Szegedi Tudományegyetem
Informatikai Intézet

SZAKDOLGOZAT

Ujszászi János

2022

Szegedi Tudományegyetem
Informatikai intézet

DXF fájlba mentett épületterv dokumentum megjelenítése és felületszámítása

Szakdolgozat

Készítette:
Ujszászi János
BSc programtervező informatikus szakos hallgató

Témavezető:
Dr. Ferenc Rudolf
egyetemi docens

Szeged

2022

Feladatkiírás

Egy épület tervezése során mindig nagy munka mind a tervezőnek, mind az épület leendő tulajdonosának a költségek megállapítása. Ezek a számok az emberi tényező miatt gyakran hibásak, és már csak egy szakágon belül, egy felület mennyiségére is ritka, hogy a tervező, a kivitelező és a tulajdonos ugyanazt az értéket számolja ki. Mindhárom szereplő számára előre lépést jelenthet, ha a számított mennyiségek pontosak.

A hallgató célja egy online platform kialakítása, ahol a felhasználó feltöltheti a tervezőtől kapott DXF fájlt. A feltöltött fájlt parsolja, és tárolja szerver oldalon, a parsolt adatokat relációs adatbázisban menti, canvas objektumban megjeleníti. A számítási algoritmus elméleti szinten tárgyalja magas szintű tervezési metodikával.

A hallagató a feladat megvalósítása közben az előforduló problémaköröket kibontja detektálástól, hibakeresésen keresztül a megoldásig. A fellelt dokumentációk felhasználásának ismertetésével.

Tartalmi összefoglaló

- ***Téma megnevezése:***

DXF fájlba mentett épületterv dokumentum megjelenítése és felületszámítása

- ***A megadott feladat megfogalmazása***

A hallgató célja egy online platform kialakítása, ahol a felhasználó feltöltheti a tervezőtől kapott DXF fájlt. A feltöltött fájlt parsolja, és tárolja szerver oldalon, a parsolt adatokat relációs adatbázisban menti, canvas objektumban megjeleníti. A számítási algoritmus elméleti szinten tárgyalja magas szintű tervezési metodikával.

- ***A megoldási mód***

Egy HTML/CSS alapú online felületen drag&drop módon feltölthető platform kialakítása. A feltöltött fájl tárolása szerver oldalon és annak parsolás back end oldalon java osztály hierarchiába. Az osztályba mentett adatokat relációs adatbázisban is tárolom. Az osztályokban mentett egyedeket JSON adatformátumként kerül visszaadásra a frontend oldalra, ahol JavaScriptben kialakított bővebb osztálystruktúrába mentem a parsolt adatokat. Minden specifikus Entitás osztálynak felüldefiniált rajzolási eljárása készül, ami egy canvas hívást valósít meg.

A falazatok és egyéb objektumok meghatározására és mérésére kettő lehetséges eljárást ismertetek és a megvalósítás során gyűjtött tapasztalatok alapján.

- ***Alkalmazott eszközök, módszerek***

Az online felületet HTML5, CSS felhasználásával készítem el.

A parsolt fájl osztály leképezéseit H2 relációs adatbázisba mentem. Az adatbázisból a fájl érdemi része visszaállítható.

Backend:

- Java
- SpringBoot

Eredmény visszaírás: JavaScript

- ***Elért eredmény***

A kiírt feladatot sikerült megvalósítani. A fájlt tároltam és feltöltöttem szerver oldalra. Sikerült egy hatékony parsolo eljárást készíteni, amivel a teljes fájlstruktúrát egyetlen bejárással, osztályhierarchiában és adatbázisba tároltam. A megjelenítés során a fontos és elengedhetetlen objektumokat visszarajzoltam és arányosítottam az browser méretéhez.

A kódolás során modul és E2E teszteléseket hajtottam végre.

Kalkulációs megoldások kidolgozása és azok elemzése.

- ***Kulcsszavak***

DXF feldolgozás, parsolás, canvas objektum, osztály struktúra, visszarajzolás, kalkuláció, tervező, megrendelő, kivitelező

Tartalomjegyzék

Feladatkiírás 2
Tartalmi összefoglaló 3
Bevezetés 7
1. Lakóház költség meghatározása 9
1.1 Tervezői költség meghatározása 9
1.2 Kivitelezői költség meghatározás 9
1.3 Megrendelői költség meghatározás 10
2. DXF felépítése, adatkapcsolatok kapcsolatok 12
2.1. Általános ismertető 12
2.2. Felépítése 12
2.2.1. Header 12
2.2.2. Classes 13
2.2.3. Tables 15
2.2.4. Blocks 16
2.2.5. Entities 16
2.2.6. Object 17
3. User interfész 18
3.1. Beviteli képernyő 18
3.2. Eredmény visszajelző képernyő 19
4. Parsolás kezelése 21
4.1. BackEnd oldali adatszerkezetek használata és osztály struktúra 21
4.2. Parsolás menete 22
4.3. Parsolt adat átadása FrontEnd felé 22
5. Adatbázis felépítése, és tárolás 23
5.1. Táblák 23
6. Elemek visszarajzolása Javascripttel 25
6.1. Osztály struktúra 27
6.2. Segédeljárások az ENTITY osztályban aszámtásokhoz 28
6.3. Entitás típusokhoz felüldefiniált draw eljárások bemutatása 29
6.3.1. 3dface 30
6.3.2. Arc 31
6.3.3. LINE 32

6.3.4. Circle 33
6.3.5. AttDef, MTEXT, TEXT, AttRib 34
6.3.6. POINT 34
6.3.7. LWPOLYLINE 35
6.3.8. SOLID 36
6.3.9. HATCH 37
6.4. Arányok kezelése 40
7. Számítási algoritmus megoldási lehetőségei elméleti szinten 40
7.1. Objektum felismerés és mérése 40
7.1.1. Képfeldolgozás pásztázással 40
7.1.2. Folytonos HATCH keresés 40
8. Alkalmazás tesztelése 42
8.1. Modul szintű tesztelés 42
8.2. EndtoEnd tesztelés 42
Irodalom jegyzék 48
Nyilatkozat 49
Köszönetnyilvánítás 50
Mellékletek és elektronikus melléklet 51

Bevezetés

Egy épület tervezése során mindig nagy munka mind a tervezőnek, mind az épület leendő tulajdonosának a költségek megállapítása. Ezek a számok az emberi tényező miatt gyakran hibásak, és már csak egy szakágon belül, egy felület mennyiségére is ritka, hogy a tervező, a kivitelező és a tulajdonos ugyanazt az értéket számolja ki. Mindhárom szereplő számára előre lépést jelenthet, ha a számított mennyiségek pontosak.

A szereplők között egy tévesen meghatározott érték, már a bizalmi kapcsolat kialakulása előtt, rányomhatja a bélyeget a későbbi kapcsolatra.

- A tervező:
    - Hiányosan jelöli ki tervező szoftverbe a számításban érintett objektumokat,
    - Vagy csak rosszul állítja be az objektumra vonatkozó számítási paramétereket.
- A tulajdonos:
    - Excel táblás struktúrában szinte 100%-os a hibázás lehetőség
    - Hozzá nem értés, egy objektum valós méretét tekintve.
- A kivitelező:
    - Beleszámolja az ablak helyét is a falazásba.
    - Felhasznált anyag alapján számol.

Ezek a példák is jól mutatják, hogy nagyon nehezen tud jól indulni egy ilyen kapcsolat. Ahhoz, hogy egy mindenki számára használható eszközt készítsek, fel kell mérni a szükséges igényeket.

Alapvető igények, elvárások az alkalmazással kapcsolatban:

- Tervezői oldalról egyértelmű elvárás, hogy a szellemi tőke megmaradhasson a tervezőnél. Ezért a nyers tervezésben érintett fájl formátum felhasználása nem lehet opció.
    - Mindhárom oldalról elvárás a kényelem és a gyors számítás. Automatizálással több embernapnyi munka váltható ki egyszerűen
- Mindhárom oldalról elvárás az egyszerűség
    - Egy újabb bonyolult, több opciós paraméterezéssel működő eszköz nem használható hatékonyan

- Eredmények átláthatósága.
    - Opciót kell biztosítani a felhasználó felé, hogy meghatározza, az eredményben visszaadott objektum típusokat, egyedeket.
    - Amennyiben nem definiálja elvárásait a felhasználó, akkor mindent vissza kell adni, ezzel elkerülve, hogy a rengeteg adatban elvesszen a mezei felhasználó.

1. Lakóház költség meghatározása

1.1 Tervezői költség meghatározása

Minden épület tervezés ügyfél egyeztetésekkel indul. Az ügyfél szeretne egy épületet, ami az igényeinek megfelelő és ehhez van egy reális vagy éppenséggel irreális költségkerete. A tervező feladatai közé tartozik, hogy a költség keretnek megfelelő épületet tervezzen. Ez a feladat egy a sokból, és ez csak egy paraméter az épület tervezése során, ami sok feladat mögé kerül, mert fontosabb, hogy az épület:

- biztonságos legyen
- megfeleljen a helyi és országos építésihatóság által előírt szabályoknak
- megfelelő és időtálló anyagokkal valósuljon meg a kivitelezés
- illeszkedjen a helyi és/vagy területi építési környezetbe, stílusba

Sajnos a költségvetési kiírás sok esetben messze van a gyakorlati megvalósítástól, illetve az objektumok alapján számol. Ergo, ha egy objektum nincs megnevezve, címkézve azt csak manuálisan tudja hozzátenni a tervező. Ideális esetben minden lerajzolt vonal, vagy réteg költsége becsülhető lenne, de ehhez minden lerajzolt vonalhoz azonosítás szüksége, hogy az az objektum éppen micsoda. Ez a címkézés időigényes. A tervezők számára ideális megoldás lehet viszont az, hogy ha a tervezés során részfázisokat kiexportálnak, és azokra végeztetnek költségelemzést.

1.2 Kivitelezői költség meghatározás

A mai építőipari környezetben egyetlen hibásan elkészített árajánlat komoly veszteséget jelenthet egy kivitelező számára, nem csak a félreszámolás miatt, hanem a folyamatos áremelkedés okán is.

Több szempontból sem előnyös, ha a kivitelező számolja ki a mennyiségeket, mivel

- időt és energiát visz el az értékesebb helyszíni szak tevékenységtől
- általában a számolások munka után második műszakban történik, ami növeli a hiba lehetőséget
- hiba esetén bizalmatlanság alakulhat ki.
- visszaélésre adhat lehetőséget mind az ügyfél, mind a kivitelező szemszögéből.

Az ideális támogató rendszer kiszámolja a mennyiségeket és anyag bontásba adja át a kivitelező felé azokat. Egy leolvasott mennyiséghez nagyon egyszerű a rétegrend további elemét kikalkulálni.

Szemléltetés egyszerű példával: Külső falazat.

- Bentről kifelé indulva.
    - A falazatot, csupasz falat belülről borítja valami.
        - Burkolat
            - A burkolat alá vakolat kell
            - Vagy gipszkarton
                - alá vázszerkezet
                - vagy ragasztó
        - Festék
            - A festék alá glett
                - alá vakolat
                - vagy gipszkarton

Ezzel az egyszerű példával szemléltehető, hogy egy meghatározott külső falazat belső felületéhez mennyi különböző anyag mennyiség, és munkabér költség határozható meg.

1.3 Megrendelői költség meghatározás

A megrendelő a tervezői és kivitelezői hibáknak és visszaéléseknek az elszenvedője. Ha nincs a kezében egy eszköz vagy nem kér fel borsos áron ellenőrt, akkor marad az Excel tábla és több hetes számolgatási procedúra.

Az ilyen jellegű ellenőrző tevékenység a következő problémákat hordozza:

- Ha már ellenőrzés szükséges a részünkről akkor már probléma van.
    - Vagy a tervező rontott el valamit vagy a kivitelező számolt rosszul, de a két szám eltér, ezért szükséges az ellenőrzés.
    - Az ügyfél szintű ellenőrzés általában kezdetleges eszközökkel valósul meg, rutin és tapasztalat nélkül.
    - Nagy mértékű hibalehetőség
    - Gyakori hiba kellemetlené teszi a kapcsolatot, a hitelesség csökken.

- Kezdeti költségelemzés meghatározásakor nagy mértékű pontatlanság.

A probléma gyökerét tehát az jelenti, hogy nincs hatékony segítség a megrendelő kezében.

2. DXF felépítése, adatkapcsolatok kapcsolatok

2.1. Általános ismertető

A DXF egy fix szabályrendszer alapján felépített, ASCII-alapú szöveges állomány. Két egymást követő sor mindig összetartozik. Az első sor egy típuskódot ad meg, ami három karakter hosszú balról padolva SPACE karakterrel. a második pedig a típushoz tartozó értéket írja le. A fájl több szekcióból állhat, de a szekciók közül csak az ENTITIES kötelező. Az ENTITIES szekció tartalmazza a rajzelemek kirajzolásához szükséges alapvető geometriai adatokat. A fájl végén EOF kulcsszó található.

Minden szekció előtt meg kell adni, hogy egy új szekció következik a „ 0”, SECTION kóddal, ezt követően „ 2”, Szekció neve következik. A tartalmat követően a szekció „ 0” kóddal és „ENDSEC” kulcsszóval zárul.

2.2. Felépítése

A következő alfejezetben ki fogom fejteni a dokumentáció alapján felhasználható szekciókat és ismertetem azok felhasználási lehetőségeit.

A DXF fájlformátum kötött és a szekciók sörrendje kötött, de az ENTITIES szekción kívül, a szekciók szabadon elhagyhatok. Ettől függetlenül érdemesnek tartottam mindegyik szekcióról egy rövid leírást készíteni még akkor is, ha ezeket a szekciókat a feldolgozó logika nem érinti. A rövid bemutatás a formátum sokrétűségére és megannyi felhasználhatóságára próbál rámutatni.

2.2.1. Header

A HEADER szekcióban az általános fájladatok találhatóak a beállított változók formájában. A változók elnevezése általában $ karakterrel kezdődnek. A teljesség igénye nélkül, itt található például a fájl formátumának verzió száma, a kép méretéhez köthető különböző változók, az entitásokhoz társítható általános preferenciák, mint például, hogy egy körív előre definiált forgásiránya ($ANGDIR) óramutató járásával ellentétes, vagy nem.

$EXTMAX	10, 20, 30	X, Y, és Z koordináta értékek, amik leírják egy a rajztér legfelső jobb sarkát.
$EXTMIN	10, 20, 30	X, Y, és Z koordináta értékek, amik leírják egy a rajztér legalsó jobb sarkát.

2-1. táblázat Header formátum leírás hivatalos dokumentációból magyarosítva

Az dolgozatban a HEADER szekcióból fogom meghatározni a rajztér méretét, aminek a jellemzőit az 1. táblázat mutatja be. A fájl parsolás és visszarajzolás szempontjából is fontos adatokat tartalmaz.

2.2.2. Classes

A CLASSES szekcióban azok az osztályok találhatóak, amiket az exportáló alkalmazás példányosít és később megjelennek a BLOCKS, ENTITIES és OBJECT szekcióban. Az osztályokat előre definiált mezőkkel lehet leírni és minden mezőnek a kitöltése kötelező. A mezők a 2. táblázatban találhatóak.

CLASSES szakasz csoportkódok	
Csoportkód	Leírás
0	Bejegyzés típusa (CLASS). Meghatározza egy CLASS bejegyzés kezdetét
1	Osztály DXF-bejegyzésneve; mindig egyedi
2	C++ osztálynév. Olyan szoftverekhez kapcsolva használható, amelyek meghatározzák az objektumosztályok viselkedését; mindig egyedi
3	Alkalmazás neve. A Riasztás mezőben teszi közzé a rendszer, amikor egy ezen szakaszban listázott osztálydefiníció aktuálisan nincs betöltve

90	Proxyképességek jelzője. Bitkódolt érték, amely megadja ezen objektum proxyként meglévő képességeit: 0 = Semmilyen művelet nem engedélyezett (0) 1 = Törlés engedélyezve (0x1) 2 = Transzformáció engedélyezve (0x2) 4 = Színváltoztatás engedélyezve (0x4) 8 = Fóliaváltoztatás engedélyezve (0x8) 16 = Vonaltípus-változtatás engedélyezve (0x10) 32 = Vonaltípuslépték-változtatás engedélyezve (0x20) 64 = Láthatóságváltoztatás engedélyezve (0x40) 128 = Klónozás engedélyezve (0x80) 256 = Vonalvastagság-változtatás engedélyezve (0x100) 512 = Nyomtatási stílus névváltoztatása engedélyezve (0x200) 895 = Minden művelet engedélyezve, kivéve a klónozást (0x37F) 1023 = Minden művelet engedélyezve (0x3FF) 1024 = Letiltja proxyfigyelmeztető párbeszédpanelt (0x400) 32768 = R13 formátum proxy (0x8000)
91	Egyéni osztály példányszáma
280	Proxy volt jelző. 1 értékre állítva, ha az osztály nem volt betöltve ezen DXF-fájl létrehozásakor, egyéb esetben 0
281	Rajzelem jelző. 1 értékre állítva, ha az osztály az AcDbEntity osztályból származik, és a BLOCKS vagy az ENTITIES szakaszban tárolható. Ha értéke 0, előfordulhat, hogy a példányok csak az OBJECTS szakaszban jelennek meg

2-2. táblázat Classes szekciót leíró adatok hivatalos dokumentációból

Vannak előre definiált osztályok is, amik fix elnevezési párosítással érthetőek el. Ennek a leírásnak a formátuma a 3. táblázatban látható.

Alapértelmezett osztály értékek				
DXF rekord név kód 1	C++ osztály név kód 2	Kód 90	Kód 280	Kód 281
ACDBDICTIONARYWDFLT	AcDbDictionaryWithDefault	0	0	0

2-3. táblázat Default classes

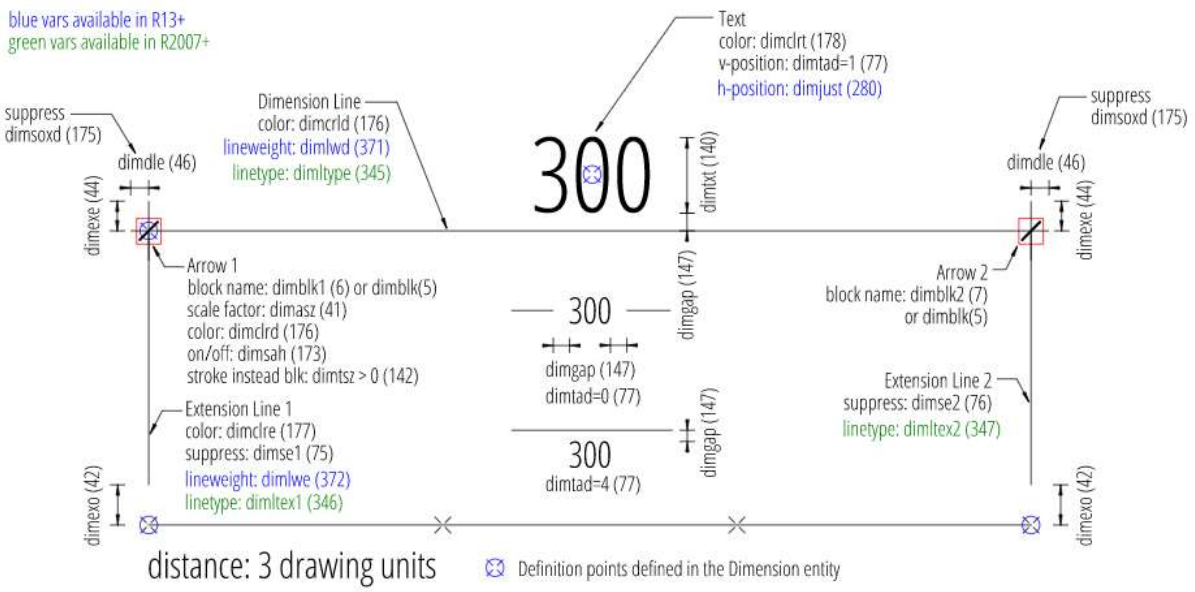
Az dolgozatban a CLASSES szekciót nem dolgozom fel. Figyelmen kívül hagyom, mert a visszarajzolás szempontjából nem releváns.

2.2.3. Tables

A TABLE szekcióban a 4.táblázatban megtalálható típus definíciók deklarálhatók. Az 1. ábra egy DIMSTYLE objektum teljes leírását is bemutatja.

Név	Kód	Leírás
Linetype table	LTYPE	A DXF fájlban megtalálható LINE entitásnál megjelenő vonaltípus definíciókat tartalmazza.
Layer table	LAYER	Rétegeket lehet definiálni
Text Style table	STYLE	A szöveg stílusokat tartalmazza
View table	VIEW	A rajzterület elrendezéseinek nézeteit tárolja. Nem befolyásolja a rajzot, de az AUTOCAD alkalmazásnak segít a feldolgozásban.
User Coordinate System table	UCS	Nevesített vagy nem megnevezett felhasználói koordináta rendszer, ami a CAD alkalmazások használnak.
Viewport configuration table	VPORT	A DXF fájlban megtalálható VIEWPORT entitásnál megjelenő elem definíciókat tartalmazza.
Dimension style table	DIMSTYLE	A DIMENSION Entitások stílusmeghatározása
Application Identification table	APPID	Alkalmazásoknak fenntartott definíciós lehetőségek

2-4. táblázat Table szekció típus definiciók



2-1. ábra DIMSTYLE ábrázolása (https://ezdxf.readthedocs.io/)

Az dolgozatban ezt szekciót nem dolgozom fel. Figyelmen kívül hagyom, mert az alapobjektumok visszarajzolása szempontjából nem releváns információt hordoz.

2.2.4. Blocks

A blokkok entitások gyűjtemény, amik több példányban elhelyezhetők a rajztérben, eltérő elrendezésben, eltérő helyen. Egy blokk bejegyzés a BLOCKS szekcióban BLOCK bejegyzéssel kezdődik és ENDBLK-val zárul közben INSERT, ATTRIB, ATTDEF block referenciákat kezel. Az dolgozatban a szekciót nem dolgozom fel. Figyelmen kívül hagyom, mert a vizsgált tesztfájlok adatai alapján nem érdemi a hivatkozott entitások előfordulásának mennyisége.

2.2.5. Entities

Az EINTITIES az egyetlen kötelező szekció. A rajzi elemeket tartalmazza. Az ENTITY-k „ 0” kóddal kezdődik. ez mutatja meg a ENTITY típusát. Az ENTITIES végét szintén ENDSEC zárja. Közben akármennyi ENTITY előfordulhat. A szekciónak előforduló entitásoknak vannak általános típus független és típusfüggő jellemzői. Az általános jellemzők a „ 0” típus definíció, elnevezések, hivatkozások más szekciókra. A specifikus jellemzők között

is átjárás van, de vannak kirajzolhatósághoz elengedhetetlen jellemzők, amiket specifikusan a 6.2 fejezetben fogok kifejteni.

2.2.6. Object

Az ENTITIES-hez hasonló általános tulajdonságokkal rendelkező szekció, de itt nem grafikus, megjelenítendő elemek találhatóak. Az OBJECT szekcióban megtalálható elemek például a DICTIONARY, GEODATA, MATERIAL.

Az dolgozatban a szekciót nem dolgozom fel. Figyelmen kívül hagyom.

3. User interfész

A User interfész egyképernyős webalkalmazásban jelenik meg. Az alkalmazás UI kialakítása során HTML, CSS, és Javascript technológiák falhasználása történt. Az animációk megvalósítása @keyframes használatával készült. A visszarajzolási felület canvas

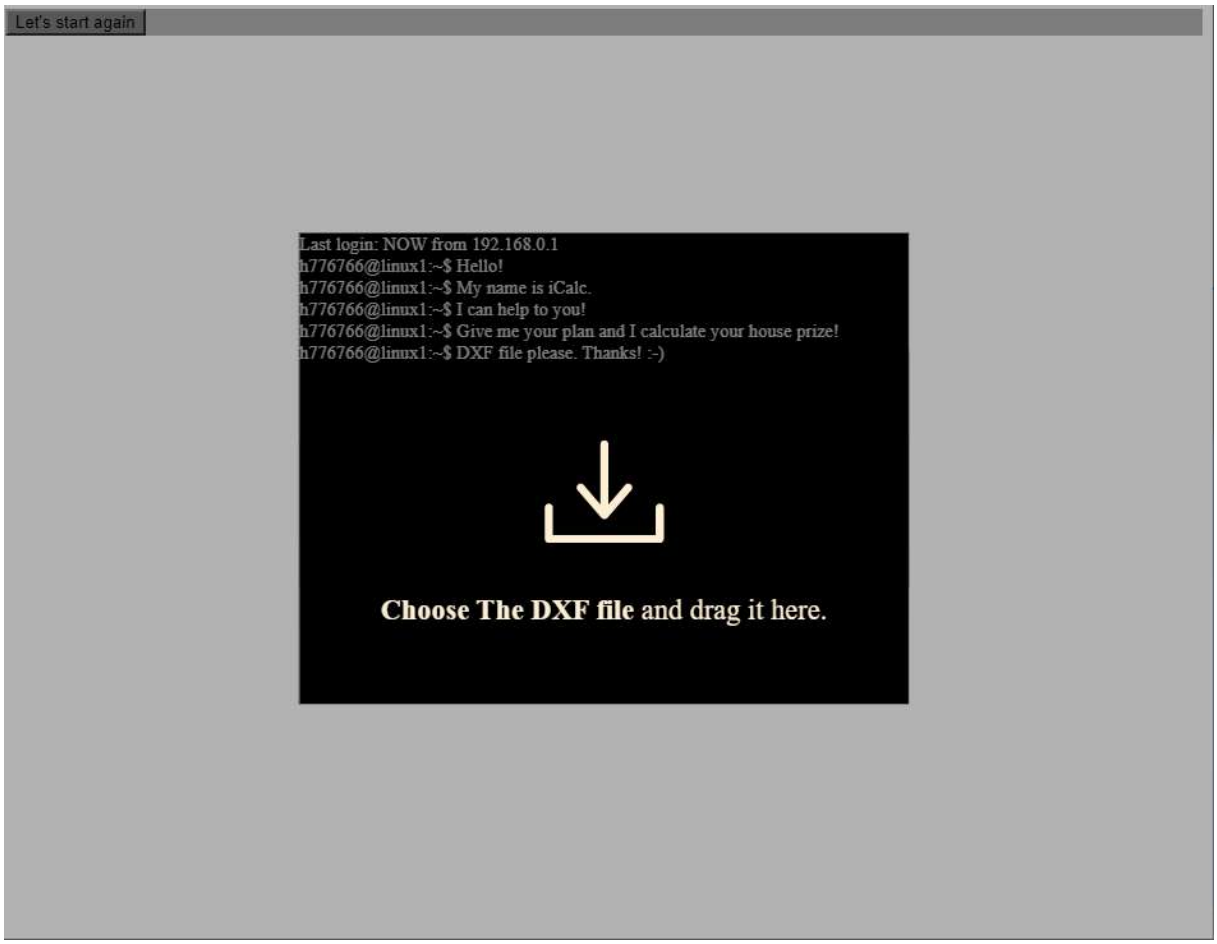
3.1. Beviteli képernyő

Az alkalmazás induláskor megjelenő háttár egy adásmentes TV képernyőre hasonlítható, amiből kettő másodperc elteltével egy terminál lesz látható. A terminálban üdvözlő üzentek, illetve bemutatkozás jelenik meg sorról sorra.

Az animációk CSS stíluslappal egymás után mennek végbe a következő sorrendben:

- A @keyframes backgroundchg segítségével a kezdeti noise.gif képet először fehérre módosul
- Majd ugyanebben a @keyframe-ben szürkére.
- Közben a megjelenítem fokozatosan a terminál felületét
- A terminálban miután a háttér szürkére váltott a @keyframes cursor-visible animációval egymás után jelenítem meg a kiírt szöveget.
- Végül megjelenik a drag&drop area-t (2. ábra)

A bemutatkozást követően előtűnik a drag&drop felület ahová a felhasználó feltöltheti a kezelendő DXF fájlt. A fájl felületre mozgatása módosítja a feltöltés alatt megtalálható szöveget A fájl elengedés és a feldolgozó eljárás megkezdése szintén visszajelzésre kerül a felhasználó számára. Ha fájl mérete túl nagy vagy a kiterjesztése nem DXF, azt alert formájában visszajelzem a felhasználó felé, és ismételt fájlfeltöltés lehetséges.



3-1. ábra kezdő képernyő

3.2. Eredmény visszajelző képernyő

A beolvasást követően a képernyőről eltűnik a terminál szöveges felület és browser méretével arányosan 90%.-osra nagyítom a terminál DIV objektumát. Az animációt követően a terminál DIV objektuma eltűnik és ugyanebben a pillanatban megjelenítem az azonos méretű és pozíciójú canvas objektumot. A canvas jobb oldalának 220 pixel nagyságú részében egy calculation rész helyezkedik el, ahova majd a számítási eredményeket írnám vissza. A maradék rajzfelület 1%-os kertet kap. A képernyő végállapotát a 3. ábra mutatja be. A visszarajzolás során a rajzokat arányosítani kell, ezért minden esetben az eredeti méretarányokat megtartva adom vissza az eredményt a felületre. Az arányosítással kapcsolatos teljes leírás az 5.3. fejezetben lesz elérhető.

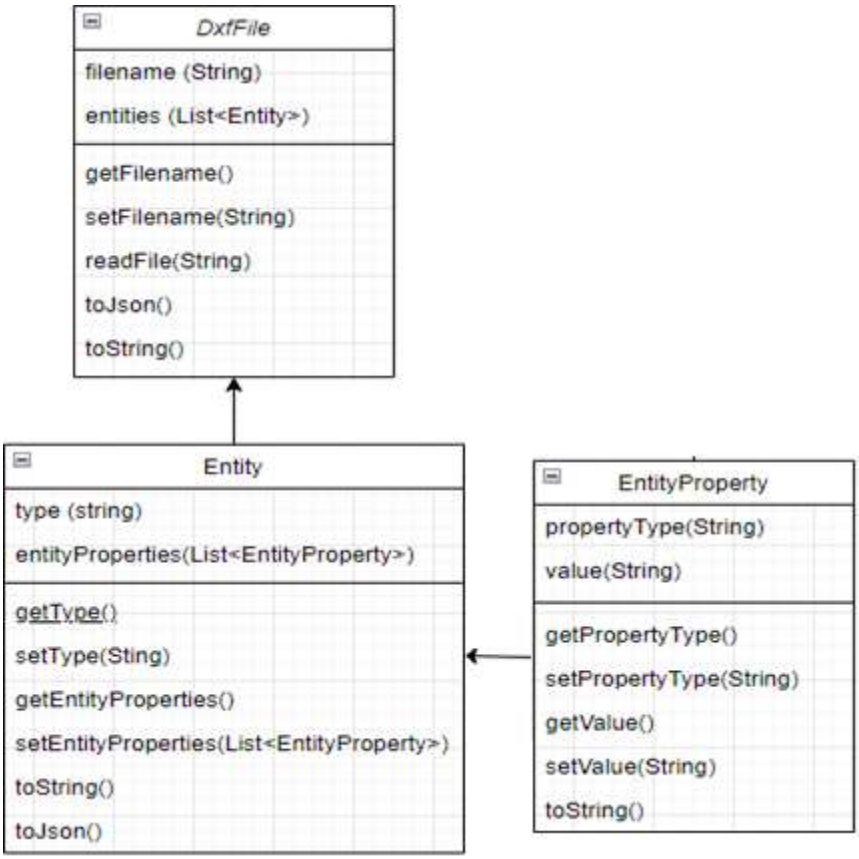
3-2. ábra Eredmény visszaírása képernyő

4. Parsolás kezelése

Az adatok parsolása során kihasználom a formátum legfontosabb szabályát, miszerint a teljes formátum adatpárokból épül fel. Tehát egy típus után a következő sorban egy érték fog következni. Mivel nagy az adathalmaz, de a parsolásnak gyorsnak kellett lennie, ezért a kezdeti feldolgozó eljárás a fájl első bejárása során a teljes fájlt kezelte és osztályokba strukturálta. Az átadott fájlokat backend oldalon mentem.

4.1. BackEnd oldali adatszerkezetek használata és osztály struktúra

A DXF file backend oldali kezelését három osztállyal lehet megvalósítani, amik között erős aggregációs kapcsolat van (4.ábra).



4-1. ábra backend class diagram

4.2. Parsolás menete

A parsolás eljárás során a HEADER szekcióból a $EXTMIN és $EXTMAX értékeket gyűjtötte össze, mert ezek az adatok határozzák meg a rajztábla jobb felső és bal alsó pontját, ami ahhoz szükséges, hogy a visszarajzolt kép elhelyezkedése és mérete megfelelő legyen Ezeket az adatokat entitásként kezeli a parser. A header szekcióból érkező kettő speciális entitást a tömb elejére helyeztem el. Ezeknek az entitásoknak négy entityproperty-vel kellett dolgoznia. A 10,20,30 egy pontot írt le a 9-es pedig figyelmen kívül hagyható volt. A parsolás további része akkor kezdődik, ha a fájlban elérem a ENTITIES szekciót.

A while futása közben segéd változókkal határozom meg hogy éppen milyen osztályból kell példányosítani. Az entitás kezdetekor („ 0” típus ID esetén) kigyűjtöm az entitás típusát és létrehozom a EntityProperties listát, amiben gyűjtöm a következő „ 0” értékig az entityproperty értékpárokat. A lista zárásával az entity objektumot példányosítom és ürítem az entityproperty listát. A boolean segédváltozókat a kezdeti állapotba állítom. A parsolás metódus egészen a ENDSEC részig fut. Ezt követően a file feldolgozása már nem hoz létre entitásokat. A feldolgozás végén egy kapcsolható logolás található, ami kilistázza a kigyűjtött entitásokat. Feldolgozási idő gyorsítása miatt volt szükség a logolás kapcsolhatóságára, mert a logolás gyakorlatilag újra végig olvassa az összegyűjtött entitások tömbjét.

4.3. Parsolt adat átadása FrontEnd felé

A feldolgozott adatok visszaküldése frontend oldalra a JSON formátumban valósult meg. Az osztály struktúra DxfFile és Entity osztályai a toJson eljárás segítségével állították elő a JSON formátumot, a JSON eljárás végén az osztály felüldefiniált toString eljárását használtam, így a return már egy valid JSON formátumot eredményezett. Azért ezt a formátumot használtam, mert a JSON formátum a JS frontendnek az egyik legmegfelelőbb struktúra, mert a visszatérő stringet a json() hívással könnyen kezelhető objektummá lehet alakítani.

5. Adatbázis felépítése, és tárolás

A parsolt adatokat backend oldalon táblákba rendezem. A rendezett adatokból, egy fájl visszaalakíthatóvá, de az aktuális parsoló logika szándékosan szűri a tárolandó adatok mennyiségét a szekciók segítségével. Az adatbázis egy H2 adatbázis lett.

5.1. Táblák

A „DXF_FILE” nevű tábla fogja tartalmazni a fájl feltöltésének eseménykor elérhető adatokat.

A mezők listáját és tulajdonságait a 5. táblázat írja le.

Mezőnév	Megnevezés	Megkötés	típus
ID	szekvenciából osztott unique index	not null constraint	Long
FILENAME	fájl név	not null constraint	String

5-1. táblázat DXF_FILE táble leírása

A „DXF_ENTITY” nevű tábla tartalmazza az entitás fő adatait az ENTITIES szekcióból.

A mezők listáját és tulajdonságait a 6. táblázat írja le.

Mezőnév	Megnevezés	Megkötés	típus
ID	szekvenciából osztott unique index	not null constraint	Long
DXF_FILE_ID	Foreign key a file_basic_data táblához	not null constraint	Long
TYPE	Entitás típusa	not null constraint	String

5-2. táblázat DXF_ENTITY tábla leírása

A „ENTITY_PROPERTY” nevű tábla tartalmazza az entitáshoz köthető tulajdonságok azonosító típuskódját és az ahhoz tartozó értéket. adatait az ENTITIES szekcióból.

A mezők listáját és tulajdonságait a 7. táblázat írja le.

Mezőnév	Megnevezés	Megkötés	típus

ID	szekvenciából osztott unique index	not null constraint	Long
DXF_ENTITY_ID	Foreign key a entities_basic_data táblához	not null constraint	Long
PROPERTY_TYPE	EntitásPropery típusa		String
VALUE	EntitásProperty típushoz tartozó érték		String

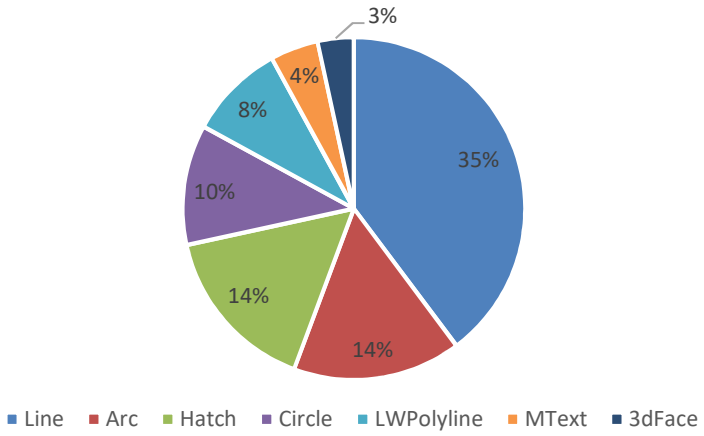
5-3. táblázat ENTITY_PROPERTY tábla leírása

6. Elemek visszarajzolása Javascripttel

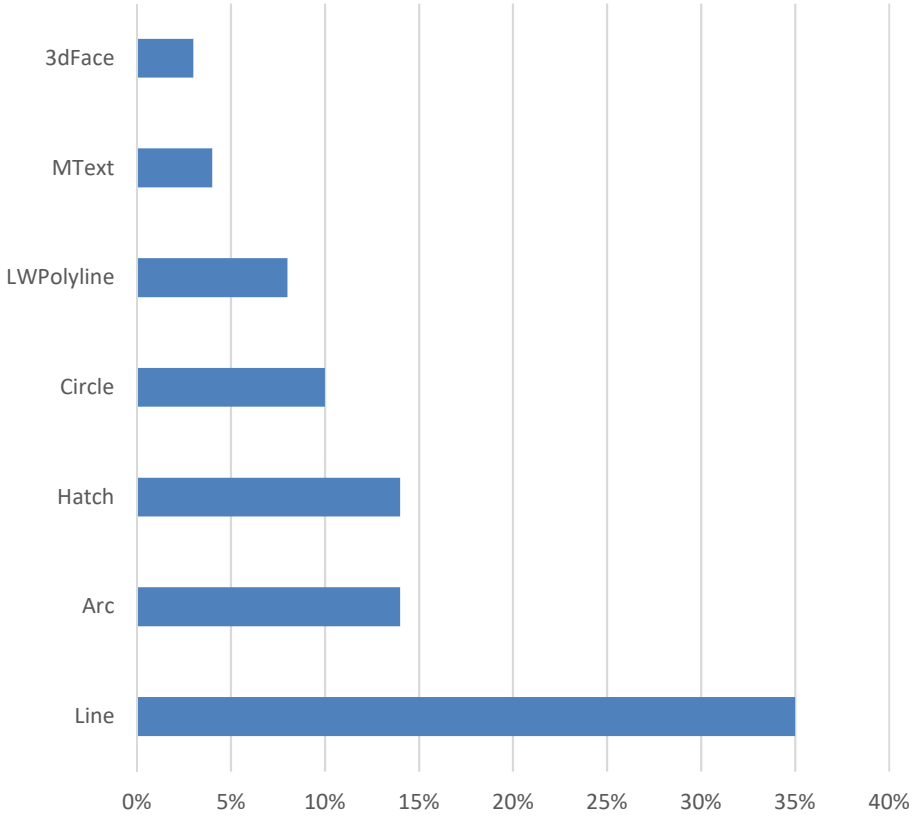
A visszarajzolás során a JSON struktúrában érkező adatokat JavaScript osztályszerkezetű objektumokba szervezem. A JSON adatok első elemei a $EXTMAX és $EXTMIN entitások voltak. Ezek feldolgozását a cikluson kívül kezelem, hogy példányosítani lehessen egy DxfFile objektumot, ahova a további entitásokat és a hozzájuk tartozó entitástulajdonságok típus, érték párjait tároltam el.

Az aktuális hivatalos DXF formátum leíró dokumentáció alapján 45 grafikus entitás objektum kezelése lehetséges a formátummal. Mind a 45 grafikus entitás lekódolása nagyon sok időt venne igényben, ezért meg kellett határoznom a projekthez szükséges feltétlenül lekódolandó, megjelenítendő entitás típusokat. Az interneten nem találtam olyan jellegű kimutatást, amiből ki lehetett volna indulni, és olyan segéd eljárást sem, ami hasonló módon szűrte volna a fájlok tartalmát, ezért a parsolo eljáráshoz készült egy modul, ami összegyűjti a parsolás során feldolgozott entitás típusokat. A modul segítségével tároltam egy fájlba az összes leparsolt fájlban fellelhető entitás típust, és ebből egy xls fájlba csináltam egy rendezett kimutatást. Körülbelül 120 fájl adataiból indultam el. A kimutatás alapján készítettem egy listát a leggyakrabban használt entitásokról. A hivatalos DXF formátum leíró dokumentáció alapján megvizsgáltam, hogy ezek valóban rajzoló vagy textúrális entitások-e.

Az eloszlást a 3. ábra a sorrendet pedig a 4. ábra írja le.



6-1. ábra Entitás eloszlás

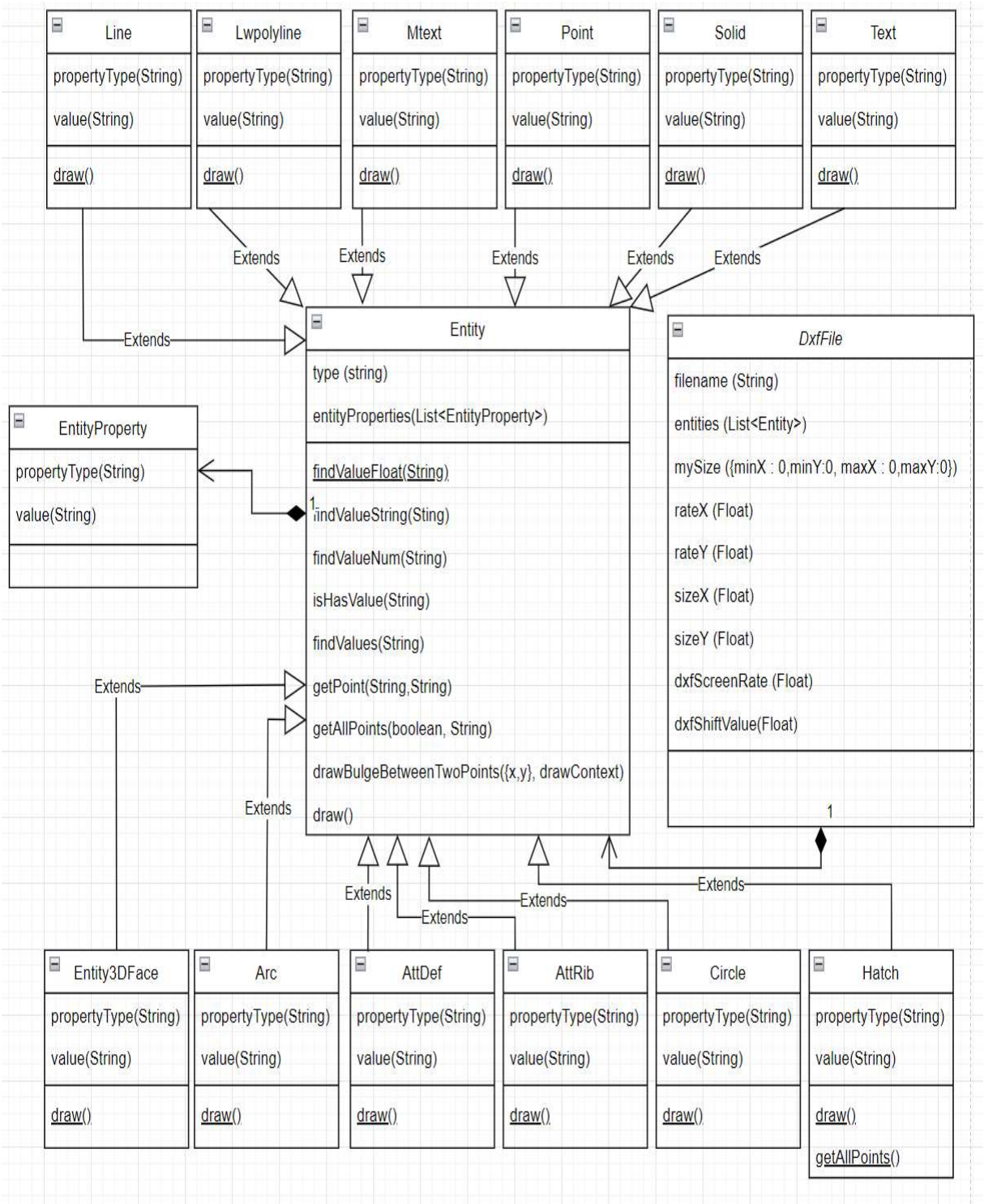


6-2. ábra Entitás sorrend

A lista alapján ezek az entitások lefejlesztése elengedhetetlen volt az objektum visszarajzolásához. A DXF fájlleíró dokumentáció alapján kiválasztottam azokat az entitásokat, amik a fenti típusok alapján könnyen lefejleszthető voltak. A később beolvasott tesztfájlok alapján a kiválasztott entitások hozták az elvárt eredményt.

6.1. Osztály struktúra

Az JavaScript osztály struktúrát az 5. ábra írja le le.



6-3. ábra JavaScript class diagram

6.2. Segédeljárások az ENTITY osztályban a rajzpont számtásokhoz

A util jellegű segéd eljárások legyártása szükséges volt a hatékony kódoláshoz. A pontadatok legyűjtése, listázása és értékek keresésére megvalósított eljárásokat fogom a fejezetben kifejteni.

6.2.1. Pontadatok kezelése:

getPoint: Kettő beérkező típus alapján visszaad egy point objektumot

Mivel nem csak egy fix pártípust tud leírni egy pontot, ezért bejövő paraméterként szükséges volt megadni a keresendő típus értékeket. A visszaadott objektum könnyen kezelhető a hívó oldalon. A következő eljárásban megjelenik a bulge, mint adat.

A bulge egy ívet jelent a két pont között az egyenes helyett. A 1.0 értékű bulge jelenti a tökéletes félkörívet a két pont között a -1.0 pedig a negatív irányba határozza meg ugyanezt. A bulge értéke a két pont között megtalálható szakaszfelező hosszát arányosítja. Tehát az ív legfelső/legalsó pontja pont olyan magasan lesz, mint a szakaszfelező hosszának és a bulge értékének a szorzata.

getAllPoints: Bejövő paramétereként megjelenik az isbulge boolean adat és a bulge adat típusa. Az eljárás egy point objektum listát ad vissza, ahol meg tud jelenni a bulge adata is.

A bulgeType értéke eltérő az entitások között ezért szükséges megadni azt az eljárás híváskor.

6.2.2. Érték meghatározás

A következő eljárások bejövő típushoz adnak vissza értékeket.

findValueFloat: Egy bejövő property típushoz ad vissza float értéket.

findValueString: Egy bejövő property típushoz ad vissza string értéket

findValueNum: Egy bejövő property típushoz ad vissza string értéket

isHasValue: Egy bejövő property típushoz visszaadja, hogy van-e az entitásban ilyen típusú property.

findValues: Egy bejövő property típushoz adja vissza listában az összes előforduló értéket

6.2.3. Bulge kezelés

drawBulgeBetweenTwoPoints: bejövő paramétereként kettő point objektum illetve a rajztér a rajzoláshoz. Visszatérési érték nincs a függvény a lefutás végén rajzol.

A bejövő paraméterben meghatározható, hogy az első és második pont közé egy vonalat kell rajzolni, ha null vagy 0.0 értékű az első pont bulge értéke, vagy egy körívet, ha ettől eltérő.

Több matematikai művelet szükséges a teljes folyamat megvalósításához.

- Megkeresem a domborítandó, vagy homorítand szakasz közepét
- A szakaszfelező pontot elviszem az origóba
- Elforgatom az origó körül 90 fokkal.
- Kiszámítom a domborítás/homorítás tetejét
- Megállapítom a szakaszfelező merőleges hosszát az eltolt középpont és a domborítás/homorítás távolságából.
- Ezt a pontot visszateszem a helyére
- Kiszámolom a szakasz kezdő, végpontja és domborítás teteje pontokkal meghatározott háromszög köré rajzolható kört
- Meghatározom a szögeket, amik meghatározzák a levágandó körív valós hosszát.

6.3. Entitás típusokhoz felüldefiniált draw eljárások bemutatása

Minden rajzolható entitás őse az ENTITY osztály, amiben megtalálható a draw() függvény és amit felüldefiniálok kiterjesztés esetén. A következő fejezetben ezeket a felüldefiniálásokat fogom bemutatni.

A folyamat megkezdése előtt a rajzvászonra meghívom a beginpath() eljárást, ami nem engedi, hogy olyan elem legyen a rajzolási metodikában, amire még nem hívódott fill() mint kitöltés vagy stroke() mint kirajzolás parancs.

6.3.1. 3dface

Az entitás négy darab zárt vonalhalmazt tartalmaz. Specializációja, hogy bináris változóval jelölni lehet, hogy egy vonal látszódik vagy nem.

6.3.1.1. Rajzolásban érintett entityproperties

A rajzolásban érintett entitás tulajdonságokat a 8. táblázat írja le.

Típuskód	Leírás
39	A rajzolás során használt vonalhossz vastagsága. Ha nem érkezik változó szinten definiálható alapértelmezett vonalvastagság.
70	Bináris blocktype flag. Azt jelöli, hogy melyik vonal látható és melyik nem.0-16ig vehet fel bináris értékeket. Az egymás utáni helyiértékek jelölik, hogy a négy vonal közül melyik látszódik.
10	első pont X koordinátája
20	első pont Y koordinátája
11	második pont X koordinátája
21	második pont Y koordinátája
12	harmadik pont X koordinátája
22	harmadik pont Y koordinátája
13	negyedik pont X koordinátája
23	negyedik pont Y koordinátája

6-1. táblázat rajzolásban érintett 3DFACE property-k

6.3.1.2. Visszarajzolás kezelése

A „ 70”-es kódhoz tartozó értéket stringként karakteresen értelmezem hátulról. Abban az esetben, ha a karakter 0 értékű, akkor kirajzolom a megfelelő két pont közötti vonalalt, ellenkező esetben a feldolgozással és nem történik rajzolás.

A canvas szinten a moveto(), lineto() függvényeket használom a valós rajzolás elvégzéséhez.

6.3.1.3. Felmerült problémák kezelése

A fejlesztés során nem volt egyértelmű, hogy honnan indítom a bináris adat feldolgozását. Valójába, a lenti kódrészleten (**Hiba! A hivatkozási forrás nem található.**) látható, hogy a bináris stringet hátulról kell olvasni, de a pontokat elölről kell feldolgozni.

```
class Entity3DFace extends Entity {
    constructor(type,Entityproperties) {
        super(type, Entityproperties);
    }

    draw() {
        let drawContext = drawSheet.getContext('2d');
        drawContext.lineWidth = this.findValueNum(' 39')==null ? defaultLineWidth : this.findValueNum(' 39');
        let binaryLineVisibility = String(this.findValueNum(' 70')==null ? 0 : this.findValueNum(' 70').toString(2)).padStart(4,'0');
        drawContext.beginPath();
        if (binaryLineVisibility.charAt(3)==='0') {
            console.log('1line');

            drawContext.moveTo(this.findValueFloat(' 10'),this.findValueFloat(' 20'))
            drawContext.lineTo(this.findValueFloat(' 11'),this.findValueFloat(' 21'))
        }
        if (binaryLineVisibility.charAt(2)==='0') {
            console.log('2line');
            drawContext.moveTo(this.findValueFloat(' 11'),this.findValueFloat(' 21'))
            drawContext.lineTo(this.findValueFloat(' 12'),this.findValueFloat(' 22'))
        }
        if (binaryLineVisibility.charAt(1)==='0') {
            console.log('3line');
            drawContext.moveTo(this.findValueFloat(' 12'),this.findValueFloat(' 22'))
            drawContext.lineTo(this.findValueFloat(' 13'),this.findValueFloat(' 23'))
        }
        if (binaryLineVisibility.charAt(0)==='0') {
            console.log('4line');
            drawContext.moveTo(this.findValueFloat(' 13'),this.findValueFloat(' 23'))
            drawContext.lineTo(this.findValueFloat(' 10'),this.findValueFloat(' 20'))
        }
        drawContext.stroke();
    }
```

6-4. ábra 3dFace forrás

6.3.2. Arc

Az entitás egy körív objektumot ír le. Nagyon hasonlóan, mint a canvas objektum. A lényegi különbség a rajzolás kezdő pontja.

6.3.2.1. Rajzolásban érintett entityproperties

A rajzolásban érintett entitás tulajdonságokat a 9. táblázat írja le.

Típuskód	Leírás
39	A rajzolás során használt vonalhossz vastagsága. Ha nem érkezik változó szinten definiálható alapértelmezett vonalvastagság.
10	A kör középpontjának X koordinátája
20	A kör középpontjának Y koordinátája
40	Sugár

50	Körív kezdetének szöge
51	Körív zárásának szöge
210	bejárási irány (default = 0 (óramutató járása szerint), 0, 1)

6-2. táblázat rajzolásban érintett ARC property-k

6.3.2.2. Visszarajzolás kezelése

A canvas objektum és az ARC leíró entitás ugyanazokat az értékeket veszi fel. A szögeket radiánban kellett megadni. Az óramutató járásának kezelés is megegyezett, ezért azt is csak át kellett adni a rajzfelületnek az arc() függvényhívással.

6.3.2.3. Felmerült problémák kezelése

Az arc entitás és a canvas objektum körív rajzolónak eltérő kezdőponttal indítja a rajzolást. Ezzel sajnos nem voltam tisztában, ezért nagyon sokat kellett debugolni. Végül az 5.ábrán látható szögátalakítással lehetett kezelni az eltérő kezelést.

$$\pi/180 * -1 * (\varphi - 360)$$

6-5. ábra szögszámítás

6.3.3. LINE

Az entitás egy darab zárt vonalhalmazt tartalmaz.

6.3.3.1. Rajzolásban érintett entityproperties

A rajzolásban érintett entitás tulajdonságokat a 10. táblázat írja le.

Típuskód	**Leírás**
39	**A rajzolás során használt vonalhossz vastagsága. Ha nem érkezik változó szinten definiálható alapértelmezett vonalvastagság.**
10	**a kezdőpont X koordinátája**

20	**a kezdőpont Y koordinátája**
11	**a végpont X koordinátája**
21	**a végpont Y koordinátája**

6-3. táblázat rajzolásban érintett LINE property-k

6.3.3.2. Visszarajzolás kezelése

A visszarajzolás során először rajzolás nélkül el kell mozgatni a rajzolás kezdőpontját a 10,20-es value értékekre a moveTo() függvénnyel. Ezt követően rajzolható ki a vonal a lineTo() függvénnyel.

6.3.4. Circle

Az entitás egy speciális körív objektumot ír le. Eltérése a körívtől, hogy nincsenek szög értékek és bejárási irány

6.3.4.1. Rajzolásban érintett entityproperties

A rajzolásban érintett entitás tulajdonságokat a 11. táblázat írja le.

Típuskód	Leírás
39	A rajzolás során használt vonalhossz vastagsága. Ha nem érkezik változó szinten definiálható alapértelmezett vonalvastagság.
10	kör középpontjának X koordinátája
20	kör középpontjának Y koordinátája
40	Sugár

6-4. táblázat rajzolásban érintett CIRCLE property-k

6.3.4.2. Visszarajzolás kezelése

A canvas objektum arc-ként valósul meg ezért a kezdő szöget 0 értékkel állítom be a vég szöget pedig Math.PI * 2 értékkel. A rajzolás iránya nem releváns.

6.3.5. AttDef, MTEXT, TEXT, AttRib

Szöveg megjelenítésére alkalmas entitás. A projekt szempontjából csak a szöveg megjelenítése a lényeg. Az érdemi változást az entitások között nem kell külön részletezni.

6.3.5.1. Rajzolásban érintett entityproperties

A rajzolásban érintett entitás tulajdonságokat a 12. táblázat írja le

Típuskód	Leírás
1	megjelenítendő szöveg
10	a kezdőpont X koordinátája
20	a kezdőpont Y koordinátája
40	a szöveg mérete

6-5. táblázat rajzolásban érintett Text jellegű property-k

6.3.5.2. Visszarajzolás kezelése

A 10, 20 értékkel megtalálható kezdő pontra mozgatom a kurzort. Beállítom a 40-es értékben érkező betűméretet. Ha nem érkezik akkor a kód szinte meghatározott default értékkel dolgozik.

6.3.6. POINT

6.3.6.1. Rajzolásban érintett entityproperties

A rajzolásban érintett entitás tulajdonságokat a 13. táblázat írja le.

Típuskód	Leírás
10	a kezdőpont X koordinátája
20	a kezdőpont Y koordinátája
39	Kitöltött pont sugara

6-6. táblázat rajzolásban érintett POINT property-k

6.3.6.2. Visszarajzolás kezelése

Egy kitöltött kört kell megjeleníteni. A 10,20 értékkel megtalálható kezdő pontra mozgatom a kurzort. Beállítom a 39-es értékben érkező pont méretének a felét a sugár értékének. A fill() paranccsal és az arc canvas rajzolással a rajztérre kitöltött kör rajzobjektumot tudok létrehozni.

6.3.7. LWPOLYLINE

Egy n vonalból álló objektum, ahol n természetes szám. Minden vonal az utána következő ponttal van összekötve. A formátum jelöli, hogy zárt vagy nyitott a vonalhalmaz.

Az LWPOLYLINE objektum kezeli a pontok közötti görbe vonalat is, így lehetséges bulge érték is a pont leírásban. A következő ábra fogja megmutatni a valódi formátumokat.

```
LWPOLYLINE
  5
4A9
330
4A8
100
AcDbEntity
  8
0
100
AcDbPolyline
 90
        4
 70
     1
 43
0.0
 10
0.0
 20
1192.066361318109
 10
2199.215790113099
 20
1192.066361318109
 10
2199.215790113099
 20
0.0
 10
0.0
 20
0.0
  0
```

```
LWPOLYLINE
  5
6B4
330
1F
100
AcDbEntity
  8
0
100
AcDbPolyline
 90
        4
 70
     0
 43
0.0
 10
73.73410651759013
 20
46.57321826871589
 10
76.90070337271527
 20
46.57321826871589
 42
0.9999999999999998
 10
86.90070337271526
 20
46.57321826871589
 10
89.85873493271322
 20
46.57321826871589
  0
```

6-6. ábra LWPolyline 42-es bulge nélkül és 42-es bulge értékkel

6.3.7.1. Rajzolásban érintett entityproperties

A rajzolásban érintett entitás tulajdonságokat a 14. táblázat írja le.

Típuskód	Leírás
43	A rajzolás során használt vonalhossz vastagsága. Ha nem érkezik változó szinten definiálható alapértelmezett vonalvastagság.
10	a 0…n-edik pont X koordinátája (a)
20	a 0…n-edik pont Y koordinátája
42	a 0…n-edik pont bulge értéke
70	Zárt polyline 0 érték. nyitott polyline 1-es érték

6-7. táblázat rajzolásban érintett LWPOLYLINE property-k

6.3.7.2. Visszarajzolás kezelése

Egy pont kettő fix és 1 opcionális mezővel rendelkezik. A 10-es és 20-as koordináta érték minden esetben érkezik. Ha az vonalak között van görbe vonal akkor az egyenes vonalakhoz is érkezhet bulge type 0.0 értékkel. Ha kizárólag egyenesek vannak az entitásba akkor nem érkezik bulge érték egyik ponthoz sem.

A „ 70” -es típus kód mutatja meg hogy a vonallánca az nyitott vagy zárt. Nyitott esetben a vonallánc végpontja a kigyűjtött pontok utolsó pontja. Zárt esetben az utolsó pont a kezdő pont lesz.

6.3.7.3. Felmerült problémák kezelése

Nehezen sikerült detektálni, hogy mikor érkezik egy pont listában bulge érték. Több fájl teljes vizsgálata alapján tudtam leszűrni a bulge-re vonatkozó fenti ténymegállapításokat.

6.3.8. SOLID

A solid grafikai objektum egy 3 vagy 4 pontból álló kitöltött alakzat.

6.3.8.1. Rajzolásban érintett entityproperties

A rajzolásban érintett entitás tulajdonságokat a 15. táblázat írja le.

Típuskód	Leírás
39	A rajzolás során használt vonalhossz vastagsága. Ha nem érkezik változó szinten definiálható alapértelmezett vonalvastagság.
10	első pont X koordinátája
20	első pont Y koordinátája
11	második pont X koordinátája
21	második pont Y koordinátája
12	harmadik pont X koordinátája
22	harmadik pont Y koordinátája
13	negyedik pont X koordinátája
23	negyedik pont Y koordinátája

6-8. táblázat rajzolásban érintett SOLID property-k

6.3.8.2. Visszarajzolás kezelése

Első lépésként moveTo függvényhívás segítségével beállításra kerül a grafikai objektum kezdő pozíciója. Ezt követően, ha volt 13 és 23-as típusú pontleíró tulajdonsága akkor négy vonallal körbezárt alakzatra futott le a fill eljárás, ellenkező esetben a zárt alakzat amire fill parancs ki lett adva csak három vonalból állt.

6.3.9. HATCH

A HATCH (kitöltés) objektum a legösszetettebb feldolgozott objektum, ami szerepel a projektmunkában. Az objektum leírás felhasználja az ARC, LINE, LWPolyline, ELIPSE, SPLINE objektumokat. Az adatgyűjtés során látható volt, hogy kiemelkedően sok entityproperty-vel rendelkezik az objektum. A legkülönfélébb objektumok előállíthatók vele, ezért nagyon érdekes volt a munka az objektummal. Ennél az objektumnál a pontok összegyűjtés módját is felül kellett definiálni, mert az öt különböző esetben, máshogy kell megállapítani azokat.

6.3.9.1. Rajzolásban érintett entityproperties

A rajzolásban érintett entitás tulajdonságokat a 16. táblázat írja le.

Általános leíró adatok:

Típuskód	Leírás
91	A feldolgozandó összeköttetések darabszáma
72	A HATCH-ben szereplő pontok közötti összekötések típusa *1 = Line; 2 = Circular arc; 3 = Elliptic arc; 4 = Spline 0 = Polyline*

6-9. táblázat rajzolásban érintett HATCH property-k alap definíciók

A POILYLINE jellegű összekötés típushoz tartozó entitás tulajdonságokat a 17. táblázat írja le.

Típuskód	Leírás
72	Bulge flag boolean
73	Zárt-e boolean
93	Csúcsok száma
10	a 0…n-edik pont X koordinátája (a)
20	a 0…n-edik pont Y koordinátája
42	a 0…n-edik pont bulge értéke

6-10. táblázat POILYLINE jellegű összekötés típushoz tartozó entitás tulajdonságok

A LINE jellegű összekötés típushoz tartozó entitás tulajdonságokat a 18. táblázat írja le.

Típuskód	Leírás
10	első pont X koordinátája
20	első pont Y koordinátája
11	második pont X koordinátája
21	második pont Y koordinátája

6-11. táblázat LINE jellegű összekötés típushoz tartozó entitás tulajdonságok

Az ARC jellegű összekötés típushoz tartozó entitás tulajdonságokat a 19. táblázat írja le.

Típuskód	Leírás
10	a 0…n-edik pont X koordinátája (a)
20	a 0…n-edik pont Y koordinátája

40	a 0…n-edik ponthoz tartozó sugár
50	a 0…n-edik ponthoz tartozó körív kezdetének szöge
51	a 0…n-edik ponthoz tartozó körív zárásának szöge
73	a 0…n-edik ponthoz tartozó bejárási irány (default = 0 (óramutató járása szerint), 0, 1)

6-12. táblázat ARC jellegű összekötés típushoz tartozó entitás tulajdonságok

Az ELLIPSE jellegű összekötés típushoz tartozó entitás tulajdonságokat a 20. táblázat írja le.

Típuskód	Leírás
10	a 0…n-edik pont X koordinátája (a)
20	a 0…n-edik pont Y koordinátája
11	a 0…n-edik ponthoz tartozó hosszabb fő átló X koordinátája
21	a 0…n-edik ponthoz tartozó hosszabb fő átló Y koordinátája
40	a rövidebb fő átló hossza
50	a 0…n-edik pont hoz tartozó körív kezdetének szöge
51	a 0…n-edik ponthoz tartozó körív zárásának szöge
73	a 0…n-edik ponthoz tartozó bejárási irány (default = 0 (óramutató járása szerint), 0, 1)

6-13. táblázat ELLIPSE jellegű összekötés típushoz tartozó entitás tulajdonságok

6.3.9.2. Visszarajzolás kezelése

Egy HATCH entitás során a pontok és azok között megrajzolandó összeköttetések legyűjtése is kihívást jelent. A „ 72” -es típus kód alapján három különböző képpen gyűjtöttem össze a pontoknak a halmazát. A fenti táblázatban megtalálható objektumként szerepelt egy-egy pont rekord. A visszarajzolást szintén a „ 72”-es kód vezérlete.

Attól függetlenül, hogy nem találtam olyan HATCH mintát, ahol egy HATCH-en belül több különböző összeköttetés megjelent, a kód úgy készült el, hogy tudja kezelni, mert a fájlleíró dokumentáció nem kötötte ki ezt feltételként.

6.4. Arányok kezelése

Ahhoz, hogy a megfelelő pontok a megfelelő helyre kerüljenek arányosítani kellett a reszponzív megjelenítő felületet a DXF kép méretéhez. A parsolás első két entitása a header szekcióból lett kinyerve. Ezek a mezők mutatták meg az arányosításokat és az esetleges eltoláshoz köthető feladatokat. A DXFFile osztály példányosítása során a konstruktor már a bejövő adatokból kalkulál felhasználható értékeket. Az elem példányosításakor átadásra kerül a dxf fájl rajztábla mérete. Ezek a koordináta adatok nagyon ritkán kezdődnek 0,0 értékkel. Az arányosításhoz szükséges kalkulációt eltolással kellett kezdeni, mert a canvas rajztábla koordinátája 0,0 értékkel indul. A folyamatot ezen a ponton ketté kellett választani. Ha minden oldalra elvégezné, az arányosítást akkor a kép torzul. Egyik oldalt lehetett csak arányosítani a DXF adatok és canvas értékeknek megfelelően. A másikat a már kikalkulált aránnyal kellett számolni.

7. Számítási algoritmus megoldási lehetőségei elméleti szinten

A dolgozat további részében elméleti szinten kidolgozott megoldási lehetőségeke fogok bemutatni a számítások és kalkulációk későbbi elvégzéséhez.

7.1. Képfeldolgozás pásztázással

A kirajzol és elkészült képen soronként keresem az első kettő és utolsó kettő be- és kilépési pontot. Egy ház tervénél ezeknek a ki és belépő ponthalmazoknak egy-egy párhuzamos vonalat kell alkotnia. A detektált falak vég és kezdőpontjaihoz DIMENSION jellegű objektumot kell keresni. A DIMENSION objektumba tárolja a fájl az elemhez tartozó méreteket. A DIMENSION text értékében megjelölt szám alapján a képpontok száma már arányosítható.

7.2. Folytonos HATCH keresés

A dolgozat megírása során több olyan lehetőséggel találkoztam, amivel a kalkuláció eredményesen elvégezhető, de véleményem szerint a megfelelő megoldást ez a fejezet tárgyalja.

A folyamat kezdetekor beolvasásra kerül a teljes ponthalmaz az összes objektumhoz. A ponthalmazokat a megfelelő osztályhoz társítható módon állítom elő, de összegezve egy-egy tömbbe tárolom őket. A kapott tervrajzon megállapítom a kép legszélső kettő párhuzamos oldalpárjának kezdő pontjait. Ugyanezt a folyamatot elvégezem fordítva is, így meglesz a négy legszélsőbb pontja az objektumnak. A legszélső pontot viszont definiálni kell mert az X és Y tengely vizsgálata közben eltérhet a két legszélső pont. Abban az esetben, ha X és Y tengely vizsgálata esetén is ugyanaz a legszélső pont akkor tárolható. Ellenkező esetben fel kell venni az eltérő pontokat is legszélső pontnak, mert ritka a négyszögletes ház a mai világban.

A detektált pontokat rekurzívan vizsgálom és tárolom, hogy mennyi hívást követően lehet elérni a legközelebbi ponthoz. Tehát vizsgálni kell, hogy egy entitáson belül szerepel-e a két pont vég és vagy kezdőpontként e. A rekurzió a detektált pontok számánál nem lehet több. Abban az esteben ha több kiesik a tömbből. A megmaradt tömbök halmazából HATCH keresési algoritmust kell futtatni, aminek lényege a zártság. Visszatérünk a kezdeti adathalmazhoz és minden objektumot megvizsgálva zárt HATCH-et vagy zárt de összefüggő HATCH-eket kell keresni.

8. Alkalmazás tesztelése

8.1. Modul szintű tesztelés

Minden létrehozott entitás külön tesztfájllal lett tesztelve. A teszt folyamatot debugger segítségével soronként is el tudtam végezni. Továbbá az változók és az eredmények nyomon követhetőségé érdekében külön loglevel változót deklaráltam, ami kezeli a túl részletes log adatok átadását a console felé. A teszteléshez szükséges logolást mind java mind JS oldalon elkészült.A tesztelés során főleg javascript szintaktikai és funkcionális hibát sikerült kiszűrni.

Például tanulságos eredményeket tud produkálni a JS-ben ha az if vizsgálat során a két változó közé csak egy egyenlőség jel kerül.

8.2. EndtoEnd tesztelés

A testfile könyvtárban több ellenőrzött fájl van, aminek a beolvasása rendben lezajlott, és több online beolvasóval is összehasonlítottam.

9. Eredmények összefoglalása

„A megadott feladat megfogalmazása

A hallgató célja egy online platform kialakítása, ahol a felhasználó feltöltheti a tervezőtől kapott DXF fájlt. A feltöltött fájlt parsolja, és tárolja szerver oldalon, a parsolt adatokat relációs adatbázisban menti, canvas objektumban megjeleníti. A számítási algoritmus elméleti szinten tárgyalja magas szintű tervezési metodikával.„

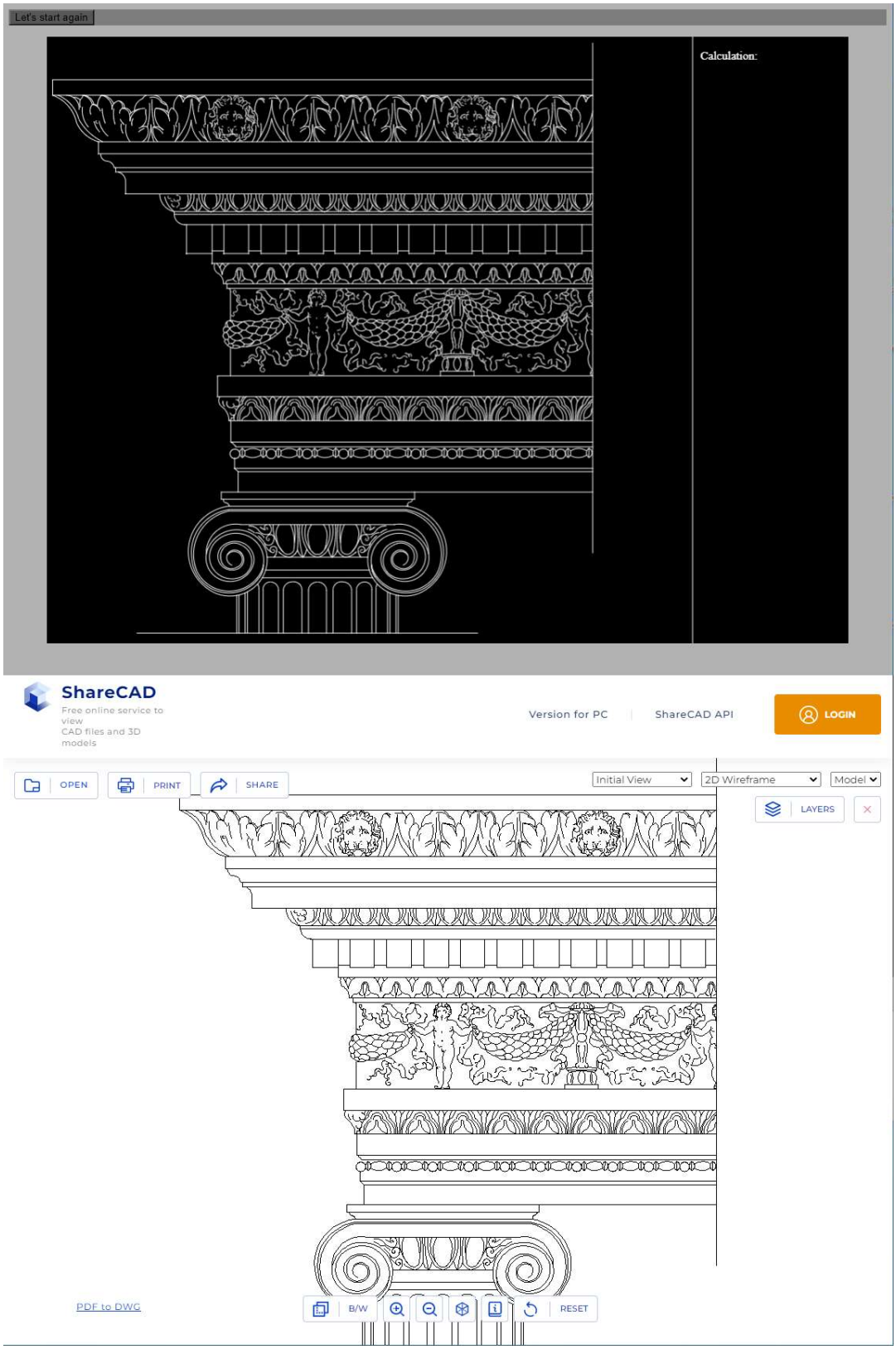
A feladat megvalósítása és a tervek kidolgozása közben a kezdeti elképzeléstől eltérően több technológiát felhasználó platform jött létre. A dolgozat backend oldala végül az adatbázis kezelés átlátható megvalósítása érdekében Spring alapon valósult meg. A mélyebb JavaScript szintű osztályhierarchia miatt modern JS-ben meg megtalálható modul megoldást kellett elsajátítanom a megfelelő fájlstruktúra kialakítása érdekében.
A dokumentációk értelmezése és az információ gyűjtés is egy ráfordítási időben alultervezett feladat volt.

A szakdolgozat fő eredményét leginkább képekben tudom bemutatni önöknek, ezért egymás mellé helyeztem néhány kirajzolást, egy hivatalos online visszarajzoló programmal és általam készített ICalc programmal. Minden esetben jelölni a fogom a kirajzolás idejét és a kép méretét mindkét alkalmazásnál. Megközelítőleg 8-12x gyorsulást sikerült elérni. Hogy ez mérhető legyen az animate.js fájlban a setTimeout eljárásban (159. sor) a vizualizációs timeout értékét 2000-ről 1-re kell módosítani.

Ceco.NET-Architecture-Fe-393.dxf fál feldolgozása 580kb

ICalc: 497ms

ShareCad.com: 3,5sec



Ceco.NET-Architecture-Fe-393.dxf fál feldolgozása 88kb

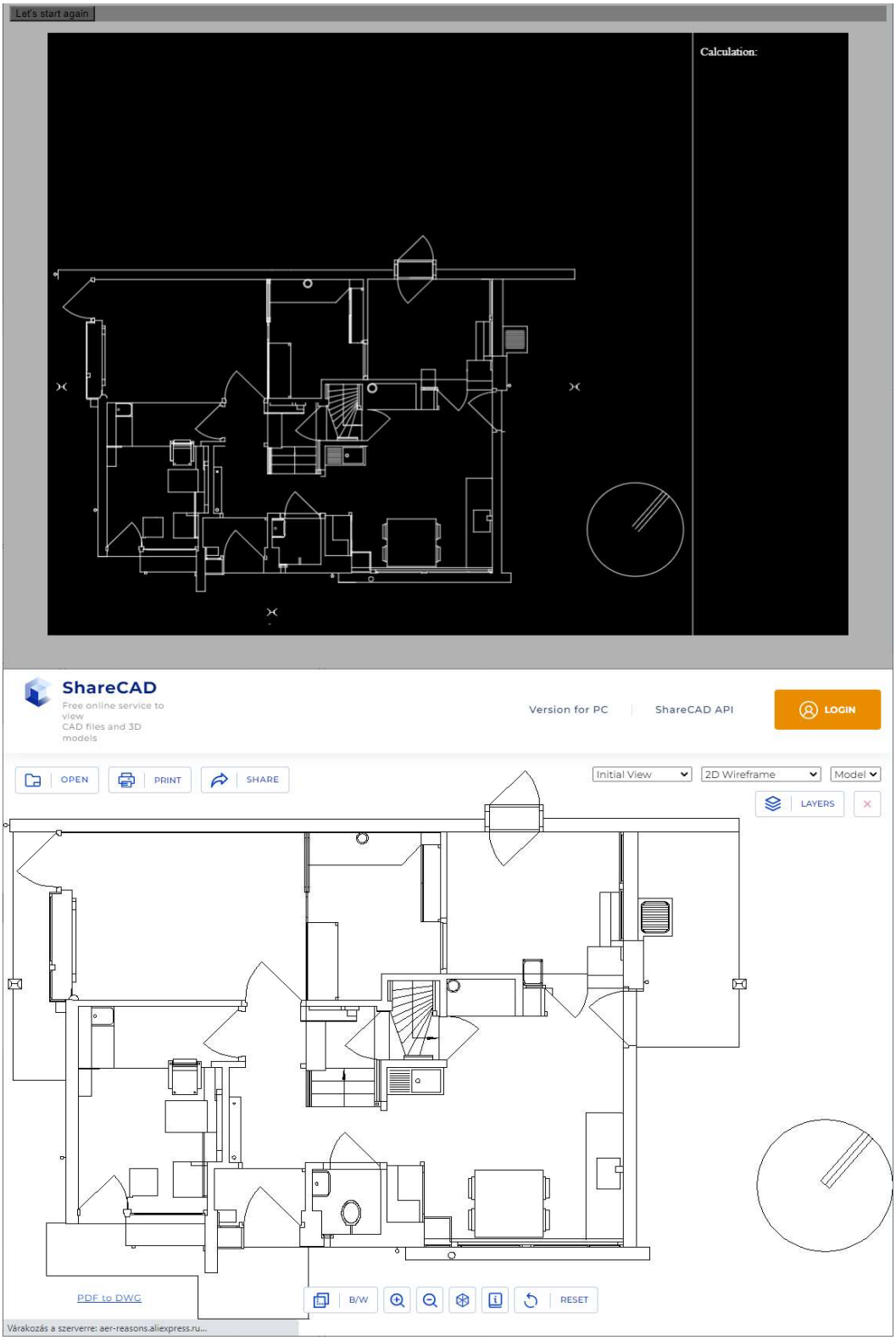
ICalc: 88ms

ShareCad.com: 2sec

Architecture306.dxf fájl feldolgozása 230kb

ICalc: 220ms

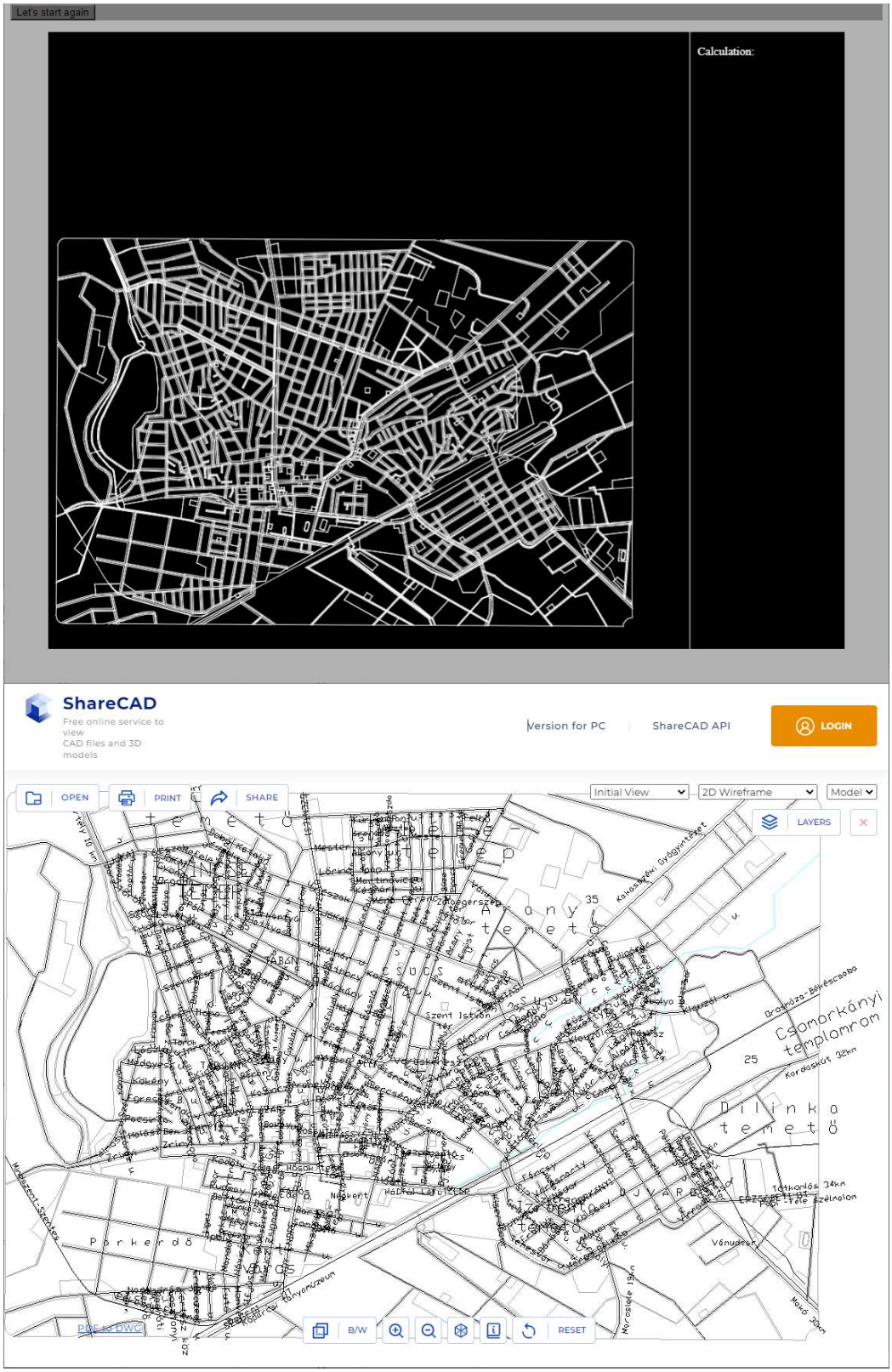
ShareCad.com: 2,5sec



hódmezővásárhely.dxf fájl feldolgozása 1.1mb

ICalc: 464ms

ShareCad.com: 7sec



Irodalomjegyzék

1. https://images.autodesk.com/adsk/files/autocad_2014_pdf_dxf_reference_enu.pdf
    a. Utolsó megtekintés: 2022:04.15.
2. https://ezdxf.readthedocs.io/en/stable/dxfinternals/filestructure.html
    a. Utolsó megtekintés: 2022:04.15.
3. https://www.loc.gov/preservation/digital/formats/fdd/fdd000446.shtml
    a. Utolsó megtekintés: 2022:04.15.
4. https://damassets.autodesk.net/content/dam/autodesk/www/developer-network/platform-technologies/autocad-dxf-archive/acad_r12_dxf.pdf
    a. Utolsó megtekintés: 2022:04.15.
5. https://docs.fileformat.com/cad/dxf/
    a. Utolsó megtekintés: 2022:04.15.
6. https://www.inf.u-szeged.hu/~katona/gis.pdf
    a. Utolsó megtekintés: 2022:04.15.
7. https://help.autodesk.com/view/ACD/2017/HUN/?guid=GUID-DBD5351C-E408-4CED-9336-3BD489179EF5
    a. Utolsó megtekintés: 2022:04.15.
8. SZTE 04_DXF segédlet.pdf
9. dxf_felepítes_99old.pdf
10. https://github.com/fuzziness/kabeja
    a. Utolsó megtekintés: 2022:04.15.

Nyilatkozat

Alulírott Ujszászi János programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

2022. április. 10.

aláírás

Köszönetnyilvánítás

Szeretném megköszönni a segítséget oktatómnak Tóth Zoltánnak.

Kollégáimnak Verner Gábornak, Gulyás Ferencnek és Dr. Ugron Balázsnak.

Mellékletek és elektronikus melléklet

https://github.com/JannY0927/iCalc_V0