Igény szerint a csomópontokat át lehet méretezni, de a csomópont automatikusan elkezd szélesedni, mielőtt a beleírt karakterek hossza elérné a csomópont széleit.

5. Ha ezzel is megvagyunk, célszerű valamilyen szempont alapján (például időrendi sorrend szerint) egymás után elhelyezni a csomópontokat balról jobbra, illetve fentről lefelé.

6. Ezután összekötjük az általunk összekapcsolni kívánt csomópontokat egymással. Ha teljes folyamatábrát szeretnénk kapni, akkor figyeljünk az alábbi szempontokra:

   - Legyen legalább 1-1 kezdő- és végállapot úgy, hogy a kezdőállapotból csak kifelé, a végállapotba pedig csak befelé vezet legalább egy összekötő vonal.
   - Minden csomópontba el lehessen jutni (tehát a kezdő- és a végállapotokat kivéve mindegyik csomópont legalább két másikkal összeköttetésben legyen úgy, hogy legalább egy bevezető és legalább egy kivezető vonal tartozik hozzá).
   - Ne legyen olyan csomópont, amiből már nem lehet végállapotba jutni a vonalakon keresztül.

7. Végül, ha minden simítást elvégeztünk a folyamatábránkon (megfelelő méretűekre változtattuk a csomópontokat, az általunk kívánt helyekre mozgattuk őket, stb.), utolsó lépésként már csak el kell azt mentenünk, és készen is vagyunk.

## 3.5. UX design szempontok a programnál

Ezen alfejezet megírásához a(z) [**?**]. forrás blogbejegyzését használtam alapul. Az itt említett szempontok, meghatározások alapján építettem fel ezt a részt, és azt egészítettem ki a dolgozat programjára jellemző tulajdonságokkal, egyéni gondolatokkal.

### 3.5.1. Mi az a UX design?

A **felhasználói élmény tervezés** (*user experience design*, UX design) egy tervezési folyamat, ami a felhasználói elégedettséget hivatott javítani a felhasználó és a termék között azáltal, hogy növeli a használhatóságot, a hozzáférhetőséget és az élményt. Lényegében azt kell megtervezni, hogy az elkészült alkalmazás vagy termék jó benyomást keltsen a felhasználóban bizonyos szempontokat figyelembe véve. Ez azért fontos, mert ha a felhasználó elégedett a termékkel, akkor azt szívesebben, örömmel használja, és a pozitív visszajelzés több szempontból is hasznos tud lenni. A használó számára is élményt nyújt, és a készítő is elégedett lesz a termékével.

Nélkülözhetetlen egy, a UX-szel kapcsolatos fogalmat is megemlíteni: a **UI**-t. Ez a *user interface* (magyarul: **felhasználói felület**) rövidítése, amely a program és a felhasználó közötti kommunikációért felelős. Többféle felhasználói felületek létezik (például szöveges, parancssoros, grafikus). Esetünkben *grafikus felhasználói felület*ről beszélhetünk, hiszen a program grafikus (rajzos, szöveges) elemeket tartalmaz, nem kell parancsokat kiadni bizonyos funkciók elérése érdekében.

### 3.5.2. UX alapkövetelmények

Nézzük meg, hogy milyen alapkövetelményei vannak a UX-nek egy adott weboldal szempontjából, illetve hogy ezeknek az alapkövetelményeknek mennyire felel meg a szakdolgozat programja.

1. **Legyen hasznos:** A célközönségnek kell a legnagyobb fókuszban lennie: jelen esetben ez a felhasználók, akik számára készül a program. Ezáltal úgy kell azt megvalósítani, hogy az érdemleges információt tartalmazzon, felkeltse a felhasználó érdeklődését, és így hasznosnak találja a programot.

    - A program hasznosságát az mutatja, hogy az valós szervezeti folyamatok modellezését segíti elő azáltal, hogy lehetővé teszi a folyamatok ábrázolását. Ennek szükségességéről a 2. fejezetben olvashatunk.

2. **Legyen könnyen használható:** Ne legyen túlzsúfolt az oldal, hanem törekedjen az átláthatóságra. Rajta a navigáció legyen következetes és mindenki számára egyértelmű. Ha az alkalmazás nyújtotta lehetőségeket egyszerűen, könnyedén lehet kezelni, az mindenképp pozitív élményt jelent a felhasználó számára.

    - Az alkalmazás nincs túlbonyolítva, kivitelezése átlátható, használata egyszerű. Ami pedig még inkább megkönnyíti a felhasználó eligazodását, hogy a műveletet végrehajtó gombokra, ha ráviszi az egérmutatót, akkor azok egy rövid leírást (*tooltip*et) jelenítenek meg számára, amely a könnyebb használatot segíti elő. Például a *mentés* gombnál leírja, hogy mi fog történni annak megnyomása esetén.

3. **Legyen kívánatos:** Fontos szempont a testreszabottság. Elvárás lehet a célközönség részéről a kifogástalan esztétikai élmény. Ezáltal lényeges, hogy minél letisztultabb felületet biztosítson az alkalmazás. Nem mindig a legfelkapottabb a célra legmegfelelőbb, viszont kívánatossá teheti a programot.

    - A program ennél az alapkövetelménynél hiányt szenved, ugyanis nem használja a legmodernebb és legnépszerűbb JavaScript könyvtárakat (sőt, egyáltalán nem használ semmilyen könyvtárat). Ugyanakkor megmutatja, hogy az alap JavaScript, HTML Canvas és némi CSS felhasználásával is elkészíthető egy ilyen alkalmazás.

4. **Legyen könnyen kereshető:** Ez az alapkövetelmény azt írja elő, hogy az adott oldalon / alkalmazásban a keresés hatékony legyen. A felhasználók hamar megtalálják azt, amit keresnek, és hogy egyáltalán az adott oldal rendelkezzen a felhasználó számára a keresett információval.

    - Ezt azért teljesíti könnyen a programunk, mert amint betölt az oldal, mindent a szemünk előtt látunk rajta. Alapvetően nincsenek rejtett dolgok, azonnal elkezdhetjük a folyamatmodellezést, minden egyszerűen a kezünk ügyébe kerül. Egyedül a mentés és a csomópont szerkesztés, ami bizonyos funkció végrehajtásakor jelenik meg a felhasználó számára, de ezt az 5. fejezet részletesen tárgyalja majd.

5. **Legyen hozzáférhető:** Vagyis mindenki számára elérhető. A blogbejegyzés különös hangsúlyt fektet a fogyatékkal élőkre, hiszen számukra kellene leginkább biztosítani a könnyű kezelhetőséget és használhatóságot. Ennek ellenére mégsem teljesül sok weboldal esetében ez az alapkövetelmény.

    - Az alkalmazásunk, mint ahogy azt a korábbi pontokban is már olvashattuk, egyszerű használatot és könnyű megértést kínál a felhasználó számára. Ezáltal bárki percek alatt megtanulhatja azt kezelni úgy, hogy képes legyen folyamatábrák készítésére.

6. **Legyen megbízható:** Tehát legyen optimális az oldal azáltal, hogy ne csak szép legyen, hanem gyors is. Ne legyenek rajta felesleges hivatkozások, vagy bármi nem oda illő tartalom, hiszen az összezavarhatja a felhasználót, ami azt eredményezheti, hogy nem fogja többször meglátogatni az oldalt. Ezek elkerülése pozitív benyomást kelthet a felhasználóban, hiszen azt találja az oldalon, amit keresett.

    - Nem tartalmaz a program felesleges mondanivalókat, csupán annyi rajta a tartalom, amennyire szükség van, se több, se kevesebb. A gyorsaságát többek között az adja, hogy nem használ semmilyen külső eszközt (például könyvtárat), illetve nem tartalmaz felesleges nagyméretű objektumokat, amik szintén lassíthatják egyes funkciók elérését.

# 4. fejezet

# Tervezés

## 4.1. Bevezetés

A specifikációk alapján a programban a felhasználónak kell tudnia csomópontokat hozzáadni, szerkeszteni, és azokat összekötni más csomópontokkal. Továbbá az éppen aktuális munkáját a gráf szerkesztőn el kell tudni menteni, és a program újraindításánál visszatölteni azt. Ez utóbbi megvalósításához szükség van valamilyen "tárolóra", vagyis egy szerverre és egy adatbázisra. Ennél a fejezetnél már nem csak a látható programrészen van a hangsúly, hanem legalább ugyanakkora figyelmet kap a szerver oldal is az adatbázissal, hiszen azok is kiemelkedő részei az alkalmazásnak.

## 4.2. Technológia és architektúra a szerver oldalon

Ahhoz, hogy a bevezetőben említett specifikációt megvalósító szoftvert meg tudjuk írni, először kijelöljük a megfelelő technológiákat és az architektúrát. A kliens-szerver architektúra megfelelő erre az alkalmazásra. A kliens egy böngészős frontend, ami egy API-n keresztül tud kommunikálni a backend szerverrel, amely perzisztensen tárolja az adatokat egy relációs adatbázisban. A kliens esetünkben a böngészőben futó **JavaScript** alkalmazás, míg a szerver egy **Python** nyelven íródott **Falcon API**-t használ, valamint **SQLite** adatbázist a tárolásra.

Hogy ez működjön a számítógépünkön, először telepíteni kell a **Python**-t, és a **pip**-et. A Python-t egyszerűen le lehet tölteni a hivatalos honlapról[**?**]. Windows felhasználóként a Windowshoz készült 64-bites setup segítségével telepítettem fel az alkalmazást. A **??** ábrán látható, hogy a Python telepítésével egyidejűleg lehetőségünk van a pip telepítésére is, ehhez azonban manuális telepítés szükséges.

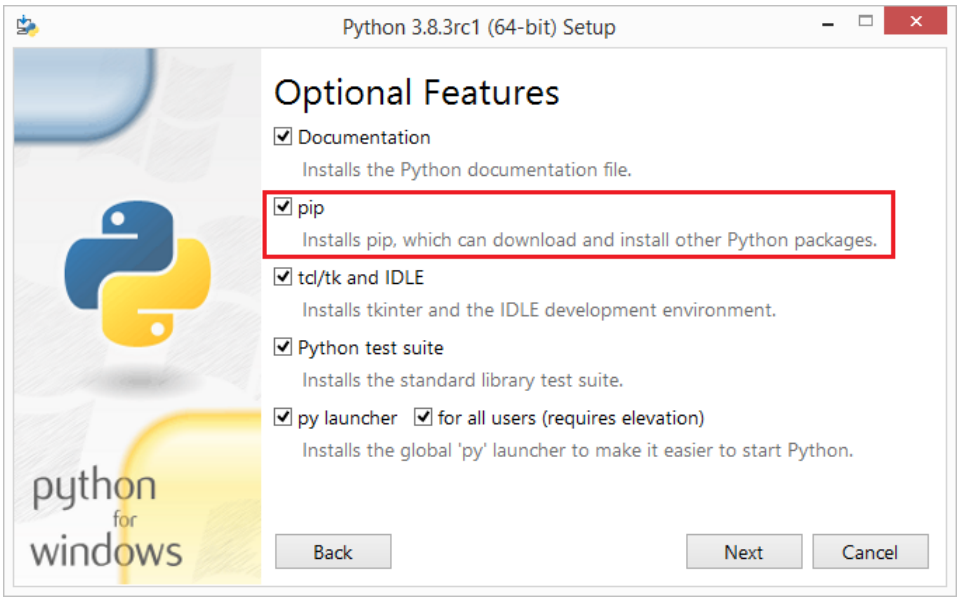
A telepítés következő lépésében célszerű bepipálni a 4. pontot, ami a környezeti változókhoz hozzáadja a Python-t (`Add Python to environment variables`), hiszen ennek elmulasztása esetén láthatósági problémák jelentkezhetnek használat közben.

Ezek után a program működéséhez elengedhetetlen csomagokat telepítjük: az **SQLAlchemy**-t és a korábban említett **Falcon**-t. A telepítés a következő formátum megadásával történik: `python -m pip install [csomag neve]`

Tehát például a Falcon-t az alábbi parancs kiadása telepíti:

```
python —m pip install falcon
```

Az SQLAlchemy-t pedig így a következő paranccsal fogjuk tudni telepíteni:




4.1. ábra. Python telepítése pip-pel együtt

```
python -m pip install sqlalchemy
```

Az SQLite3-at is érdemes telepítenünk, ha manuálisan akarjuk létrehozni az adatbázist és a táblákat. A telepítést itt is a hivatalos SQLite oldalon[?] keresztül tehetjük meg.

Ezt követően egy webszerver telepítése következik, például **waitress** vagy Unix-szerű rendszerek esetében a **gunicorn**, amivel elérhetjük a programot egy porton keresztül. Esetünkben a 8000-es portot használjuk. A backend szerver egy API-n keresztül szolgálja fel a statikus elemeket is, mint a HTML, CSS, JavaScript fájlokat.

Itt láthatjuk az útvonalakat, amelyeken az API felszolgálja a statikus elemeket:

```
api.add_route('/js/{filename}', StaticJS())
api.add_route('/css/{filename}', StaticCSS())
api.add_route('/{filename}', StaticHTML())
```

A backend programban a Falcon keretrendszer kezeli az API-t, és egy adatbázis-kezelő segítségével csatlakozunk az adatbázishoz az alábbi módon:

```
api = falcon.API()
dbms = mydatabase.MyDatabase(mydatabase.SQLITE, username='', password='',
dbname='mydb.sqlite')
```

Az adatbázis-kezelő osztály tartalmaz metódusokat, amikkel az alapvető műveleteket elvégezhetjük, mint például a lekérdezést, beszúrást, törlést.

```
# beszuras, torles
    def execute_query(self, query=''):
            if query == '' : return
            print (query)
            with self.db_engine.connect() as connection:
                try:
```

```
                    connection.execute(query)
                except Exception as e:
                    print(e)
def get_all_data(self, table='', query=''):
            query = query if query != '' else "SELECT *
FROM '{}';".format(table)
            print(query)
            returnData = []
            with self.db_engine.connect() as connection:
                try:
                    result = connection.execute(query)
                except Exception as e:
                    print(e)
                else:
                    for row in result:
                        print(row) # print(row[0], row[1], row[2])
                        d = dict(row.items())
                        returnData.append(d)
                    result.close()
            print("\n")
            return returnData
```

A szerver egy REST API-n keresztül hallgat a kérésekre. Ezen az API-n szolgálja fel a statikus fájlokat, köztük a kliens programot is, és itt kommunikál a klienssel. Az `/api/nodes` végponton HTTP GET kéréssel felszolgáljuk az adatokat JSON formátumban az alábbi módon:

```
[
   {
      "text":"folyamatElso",
      "height":50,
      "width":688,
      "radius":0,
      "y":540,
      "x":887,
      "type":"rectangle",
      "id":1
   },
   {
      "text":"folyamatMasodik",
      "height":50,
      "width":156,
      "radius":0,
      "y":512,
      "x":446,
      "type":"rectangle",
      "id":2
   },
   {
```

```
        "text":"",
        "height":0,
        "width":0,
        "radius":40,
        "y":122,
        "x":55,
        "type":"start",
        "id":3
    },
    {
        "text":"kezdo",
        "height":0,
        "width":0,
        "radius":40,
        "y":675,
        "x":1195,
        "type":"start",
        "id":4
    }
]
```

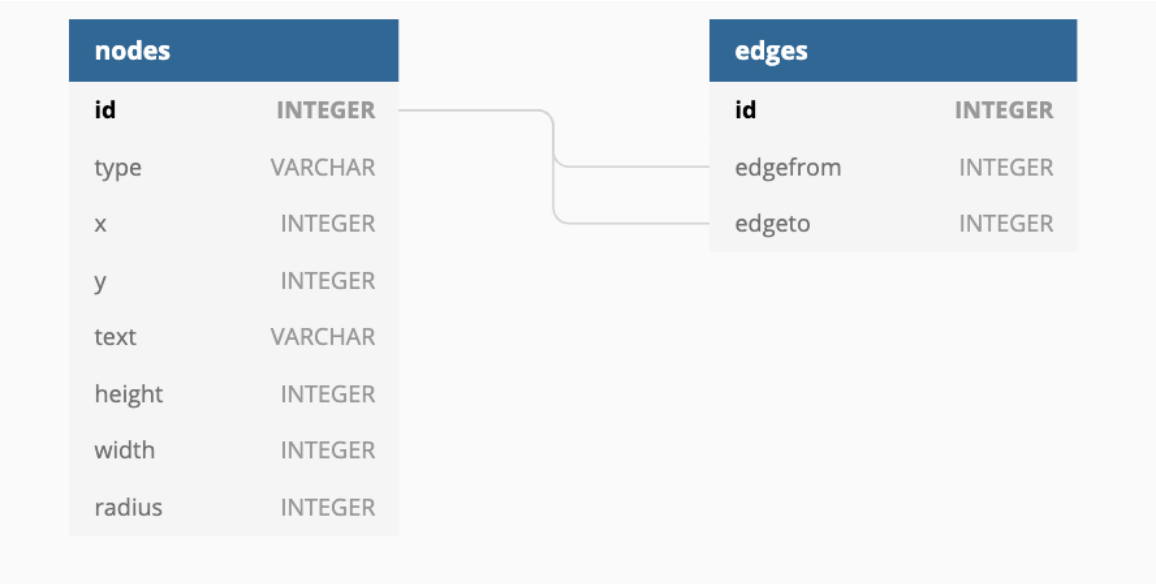
Ugyanezen a végponton HTTP POST kérés után mentjük az adatokat, amelynek a sikerességéről egy JSON válasz tájékoztat a következőképpen:

```
[
    {
        "text":"",
        "height":50,
        "width":100,
        "radius":0,
        "y":236,
        "x":678,
        "type":"rectangle",
        "id":1
    },
    {
        "text":"",
        "height":50,
        "width":100,
        "radius":0,
        "y":363,
        "x":1212,
        "type":"rectangle",
        "id":2
    }
]
```

Az API azért REST, mert követi a REST architektúrát, azaz a web URL és kérés metódus által pontosan azonosítható, mit akar a kliens. Például ha a metódus GET és az URL `/api/nodes` az azt jelenti, hogy a kliens le szeretné kérni az összes csomópontot.

A szerver a már fentebb említett SQLite adatbázissal is kommunikál, itt perzisztensen tároljuk az adatokat, így a következő indításnál is megmarad minden a szerkesztőben.

Az adatbázis két táblát használ, a **nodes** és az **edges** táblákat. Először manuálisan kell létrehozni őket, az alábbi ER diagram szerint:




4.2. ábra. SQL táblák ER diagramja

SQLAlchemy-ben így néz ki a fenti séma:

```
            nodes = Table(NODES, metadata,
                        Column('id', Integer, primary_key=True),
                        Column('type', String),
                        Column('x', Integer),
                        Column('y', Integer),
                        Column('text', String),
                        Column('height', Integer),
                        Column('width', Integer),
                        Column('radius', Integer)
                        )
            edges = Table(EDGES, metadata,
                        Column('id', Integer, primary_key=True),
                        Column('edgefrom', Integer,
ForeignKey("nodes.id"), nullable=False ),
                        Column('edgeto', Integer,
ForeignKey("nodes.id"), nullable=False)
                        )
```

## 4.3. A kliens oldal

A JavaScript kliens program 9 osztályból áll. Az osztályok mind külön feladatot látnak el. Az **Editor** osztály körül öleli a többi osztályt, és megosztja közöttük az információkat. A **Graph** osztály kezeli a szerkesztőt. A szerkesztő és tartalma megrajzolása ebben az osztályban valósul meg. Minden, a szerkesztőn megjelenő elem, ennek az osztálynak valamelyik adattagjában megtalálható. Két fontos adattagja a **nodes** és **edges** tömbök, melyek a **Node** és **Edge** osztályok, vagy ezen osztályok egyik leszármazottjának a példányait tartalmazzák. A Node osztály tartalmazza a csomópontokat a különböző alakzataival, míg az Edge osztály az alakzatokat összekötő vonalakat valósítja meg.

A Node osztálynak 3 közvetlen leszármazottja van: **Diamond**, **Rectangle** és **Circle** (magyarul: rombusz, téglalap és kör) melyek mind implementálják a Node osztály konstruktorát. A Circle osztálynak további két leszármazottja van: a **Start** és az **End**, melyek a kezdő- és végállapotokat jelölik.

Ezen osztályoknak mind vannak saját metódusai, melyek ugyanazokat a funkciókat látják el, csak a részletekben térnek el. Ilyen például a *resize()* és *draw()* metódus. Mivel minden alakzatot meg kell rajzolni, vagy adott esetben átméretezni, ezek a metódusok minden alakzatot megvalósító osztályban megtalálhatóak, és a legtöbb funkció végrehajtásakor ezek meg is hívódnak.

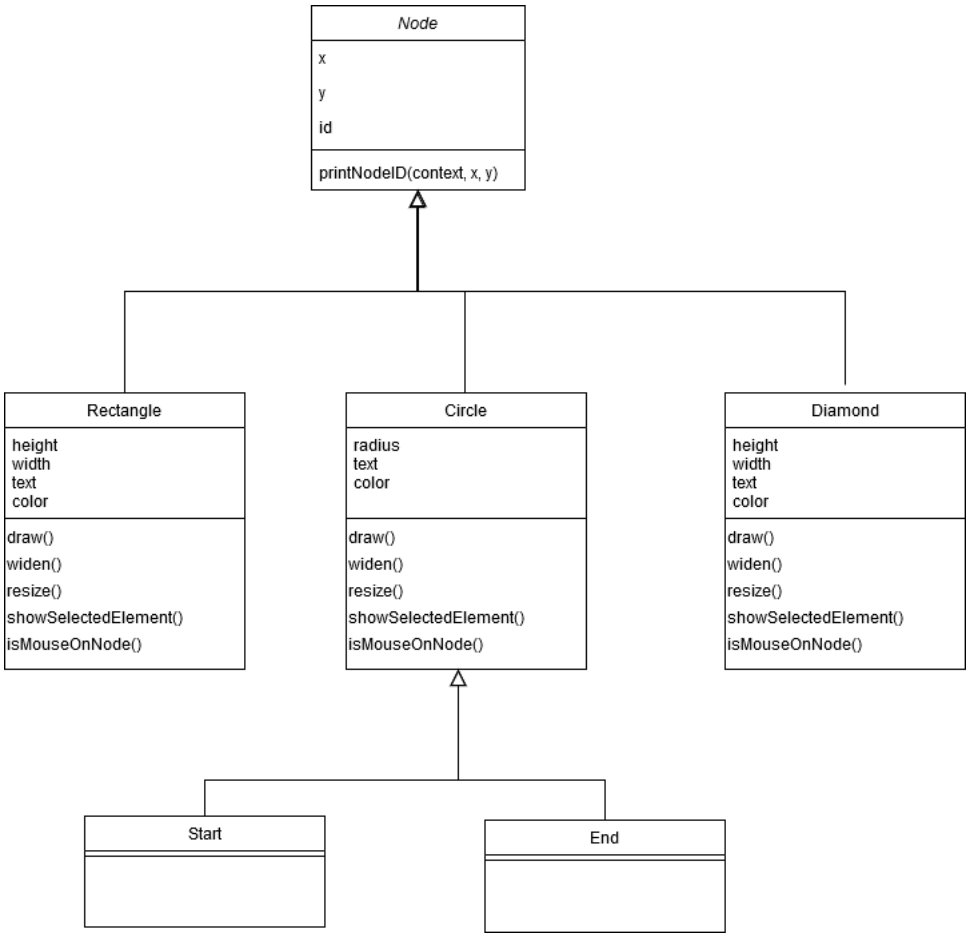
A frontend egy API-n, **HTTP kérések**kel kommunikál a szerverrel. A mentés úgy zajlik, hogy a frontend alkalmazás összegyűjti az összes csomópontot és az őket összekötő vonalakat, majd egy objektumot csinál minden egyes csomópontból és vonalból csak a szükséges adatokkal, ezt követően pedig hozzáadja egy tömbhöz. Ezután JSON formátumba alakítja a tömbben lévő objektumokat, amit egy **HTTP POST** kéréssel elküldünk a szervernek.

A lekéréshez **HTTP GET** kérést használunk, itt JSON formátumú tömböt kapunk benne a megfelelő adatokat tartalmazó objektumokkal. Ezeket az adatokat visszaalakítjuk, hozzájutva így a megfelelő osztályok konstruktorainak paramétereihez. Ezután a paraméterekkel osztályt hozunk létre belőle, és ezeket az osztályokat hozzáadjuk a Graph osztály nodes és edges adattagjaihoz. Innentől kezdve tudjuk reprezentálni őket a kliens alkalmazással.

# 5. fejezet

# JavaScript implementáció

## 5.1. A kliens program működése




5.1. ábra. A program UML diagramja

A kliens program a következőképpen működik: A graphEditor.html dokumentum betöltése után létrehozzuk az Editor osztály egy példányát. Ezután inicializáljuk a szerkesztőt egy függvény segítségével. Itt beállítjuk a program működéséhez elengedhetetlen adattagokat, események kezelését. Ilyen adattagok a graph, canvas, context, themes.

Végül egy metódus segítségével átméretezzük az adott böngészőablak méretéhez arányosra a canvas-t, biztosítva így, hogy bármilyen méretű képernyőn használható legyen a program.

Az indítás után a program egy aszinkron GET kéréssel lekéri az adatokat a szervertől JSON formátumban. Az adatokból létrehozza a megfelelő osztályokat a konstruktor paraméterekkel. Ezután a csomópontokat hozzáadja a nodes adattaghoz.

```
function GetEditorData() {
   fetch(apiURL+'/nodes').then(response => response.json())
       .then(data => {
           console.log(data);
           data.map(node => {
               let newNode = {};
               switch (node["type"]) {
                   case "rectangle":
                       newNode = new Rectangle(node["x"],node["y"],
                          node["id"],node["height"], node["width"],
                          node["text"], editor.graph.themes[editor.
                          graph.selectedtheme].rectangleColor);
                       break;
                   case "circle":
                       newNode = new Circle(node["x"],node["y"],node["
                          id"],node["radius"], node["text"], editor.
                          graph.themes[editor.graph.selectedtheme].
                          circleColor);
                       break;
    //...a tobbi csomopontra hasonlokeppen
               }
               let index = editor.graph.nodes.push(newNode) -1;
               editor.graph.selectedIndex = index;
               editor.addTextToNode(node["text"]);
           })
```

Ha ez sikerült, a függvény egy Promise-szal (ígéret) jelzi, és lekérhetjük az összekötő vonalakat. Mivel aszinkron az előző kérés is, előfordulhat olyan, hogy az összekötő vonalak hamarabb megérkeznek, mint maguk a csomópontok, és ez a programban végzetes hibához vezetne. Ezért van szükség a Promise-ra.

```
       .then(r => {
           //itt mar betoltottuk a nodes-okat
```

Ezután egy másik GET kéréssel lekérjük az összekötő vonalakat, és a megfelelő adatokból létrehozzuk az Edge osztály példányait, és hozzáadjuk őket az edges adattaghoz, így a következő rajzolásnál már látszódnak a gráfok.

A szerkesztő állapotát a következőképpen tudjuk elmenteni: A mentés gombra kattintva először átalakítjuk a nodes és edges adattagokban található osztályokat JSON adattá, majd egy aszinkron POST kéréssel mentjük az adatokat a szerveren.

A csomópontokat megvalósító osztályokat univerzálisan kezeljük: amilyen tulajdonsággal nem rendelkeznek, egyszerűen 0-ra állítjuk, úgyse foglalkozunk velük.

```

       let nodeJSON = {
           "type": type,
           "x": node.x,
           "y": node.y,
           "id": node.id,
           "text":node.text,
```

```
            "height": node.height ? node.height : 0,
            "width": node.width ? node.width : 0,
            "radius": node.radius ? node.radius : 0,
            //"theme": editor.graph.selectedtheme
        };
        nodes.push(nodeJSON)
```

Az összekötő vonalakat is csak azután küldjük el a szervernek, hogy megbizonyosodtunk arról, hogy a csomópontokat hibamentesen elmentettük. Az SQLite adatbázisban olyan idegen kulcs relációval kapcsolódnak a vonalak a csomópontokhoz, hogy ha a csomópont nem létezik, nem tudjuk elmenteni az összekötő vonalat.

Az eseménykezelés a következőképp zajlik: A billentyűk lenyomásához és felengedéséhez az editor megfelelő metódusát rendeljük.

```
window.addEventListener("keydown", editor.keyDown, false);
window.addEventListener("keyup", editor.keyUp, false);
editor.graph.canvas.addEventListener("mousedown", editor.mouseDown.
    bind(editor), false);
editor.graph.canvas.addEventListener("mousemove", editor.mouseMove.
    bind(editor), true);
editor.graph.canvas.addEventListener("mouseup", editor.mouseUp.bind(
    editor), false);
editor.graph.canvas.addEventListener("mousewheel", editor.mouseWheel.
    bind(editor), false);
```

Például a *mouseDown* eseménykezelő metódus úgy működik, hogy ha lenyomjuk az egér bal gombját, a függvény újra rajzoltatja a gráfokat, majd kiszámítja, hogy az egér rajta van-e valamely csomóponton. Ha rajta van, az lesz az éppen kijelölt csomópont, és a csomópont osztályban jelez, hogy húzható (*drag*) állapotba került, azaz az éppen kijelölt csomópont követi az egérmutató mozgását.

```
mouseDown(event)
{
    this.graph.clear();
    this.graph.draw();
    const mouse = this.calcMouseEvent(event);
    this.graph.selectedIndex = null;
    for (let i = 0; i < this.graph.nodes.length; ++i) {
        let node = this.graph.nodes[i];
        if (node.isMouseOnNode(mouse['x'], mouse['y'])){
            this.graph.selectedIndex = i;
        }
    }
    this.graph.dragStart = {
        x: event.clientX - this.graph.canvasPosition.left,
        y: event.clientY - this.graph.canvasPosition.top
    };
    this.graph.drag = true;
}
```

Ha a megfelelő helyre húztuk a csomópontot, és felengedtük az egér bal gombját, a *mouseUp* eseménykezelő metódus jelzi a csomópontokat kezelő osztálynak, hogy a csomópont már nem húzható, azaz nem követi tovább az egeret.

```
mouseUp(event, context)
{
    this.graph.clear();
    this.graph.draw();
    const mouse = this.calcMouseEvent(event);
```

```
6       this.graph.drag = false;
7   }
```

A csomópontokat úgy tudjuk összekötni, hogy a kiinduló (forrás) csomópontra rákattintunk, miközben nyomva tartjuk a `Shift` billentyűt. Egy másik (cél)csomópontra kattintva, miközben nyomva tartjuk az `Alt` billentyűt, kijelölhetjük az összekötendő csomópontot. A `Ctrl` billentyűvel, és bal egér kattintással törölni tudjuk a csomópontot.

Vonalat törölni szintén a Shift és a Alt billentyűk segítségével lehet (úgy, mint a létrehozásnál), itt viszont a jobb egérgombot kell lenyomnunk a billentyűk mellé.

Ezek között az `event.which` száma tesz különbséget: Az 1-es a bal egérgombra vonatkozik, a 3-as pedig a jobbra.

```
1   if (event.shiftKey === true) {
2       this.graph.drawEdge['from'] = this.graph.nodes[selectedIndex];
3   }
4   if (event.altKey === true) {
5       this.graph.drawEdge["to"] = this.graph.nodes[selectedIndex];
6       if (event.which === 1 ){
7           //node["id"];
8           this.addEdge(this.graph.drawEdge["from"], this.graph.drawEdge[
                "to"]);
9           this.graph.clear();
10          this.graph.draw();
11      } else if (event.which === 3){
12          for (let j = 0; j < this.graph.edges.length; ++j) {
13              if(this.graph.nodes[selectedIndex] === this.graph.edges[j
                    ].to)
14              {
15                  this.graph.edges.splice(j, 1);
16                  j = j-1;
17                  this.graph.clear();
18                  this.graph.draw();
19              }
20          }
21      }
22
23  }
24  if (event.ctrlKey === true) {
25      for (let j = 0; j < this.graph.edges.length; ++j) {
26          if(this.graph.nodes[selectedIndex] === this.graph.edges[j].to
                || this.graph.nodes[selectedIndex] === this.graph.edges[j
                ].from)
27          {
28              this.graph.edges.splice(j, 1);
29              j = j-1;
30          }
31      }
32      this.graph.nodes.splice(selectedIndex, 1);
33      this.graph.clear();
34      this.graph.draw();
35  }
```

Az automatikus és manuális méretnövelés a következőképpen működik: Minden csomópontnak van megfelelő metódusa a feladathoz, ami a csomópont saját tulajdonsága alapján elvégzi a feladatot. Például a *widen(length)* metódus elfogad egy paramétert, amely ha túl nagynak bizonyul, növeli a csomópont méretét. A téglalap esetében a

szélességet növeljük:

```
widen(length){
    if(15 > (this.width/length))
    {
        this.width = parseInt(this.width) + 14;
    }
}
```

Viszont például egy körnél a sugarat növeljük:

```
widen(length){
    if(8 > this.radius / length)
    {
        this.radius += 7;
    }
}
```

Átméretezni egy csomópontot a *resize()* metódussal lehet, itt is hasonlóan a csomópont tulajdonságaihoz megfelelő logikával:

```
resize(){
    this["width"] = parseInt(document.getElementById("width").value);
    this["height"] =  parseInt(document.getElementById("height").value)
        ;
}
```

## 5.2. Felhasználói kézikönyv a programhoz

Ez az alfejezet írja le a már korábban többször is említett konkrét kezelését az elkészült gráf szerkesztőnek.

### 5.2.1. Csomópont hozzáadása

A bal oldali eszköztárban rákattintunk a hozzáadandó elemre, így az megjelenik a szerkesztő felületen egy előre definiált helyen. Ezt követően az adott csomóponton a bal egérgombot nyomva tartva húzással tudjuk azt mozgatni. Fontos, hogy csak a szerkesztő felület területén belül, annak szélét nem érheti el, tehát nem tudjuk "levágni" a csomópont egy részét a canvas-on túli területre való mozgatással.

### 5.2.2. Csomópontok összekötése vonallal

Az összekötésnél figyelembe kell vennünk, hogy a program megkülönböztet forrás- és cél csomópontot, hiszen ez alapján dönti el, hogy a vonal melyik végén lesz a nyíl (vagyis melyik csomópontból melyik másik következik).

**Forrás csomópont kijelölése:** A szerkesztő felületen a `Shift` gomb lenyomása mellett az adott csomópontra bal egérgombbal való kattintás fogja kijelölni a forrás csomópontot.

**Célcsomópont kijelölése:** A Shift gombot elengedve, helyette az `Alt` gomb lenyomása mellett egy másik csomópontra való kattintás az egér szintén bal gombjával fogja meghatározni a célcsomópontot, vagyis azt, hogy hová lesz bekötve az adott vonal.

Ezt követően már meg is jeleníti a program a kívánt összeköttetést a két csomópont között.

### 5.2.3. Csomópontok közötti vonal törlése

Vonalat letörölni hasonlóképpen lehet, mint ahogy azt létrehoztuk. Itt először a `Shift` gomb lenyomásával egyidejűleg a jobb egérgombot kell megnyomnunk azon a csomóponton, amelyikből a törölni kívánt vonal kiindul. Ezt követően az `Alt` gomb nyomva tartása mellett szintén jobb egérgombbal való kattintás azon a csomóponton, amelybe vezet a vonal, fogja letörölni az adott vonalat.

A jobb kattintásra megjelenő, legtöbb böngészőbe beépített szerkesztő menü le van tiltva a canvas teljes területén, hogy vonal törlése esetén (ami jobb egérgomb kattintást igényel) ne zavarja meg a felhasználót.

### 5.2.4. Csomópont törlése

A törlést a `Ctrl` gomb nyomva tartása közben a törölni kívánt csomópontra való kattintás teszi lehetővé. Ilyenkor a csomóponthoz kapcsolt vonalak is törlésre kerülnek a csomóponttal együtt.

### 5.2.5. Szöveg hozzáadása csomóponthoz

Amelyik csomópontra szöveget szeretnénk írni, azon egy dupla kattintás meg fogja nyitni az eszköztár aljában a szöveg hozzáadásához szükséges részt. Ez a rész addig nem látható, amíg rá nem kattintunk egy csomópontra.

Az itt beírt szöveg azonnal megjelenik az adott csomóponton, illetve, ha a beírt szöveg hossza meghaladja a csomópont szélességét, akkor az automatikusan szélesedni kezd.

Ha lekattintunk az adott csomópontról, majd később visszakattintunk rá (bármelyikre, amelyiken szöveg található), akkor annak a szövege megjelenik azon a helyen, ahol előzetesen beírtuk a csomópont szövegét.

### 5.2.6. Csomópont átméretezése

Az átméretezni kívánt csomópontra való dupla kattintással (a szöveg hozzáadásához hasonlóan) érhető el a csomópont átméretezése. Az új méreteket az eszköztár aljában a kattintásra megjelenő felületen lehet megadni.

Hogy legyen mihez viszonyítani, az adott csomópont aktuális mérete megjelenik azon a helyen, ahová majd az új méreteket fogjuk megadni.

A téglalap esetén először annak szélességét, majd a magasságát kell megadnunk. A kör alakú csomópontok (circle, start, end) esetében csak a kör sugarát kell megadnunk.

A csomópontok egy előre definiált minimális méretnél kisebbre nem méretezhetőek. Ezek az értékek az alábbiak:

- Téglalap esetén 50x50,
- Rombusz esetén szintén 50x50 (ennek szélessége minden esetben megegyezik a magasságával),
- Kör esetén pedig 20

Miután beírtuk az új méreteket, az ezek alatt található `Resize` gomb megnyomása fogja átméretezni a kiválasztott csomópontot.

### 5.2.7. Téma megváltoztatása

Az adott témát megváltoztatni az eszköztár alatt található **graph themes** (gráf témák) felirat melletti legördülő menüben lehetséges. Egy másik téma választása újabb színekkel látja el a csomópontokat. Természetesen egyszerre több témát is használhatunk. Ilyenkor kiválasztunk egyet, felvisszük a kívánt elemeket, majd témát váltunk, és az új téma új színeivel további elemeket adunk hozzá a folyamatábrához.

### 5.2.8. Folyamatábra mentése

Az elkészült folyamatábrát a Témák melletti **Save** gombra kattintva menthetjük el. Ez a funkció minden, a szerkesztő felületen megtalálható elemet (csomópontok, összekötő vonalak, szövegek) elment. Viszont ha a Mentés gomb megnyomása után további szerkesztéseket hajtunk végre, és nem nyomunk ismételten mentés gombot, akkor ez utóbbi módosítások már nem lesznek eltárolva az adatbázisban.

Fontos még megjegyezni, hogy a gráf szerkesztő program a szerver futtatása nélkül is használható, viszont ebben az esetben (értelemszerűen) a mentés funkciót nem tudjuk használni.

# 6. fejezet

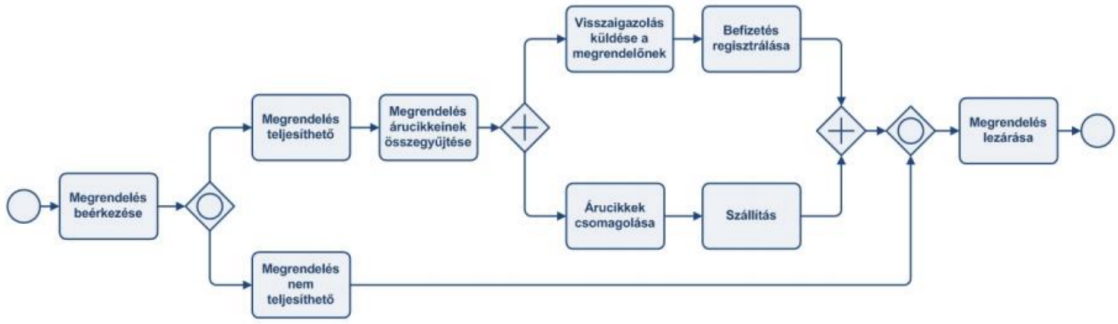
# Folyamat példák

## 6.1. Áttekintés

Ez a fejezet bemutatja néhány példa segítségével, hogy hogyan és milyen folyamatokat lehet a program elkészülte után annak használatával modellezni. A példák mellé leírás is társul, hogy érthető legyen a modellezett folyamat.

## 6.2. Első példa

Először nézzünk egy, már létező példát arra, hogy hogyan is néz ki egy olyan folyamatábra, amely egy üzleti folyamatot modellez. Ehhez tekintsük az alábbi *BPMN* (Business Process Model and Notation) diagramot (**??** ábra).



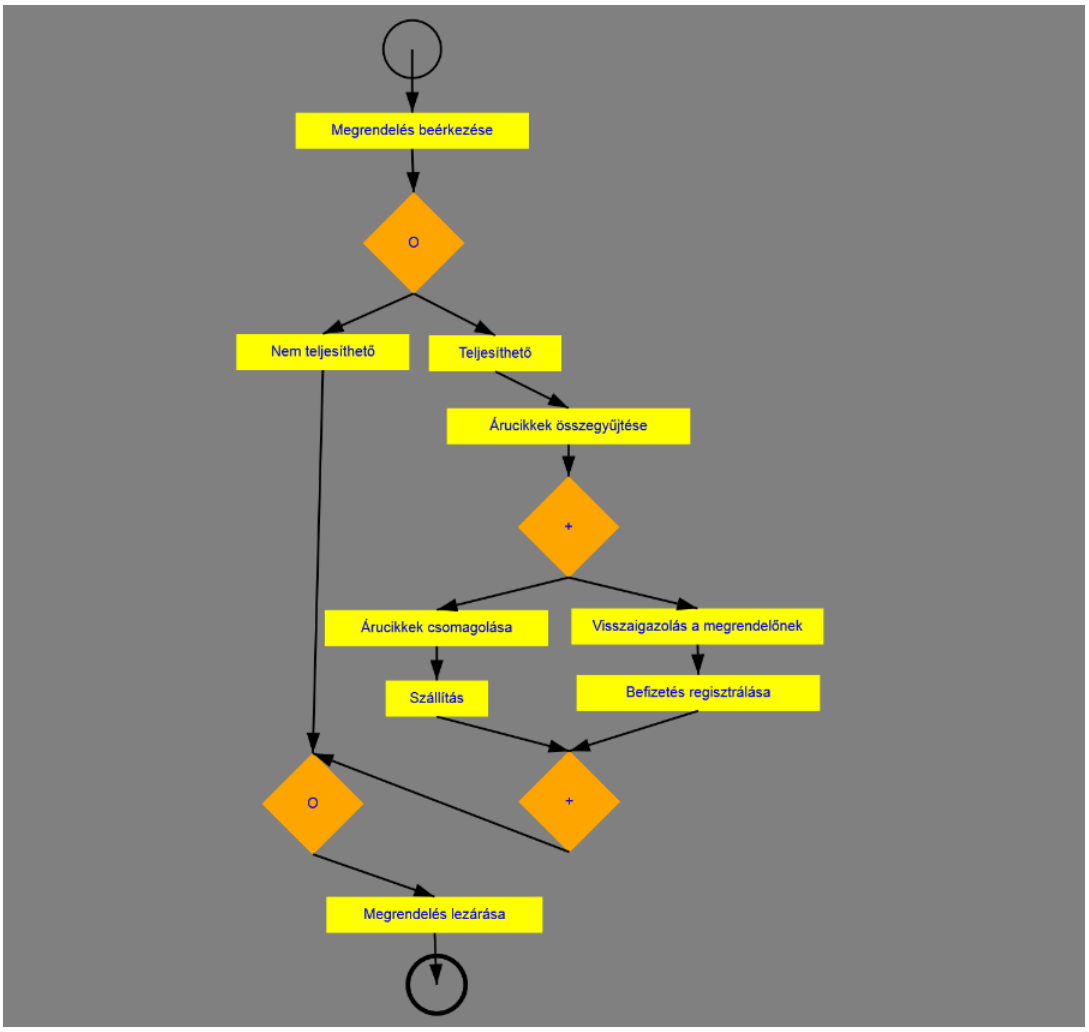

6.1. ábra. BPMN diagram (forrás: [**?**])

Az ábra egy kiváló példa teljes folyamatábrára: Van egy-egy kezdő- és végállapota, minden csomópontba el lehet jutni, és bármelyik csomópontból a végállapotba mehetünk a vonalak és a csomópontok mentén.

A folyamatábrán egy megrendelési folyamatot láthatunk. Első lépésben az adott vállalathoz beérkeznek az általuk megrendelt termékek. Itt máris útelágazáshoz érkezünk: ha a megrendelés nem teljesíthető valamilyen oknál fogva (például aktuálisan nincs rá kapacitása a cégnek, vagy nincs jelen az a személy, aki a megrendeléseket kezeli), akkor máris lezárásra kerül, és végállapotba jutunk a folyamatábrán. Ha viszont minden feltétel adott a megrendeléshez, akkor az teljesíthető. Ekkor összegyűjtik a megrendelés árucikkeit. Ezt követően egy újabb átjáró (vagy elágazás) következik. Láthatjuk, hogy az eddigi két rombusz elem más-más jelölést tartalmaz. Az előbbi egy

eldöntendő (igaz vagy hamis) csomópont volt, utóbbi után pedig párhuzamos feladat végrehajtás következik. Egyfelől az összegyűjtött árucikkeket becsomagolják, majd ezt követően elszállítják a megrendelés helyére, ezzel egyidejűleg pedig értesítik a megrendelőt egy visszaigazolással, és regisztrálják a befizetést. Két újabb rombusz alakzat zárja az azonos jellel megjelölt korábbi útválasztó csomópontokat, és végül lezárásra kerül a megrendelés. Az egész folyamat pedig egy kezdő- és egy végállapot közé van fogva.

A fenti ábra után tekintsük meg a **??** ábrán, hogy pontosan ugyanez a folyamat hogyan modellezhető a szakdolgozat programjának segítségével.




6.2. ábra. BPMN diagram a saját programmal készítve

Némi eltérés felfedezhető a két ábra között. A szakdolgozat alkalmazás színes csomópontokat használ, ami látványosabbá teszi a modellezést, azonban azok összekötésére kisebb hangsúlyt fektet. Előbbi ábrán a csomópontok szövegei több sorba való tagolással vannak megvalósítva, így minden tevékenységet jelölő csomópont azonos méretű. Hosszabb szövegek esetén viszont már nem lesz meg ez az egység. A szakdolgozat programjában ellenben minden csomópont pontosan azonos méretűre hozható a benne lévő szöveg hosszától függetlenül.

## 6.3. Második példa

Második példaként hozzunk létre egy létező diagram mintától függetlenül egy folyamatábrát. Tekintsünk egy szerződéskötési folyamatot, melyhez a(z) [**?**] forrásban leírtakat veszem alapul, amely forrás a dolgozat 2. fejezetéhez hasonlóan a(z) [**?**] hivatkozásban található egyik blogbejegyzés.

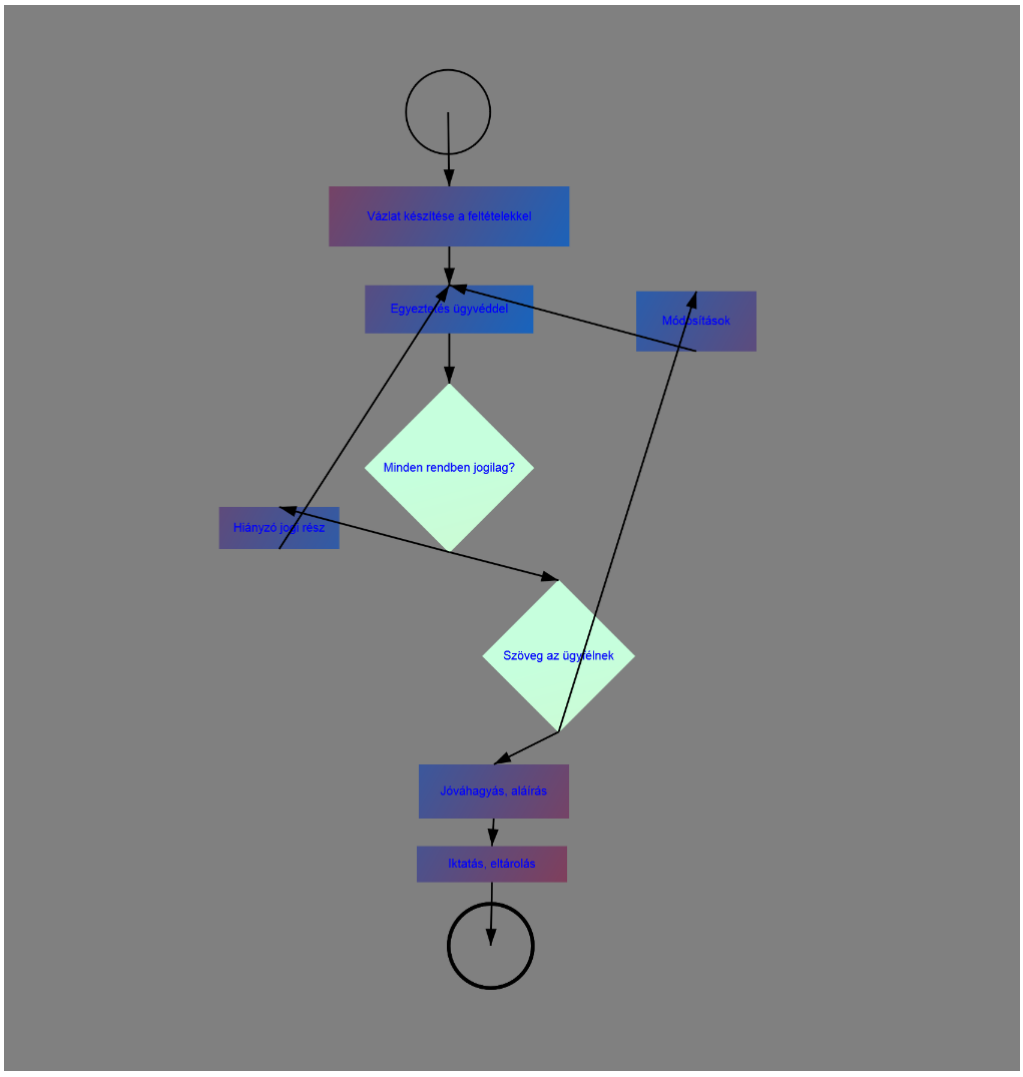
A folyamat előfeltétele, hogy már egy korábbi (ajánlattevési) folyamatot az ügyfél, akivel a szerződéskötés esedékes, elfogadott. Ezt követően kerül sor a pontos megegyezés (vagyis a szerződés) írásos formában való rögzítésére. Ennek egy egyszerű folyamatát írja le a forrásban említett blogbejegyzés az alábbi lépésekkel:

1. Első lépésben vázlatot kell készíteni a megegyezett feltételek mellett.
2. Ezt követi a vázlat egyeztetése egy ügyvéddel, vagy egy hozzá értő jogi személlyel.
3. Ha jogilag rendben van minden, akkor a megfogalmazott szöveg kiküldése az ügyfél számára a következő lépés.
4. Az ügyfélnek lehetősége van módosításokat kérvényezni, ebben az esetben a feladat ezek felülvizsgálata, a módosítások egyeztetése, visszaküldése az ügyfélnek.
5. A módosítások után célszerű ismét leellenőrizni, hogy jogilag a helyén van-e a szerződéstervezet.
6. Ha ez is megtörtént, már csak jóvá kell hagynia a két félnek a szerződést, azt aláírásukkal hitelesíteni.
7. Végül iktatásra kerül a szerződés, valamint eltárolásra.

A fenti folyamat ábrázolására többféle lehetőségünk is van. Nézzünk rá egy példát (lásd: **??** ábra).

Először átállítottam a témát, hogy ne csak az alapértelmezett legyen használva, hanem színátmenetes csomópontok is legyenek. Ezután, hogy ne csak tevékenység (téglalap alak) legyen benne, létrehoztam két átjárót is (rombusz alak), ugyanis ha jogi probléma merül fel (például hiányzik a megfogalmazásból valami), akkor ismételten szükség van az ügyvéddel való egyeztetésre, illetve az ügyfélnek küldött szövegen is vagy módosít az ügyfél, vagy következhet a jóváhagyás. Továbbá kihasználtam, hogy a csomópontokat át is lehet méretezni, ezáltal különböző nagyságokat állítottam be az egyes elemeknek.

A fent említett két folyamat példán keresztül még számtalan létezik, és természetesen a felhasználó, aki használni fogja a programot, magától is találhat ki, vagy módosíthat kedvére már létező üzleti folyamat modelleket.




6.3. ábra. Szerződéskötési folyamat modellezése

# 7. fejezet

# Összefoglalás

A dolgozat az üzleti folyamatok modellezésének megvalósítását célozta meg elméleti és gyakorlati példák bemutatásával, valamint a hozzá elkészített alkalmazással. Igyekezett felhasználóbarát központúvá tenni az alkalmazást annak működését és kinézetét illetően.

A program kliens oldalon JavaScriptet használt, amihez CSS-t alkalmazott, szerver oldalon pedig egy Python alapú mikrokeretrendszert, a Falcon-t. A gráf szerkesztés során használt adatokat SQLite adatbázisban tárolta.

Az alkalmazás a használat közbeni integritási feltételek ellenőrzésével próbált kiemelkedni a többi hasonló gráf szerkesztő alkalmazás közül. Azonban még sok továbbfejlesztési lehetőség tárul elénk, hiszen ezen ellenőrzések sokaságának köszönhetően további, bonyolultabb validálási feltételeket is meghatározhatunk a jövőben.

Ha lesz legalább egy szoftverfejlesztő csapat, aki felkapja az alkalmazást, és továbbfejleszti az integritási feltételeket ellenőrző részét, akkor már megérte elkészíteni az ehhez alapul szolgáló gráf szerkesztő programot.

# Irodalomjegyzék


[1] Forrás: `https://xflower.hu/blog/`, *Az üzleti folyamatokról bővebben* szavakkal kezdődő bejegyzések.

[2] Forrás: `https://hu.wikipedia.org`.

[3] Az oldal elérhető itt: `https://www.arukereso.hu/`.

[4] Forrás: `www.piacesprofit.hu`.

[5] Forrás: `https://app.diagrams.net/`.

[6] A blogbejegyzés, amely alapjául szolgált az alfejezetnek: `https://www.newconcept.hu/blog/felhasznaloi-elmeny-tervezes-ux-design`.

[7] Letölthető itt: `https://www.python.org/downloads/`.

[8] Hivatalos oldal a letöltéshez: `https://www.sqlite.org/download.html`.

[9] Forrás: `https://people.inf.elte.hu/fekete/algoritmusok_msc/workflow/hallgatoi_esszek/Bene_Katalin_Workflow.pdf`, 10. oldal, 2. ábra.

[10] Forrás: `https://xflower.hu/blog/`, "Az üzleti folyamatokról bővebben - Hogyan határozzam meg folyamataimat?" című blogbejegyzés.

# A melléklet tartalma

A dolgozat melléklete a következőket tartalmazza:

- `dolgozat.pdf`: A szakdolgozat PDF formátumban.
- `dolgozat`: A szakdolgozat LaTeX forráskódját tartalmazó jegyzék.
- `program`: Az elkészített programok forráskódja.

A dolgozat az elkészített program forráskódjával együtt `GitHub`-on is megtalálható az alábbi címen:

`https://github.com/utrymate/szakdolgozat`

## A program futtatása

Ha telepítettük a 4-es fejezetben leírt alkalmazásokat, akkor a következőképpen tudjuk elindítani a programot:

- A `program` jegyzékben található `server` jegyzékbe lépünk a Parancssor segítségével.
- Ha a `waitress`-t telepítettük, akkor az alábbi parancs indítja el a szervert: `waitress-serve -port=8000 server:api`
- A klienset a `http://localhost:8000/graphEditor.html` oldalon keresztül érhetjük el.
- Az adatbázis a `server` jegyzékben jön létre (ha még nem létezik) `mydb.sqlite` néven.