Budapesti Corvinus Egyetem

Gazdaságtudományi Kar

Számítástudományi Tanszék

# Hivatkozás kezelés számonkérés támogatása

Készítette: Mátrai Tibor

Gazdaságinformatikus

2022

Burka Dávid

# Tartalom

# Bevezetés

A dolgozat témája a Budapesti Corvinus Egyetem Gazdaságinformatikus BSc szakának hivatkozás kezelés számonkérésének a támogatása. Ez a dolgozat a Szakszeminárium 1 tárgy keretei közt valósul meg és a dolgozat javításának automatizálása lenne a támogatás. A javítás automatizálására az igény a tantárgy oktatóiban merült fel, akik számára a számonkérések javítása a dolgozat súlyához képest aránytalanul sok időt vett igénybe. A nehézséget a dolgozat szöveges felépítése okozta, ami képes volt a hallgatók gyakorlati tudását ellenőrizni, amire a felelet választós dolgozat nem képes. Mivel a szöveges felépítés jobban kiszolgálja a célt, ezért ennek a dolgozatnak a támogatását kell megoldani.

A kutatás során a cél meghatározni, hogy a számonkérés automatizálása képes-e olyan hozzáadott értéket teremteni, amely szignifikáns mértékben megkönnyíti az oktatók munkáját. A cél elérése érdekében megvalósításra kerül egy támogató applikáció, melynek pontos követelményei az oktatókkal folytatott interjúk alapján kerülnek kialakításra.

A dolgozat első felében a készített megoldás során használható projekéletciklus modelleket fogom bemutatni, szó lesz lineáris és agilis módszertanokról. A következő részben elemzem a lehetséges szoftveres megoldásokat, bemutatom röviden a történelmüket, keletkezésüket és működésüket. Ezek után szó lesz az általam választott szoftver megoldás által alkalmazott keretrendszerekről és programozási nyelvekről.

Bemutatásra kerül az általam készített projekt, kifejtem hosszabban a felvetett problémát, majd a kutatásaim alapján elemzem a lehetséges megoldásokat és bemutatom a döntéseimhez vezető gondolatmenetet. Ezután részletesen ismertetem az általam kínált megoldást a vízesés modell fázisait követve. A projekt fázisokat követően szó lesz az applikáció jövőjéről: továbbfejlesztéséről és támogatásáról.

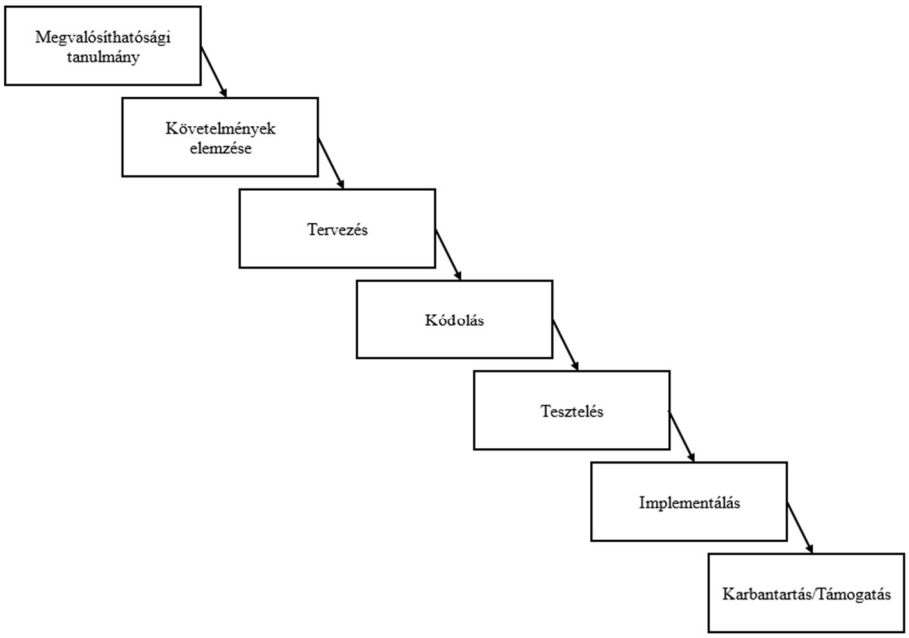
# 1. Projektéletciklus modellek

Ebben a fejezetben bemutatásra kerülnek az egyéni fejlesztés kapcsán felmerült projekt életciklus modellek. Ezen modellek közül elsőként a vízesésmodell lesz ismertetve, mert a saját fejlesztés során az oktatók elfoglaltságából adódó kevés kommunikáció miatt ez a modell volt a legalkalmasabb a fejlesztés lebonyolítására. A szoftver támogatás szakaszában, a sok éves használat érdekében folyamatos fejlesztésre van szükség és ezeknek a fejlesztéseknek a lehető legjobban le kell fednie az oktatók igényeit, ezért erre az időszakra agilis életciklus modellel kell tervezni. Így bemutatásra kerül az agilis életciklus modellek közül az exteme programozás és a scrum. A kutatómunka során a források a scrum modellt találták a legnépszerűbbnek, azonban abban az esetben, ha egyszemélyes tovább fejlesztésre kerül sor, az extreme programozás jobban megfelel a célnak.

## 1.1 Vízesésmodell

A vízesésmodell volt a tradicionális módja a kis és nagy informatikai projekteknek a kilencvenes évektől egészen a kétezer-tízes évekig (Dima - Maassen, 2018), a 2018-ban készült felmérés szerint a megkérdezett szakértők 32% még mindig ezt használta (Dima - Maassen, 2018). Ez a modell alapvetően öt elemből áll, de némely forrás hatot, illetve hetet említ, ezeket a lépéseket a következőkben fogom részletezni. Ezek a lépések akkor követik egymást, ha az előző már teljesen befejeződött, amennyiben egy korábbi lépésen szeretnénk módosítani, akkor azon lépés után következő összes lépést újra el kell végezni (Stober – Hansmann, 2010).

Az 1-es ábrán Chandra (2015) által megfogalmazott hét lépés látható, a két plusz elem Stober és Hansmann (2010) által megfogalmazotthoz képest a megvalósíthatósági tanulmány és a tesztelés utáni implementáció. A következőkben viszont Stober és Hansmann 2010-es könyvében meghatározott öt fázis (Követelmények, Tervezés, Implementáció/Kódolás, Tesztelés, Támogatás) szerint fogok haladni és közben kitérek arra, hogy ők ezeket a plusz lépéseket mikor végzik el.




*2. ábra vízesésmodell (forrás: Chandra, 2015)*

### 1.1.1 Követelmények

Stober és Hansmann (2010) szerint a vízesésmodellel végrehajtott projektek első fázisának neve a követelmények. Ezt a fázist több részre bontják, első lépésként azonosítják a projekt részvevőit és elemzik, hogy kinek mekkora a beleszólása a projektbe. Csak ezután kezdik el begyűjteni a résztvevők igényeit. Ez azért fontos, hogy pontosabb képet kapjanak a következő lépésen dolgozók, hiszen, ha ebben a szakaszban rosszul fogalmazzuk meg a követelményeket, az majd csak az utolsó lépéseknél fog kiderülni. Egy változtatást akkor bevezetni már nagyon költséges, erről majd a későbbiekben bővebben is lesz szó.

A résztvevők azonosítása után megkezdődik az igények felmérése, begyűjtése. Az igényekből egy elemzés során lesznek követelmények, melyeket két fő csoportba sorolnak, a funkcionális és a nem funkcionális követelmények közé (Stober – Hansmann, 2010). Ezeket a követelményeket dokumentálják és adják tovább a következő fázisnak.

Chandra (2015) ezt a fajta követelmények fázist veszi ketté, első lépésként begyűjti a követelményeket, amely az 1. ábrán megvalósíthatósági tanulmányként szerepel. Ebben a részben megnézi, hogy a begyűjtött követelmények közül melyik megvalósítható a rendelkezésre álló eszközökkel. Az 1. ábrán látható második lépésként pedig elemzi a követelménylistát és ezután pedig dokumentálja azokat.

### 1.1.2 Tervezés

A tervezés fázisban a tervező csapat egy részletes tervet készít az egész rendszerről és minden egyes komponensről. Ez olyan részletességgel történik, hogy ezeket a komponenseket a fejlesztők már rögtön kóddá tudják alakítani (Stober – Hansmann, 2010). Ennek a résznek az eredményét Chandra (2010) a kódolás alaprajzának nevezi. Több módszer is van ennek az „alaprajznak" az elkészítésére, Stober és Hansmann 2010-ben hármat említett.

Elsőként a Use Case-modellt, amely a dokumentált követelményeket bontja fel Use Case-ekre (használati esetekre) és ezek helyezi el egy táblázatban. A Use Case-ek lekódolásával készül el a program. A második módszer az úgynevezett Unified Modeling Language (UML), ami egy szabványos, általános célú modellező nyelv, az objektumokat tudják jól ábrázolni az objektum orientált programozáshoz. A harmadik mód pedig folyamatábra segítségével történik, ahol a követelmények alapján készül a folyamatábra, amely tartalmazza a program által összes bejárható lépéseket (Stober – Hansmann, 2010). Chandra (2015) is ugyanezeket a modelleket sorolta fel.

### 1.1.3 Implementáció/kódolás

Az implementáció fázisban készül el a fejlesztők által a programkód (Chandra, 2015). Az előző fázisból megkapott modell segítségével a programozók megírják a tényleges szoftvert. Ebben a szakaszban derülnek ki a problémák, ha nem volt megfelelően elvégezve a tervezési szakasz. Amennyiben itt hibát találnak, akkor vissza kell menni az előző szakaszba és újra kell tervezni a folyamatokat vagy használati eseteket. Ebben a szakaszban is zajlik már tesztelés, ezt fejlesztői vagy integrációs tesztnek nevezzük. Itt csak azt akarjuk biztosítani, hogy a tesztelő csapatnak nem lesznek alap problémái, és ők fókuszálhatnak az összetettebb tesztesetekre (Stober – Hansmann, 2010). Ha egy hibát csak a tesztelők találnak meg, azt 10-15-ször annyiba kerül majd kijavítani, mint ha már a fejlesztő megtalálná (McConnel, 2004).

### 1.1.4 Tesztelés

Ez előbb említett integrációs teszt sikeres lefutása után kezdődik meg a teszt csapat általi tesztelés. Ennek a célja megtalálni az összes hibát az adott rendszerben a kiadás vagy átadás előtt (Stober – Hansmann, 2010). Ez egy lehetetlen feladat, ezért a tesztelők általában addig tesztelnek, amíg elérnek egy becsült százalékot a tesztesetek között vagy addig, amíg a talált hibák az eltelt idő alatt egy ellaposodó tendenciát mutatnak (Stober – Hansmann, 2010).

A tesztelés szakaszának tartalmaznia kell különböző teszt típusokat. Ezek a szoftver különböző elemeire koncentrálnak. Ezek a tesztek lehetnek: integrációs és funkcionális tesztek,

globalizáció ellenőrző teszt, fordítás ellenőrző teszt, rendszer ellenőrző teszt, teljesítmény teszt, elfogadási teszt (Stober – Hansmann, 2010).

A tesztelés sikeres elvégzése után átadják a kész szoftvert a megrendelőnek vagy piacra bocsájtják. Ezt a műveletet Chandra (2015) külön szakaszba sorolja implementáció néven, mivel ekkor implementáljuk a szoftvert a megrendelő rendszerébe.

### 1.1.5 Támogatás és karbantartás

A támogatás fázis egyből elkezdődik, amint átadják a szoftver a megrendelőnek. A támogatás több szintű lehet, vannak esetek amikor csak hívás központon keresztül segítenek a felhasználónak. Komolyabb problémáknál egy specialista segíthet a probléma megoldásában. A legkomolyabb esetben pedig a fejlesztők javítják ki a talált hibát a rendszerben (Stober – Hansmann, 2010). Az innen érkező hibák javítása akár 10-100-szorosa is lehet annak, mintha már az implementációs fázisban megtalálták volna (McConnel, 2004).

### 1.1.6 A vízesés modell előnyei és hátrányai

A vízesésmodell legnagyobb előnye, hogy akkor a leghatékonyabb a projekt végrehajtása, ha az elején mindent megtervezünk és egy komplett befejezett követelmény listával dolgozhatunk (Stober – Hansmann, 2010). A vízesésmodellnek pedig ez a lényege, hogy addig nem állunk neki a tervezésnek, ameddig a követelmények fázis nem teljes. A modell következő nagy előnye, hogy jól definiáltak a fázisok (Chandra, 2015), és ennek köszönhetően könnyen megmondható, hogy hol tart a projekt, és könnyen becsülhető a hátra lévő idő is, feltéve, hogy nem lépnek fel hibák. Előnye továbbá, hogy a kódolási és implementációs fázis egyszerűen megvalósítható (Chandra, 2015), köszönhetően a korai részletes tervezésnek és annak, hogy a folyamat végrehajtása közben nem érkeznek új követelmények. A Stober és Hansmann (2010) féle követelmény gyűjtés során végzett résztvevő azonosításnak köszönhetően, Chandra (2015) felsorolja előnyként azt, hogy a projekt során jól ismerjük a végfelhasználókat és ezáltal személyre szabottabb végterméket adhatunk át.

A vízesésmodell fő hátránya, amit Chandra (2015), Dima és Maassen (2018), Stober és Hansmann (2010) is említ az a rugalmasság hiánya. Ez Stober és Hansmann (2010) által megfogalmazott szabályból is következik, miszerint addig nem léphetünk a következő fázisba, ameddig az előzőt teljesen el nem végeztük. Az ezzel a modellel végrehajtott projektekben nem lehet vagy nagyon költséges közben követelményt változtatni, hiszen ebben az esetben elölről kell kezdeni a tervezési fázist. Nem tud visszajelzés érkezni a megrendelő felől (Chandra, 2015), hiszen neki csak kétszer van betekintése a projektbe, a követelmény és az átadási

fázisokban (Stober – Hansmann, 2010). A vízesésmodellt mára már primitívnek nevezik, de rövid, egyszerű projektek elvégzésére még mindig a leghatékonyabb eszköz (Chandra, 2015).

## 1.2 Agilis életciklus modellek

A vízesés modell rugalmatlansága miatt megjelent az igény új fajta projekt életciklus modellekre. 2001-ben jelent meg az agilis manifesztum, amely összefoglalta az előző évtized törekvéseit egy kiadásban (Abrahamsson - Salo - Warsta, 2002). Ez a kiállítmány négy fő kijelentésből állt:

1. „Az egyének és a személyes kommunikáció a módszertanokkal és az eszközökkel szemben”
2. „A működő szoftver az átfogó dokumentációval szemben”
3. „A rendelővel történő együttműködés a szerződéses egyezéssel szemben”
4. „A változtatás iránti készséget a tervek szolgai követésével szemben”

A felsorolt négy pontban mindig az első kijelentést tartják fontosabbnak a második helyett (Abrahamsson et. al. 2002, 11. old.). Ezen kijelentések mindegyike a vízesés modell gyengeségeire utal. Az első kijelentés a kommunikáció fontosságára utal, ami hiányzott a statikus vízesés modellből. A második kijelentés arra utal, hogy ne minden problémát azzal oldjanak meg a fejlesztők, hogy a program részévé teszik és dokumentálják, hanem ha valamely funkció nem felel meg a megrendelő igényeinek, akkor azt módosítani kell. A harmadik kijelentés is a szerződésben esetleg rosszul megadott részek módosítására ad lehetőséget, hogy a végtermék jobban kiszolgálja a felhasználói igényeket. A negyedik kijelentés is a változtatások fogadására biztosít lehetőséget.

Jelenleg többféle módszertan van agilis fejlesztésre, mindnek meg van az előnye és a hátránya. A 2018-ban Dina és Maassen által készült kutatás során a válaszolók 68 százaléka használt valamilyen agilis módszertant, közülük 77 százalék használt Scrumot, 15 százalék tesztvezérelt fejlesztést és a maradék 8 százalék pedig nem említette, hogy milyen módszertant használ. Míg a Szabó Bálint és Ribényi Máté által 2018-ban készült kutatás alapján az ott megkérdezettek 94 százaléka dolgozott már olyan projekten, amit Scrum módszertannal vezettek végig, 65 százalékuk kanbant és 40 százalékuk pedig extreme programozást használt a munkája során. Ezekből az adatokból kiindulva és történelmi jelentősége miatt, a következőkben az extreme programozást és a Scrumot fogom bemutatni.

### 1.2.1 Extreme programozás

Az extreme programozást először 1999-ben említette meg Kent Beck az egyik könyvében (Stober – Hansmann, 2010), tehát már az agilis manifesztum kiadása előtt. Ennek a modellnek 12 elgondolás az alapja, amelyeknek megnevezése más lehet a forrásoktól függően, de lényegben ugyanazok. Ezek közül több forrás is említi a páros programozást, az egyszerű kódra törekvést, a kiadások során törekvést a kis mennyiségű újdonságra, a folyamatos tesztelést, a folyamatos kapcsolatot a megrendelővel és a story cardok használatát (Abrahamsson et. al. 2002, Stober – Hansmann, 2010, Szabó - Ribényi, 2018). A story cardok hasonlóak a use case-ekhez, azonban ezek teljes funkciókat tartalmaznak.

Az extreme programozás folyamata Abrahamsson et al. (2002) szerint öt lépésből áll. Ezek rendre: feltérképezés, tervezés, iteráció és kiadás, termékkialakítás, karbantartás és a termék halála. A feltérképezés fázisban a megrendelő összeírja story cardokra az általuk megvalósítani kívánt funkciókat és követelményeket, és közben a fejlesztő csapat ismerkedik a keretrendszerekkel és a technológiákkal.

A tervezés fázisában prioritás szerint sorba rendezik a story cardokat, ezen a sorrenden a megrendelő ezután változtathat. Az első tervezés feladata kiválasztani azokat a követelményeket a story cardokról, amelyek nélkül nem lehet működő kiadást készíteni (Abrahamsson et al. 2002).

A tervezés után kezdődnek az iterációk és kiadások, az extreme programozás során az első kiadás előttiek kivételével, minden iteráció egy story card tartalmával foglalkozik, azaz minden iteráció egy követelménnyel vagy funkcióval foglalkozik. Az első kiadás előtt azért nem, mert az itt keletkezett programsorok magukban még nem képeznek működő alkalmazást. Ezt követően viszont minden iteráció után egy működő programnak kell elkészülnie, amelynek készen kell lennie éles használatra. Ennek a fázisnak része az adott iterációban újonnan hozzáadott funkció tesztelése is, a tesztesetek megírása a megrendelő feladata (Abrahamsson et al. 2002).

A következő fázis feladata az elkészült kiadás további tesztelése, hogy azt tényleg átadhassuk a felhasználóknak. Itt is adhat hozzá még a megrendelő új követelményt, ezeket már gyorsabban kell végre hajtani, mint az előzőket, ezek az igények a tervezés folyamata során kerülnek bele a prioritás listába és onnan fejlesztés alá (Abrahamsson et al. 2002).

A karbantartás fázis hasonló, mint a vízesésmodellnél, azzal a különbséggel, hogy itt még adhatnak hozzá új funkciót. Az extreme projekteknek a vége a halál fázis. Ez akkor történik, amikor a megrendelő már nem akar új funkciót hozzáadni, ekkor készítik el a teljes

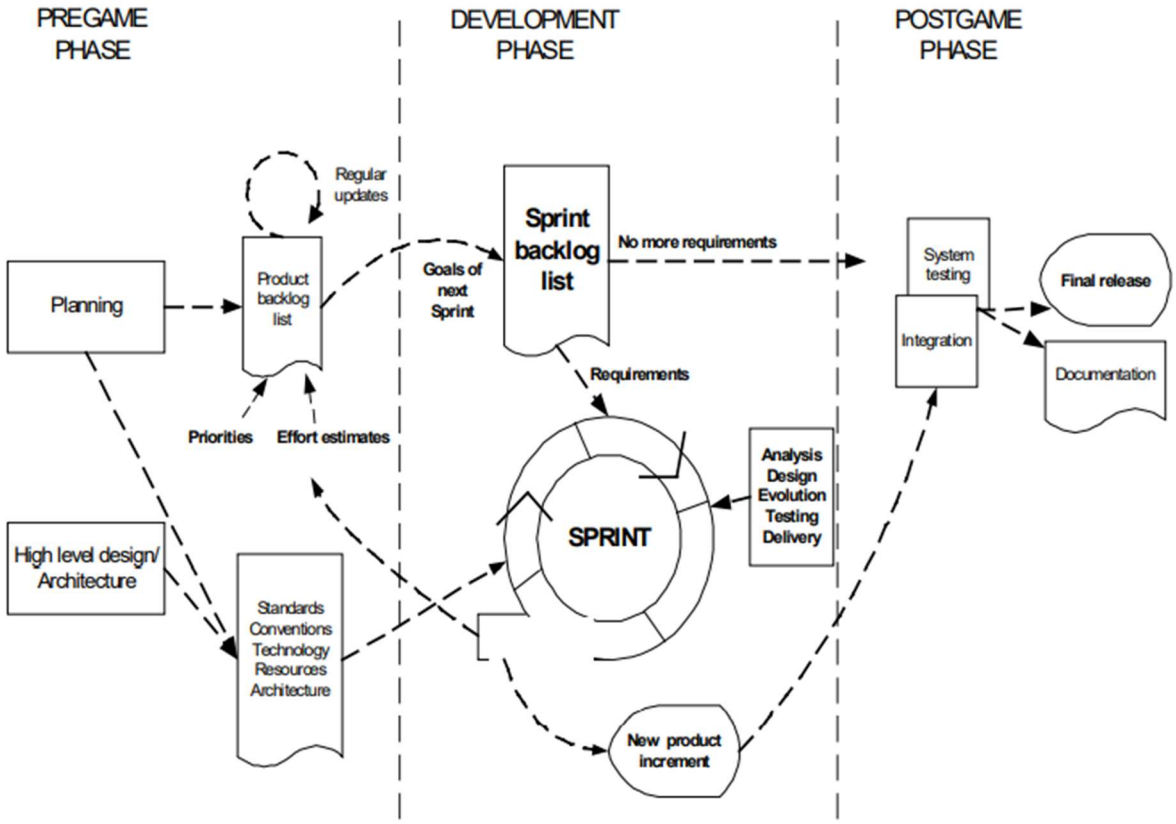
projekt részletes dokumentációját és teljesen átadják a megrendelő cégnek (Abrahamsson et al. 2002).

Az extreme programozás kis és közepes fejlesztő csoportoknak alkalmas, és csak olyan megrendelők esetében, akik képesek rendszeres visszajelzésre (Abrahamsson et al. 2002). Nem ajánlott nagy projektekhez, ebben az esetben nem alkalmazható az alap elgondolások közül több sem, mint például a kis kiadások, az egyszerűség és a tervezés egyszerűsítése (Stober – Hansmann, 2010).

### 1.2.2 Scrum

A Scrum keretrendszer először 1986-ban került dokumentálásra Takeuchi és Nonaka cikkében, ahol egy adaptív, rendszerező, és gyors termék fejlesztési eljárást írnak le. Ezt az eljárást 2002-ben írta át Schwaber és Beedle informatikai fejlesztési projektekre. A Scrum modell arra koncentrál, hogy a fejlesztő csapatnak hogyan kell együtt dolgoznia úgy, hogy képesek legyenek megbirkózni a folyamatosan változó környezeti tényezőkkel (követelmények, időkeret, erőforrások, technológia) (Abrahamsson et al. 2002).

Abrahamsson et al. (2002), ahogy a 2. ábrán látható, három részre bontja a scrum folyamatát: a pregame-re, a fejlesztésre és a postgame-re. Az első részt tovább bontják két alfázisra, melyek a tervezés és az architektúra. A tervezés során elkészítik a Product backlog listát, amely tartalmazza az összes ekkor ismert követelményt. Ez a lista folyamatosan bővül a projekt során a hozzá adott követelményekkel. Ebben a szakaszban határozzák meg a projekt során használandó szerepeket is. Ezután következik az architektúra szakasz, amely során készül egy magas szintű terv a programról a rendelkezésre álló követelmények alapján.




*4. ábra Scrum folyamati ábra forrás: Abrahamsson et. al. (2002)*

A pregame fázis után kezdődik a fejlesztés. Ebben a szakaszban úgy nevezett sprinteket végeznek (ahogy a 2. ábrán látható). Minden sprint tartalmazza a vízesésmodellben már leírt szakaszokat (Követelmények, Tervezés, Implementáció/Kódolás, Tesztelés, Átadás), egy sprint egy hét és egy hónap közötti hosszúságú lehet. A sprintek során minden nap elején van egy tíz-tizenöt perces stand-up meeting, amely során mindenki elmondja mit csinált előző nap és mit tervez ma, ezzel elősegítve a csapaton belüli kommunikációt. A sprint része továbbá a sprint elején elvégzett úgy nevezett sprint tervezés, amely során kiválogatják, hogy mit is fognak csinálni ebben a sprintben. A végén pedig egy review található, ahol megbeszélik, hogy mi készült el és összefoglalják a tanulságokat ebből a sprintből (Abrahamsson et al. 2002). A postgame fázisban zárul le a projekt, itt készül egy utolsó teljes teszt és a projekt dokumentációja. Itt már nem érkezhet új követelmény (Abrahamsson et al. 2002).

A scrum keretrendszer meghatároz szerepeket is, amelyeket, mint már említettem, a pregame fázisban osztanak ki. Ezek a szerepek a Scrum Master, Product owner és a Team (Szabó - Ribényi, 2018). A Scrum Master feladata a keretrendszer betartatása, valamint az akadályok elhárítása a csapat elől. A Product owner feladata a termékfejlesztés sikerének elérése, továbbá az ügyféligényeinek felmérése (product backlog megírása és frissítése). A

harmadik fontos szereplő a Team, azaz a fejlesztő csapat, ők felelnek a tényleges fejlesztésért. A csapatszellem optimális kialakulásának érdekében, ez a csapat általában öt-kilenc főből áll (Szabó - Ribényi, 2018). Abrahamsson et al. (2002) említ még két fontos szereplőt, a megrendelőt, akinek feladata az új követelmények megfogalmazása, és a vezetőséget, akik meghozzák a végső döntéseket a követelmények között és a szerepek kiosztásában is.

A keretrendszer meghatároz különböző dokumentumokat is, melyek közül a product backlogról már szó esett. Szabó és Ribényi (2018) további kettőt említ. A sprint backlog tartalmazza a product backlog elemeit, de már az elemek prioritásának megfelelő sorrendben. A harmadik dokumentum pedig a Burndown Chart, amivel követni lehet a projekt haladását oly módon, hogy a grafikon függőleges tengelye a még elvégzendő feladatok számát mutatja, a vízszintes pedig az eltelt időt.

Mint már említettem, a scrum az egyik leghasználtabb agilis fejlesztési módszertan a Dima és Maassen (2018) és a Szabó és Ribenyi (2018) által végzett kutatásokon, ez jól mutatja a hatékonyságát, azonban a scrumnak is megvannak a hátrányai. Ezt a metódust csak kis csoportokra lehet alkalmazni (maximum tíz fő) (Abrahamsson et al. 2002), mert nagyobb csoportoknál már nem jól alkalmazható a napi stand-up meeting, valamint az általános csapat kommunikáció sem tud kialakulni.

# 2. Word plugin fejlesztés

Ebben a fejezetben bemutatom az automatizálás projekt közben használt vagy használható keretrendszereket. Szó lesz röviden a történetükről, működésükről, és a projekt szempontjából előnyeikről és hátrányaikról. Elsőként a Moodle rendszerről lesz szó, aminek a segítségével lehetett volna megoldani a számonkérés felelet választós szerkezetű lebonyolítását. Ezután szó lesz a Microsoft Office makrókról, amikkel már a jelenlegi szerkezetű dolgozat automatizálását lehetne megoldani, majd bemutatásra kerülnek a Microsoft által kínált Office plugin típusok.

A **Moodle** egy ingyenes Opensource e-learning rendszer, amely 2003-ban jelent meg először, legutóbbi verziója pedig 2022 januárjában 3.11.5 verzió számmal (Moodle, 2021). A Moodle az egyik legnépszerűbb e-learning rendszer, világszerte több mint 200 millió felhasználóval (Murillo, 2021).

A Moodle főbb funkciói között megtalálható a tanórai kurzusok létrehozása és kezelése, tananyag megosztás, feladat leadási felületek, tesztek, vizsgák és még sok minden más. Tesztek automatikus javítása is megoldott, valamint a pontszámok és érdemjegyek közzététele és

tárolása is. A Moodle-nek rengetek előnye van a diákok számára, csak párat említenék. A tananyagok egy helyen megtalálhatóak, leegyszerűsítve a tananyagok megtalálását. A Moodle felületen írt teszteknél az eredmény azonnal látható, így megszűnik az aggódás az eredmény miatt, mivel azonnali a visszajelzés. A tanárok számára előnyt jelent az, hogy egyhelyen tudják elérni az összes hallgatót, aki a csoportjukba tartozik. Ezek csak a legfontosabb pozitívumai voltak a Moodle rendszernek, ezeken kívül még rengeteg előnnyel jár a Moodle e-learning rendszer. A Moodle azonban nem tud mindent, az általam vizsgált számonkérésben például nem alkalmazható, ugyanis ennek a számonkérésnek a teszté alakításával elveszítené a gyakorlati részét. Emiatt a projektem során nem fogom a Moodle rendszert használni.

A Microsoft Office termékein belül az első lehetőséget az automatizálásra a **makró**k jelentették. Segítségükkel automatizálni lehetett az ismétlődő kattintásokat. A makrók Visual Basic for Applications (röviden VBA) nyelven íródnak, ami egy esemény-vezérelt programozási nyelv (Chi, 2000). Az első verziók 1993-ban jelentek meg, a legutóbbi verzió pedig az Office 2019-ben jelent meg (Microsoft 2021).

A VBA applikációknak fő tulajdonsága, hogy a fájlhoz kötődnek, ez előnyt és hátrányt is hordoz magával. Előnye a könnyű megosztás és folyamatos használhatóság, de ebből a könnyű megosztási lehetőségből fakad legnagyobb hátránya is. A makrók híresek a számítógépes vírusokról. 1999-ben az első nagy makró vírus majdnem fél milliárdos kárt okozott az Észak-Amerikai piacon, a 2014-ben elterjedt Emote nevű vírust pedig csak 7 év után tudták kiirtani az internetről (Gutfleish et al., 2021). A makrókkal járó veszélyek és a fájlhoz kötöttségük miatt nem alkalmasak az általam kitűzött projekt megvalósítására.

## 2.1 Office Add-In

Az Office Add-Inok a Microsoft cég által kínált megoldások olyan funkciók készítésére, amik alapból nem elérhetőek a Microsoft Office termékekben. A következőkben két típusú Add-Int fogok bemutatni, a VSTO-t és az újabb Office Web Add-Int. A projekt során ezek a rendszerek fogják képezni az alapját a kész terméknek, és segítségükkel ágyazom be az applikációt a Microsoft Word programba.

### 2.1.1 Visual Studio Tools for Office (VSTO)

Az első Visual Studio Tools for Office (VSTO) a 2003-as Microsoft Office verzióval együtt jelent meg, ez volt az első lehetőség arra, hogy az Office applikáción kívül módosítani lehessen az Office fájlokat. Ezelőtt is volt lehetőség a fájlok módosítására kódolás segítségével, de akkor még csak makrók rögzítésére és írására volt lehetőség VBA segítségével. Ezeket nem

lehetett általánosan használni, minden egyes új dokumentumnál az aktuális dokumentumra kellett őket testre szabni. Ez sok munkaidőt igényelt alapból is, nem beszélve a hibák elhárításáról, amik felléptek minden egyes módosításkor. Továbbá a VBA modell a védelem hiánya miatt a makró vírusok elterjedéséhez vezetett (Pirjan, 2015).

A 2003-as verzió azzal, hogy Visual Studioban lehetett használni, lehetővé tette a .NET keretrendszer használatát. Ezzel az újítással az applikációk a .NET assembly segítségével kapcsolódtak a Microsoft Office alkalmazásokhoz, amely védelmet jelentett az előbb említett vírusok ellen. Továbbá a Visual Studio és a .NET környezet lehetővé tette a C# kódolási nyelv használatát (Pirjan, 2015). Ez a verzió csak a Microsoft Office 2003 Word és Excel alkalmazásokat támogatta és azokban is csak dokumentum szintű változtatásokat tett lehetővé (Sempf - Jausovec, 2010).

2005-ben jelent meg a következő verzió VSTO 2005 SE néven. Ez a verzió már alap része volt a Visual Studio 2005 professional és magasabb verzióinak. Megjelent a többi 2003-as Office applikációk támogatása (Word, Excel, Power Point, Visio, Info-path, Outlook) és magasabb támogatást nyújtott a Windows Form controlok és a szerveroldali programozásnak. Elkezdődött a 2007-es Office támogatás, ezek főleg csak a következő verzióban lettek rendesen támogatva (Sempf - Jausovec, 2010).

A Visual Studio 2008-as professional verziójával jelent meg a harmadik verzió. Ez a kiadás lehetővé tette a COM (Component Object Model) Add-In-ok fejlesztését és ezzel lehetővé tette, hogy az Office applikációk szalagján (ribbon) jelenjenek meg a megírt funkciók, ezzel teljesen integrálni lehet az általunk megírt funkciókat az Office applikáció felhasználói felületébe (Pirjan, 2015).

A 2010-es Visual Studioval együtt jelent meg a negyedik verzió. Legnagyobb újítás a 64-bit-es rendszerek támogatása. Ebben a verzióban továbbá váltottak a Office client applikációk támogatásáról a SharePoint támogatásra, amivel lehetővé tették a SharePoint workflow-k használatát a VSTO applikációkban (Sempf - Jausovec, 2010). Ennek egy újabb kiadása a jelenlegi verzió. Támogatást kapott a 2013-as Office applikációk többsége, valamint folyamatos frissítések érkeznek a .NET keretrendszerekre is, ezáltal ez a verzió használható az összes 2013-asnál újabb Microsoft Office verzióra (Pirjan, 2015).

A jelenlegi verzió a Visual Studio 2019 verziójával fut, 4.7.2-es .NET keretrendszert használ. Kompatibilis Windows 10-en futtatott 2013-as vagy újabb Microsoft Office applikációkkal (Excel, Word, InfoPath, Outlook, PowerPoint, Project, Visio), C# és VB programozási nyelveken.

A VSTO Add-Inoknak előnye a Web Ad-inokkal szemben, hogy lokálisan futnak, nincs szükség szerverre a használathoz, valamint a lokalitás és a .NET keretrendszernek köszönhetően elérik az operációs rendszeren keresztül a lokális fájlokat, fájlszerkezeteket. Jobban integrálódnak az Office által használt felhasználói felületbe, ezáltal a fejlesztésben nem részvevő felhasználó is könnyedén tudja használni, hiszen hasonló ahhoz, amit már megszokott. Könnyebb fejleszteni, hiszen csak C# vagy VB (Visual Basic) tudásra van szükség hozzá addig, amíg nem akarjuk szerverhez kötni (Raymond, 2017).

Fő hátránya a VSTO típusú kiegészítő alkalmazásoknak, hogy csak Windows operációs rendszeren lehet őket használni, pont az előbb említett .NET keretrendszer miatt. További hátrányuk Web add-inokkal szemben, hogy a közzétételt vagy értékesítést nem támogatja a Microsoft, vagyis, ha értékesíteni szeretnénk a programunkat, akkor arra saját megoldást kell találni. Ha csak megosztanánk, akkor is saját rendszert kell rá tervezni, irodai környezetben például minden gépre fel kell telepíteni a rendszergazdák által (Raymond, 2017).

### 2.1.2 Office Web Add-In

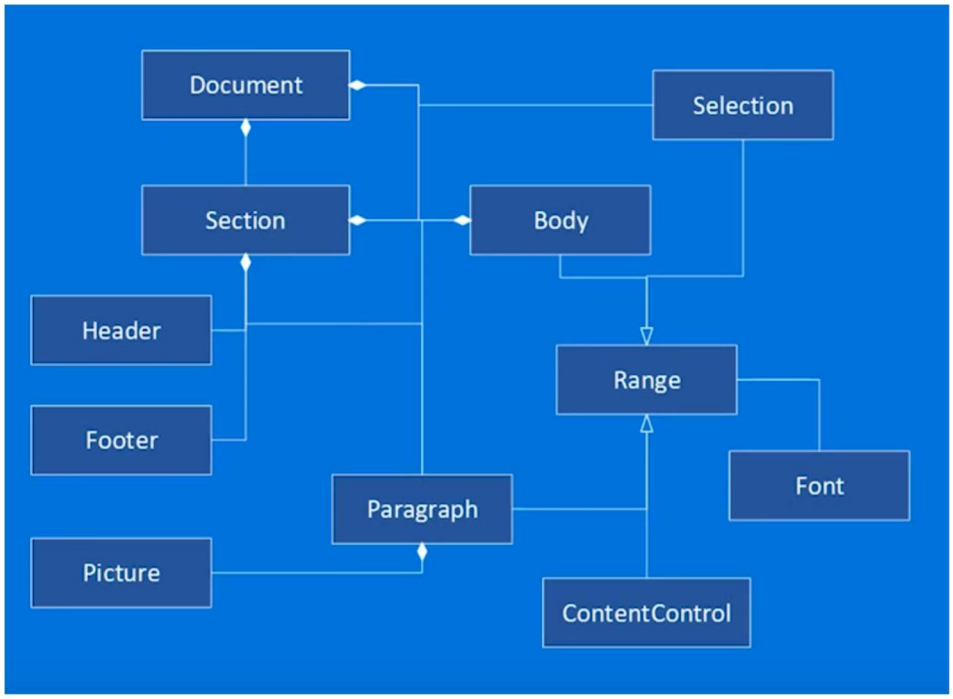
Az Office Web Add-In technológia hasonlóan a VSTO-hoz lehetőséget ad a fejlesztőknek arra, hogy saját funkciókat adjanak a Microsoft Office termékekhez. Ennek a típusú add-innak az első alapjai a 2013-as Office verzióban és az akkori Visual Studioban jelentek meg. A fő megjelenésük viszont csak a 2016-os verzióban következett be, mert ekkor kerültek bemutatásra az új fájl szerkezetek, amik közül a Wordhöz tartozó látható a 3. ábrán. Az új dokumentum szerkezet és a hozzá készült JavaScript API csomag segítségével lehetővé vált az Office web add-inok fejlesztése.

A VSTO-val ellentétben a Web Add-inok nem .NET és C# alapon futnak, a nevükből kitalálható, hogy webes alapokon működnek. HTML és CSS adja a megjelenésnek az alapját és JavaScript a funkcionalitást. A JavaScript az a Microsoft által készített API és az említett dokumentum szerkezet segítségével tud módosításokat végezni a dokumentumokon (Balmori, J., 2015). Az API (Application Programming Interfaces) definíciója: „egy olyan programozási interfészt, programozási felületet és annak részletes dokumentációját értjük, amelynek segítségével egy rendszer egy másik programhoz (esetleg rendszerprogramhoz) csatlakozhat. Ennek révén a másik programrendszer szolgáltatási használhatóak, anélkül, hogy a program belső részleteit ismerni kellene.” (industry4.hu).

A dokumentum szerkezet, az API és az API segítségével megírt JavaScript hármas adja az egyik felét egy Office Web Add-Innak, a másik fele pedig egy XML file. Ez a fájl tartalmaz alap információkat mint, az Add-In neve, leírása, verziója, tartalmazza azt, hogy az add-in

hogyan integrálódik az Office applikációba, melyik web szerveren található az add-in és tartalmazza a szükséges biztonsági feltételeket a használathoz. A Web névből továbbá adódik, hogy az applikáció nem lokálisan, hanem valamilyen webszerveren fut. Ez lehet akármilyen alapú webszerver például .NET, vagy php (Microsoft, 2021).

A Microsoft Word esetében a 3. ábrán látható az a szerkezet, amin keresztül az API tud kapcsolódni a dokumentumhoz. A 3. ábrán látható szerkezet alapján például, ha a dokumentumunkban egy kijelölt szöveget szeretnénk elérni, akkor azt a Document → Selection keresztül tehetjük meg. Így egy Range elemet kapunk vissza, ami, ahogy az ábrán is látható közvetlenül kapcsolódik a Font-hoz, szóval ezen keresztül végezhetünk változtatást a betűtípusban vagy egyéb megjelenésben.




3. ábra MS Office Word 2016 dokomentum szerkezet forrás: Balmori, J., (2015), 1:35

Az Office Web Add-In előnye a VSTO-val szemben az, hogy minden platformon működik, lehet használni Windows-on, Androidon, iOS-en, és böngésző alkalmazásban is. Nem lokálisan fut, tehát több gép is eléri egyszerre az adatbázist és nincs probléma a telepitéssel és a terjesztéssel. Céges környezetben például van lehetősége a rendszergazdának az összes felhasználónak integrálni a programot, többféle módon a saját irodájából. Továbbá a Microsoft Office-nak van saját online áruháza, ahonnan le lehet tölteni a már kész alkalmazásokat, meg lehet őket venni, és fel lehet rakni saját alkalmazást is. HTML-t és CSS-t használ a megjelenésre, tehát bármit meglehet valósítani felhasználói felületként (Balmori, 2015).

Hátránya a sok féle kódolási ismeret szükségessége, tudni kell HTML-t, CSS-t, JavaScriptet-t és azt a keretrendszert is, amit a szerver használni fog. Az előbb előnyként említettem, hogy mindent meg lehet csinálni felhasználói felületként, viszont az Office applikációk szalagjába nem lehet elhelyezni az alkalmazást úgy, mint a VSTO-k esetében, tehát nem lesz természetes a kinézete a kész applikációnak (Balmori, 2015).

## 2.2 Architektúra

A következőkben a fejlesztési projektem során használt kódolási és programozási nyelveket fogom bemutatni. Szó lesz röviden a történetükről és általános használatukról. **HTML** (Hypertext Markup Language) az a nyelv, amivel megadjuk a szerkezetét a weboldalaknak, webalkalmazásoknak. Lehetőséget ad a fejlesztőnek, hogy létre hozzon dokumentumokat fejléccel, szöveggel, táblázatokkal, listákkal, képekkel és egyéb elemekkel (W3.org, 2021).

A HTML kód először 1990-ben jelent meg, egyike a World Wide Web három alap pillérének (HTML, URL, HTTP). Az alapok letétele után folyamatos fejlesztés jellemzi a verziókat. 1995-ben jelent meg a második verzió, 1997 elején a harmadik, 1997 decemberében pedig már a negyedik verzió volt a legfrissebb. A következő verzió, a HTML 5 2014-ben lett közzé téve a W3C által, mint a hivatalos, javasolt kódolási nyelv. Ennek a kiadásnak a legújabb W3C által javasolt verziója a HTML 5.2 (W3.org, 2017)

A HTML adja a szerkezetét a weboldalaknak, a **CSS** (Cascading Style Sheets) a kinézetét. „CSS az a nyelv, amivel megadjuk a kinézetét egy weboldalnak, beleértve a színeket, elrendezést és betűtípust. Lehetővé teszi a kinézet adaptálását más képernyő méretű ezközökre. A CSS független a HTML-től és használható bármilyen XML alapú markup nyelvvel. A HTML és CSS szétválasztása egyszerűsíti a weboldalak és webalkalmazások karbantartását, hiszen egy style sheettel változtathatjuk az összes oldalunkat.” (W3.org, 2021). Az előző idézetből egyértelműen következik, hogy a CSS egy külön álló kódolási nyelv, mégis elengedhetetlen része a weboldalaknak. Mint már többször említettem, a HTML és CSS nyelveket főleg weboldalak szerkezetének és kinézetének meghatározásához használják. Én a projektem során az Office Web Add-in felhasználói felületének kialakításához fogom használni.

A **JavaScript** programozási nyelv 1995-ben látta meg a napvilágot, Brendan Eich készítette a Netscape Communications céggel együtt működve (Netscape volt ebben az időben a legelterjedtebb böngésző). Azzal a céllal készült, hogy dinamikussá tudja tenni az eddig statikus weboldalakat kliens oldali programok segítségével (Ranjan et al., 2020).

A legfontosabb történés a JavaScript korai életében az ECMA standardizáció volt. Ez a folyamat 1996 novemberében kezdődött, ekkora már teljes használatban volt weboldalakon a programozási nyelv. Az első standardizált verzió 1997-ben jelent meg, és a legutóbbi, a tízes verziót pedig 2019-ben adták ki (Ranjan et al., 2020).

A JavaScript nyelv elsősorban, a weboldalak kliens oldali dinamikájáért felel. Lehetséges vele módosítani a weboldalak statikus HTML és CSS szerkezetét. Különböző JavaScript alapú keretrendszerek segítségével lehetséges backend applikációkat is létrehozni. Én a projektem során az Office Web Add-In fejlesztésénél fogom használni, mint a kliens oldali logika.

**PHP** (Hypertext Preprocessor) egy elterjedt nyíltforráskódú általánosan használható programozási nyelv, ami kifejezetten jól használható webfejlesztéskor, mert beágyazható HTML-be (php.net, 2021). 1994-ben jelent meg először, Rasmus Lerdorf készítette saját online önéletrajzának látogatottságának figyelésére, innen ered az eredeti név „Personal Home Page Tools", 1995-ben készült el az első publikus verzió, amit más fejlesztők úgy használhattak, ahogy szerették (php.net, 2021). Ezután 1997-ben jelent meg a harmadik, majd 1999-ben a negyedik verzió, amely az új Zend Engine-nel működött. Jelenleg a hetedik verzió a legelterjedtebb, de már tesztelés alatt van a nyolcadik is (php.net, 2021).

A PHP-t három fő területen használják: szerver oldali applikációk írására, parancssori utasításokra operációs rendszereken és asztali alkalmazások fejlesztésére, bár ez nem javasolt az egyszerű grafikus felhasználói felület írási képesség hiánya miatt (php.net, 2021). Én a projektemben az Office Web Add-in szerver oldali alkalmazásának a megírásához fogom használni.

# 3. A projekt bemutatása

A szakdolgozatom témájaként a Budapesti Corvinus Egyetem Gazdaságinformatikus BSc képzésének a szakszeminárium tárgy hivatkozásszámonkérés zárthelyi dolgozatának javításának az automatizálását választottam. A következőkben ismertetem a projektem fázisait a vízesésmodell szakaszait követve. Az első alfejezetben bemutatom a projekt kezdetekor fennálló megoldandó problémát.

## 3.1 A megoldandó probléma

A hivatkozásszámonkérés dolgozat automatizálásának az ötlete akkor született meg, amikor először vettem részt a Szakszeminárium 1 tárgyon. A tanárom panaszkodott, hogy

nagyon sok idő ezeket a dolgozatokat javítani és ahhoz képest, hogy mennyi idő elmegy vele, a tárgynak a jegyét nem befolyásolja akkora mértékben. Viszont elhagyni sem lehet, hiszen a tárgynak egyik kulcs feladata a helyes hivatkozás megtanítása. A javítási idő hosszának az oka a dolgozat szerkezete volt, egy dolgozatnak a javítása sok figyelmet igényelt. A dolgozat szerkezetét azonban nem akarták megváltoztatni, mert ez a módszer a gyakorlati tudást ellenőrizte.

A dolgozat egy másfél-két oldalas Microsoft Word dokumentum volt, amely egy tudományos szöveget tartalmazott tele hivatkozásokkal és a hozzátartozó irodalomjegyzékkel. Ezekben a hivatkozásokban és az irodalomjegyzékben tíz hivatkozási hiba volt elrejtve, és a hallgatók feladata volt ezeket megtalálni és kijavítani.

Ez a szerkezet lehetővé tette, hogy azt ellenőrizzék, hogy a hallgatók felismerik-e a helyes és helytelen hivatkozásokat és azokat ki tudják-e javítani, valamint egy olyan szituációt tudott utánozni, mint ami a szakdolgozat írás közben is felmerül, a saját dolgozat átnézését.

A javítás nehézsége pont ebből a szerkezetből adódott, hiszen ezeket a dolgozatokat javítani is úgy kell, mint ahogy megoldani, nagy figyelemmel végig nézni minden egyes hivatkozást és eldönteni, hogy helyes-e vagy sem. Ez a folyamat 5-10 percet vesz igénybe dolgozatonként, ami csoportonként 30 diákkal számolva körülbelül három és félórát vesz igénybe, és általában egy tanárnak nem csak egy csoportja van. A cél tehát ennek a javítási időnek a csökkentése, valamilyen automatizálás segítségével.

## 3.2 Automatizálási lehetőségek

A számonkérések automatizálása nem egy új ötlet az egyetemek körében, már a 2017-es félévben is volt olyan tárgyam, amit teljesen automatizálva javítottak az egyetemen. Ennek a hallgatók számára az volt a legnagyobb előnye, hogy azonnali visszajelzést kaptak a dolgozatuk vagy vizsgájuk eredményéről. A tanárok számára pedig munkát spórolt meg, hiszen nem kellet a dolgozatok javításával időt tölteni.

### 3.2.1 Moodle rendszer

Ezek az automatizálások többnyire a Moodle felület segítségével valósultak meg. Ahogy már az elméleti részben is említettem, ez a rendszer rengetek előnnyel rendelkezik, ezek közül megemlíteném az ehhez a témához kapcsolódókat. Előnye egy saját fejlesztésű applikációval szemben, hogy egységes, tehát nem kell minden tantárgyhoz külön rendszert fejleszteni és a tanároknak a kezelésület megtanítani. További előnye, hogy böngésző alapú, nem igényel telepítést a gépeken, és akárhány eszközről használható ugyanolyan módon,

változás nélkül. A web alapúság előnyt jelent a diákok számára is, akár az okostelefonon is elérhetőek a tesztek, teszt eredmények. A koronavírus ideje alatt pedig az egyetem elengedhetetlen részéve vált, mivel nélküle nem valósulhatott volna meg az online oktatás és az online számonkérés.

Az online oktatás ideje alatt az általam vizsgált zárthelyi dolgozatot is online kellett lebonyolítani, a 2020/2021-es tavaszi félévben Moodle teszt váltotta fel a Word dokumentum alapú számonkérést. Ez a teszt egyszerűen tíz felelet választós kérdésből állt, amelyeknél három vagy négy válaszlehetőség közül kellett kiválasztani a helyes hivatkozást, vagy kiválasztani, melyik illik a szövegbe. Én ebben a félévben is részt vettem ezen a számonkérésen, és saját tapasztalatból mondhatom, hogy hiányzott belőle a gyakorlati felhasználásra való felkészítés, és az, hogy ezt a szakdolgozat témát írom, azt bizonyítja, hogy a tárgyfelelős tanárok is így gondolják.

Ez a számonkérés mutatja meg a Moodle egyik gyengeségét az automatizálás terén. Moodle teszt segítségével, amelyet automatikusan javít a rendszer, nehéz olyan szöveg alapú feladatsort összeállítani, amely a gyakorlati és nem csak az elméleti tudást vizsgálja. Tehát erre a problémára a Moodle nem megfelelő.

### 3.2.2 Microsoft Office megoldások

Az eredeti dolgozat formájából kiindulva a következő gondolatom az eredeti dolgozat ellenőrzésének megoldása, valamilyen Microsoft Office-on belüli megoldással volt. Erre a legegyszerűbb megoldás egy Word makró fejlesztése lett volna, viszont ahogy már az elméleti részben is említettem, a makrók sok hibát rejtenek magukban. Az Office makrók általános hibája a kód nehéz követhetősége, a dokumentáció teljes hiánya vagy az utólagos javítások, fejlesztések dokumentálásának hiánya. Általános használata is problémás, hiszen minden dokumentumra külön kell telepíteni és az sem garantált, hogy minden gépen egyforma működés várható. Ebben az esetben azonban a legnagyobb probléma a makró használattal az, hogy a forráskód könnyű terjedése miatt könnyen a hallgatók kezébe kerülhet, ami az egész számonkérést veszélyezteti. A makró egy javított dolgozat megtekintésekor is kikerül, ha azt nem távolítják el visszaosztás előtt, ami egy olyan kockázatot jelent, amit nem lehet megengedni. Tehát Office makróval nem lehet megoldani a javítást a magas kockázat miatt.

A következő Office-on belüli megoldás a valamilyen plugin vagy add-in fejlesztése. Erre, mint ahogy a kutatási részben kifejtettem, kétféle opció van, a Visual Studio Tools for Office (VSTO) és az Office Web Add-in. A következő bekezdésekben ezeknek az előnyeit és hátrányait fogom leírni a probléma megoldásával kapcsolatban.

A **Visual Studio Tools for Office** (VSTO) megoldás, mint ahogy a kutatási részben is szó volt róla, a Microsoft első megoldása a saját programok integrálására az Office környezetekben. Egy ilyen típusú plugin megoldás tudna lenni az általam felvetett automatizálási projektre. Ezeket az alkalmazásokat C# nyelven kell megírni, ami az iskola szempontjából előnyös, hiszen ez az a programozási nyelv, amit tanítanak a Gazdaságinformatikus szakon a hallgatóknak, tehát vannak tanárok, akik részt tudnának venni a fejlesztésben és a karbantartásban. További előnye lenne a Web Add-in megoldással szemben az egyszerűsége, nincs szükség hozzá web- vagy adatbázisszerverre.

Az utolsóként felsorolt előnyből adódnak a VSTO hátrányai és hiányosságai is, azáltal, hogy nincs szükség szerverre, a tárolt adatok nem tárolódnak központilag, tehát a mentett beállítások és feladatsorok nem lennének elérhetőek az összes javító oktató számára. A program lokális mivoltából adódik az a probléma, hogy minden egyes használandó gépre külön kell telepíteni a programot, tehát a program terjesztését is külön meg kell oldani, ami rendszergazda jogosultság nélkül problémás az egyetem tulajdonában lévő számítógépeken.

A projektem során nem a VSTO megoldást alkalmaztam az előbb felsorolt hátrányok miatt, továbbá, mert számomra fontos volt, hogy az újabb megoldást alkalmazzam, hogy az évek során ne, vagy csak később szűnjen meg a támogatás az elkészült programhoz.

Az **Office Web Add-in** a Microsoft legújabb megoldása az Office alkalmazások saját funkciókkal való kiegészítésére. Ez a típusú applikáció is képes megoldani a felvetett automatizálási projektet. Előnye a webes szerkezet, aminek köszönhetően nem kell programot telepíteni egyik gépre sem, csak az elérést kell beállítani minden gépen. Ezt az egyetemi felhasználókon keresztül központilag is meg lehet tenni, de ha nem is központilag tesszük ezt meg, akkor is csak az Office alkalmazásokon belül kell ezt a beállítást elvégezni. További előnye, hogy nem csak Windows-os gépeken működik, felkínálva az opciót a tanároknak, hogy nem csak az egyetemi gépeken, hanem otthon is bármilyen operációs rendszeren is tudják használni.

Ennek a megoldásnak is vannak hátrányai, első és talán legnagyobb a webes kialakításból adódik, szükség van egy webszerverre, amelyiken futhat az alkalmazás, de valamilyen szerverre más megoldások mellett is szükség lett volna a központi adat tárolás miatt. További hátránya a sok féle nyelv, amit ismerni kell a fejlesztéshez, de megoldható minden a webfejlesztéshez használt nyelvekkel (HTML, CSS, JavaScript, PHP), amelyeknek az ismerete amúgy is együtt jár.

Projektemhez az Office web add-in megoldást választottam, tehát egy Word web add-int fejlesztettem. A projekt célkitűzése egy olyan applikáció készítése, ami segítséget nyújt a

Word alapú dolgozatok javításában, különös tekintettel a Gazdaságinformatikus alap képzés Szakszeminárium 1 tárgyához tartozó hivatkozási ismereteket számonkérő dolgozatra.

## 3.3 Megoldás bemutatása

Ebben az alfejezetben bemutatom a kitűzött célra kínált megoldás projektjének menetét, a vízesésmodell fázisai segítségével. A projekt során ezt a modellt alkalmaztam az egyedüli fejlesztés miatt és a rendszeres megbeszélések elkerülése érdekében, amire nem lett volna ideje az oktatóknak. A továbbfejlesztéshez azonban agilis életciklus modellt javaslok, hogy az elkészült applikáció minél jobban lefedje az aktuális tárgyfelelős igényeit.

### 3.3.1 Követelmények

A követelmények összeállítása előtt elemezni kellett a kitűzött célt és a rendelkezésre álló eszközöket. Korábban már bemutattam ezeket a megoldásokat, amik közül a Word Web Add-int választottam, mivel alkalmas a cél megvalósítására és nekem a hozzá szükséges programozási nyelvekben volt nagyobb tapasztalatom.

A cél elemzése során megállapodtunk abban Burka Dáviddal és a tantárgy akkori tárgyfelelősével, Vas Rékával, hogy a javítás automatizálását akkor lehet kisebb hibahatárral megoldani, ha a feladatsor összeállítását is a javításhoz használt applikáció használatával kell elvégezni. Ezzel egyszerűsítve a javítást, hiszen ebben az esetben a programnak nem kell ismernie a hivatkozási szabályokat, csak azt kell tudnia, hogy mi a helyes megoldás a feladatra, amit pedig az összeállítás során el tudunk menteni. Ezzel meg is született az első kettő követelmény, az applikációnak tartalmaznia kell egy javítás és egy dolgozat összeállítás funkciót.

A két első követelmény meghatározása után készült egy Proof of Concept (PoC) alkalmazás, amelynek célja az volt, hogy megállapítsa, lehetséges-e az általunk elképzelt funkciók megvalósítása, amiben nem voltam biztos, mivel korábban nem készítettem Word Web Add-int. A PoC alkalmazás elkészült és működő képes volt, tehát folytathattuk a követelmények összegyűjtését. A Tanár Úr javaslatára következő követelményként azt fogalmaztam meg, hogy az alkalmazás képes legyen több feladatsor készítésére ugyanabból a Word dokumentumból, azaz legyen lehetőség A, B, C stb. feladatsor készítésére, lehetőleg kevesebb munkával, mint az első feladat elkészítésével. Ezt az első nagy funkció (feladatsor összeállítás) ketté szedésével terveztem megoldani.

Negyedik követelményként az előző kiegészítésére meghatároztunk egy reset funkció szükségességét, amely lehetővé teszi az eredeti szöveg visszaállítását, a különböző feladatsorok

készítése során. Továbbá megállapítottuk a törlés funkció szükségességét, amellyel már elhelyezett feladatokat lehet kiszedni a dokumentumból.

A felhasználói felülettel kapcsolatban nem állapítottunk meg részletes követelményeket, viszont leszögeztük, hogy az elkészült alkalmazásnak felhasználóbarát kezelő felülettel kell rendelkeznie és a Microsoft Word általános megjelenésétől nem nagyban eltérő felületet kell kialakítani. Ezzel az utolsó követelménnyel összeállt a következő követelmény lista:

1. Dokumentum előkészítése funkció fejlesztése, amelynek segítségével el lehet menteni az összes lehetséges feladat helyét.
2. Hibák elhelyezése funkció fejlesztése, amely lehetségesé teszi több feladatsor készítését.
3. Javítás funkció fejlesztése, amely képes megtalálni a hallgatók helyes és helytelen megoldásait.
4. Reset funkció fejlesztése, aminek segítségével vissza lehet állítani a dokumentum előkészítés utáni állapotot.
5. Törlés funkciók fejlesztése, amelynek a segítségével törölni lehet a tévesen elhelyezett feladatokat.
6. Felhasználóbarát kezelő felület, amely illeszkedik a Microsoft Word stílusához és lehetővé teszi a felsorolt funkciók használatát.

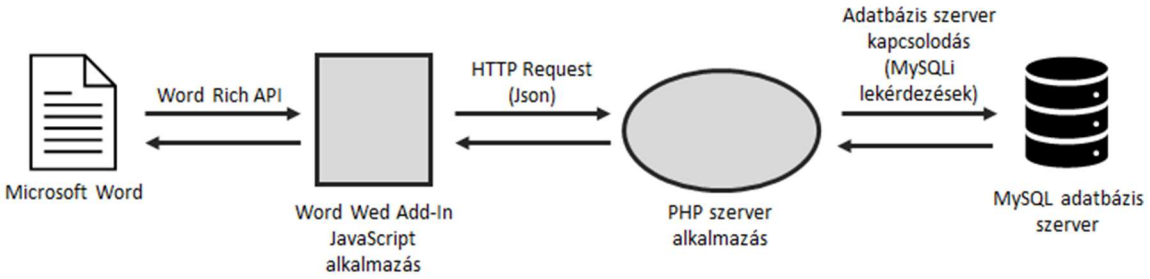
A lista átnézése után haladtam tovább a tervezési fázisra.

### 3.3.2 Tervezés

A Tervezés fázis alapjait a PoC applikáció készítése során el kellett végezni. A fázis első lépéseként megállapítottam, hogy a webes működés érdekében, és azért, hogy az eszköz használható legyen több rendszeren fennakadás nélkül, szükség lesz egy adatbázis szerverre. A fejlesztés során az egyszerűség érdekében az EasyPHP devserver megoldását alkalmaztam. Ez a megoldás egy Apache alapú php szerver, amely már tartalmazza a MySQL modult és a hozzá használt PhpMyAdmin kezelő felületet. A fejlesztés során ez a szerver adott otthont backend elemeknek és az applikációnak. A frontend alkalmazás futtatásához is szükség van webszerverre, a fejlesztés során azonban erről a szerverről a Visual Studio gondoskodik, egy ISS localhost szerver formájában. Az éles használat során használt szerverre majd az tesztelés fázisában fogok kitérni.

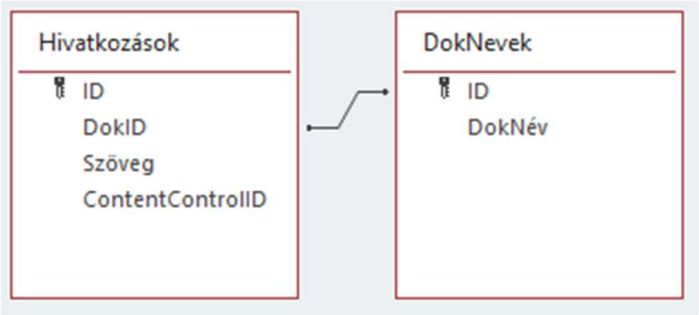
A PoC applikáció tervezése során tehát ezeken a szervereken futó kódokat, a szerverek közti kommunikációt és az adatbázis szerkezetet kellett megtervezni, amely terveket az elkészült alkalmazás tervezésénél is felhasználtam.

A 4-es ábra különböző alkalmazások közötti kapcsolatot mutatja be. A tervezés első lépéseként a Microsoft Word és a web add-in alkalmazás közötti adatátvitelt kellett megtervezni. Erre a célra az elméleti részben bemutatott 3-as ábrán látható ContentControl funkciót használtam. Ennek segítségével tárolni tudjuk a hivatkozások helyét a dokumentumban, anélkül, hogy az az elkészült program segítsége nélkül szerkeszthető lenne. Valamint egy könnyen azonosítható és könnyen módosítható felületet ad a fejlesztett funkciók számára.



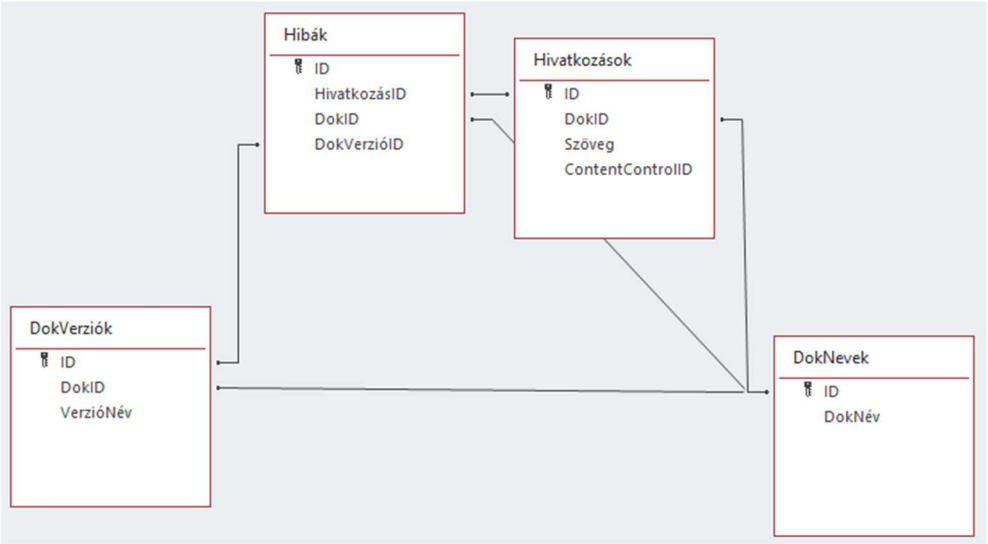

*4. ábra FrontEnd és BackEnd applikációk kapcsolata (Saját szerkesztés)*

A második feladat az adatbázis megtervezése volt. Az 5-ös ábrán látható a PoC applikáció adatbázis szerkezete. Ennek a szerkezetnek a tervezése során az egyszerűségre kellett törekedni, viszont nem lehetett csak egy tábla, mivel akkor nem bizonyosult volna az, hogy az elképzelt alkalmazás szerkezet képes több táblával dolgozni. Ez azért fontos, mert az adat duplikáció elkerülése érdekében az alkalmazás adatbázisában mindenképpen több táblára van szükség. Ezért döntöttem a két táblás szerkezet mellett. Az 5-ös ábrán látható Hivatkozások tábla szöveg oszlopa tárolja a tárolt hivatkozások szövegét, a ContentControlID pedig a dokumentumban elhelyezett ContentControl azonosítóját tartalmazza, amelynek a segítségével találja meg a program a hivatkozások helyét. A DokNevek tábla szolgál a dolgozatok nevének tárolására, ennek a táblának az ID oszlopának a segítségével történik a hivatkozások elosztása, hogy mely dokumentumhoz tartoznak.




*5. ábra Proof of Concept adatbázis szerkezet*
*(Saját szerkesztés)*

A 6-os ábrán látható az elkészült alkalmazás adatbázis szerkezete. A PoC adatbázis szerkezethez (5. ábra) képest két új táblát kellett hozzáadni. Ezeket a táblákat amiatt kellett hozzáadni, hogy a második követelményt teljesíteni tudja az alkalmazás. A DokVerziók tábla tárolja a dokumentumhoz tartozó verziók neveit, és azokat összekapcsolja a dokumentumok nevével. A Hibák tábla pedig összeköti a Hivatkozások táblát a DokVerziók táblával az ID-k segítségével. Így rögzítve, hogy melyik verzióban mely hivatkozások hibásak. Ez az adatbázis szerkezet megfelelőnek bizonyult a kódolás során, tehát nem történt rajta további módosítás.



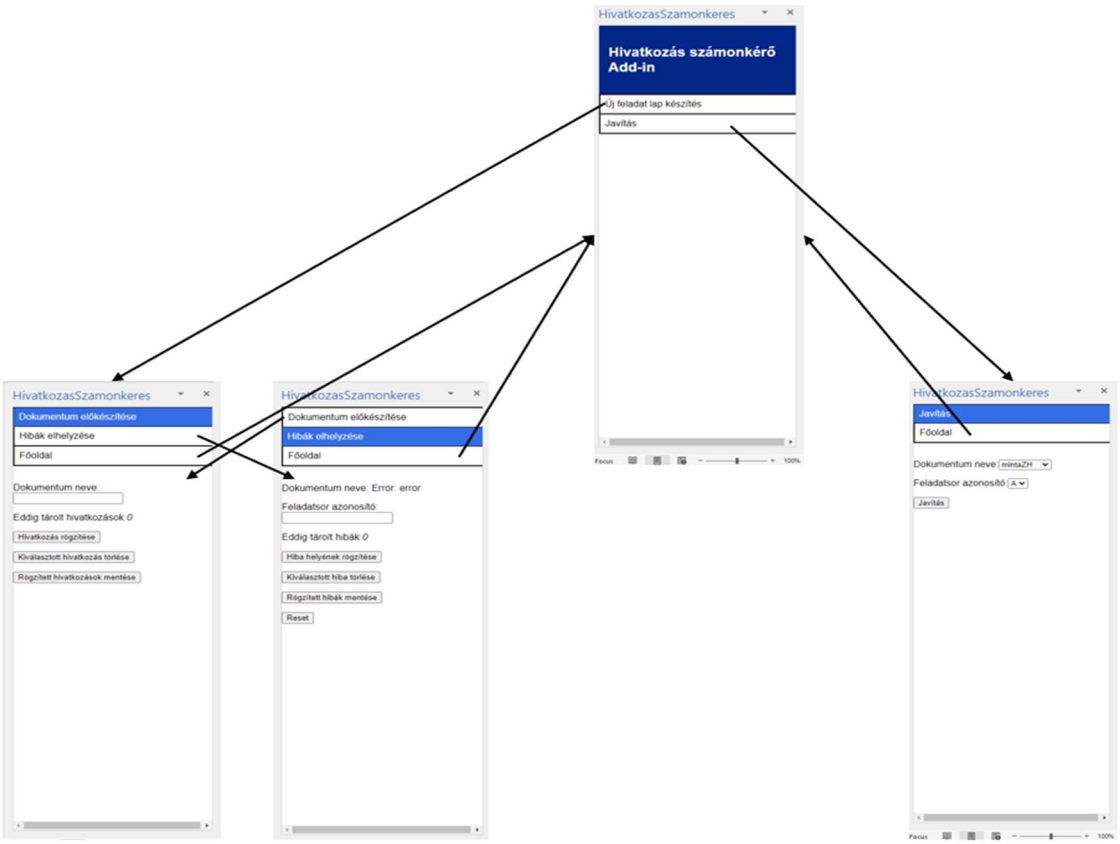

*6. ábra Alkalmazás adatbázis szerkezet (Saját szerkesztés)*

Harmadik feladatként a web add-in és az adatbázis közötti kapcsolatot kellett kialakítani. Ahogy az a 4-es ábrán látható, erre a feladatra egy php szerver alkalmazást terveztem, amelynek fő feladata a web add-in-tól beérkezett adatok felvétele az adatbázis rendszerbe. A php alkalmazással az add-in http requestek segítségével kommunikál, a tervezés során minden request típusra külön php program fájl készült. A requestek JSON szerkezetű szövegek segítségével kommunikálnak, tehát a php programok feladat ezeknek a szövegeknek a feldolgozása, adatbázisba írása vagy lekérdezés, és válasz JSON szövegek készítése. Az

adatbázist a php programok az adatbázisra való kapcsolódással képesek módosítani és lekérdezéseket futtatni. Ennek a feladatnak a megtervezésével elkészült a 4. ábrán látható applikáció kapcsolati ábra.

A következő feladat az előbb említett JSON üzenet szerkezetének megtervezése volt. A tervezés során a fő szempont az volt, hogy a kliens applikáció könnyen tudja generálni ezeket a karakter sorokat. Ennek a célnak az elérése érdekében létre kellett hozni egy olyan objektumot a JavaScript alkalmazásban, amiből a JSON.stringify funkció képes könnyen értelmezhető JSON szöveget készíteni. A követelmények során felsorolt funkciók elemzése után megállapítottam, hogy két ilyen objektumra lesz szükség, egyre a hivatkozások helyének mentésére, és egy másikra az elhelyezett hibák tárolására. Ezen objektumok szerkezete megegyezik, csak a tárolt adatok változnak. Mindkettő három tömböt tartalmaz, amelyek a hivatkozások helyének mentése esetén tartalmazzák a dokumentum nevét, a hivatkozás szövegét és a hozzáadott ContentControl azonosítóját. A hibák mentése objektum pedig tartalmazza a dokumentum nevét, a feladatsor azonosítóját, és a ContentControl azonosítót. A http requestek számának csökkentése érdekében, az objektumokban tárolt adatokat csak az adott folyamat befejezése után küldjük el, így csökkentve a szerver terhelését. Az objektumokban található tömbökben az azonos sorszámmal rendelkező adatok jelölnek egy hivatkozást vagy hibát. Az ötös követelmény (Törlés funkció) is ebben az objektumban történő törléssel valósul meg.

A kommunikációs szerkezet tervezését követően a következő feladat a felhasználói felület (hatodik követelmény) megtervezése volt. Ezt, ahogy az elméleti részben említettem, html oldalak segítségével lehet megvalósítani. A funkciók elemzése után megállapítottam, hogy szükség lesz egy főoldalra, valamint minden fő funkciónak létre kell hozni egy külön oldalt, tehát 4 html oldalra van szükség. A főoldalnak tartalmaznia kell egy linket a dokumentum előkészítés funkcióhoz tartozó oldalra, és egy másikat a javítás funkcióhoz tartozó oldalra. Minden funkcióhoz tartozó oldalnak tartalmaznia kell egy, a főoldalra vezető linket, valamint a dokumentum előkészítés oldalon kell lennie egy, a hibák elhelyezése oldalra vezető linknek. Ezen az oldalon kell egy visszavezető link a dokumentum előkészítés oldalra. A funkció oldalaknak továbbá tartalmaznia kell minden szükséges beviteli mezőt a funkciók ellátásához. A leírtak szemléltetésére szolgál a 7-es ábra, ahol az elkészült html oldalak láthatóak, a nyilak pedig a linkek hivatkozásait mutatják. Az oldalaknak egy CSS fájlt kell használniuk a könnyebb továbbfejlesztés érdekében.




*7. ábra Alkalmazás HTML oldal kapcsolat (Saját szerkesztés)*

### 3.3.3 Implementálás

Az implementálás fázis során a tervezés alatt megtervezett szerkezeteket és funkciókat kellett implementálni, és ezáltal elkészíteni az alkalmazást. Ebben az alfejezetben bemutatásra kerülnek ennek a fázisnak a lépései, és részletesen bemutatom a kód azon elemeit, amelyek az alapját képezik az elkészült alkalmazásnak. A kódolás folyamat első lépéseként elkészítettem a megtervezett html oldalak vázát  tag-ek segítségével. Ezután létrehoztam a tervezés fázisban említett linkeket az oldalakon a hozzájuk tartozó osztálynevekkel és előkészítettem a beviteli mezők helyét az oldalakon. Ezeket a mezőket akkor adtam hozzá az oldalakhoz, amikor a hozzájuk tartozó funkciót készítettem el. A html vázak elkészítése után írtam meg a hozzájuk tartozó CSS-t, ahol kialakítottam a linkek kinézetét, valamint az oldalak témáját, hogy azok hasonlítsanak a Microsoft Word kezelő felületének témájához.

A kezelő felületek alapjai elkészítése után készítettem el a kliens oldali applikációkat. A három fő funkciót (Dokumentum előkészítés, Hibák elhelyezése, Javítás) külön JavaScript programban készítettem el. A **Dokumentum előkészítés** a dokElo.js program segítségével történik. Ennek az oldalnak a feladata a dokumentumban található hivatkozások helyének rögzítése és továbbítása a php szerver számára. A rögzítés mellet képes a még nem továbbított

adatok törlésére is. A rögzített adatok a továbbítás elött egy objektumban tárolódnak, amelynek váza a tervezés során készült el. Az adatokat a Microsoft Word API segítségével nyerjük ki a dokumentumokból. Az API használásához követni kell egy sémát, különben nem működik.

A 8-as ábrán látható newCitation funkció egy jó példa az imént említett sémára, ezt a szerkezetet alkalmazom minden esetben, amikor a dokumentumon változtatást végzek vagy adatot nyerek ki belőle. A newCitation funkció feladata a kijelölt szöveg ContentControl-lá alakítása, ennek a ContentControl-nak a színezése, szövegének és azonosítójának objektumba másolása, valamint a tárolt hivatkozások számát jelző felület frissítése. A Word API kialakítása miatt async funkcióként kell megírni ezt a funkciót. A newCitation funkción belül található a Word.run funkció, aminek a segítségével végezhetőek az API műveletek, az itt indított funkció is async funkció. Mivel a változtatások során végig meg kell várni a dokumentum frissülését, ezt a feladatot látja el az await context.sync() sor, amely addig nem lépteti tovább a programot, ameddig a dokumentum nem frissül és az új adat (context) megérkezik a JavaScript applikáció számára. Az await context.sync() parancsot minden olyan változtatás után meg kell hívni, amikor az előző változtatásokkal szeretnénk dolgozni. A példában például szükség van erre a műveletre a hatodik sorban, miután elmentjük a kijelölt szöveget a range változóba, majd ezt a változtatást visszatöltjük a dokumentumba. Továbbá szükség van rá a 14. sorban, ami előtt a korábban létrehozott range változó köré létrehozzuk a wordCC ContentControlt. Ennek a ContentControlnak a visszatöltése után szükséges ismét az await context.sync() parancs, hiszen a következőkben ennek a változónak szeretnénk az adatait kinyerni és az objektumba írni. Azért van szükség a ContentControl létrehozására, mert a továbbiakban ez alapján találja meg a program a hivatkozásokat. A dokElo.js program ezt a sémát használja a törlés funkciónál is, abban a funkcióban a kiválasztott ContentControlt töröljük a hozzátartozó adatok objektumból történő törlése után.

```
async function newCitation() {

    await Word.run(async (context) => {
        var range = context.document.getSelection();
        context.load(range, 'text');
        await context.sync();

        let wordCC = range.insertContentControl();
        wordCC.font.highlightColor = 'gray';
        wordCC.appearance = 'BoundingBox';

        wordCC.load();

        await context.sync();

        text = wordCC.text;
        ccId = wordCC.id;

        jsonData.DocName[counter] = docName;
        jsonData.Text[counter] = encodeURIComponent(text);
        jsonData.CCid[counter] = ccId;
        $('#tarolt').html(jsonData.CCid.length);
        counter++;
    });
}
```

*8. ábra dokElo.js newCitation funkció (Részlet a forráskódból)*

Ahogy már említettem a dokElo.js applikáció másik fő feladata az adatok továbbítása a php szerver számára. Ezt a feladatot a 9-es ábrán látható sender funkció látja el. Ez a funkció $.ajax jQuery metódus segítségével továbbítja http request formájában az adatokat. Ennél a metódusnál az ábrán látható sorok segítségével lehet megadni a request adatait és hogy mi történjen a request sikeres és sikertelen lefutása esetén. A type változóban lehet megadni a request típusát. A projekt során végig post requesteket használok az egyszerűség kedvéért. Az url változóban kell megadni az elérési útvonalat a php szervernek és ehhez a requesthez tartozó php applikációnak. A data változóba kell beírni a küldésre váró szöveget, amit az ábrán a negyedik sorban hozok létre a tervezés során megtervezett objektumból. Ahogy a tervezésnél említettem, ezt az átalakítást a JSON.Stringify paranccsal egy lépésben el tudom végezni. A következő három változó értéke false a mi esetünkben, ezekkel a változókkal további beállításokat lehet megadni a requestnek, de a program ezeket nem használja fel. A mimeType segítségével adjuk meg a szervernek, hogy milyen típusú adatot küldünk, ezzel segítve a feldolgozást. A success változóban pedig készíthetünk egy funkciót, ami akkor fut le, ha sikeres volt a request elküldése, továbbá az itt található result változó tartalmazza a szervertől érkezett választ, ami a példánk esetében csak egy rövid szöveg, viszont más esetekben egy JSON fájl is lehet. A .fail funkció pedig akkor fut le, ha nem sikerült valamilyen okból a request, a példában ebben az esetben egy hiba üzenetet íratunk ki a programmal. Az előbb bemutatott $.ajax metódust használom az alkalmazás során mindig, ha a szerverrel kell kommunikálni. Ezen fő elem használatával elkészült a dokElo.js program. A helyes működését az összeállított JSON szöveg helyességével ellenőriztem. A következő feladat az adatbázis szerver elkészítése volt.

```
function sender() {
    $('#deleteDone').html('');

    var json = JSON.stringify(jsonData);

    jsonData.DocName.length = 0;
    jsonData.Text.length = 0;
    jsonData.CCid.length = 0;

    counter = 0;

    $('#saveDone').html(' ');

    $.ajax({
        type: "POST",
        url: "http://localhost:8080/v3/dokelo.php",
        data: json,
        contentType: false,
        cache: false,
        processData: false,
        mimeType:'multipart/form-data',
        success: function (result) {
            $('#saveDone').html(result);
        },

    }).fail((jqXHR, error) => {
        $('#saveDone').html(new Error(error), "  ", new Error(jqXHR));
    });

}
```

*9. ábra dokElo.js sender funkció (Részlet a forráskódból)*

A fejlesztés során a már említett easyPHP devservert használtam az adatbázis szerver futtatására. A beépített phpMyAdmin felület segítségével létrehoztam a megtervezett adatbázist. Ezután elkészítettem a dokElo.js sender funkciójához (9. ábra) tartozó php programot, a dokelo.php-t. Ez az applikáció a megkapott JSON szöveg adatait az adatbázisba menti. Ehhez először csatlakoznia kell az adatbázishoz, ezt a feladatot a connect.php program látja el. Ennek a programnak a meghívása után, a megkapott JSON szöveget lokális tömbökbe menti. Majd ezen tömbök segítségével elkészíti az INSERT INTO mysqli parancsot. Amennyiben a dokumentumból még nem érkezett adat, akkor a dokNevek táblához is hozzá adja a szükséges sort. Ha sikeresen megtörténik a mentés, akkor válaszként visszaadja a mentett sorok számát. Ha hiba történik, akkor a hibaüzenet lesz a válasz. Ennek a programnak a megírása után visszatértem a kliens oldali alkalmazáshoz.

A következő kliens oldali alkalmazás a hibaMentes.js. Ennek az alkalmazásnak a felépítése sokban hasonlít a dokElo.js-hoz, ezért ezt nem mutatom be olyan részletesen. Az alkalmazás feladata egy, a dokumentumban kiválasztott ContentControl adatainak tárolása, majd továbbítása a php szervernek. Itt is van lehetőség a törlésre, valamint a követelmények között említett reset funkciót is tartalmazza. A továbbítás funkcióhoz tartozik a hibamentés.php, amely szintén nagyban hasonlít a dokelo.php programhoz, ezért ezt sem fogom részletesebben

bemutatni. Ez a php program a Hibák és a DokVerzió táblába menti a kapott adatokat. A reset funkcióhoz is tartozik php program, ami egy lekérdező típusú applikáció, az ilyen típusú php programokat később fogom bemutatni.

A Dokumentum előkészítés és a Hiba mentés funkciók megírása után következett a Javítás funkció elkészítése. Ezt a funkciót a javitas.js fájl tartalmazza. A javítás során egy $.ajax metódus segítségével lekérdezzük a kiválasztott dokumentumhoz és verziójához tartozó adatokat. Ezt a lekérdezést a visszatoltes2.php program végzi, ez a program a reset funkciónál már említett lekérdező típusú applikációk egyike. Ezek az applikációk egyszerűbb felépítésűek, mint a mentés típusú programok, egy lekérdezéssel lekérdezik az adott dokumentumhoz tartozó összes elemet. A kapott eredményeket egy while ciklus segítségével egy tömbbe írják, majd a json_encode segítségével létrehozzák a JSON szöveget, amit a JavaScript alkalmazásban a result változó fog tartalmazni. A javitas.js alkalmazás esetében ezt a szöveget JSON.parse metódus segítségével egy objektumba másoljuk. Ezt az objektumot a findErrors nevű async funkció használja fel, segítségével megkeresi azokat a ContentControlokat, amelyekben nem a helyes szöveg szerepel és mögéjük íratja a helyeset, valamint háttér színezéssel látja el őket. Egy másik funkció ezután átszínezi azokat a ContentControlokat, amelyek feladatok voltak, így lehetőséget adva a tanároknak az osztályzás módjának eldöntésére.

A fejlesztés során a beviteli mezők szövegdobozok voltak, amelyekre semmilyen ellenőrzés nem vonatkozott. Ez a fejlesztéshez jó, mert egyszerű, azonban a felhasználást nagyon nehézzé teszi. Ezért, ahol már meglévő értékekből kell választani, ott legördülő listát készítettem. Ezekhez a listákhoz a lekérdezések miatt tartoznak php programok is. Ezek a programok lekérdező típusúak, azonban válaszként már a legördülő lista html szövegét adják vissza, így azokat csak be kell írni a html fájlokba. Ezt a folyamatot jQuery parancsok segítségével oldottam meg. A megmaradt szövegdobozokra pedig készült egy ellenőrző program, ami akkor ad vissza üzenetet, ha a szöveg már szerepel az adatbázisban. Ezen feladatok elvégzése után lezárult a kódolás fázis és tovább léptem a tesztelésre. Az elkészült alkalmazás forráskódja megtalálható a „Matrait/HivatkozasSzamonkeresTamogatas” github repository-ban.

### 3.3.4 Tesztelés

A kódolás befejezése után kezdődik a tesztelés fázis. Ehhez a projekthez csak rendszer szintű tesztelést lefedő tesztesetek készültek. Ezek is főleg a három fő funkciót tesztelték és azoknak az együttműködését. Ezeket a teszteseteket az egyes számú melléklet tartalmazza. A tesztelés során apró hibákat találtam, amelyeket azonnal kijavítottam. A javítások után ismét

lefuttattam a megírt teszteseteket, ezen futtatás alatt nem találtam új hibát a tesztkörnyezetben. A tesztkörnyezet ezekben az esetekben megegyeztek a fejlesztői környezettel, azaz ugyanazon a devserveren futott az adatbázis és a php applikációk, mint a fejlesztés során. Integrációs teszt nem történt a dolgozat írásának időpontjáig, azonban ez tervben van az implementálás folyamat lefutása után.

A projekt a dolgozat írásának időpontjában az implementálás fázis előtt áll, a következőkben az implementálásnak a tervezett menetét fogom bemutatni. Az implementálás első lépéseként létre kell hozni egy php alapú biztonságos webszervert. Ezen a szerveren fog futni mind a kliens oldali applikáció, mind a backend programok és az adatbázis. Azért kell php alapúnak lennie, mert ezen a nyelven készültek el a backend programok, valamint az adatbázis zökkenőmentes migrálása érdekében ez lenne a legjobb megoldás. A biztonságos megnevezés alatt azt értem, hogy követnie kell a HTTPS szabványt, mivel, ha nem ilyen szerveren van az add-in, akkor a Microsoft Word nem engedélyezi hozzáadni az add-in-ek listájához.

A szerver felállítása után migrálni kell az alkalmazásokat. El kell őket helyezni a webszerveren és át kell írni a $.ajax metódusokban a php fájlok elérését relatív elérésre, továbbá meg kell vizsgálni, hogy módosítani kell-e a header beállítását a php programoknak. Az adatbázist is migrálni kell, ezt az elkészített SQL kód segítségével könnyen meg lehet tenni, viszont, ha a teszt adatokat is szeretnénk átvinni az új szerverre, akkor azokra külön SQL parancsot kell írni. Az applikáció működőképes a régi adatok nélkül. Az új adatbázis felállítása után át kell írni a connect.php programban az adatbázis elérését, valamint meg kell oldani, hogy a nyitó ablakon található szövegdobozból eljusson a jelszó ebbe a kódba. Ezután tesztelni kell, hogy jól működik-e a szerver, és hogy meg van-e a kommunikáció a kliens oldal és a backend között. Ezt egy böngészőben lehet tesztelni a Home.html oldal megnyitásával, ahova, ha beírjuk a jelszót, és az applikáció tovább lép a HomeAPW.html oldalra, akkor megvan a kapcsolat a kliens és a backend között, és az adatbázis kapcsolat is fennáll.

A migrációs folyamat után módosítani kell a manifesztó fájlt. Erről a fájlról eddig nem esett szó bővebben, mivel eddig a pillanatig ez a fájl automatikusan módosult a fejlesztő környezet által, azonban most kézi módosítást igényel. Ezt az XML fájlt olvassa be a Microsoft Word program, és ennek a segítségével tudja meg az alkalmazás nevét, megjelenését, szükséges engedélyeit. Itt lehet megadni a támogató weboldal elérését, de a legfontosabb, hogy itt kell megadni az alkalmazást futtató webszerver elérési útvonalát.

A manifesztó fájl módosítása után elérhetővé kell tenni a Microsoft Wordnek a manifesztó fájlt. Erre több megoldás is létezik, ebben a szituációban kettő közül kell választani. Az első opció a rendszergazdák segítségével történő megosztás. Olyan vállalati környezetben,

mint az egyetem, lehetőség van a Microsoft 365 applikációkért felelős rendszergazdának megadott csoportok Microsoft Office alkalmazásaira ilyen típusú add-in alkalmazást kiosztani. Ebben az esetben a rendszergazdának kell beállítani, hogy a tárgy oktatói elérjék ezt az alkalmazást. Ez a megoldás akkor ideális, ha sok ilyen tanár van, és nekik van egy jól definiált Microsoft csoportja, amiben csak ők vannak benne. Ehhez a megoldáshoz továbbá szükséges a rendszergazda segítsége, hiszen csak ő tudja ezt a funkciót használni. Ha ezek a tényezők nem állnak fenn vagy nem pont így állnak fenn, akkor a második opciót, a megosztott mappa módszert kell használni. Ennek a módszernek a hátránya, hogy sok munkát igényel. Minden oktató összes gépén külön-külön el kell végezni ezt a beállítási folyamatot. Továbbá kétséges, hogy az iskolai hálózaton kívül hogyan működne ez a megoldás vagy további beavatkozást igényelne.

A megosztott mappa módszer alapja, mint ahogy nevéből is adódik, a hálózaton megosztott mappa. Ebben a mappában kell elhelyezni a manifesztó fájlt, majd minden gépen egyesével megadni a megosztott mappát, mint megbízható katalógust, majd ki kell választani a készített alkalmazást. Ennek a folyamatnak a részletes leírása megtalálható a kettes számú mellékletben. A két opció közül az első lenne az ideálisabb, azzal a módszerrel központilag lehetne kezelni a plugin elérhetőségét, viszont annak a megoldásnak több buktatója van, mint a másodiknak. Mivel mindössze hat oktatónak van szüksége erre az applikációra egy félévben, ezért a külön-külön beállítás sem annyira nagy munka.

Bármelyik megoldással valósul meg az integrálás, ezután implementációs tesztet kell végezni, amely az én terveim szerint a megírt tesztesetek éleskörnyezetben történő futtatását jelenti. Ezen tesztek sikeres lefutása esetén léphetünk tovább a támogatás fázisba.

### 3.3.5 Támogatás és továbbfejlesztés

A támogatás fázis alatt a fő teendők a használat betanítása az oktatók részére, valamint a felmerülő hibák elhárítása. A betanítás részeként készült egy használati útmutató a tanárok részére, ami megtalálható a hármas számú mellékletben. Ez az útmutató képek segítségével mutatja be, hogy hogyan kell használni az alkalmazást. Ezenkívül, ha van rá igény, tarthatunk rövid oktatásokat az oktatók részére. Ha ezekután is merül fel kérdés, abban az esteben emailben fordulhatnak hozzám az oktatók. A felmerült hibák jelentése is emailben történhet meg, amiknek a javítását a webes megoldásnak köszönhetően csak a webszerveren kell elvégezni, ezután nem kell újra megosztani a manifesztó fájlt.

Az alkalmazás jövője nagyban függ a jövőbeni tárgyfelelős gondolataitól és az automatizáláshoz való hozzáállásától. Az alkalmazás jelenlegi állapotában a minimum

követelményeket szolgálja ki, egy remek alap lenne a továbbfejlesztéshez, ahhoz azonban, hogy a továbbfejlesztés sikeres legyen, több közreműködésre van szükség a tárgyfelelős és a tantárgy oktatói részéről. A jelenlegi applikáció azért csak az alap követelményeket szolgálja ki, mert a fejlesztési folyamat negyedik hónapjában változott a tárgyfelelős. Nem tudtam ilyen rövid időn belül annyi változtatást behozni, amivel olyan alkalmazás született volna, ami megfelelt volna az új igényeknek.

Amennyiben továbbfejlesztésre kerül a sor, a projekt sikeressége érdekében agilis életciklus modellt kell alkalmazni. Ahhoz, hogy rövid idő alatt sikeres továbbfejlesztés tudjon történni, rendszeres kommunikációnak kell lennie a fejlesztő személy és a tantárgyfelelős között. Én a vizsgált agilis modellek közül az extreme programozást javaslom. Ez a modell nem annyira kötött, mint a Scrum (nincsenek napi meetingek sem sprintek), de ennek ellenére megköveteli minden egyes iteráció után a visszajelzést az megrendelőtől.

## 3.4 Konklúzió

A projekt kezdetekor megállapítottam, hogy a vizsgált dolgozatnak feleletválasztós tesztként történő megíratása nem tudná a gyakorlati tudást úgy ellenőrizni, mint a word dokumentum alapú dolgozat. Tehát a javítás automatizálására valamilyen Word dokumentum kezelő applikációt kell fejleszteni. A lehetséges megoldások közül a makrókat rögtön kizártam a biztonsági kockázatuk miatt, a VSTO plugint pedig az alacsonyabb támogatás és terjesztés nehézségei miatt zártam ki. A választásom tehát az Office web add-in fejlesztésére esett, a jó támogatás és a központi webes szerkezete miatt. A Proof of Concept alkalmazás sikeres fejlesztése után bebizonyosodott, hogy a Word add-in technológia képes az általunk kitűzött cél megvalósítására. Az elkészült alkalmazással a méréseim szerint egy perc alatt ki lehet javítani egy dolgozatot, ami az eredeti becsült 5-10 perces értéknél szignifikánsan jobb, akkor is, ha a dolgozat összeállításának a megnövekedett idejét is számoljuk. Ez egy szignifikáns gyorsulás, tehát az applikáció elérte a célját.

Az applikáció a dolgozat írása alatt nem került használatba a fejlesztés közepén történt tárgyfelelős váltás miatt, ami a követelmények változását okozta. Az új tárgyfelelőstől, Láng Blankától azt az információt kaptam, hogy szívesen használná a programot a jövőben, ha megtörténnek a szükséges továbbfejlesztések. Ezen változtatások implementálására a dolgozat leadásának idejéig nem került sor, az új igények felmerülése és a határidő közötti rövid idő miatt.

Az új igények fejlesztése során, a projekt során szerzett tapasztalataim alapján nem elegendő a vízesésmodell alkalmazása. A továbbfejlesztés során valamilyen agilis életciklus

modellt kell alkalmazni, melyek közül én az extreme programozást javaslom, mivel így a tárgyfelelős mindig egy működő verzión tesztelheti, hogy az elkészült program megfelel-e minden elvárásának.

# Összegzés

A dolgozat a Szakszeminárium I. tárgy hivatkozások számonkérésének automatizálásával foglalkozott, és feltette a kérdést, hogy ez az automatizálás képese olyan hozzáadott értéket teremteni a tárgy oktatói számára, ami szignifikánsan megkönnyíti ennek a dolgozatnak a javítását. Ennek a kérdésnek a megválaszolására készült egy saját fejlesztésű támogató applikáció, amely képes ezen és egyéb Microsoft Word alapú dolgozatok javítására. Az applikáció követelményeit a tantárgy akkori tárgyfelelőse közreműködésével állítottam össze.

Bemutatásra került ennek a dolgozatnak felváltási lehetőségei és egyéb lehetséges megoldások az ezen dolgozat írásának kezdetének időpontjában használt dolgozat javításának automatizálására. A felsorolt lehetséges megoldások közül kiválasztásra került a Word Web Add-In megoldás. A választás fő oka ennek a rendszernek a támogatottsága és az én korábbi ismereteim voltak. Ezután bemutatásra került a fejlesztés projektje, a vízesésmodell fázisait követve. A követelmények fázis során felsoroltam és részleteztem a projekt elején megállapított követelményeket, majd a tervezés fázis során kitértem ezen követelmények teljesítéséhez szükségek elemek tervezésére. Az implementáció szakaszban bemutattam a program forráskódjának fő metódusait és azok működését. A tesztelés fázis során ismertettem a program tesztelésének folyamatát és beszéltem az üzembehelyezés lehetséges megoldásainak előnyeiről és hátrányairól. Az utolsó fázisban bemutattam a program tervezett támogatását és az általam elképzet továbbfejlesztési lehetőségeket.

A projekt fázisainak ismertetése után összegeztem a fejlesztés konklúzióját és vizsgáltam az elért eredményt. Eszerint ennek a dolgozat írásának kezdetén használatban lévő hivatkozások dolgozat javításánál, méréseim szerint, körülbelül hétszer gyorsabb javítást lehet elérni. Ez az eredmény szignifikáns javulásnak tekinthető, tehát a dolgozat elején feltett kérdésre igen a válasz, azaz az automatizálás képes olyan hozzáadott értéket teremteni, amely szignifikánsan megkönnyíti az oktatók munkáját.

# Irodalomjegyzék


Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J. (2002). Agile software development methods: Review and analysis, *VTT publication 478, Espoo, Finland*.

Balmori, J., (2015) „Word JavaScript API”. YouTube, 2015. 11. 18. https://www.youtube.com/watch?v=EaLVfaj1mW8 Letöltés dátuma: 2021. 10. 16.

Chandra, V. (2015). Comparison between various software development methodologies. *International Journal of Computer Applications*, *131*(9), 7-10. Chi, D. (2000).

Microsoft office 2000 and security against macro viruses. *Symantec Antivirus Research Center, https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/microsoft-2000-security-against-macro-99-en. pdf, Tech. Rep*.

Dima, A. M., & Maassen, M. A. (2018). From Waterfall to Agile software: Development models in the IT sector, 2006 to 2018. Impacts on company management. *Journal of International Studies,* 11(2), 315-326.

Gutfleisch, M., Peiffer, M., Erk, S., & Sasse, M. A. (2021, October). Microsoft Office Macro Warnings: A Design Comedy of Errors with Tragic Security Consequences. In *European Symposium on Usable Security 2021* (pp. 9-22).

industry4.hu (2021) http://industry4.hu/hu/fogalomtar/api-application-programming-interfaces letöltés dátuma: 2021. 10. 16.

McConnell, S. (2004). *Code complete*. Pearson Education.

Microsoft Documentation (2021. 10. 16.) https://docs.microsoft.com/en-us/

Moodle Documentation (2021. 07. 14.) https://docs.moodle.org/

Murillo, G. G., Novoa-Hernández, P., & Rodríguez, R. S. (2021). Technology Acceptance Model and Moodle: A systematic mapping study. *Information Development*, *37*(4), 617-632.

php.net (2021. 10. 17.) https://www.php.net/manual/en/index.php

Pirjan, A. (2015). SOLUTIONS FOR REPURPOSING THE DEFAULT ACTIONS AND STATES OF THE OFFICE CONTROLS THROUGH COMPONENT OBJECT MODEL ADD-INS. *Journal of Information Systems & Operations Management*, *9*(1).

Ranjan, A., Sinha, A., & Battewad, R. (2020). *JavaScript for Modern Web Development: Building a Web Application Using HTML, CSS, and JavaScript.* BPB Publications.

Raymond, B. (2017. 05. 23.): *Comparing VSTO and Office Web add-ins,* TechGenix. http://techgenix.com/comparing-vsto-and-office-web-add-ins-video/

Sempf, B., - Jausovec, P. (2010): VSTO For Dummies. Hoboken, Wiley Publishing

Stober, T., & Hansmann, U. (2010). *Agile Software Development: Best Practices for Large Software Development Projects*. Berlin, Springer Verlag.

Szabó Bálint, Ribényi Máté (2018): Az agilis módszertanok megítélése a beosztottak és vezetők szemszögéből. Vezetéstudomány - Budapest Management Review, 49 (6). pp. 22-32.

W3.org (2017), HTML 5.2 is now a W3C Recommendation https://www.w3.org/blog/news/archives/6696 letöltés dátuma: 2021.10.17.

W3.org (2021), https://www.w3.org/standards/webdesign/htmlcss letöltés dátuma: 2021. 10. 17.

# Mellékletek

## 1. számú melléklet: Tesztelés során futtatott tesztesetek

A teszt célja a dokumentum előkészítés funkciótesztje több dokumentum esetén

| | Teszt lépések | Várt eredmény | Eredmény | 2. futtatás |
|---|---|---|---|---|
| 1 | Indítsd el a programot és ellenőrizd a szervereket(php, db). | Progarm elindul szerverek futnak | pass | pass |
| 2 | Írd be az adatbázishoz tartozó jelszót és kattints a belépés gombra | Beléptünk a szerverbe, elindul a home page. | pass | pass |
| 3 | Kattints az "Új feladat készítés" gombra | Megjelenik a dokElokeszitó.html oldal | pass | pass |
| 4 | Adj egy már létező nevet a documentum nevének. | "Létezik már ilyen dokumentum" üzenet a text box mellett. | pass | pass |
| 5 | Adj meg még nem létező nevet | Üzenet eltünt. | pass | pass |
| 6 | Rögzíts 7 hivatkozást. | Rögzített hivatkozások, szürke háttérrel és bouding boxban vannak. Eddig tárolt hivatkozások száma: 7 | pass | pass |
| 7 | Töröljí ki kettő teszőleges hivatkozást | Törlöt hivatkozások formázása megszünik, számláló 5-öt mutat | pass | pass |
| 8 | Adj hozzá 5 új hivatkozást. | Formázást megtörtént számláló 10-et mutat. | pass | pass |
| 9 | Készíts képeznyő képet a tárolt hivatkozásokról. | Képernyő kép kész | pass | pass |
| 10 | Rögzísd a tárolt hivatkozásokat a gombbal. | Hivatkozások rögzítve üzenet. | pass | pass |
| 11 | Nézd át az adatbázist. | 10 új elem hozzáadva, a helyes szövegekkel, törölt hivatkozás nincs közte. | fail | pass |
| 12 | Mensd el a dokumentumot. | Documentum elmentve | | pass |

Dokumentum előkészítés funkció

| | | | | |
|---|---|---|---|---|
| | A teszt célja a hibák elhelyezése funkció ellenőrzése | Az előtt a teszt előtt el kell végezni a Dokumentum előkészítés tesztet. | | |
| | Teszt lépések | Várt eredmény | Eredmény | |
| 1 | Indítsd el a programot és ellenőrizd a szervereket(php, db). | Progarm elindul szerverek futnak | pass | |
| 2 | Nyisd meg a dokumentum előkészítése test során készített dokumentumot. | Dokumentum megnyílt formázások még mindig szerepelnek. | pass | |
| 3 | Írd be az adatbázishoz tartozó jelszót és kattints a belépés gombra | Beléptünk a szerverbe, elindul a home page | pass | |
| 4 | Kattints az "Új feladat készítés" gombra | Megjelenik a dokElokeszito.html oldal | pass | |
| 5 | Felül kattints a hibák elhelyezése gombra | Megjelenik a hibamentes.html oldal, 3s később megjelenik a legördülő lista a már létező dokumentumok neveivel. | pass | |
| 6 | Válazd ki a megnyitott dokumentum nevét a listából | Név kiválasztva. | pass | |
| 7 | Adj meg egy feladatsor azonosítót | Feladatsor azonosító megadva. | pass | |
| 8 | Válassz ki egy megjelőlt hivatkozást, ronts el benne valamit és rögzísd a hiba helyének rögzítése gombbal. | Kiválasztott hivatkozás helytelen és háttér szine sárga lett. Eddig tárolt hibák számláló 1-et mutat. | pass | pass |
| 9 | Ismételd meg a nyolcas lépést 4 ször. | 5 hivatkozás sárga háttérszinü és helytelen. Eddig tárolt hibák számláló 5-öt mutat. | pass | pass |

Hibák elhelyezése funkció

| | | A teszt elvégzése előtt végig kell csinálni a hibák elhelyezése tesztet. | |
|---|---|---|---|
| | A teszt célja a javítás funkció tesztje | | |
| | Teszt lépések | Várt eredmény | Eredmény |
| 1 | Nyisd meg a hibák elvézése során született kész fájlt. | dok megnyitva | pass |
| 2 | A képernyő képek segítségével, javíts ki 3 elhelyezett hivát, és ronts el 1 olyan hivatkozást ami meg van jelölve de nem volt mentve hibásként. | 3 hiba kijavítva, 1 hivatkozás elrontva. | pass |
| 3 | Mensd el a dokumentumot másként. | Dokumentum elmentve. | pass |
| 4 | Indítsd el a programot és ellenőrizd a szervereket(php, db). | Progarm elindul szerverek futnak | pass |
| 5 | Írd be az adatbázishoz tartozó jelszót és kattints a belépés gombra | Beléptünk a szerverbe, elindul a home page. | pass |
| 6 | Kattints a Javítás gombra | Megjelenik a javítás oldal, 3s után megjelenik a dokumentum neveket tartalmazó legördülő lista | pass |
| 7 | Válaszd ki a dokumentum nevét. | Dokumentum név kiválasztva, 3 s múlva megjelenik a feladatsor azonosító legördülő lista, annyi eleme, van ahány feladat sort készítettünk. | pass |
| 8 | Válaszd ki a feladatsor azonosítót. | Feladatsor azonosító kiválasztva. | pass |
| 9 | Kattints a Javítás gombra | Lezajlott a javítás | pass |

Javítás funkció

# 2. számú melléklet: Add-in telepítési útmutató

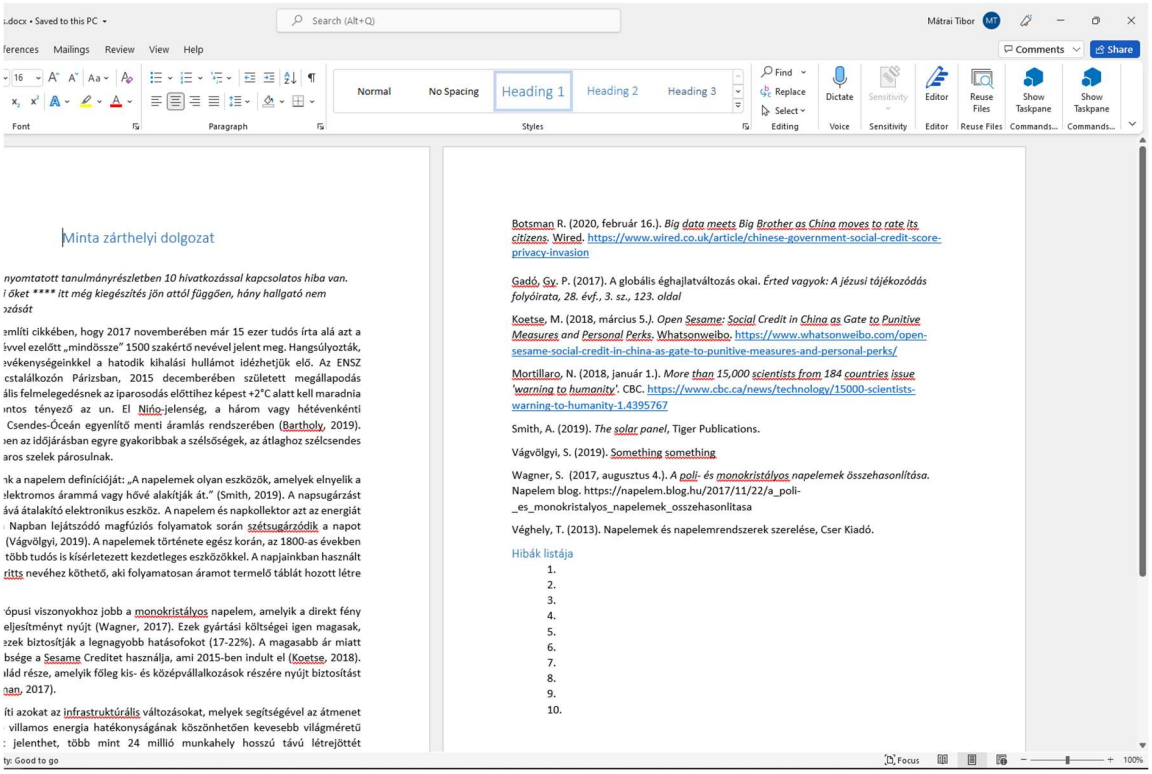
Az útmutató megtalálható a https://docs.microsoft.com/en-us/office/dev/add-ins/testing/create-a-network-shared-folder-catalog-for-task-pane-and-content-add-ins linken.

## 3. Számú melléklet: Használati útmutató

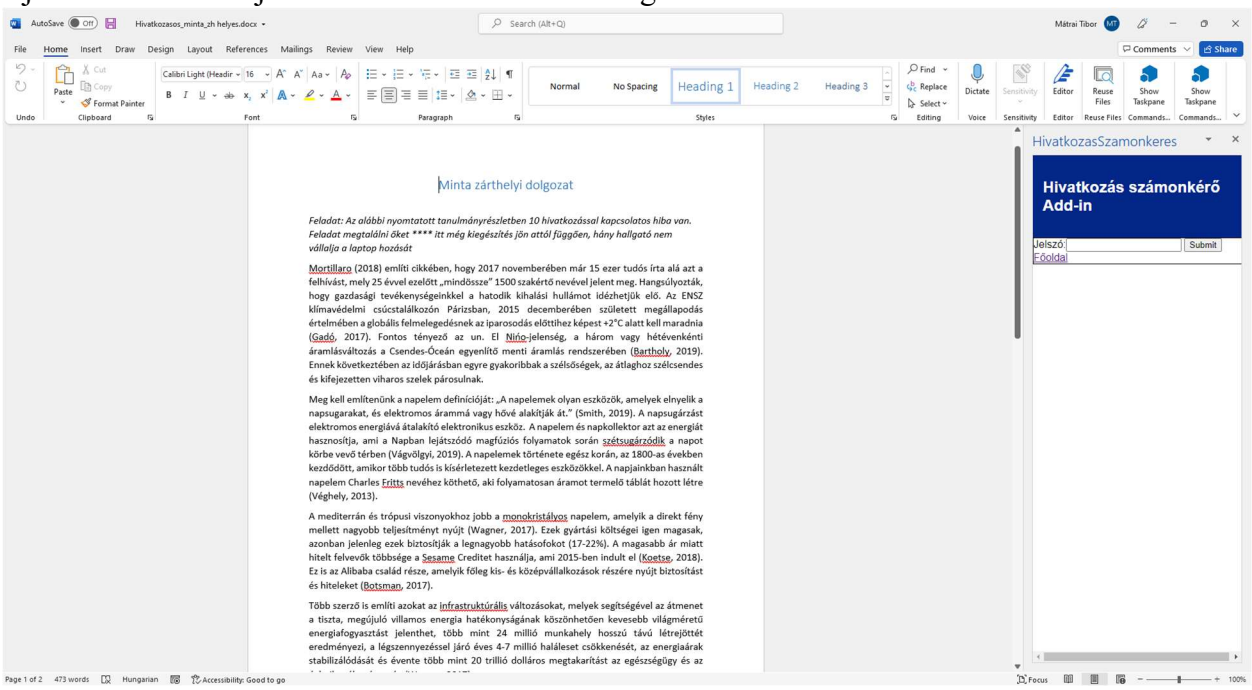
# HivatkozásokZH javító alkalmazás Használatiútmutató

## Az applikáció használata

1. Nyissa meg a ZH alapját képező helyes hivatkozásokat tartalmazó dokumentumot.
2. Kattintson a jobb felső sarokban található Ikonra (Show Taskpane) (önnek csak egynek kell lennie).

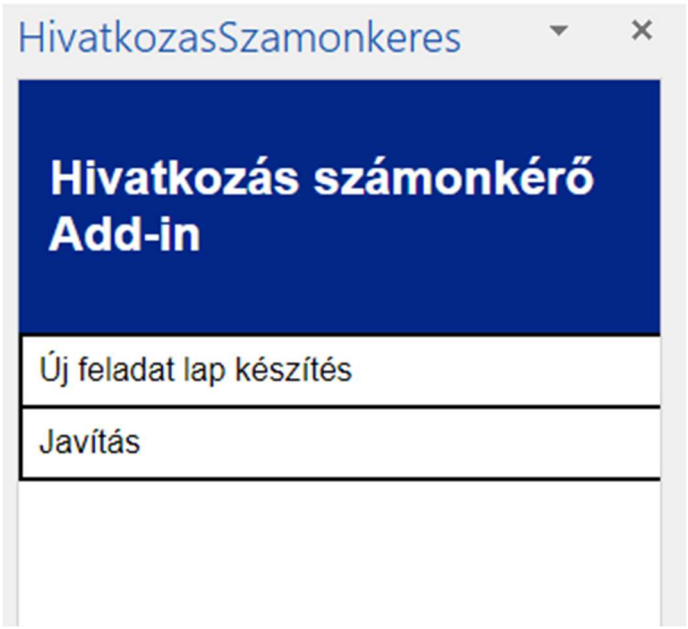


3. Írja be az adatbázis jelszót és kattintson a Submit gombra.



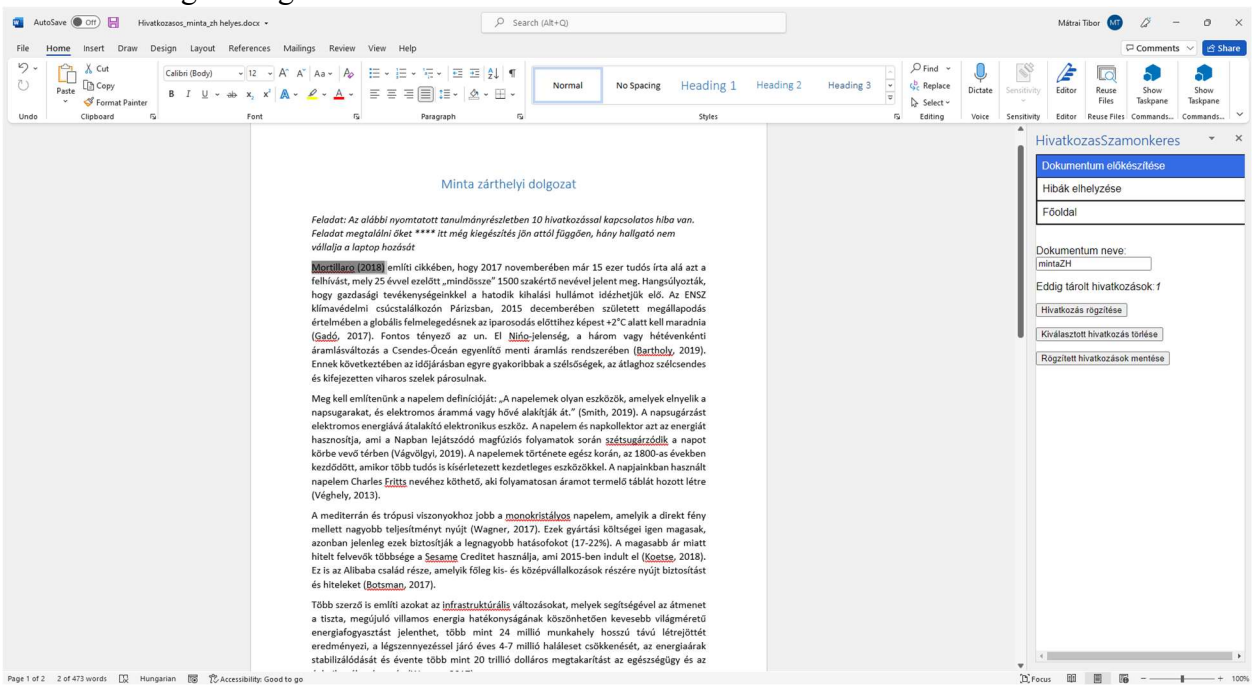

# Dokumentum előkészítés

1. Új dokumentum készítéséhez a jelszó oldal után válasza az Új feladat lap készítése opciót.

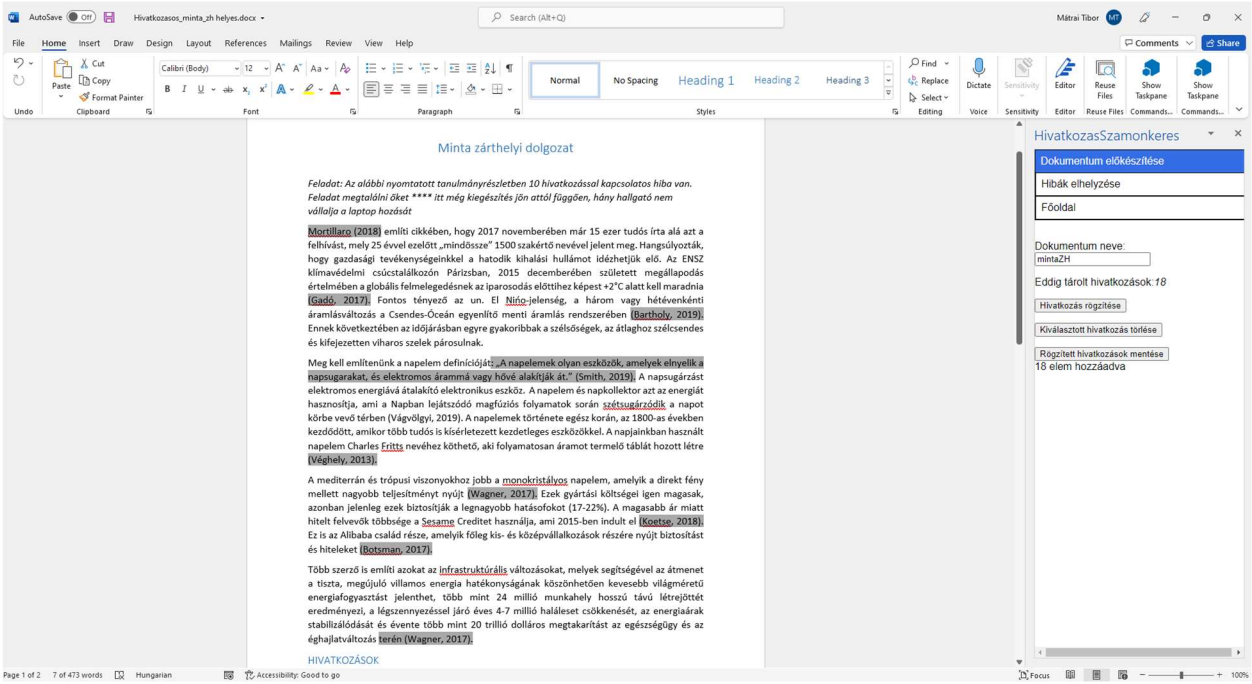



2. Adj meg a dokumentumnak egy egyedi még nem használt nevet.

3. Jelöljön ki egy olyan pozíciót, ahova hibát szeretne rakni, majd kattintson a Hivatkozás rögzítése gombra.



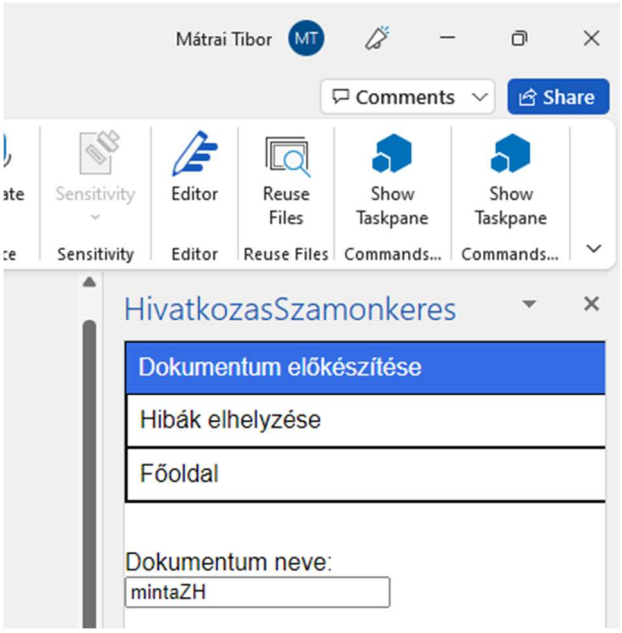

4. Jelölje ki az összes olyan pozíciót, ahol a jövőben hiba lesz. Ha esetleg kijelölne olyat, amit még sem szeretne, kattintson bele abba a hivatkozásba és a Kiválasztott hivatkozás törlése gombbal törölhetted.
5. Ha végzett a jelölésekkel kattintson a Rögzített hivatkozások mentése gombra. Ezután már nem tud hivatkozást törölni, az adatbázisban megmarad.



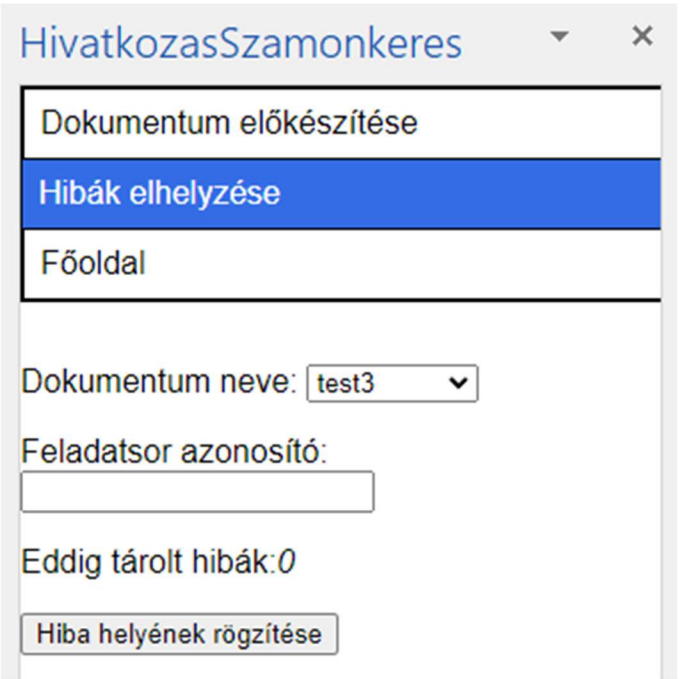

6. Ezután mentse el a dokumentumot, hogy máskor már csak a hibákat kelljen elhelyezni benne.

## Hibák elhelyezése

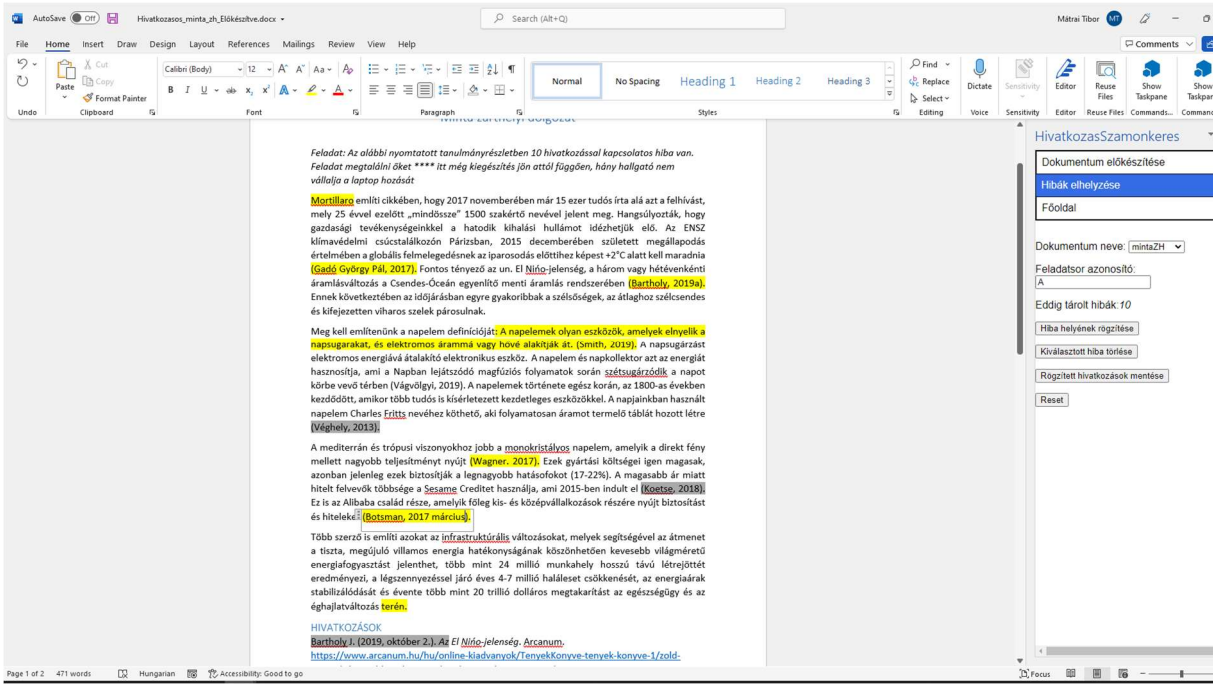
1. A Hibák elhelyezése oldalt a fő oldalról az Új feladat lap készítés gombon keresztül éri el.




2. Nyissa meg az előkészített dokumentumot, válasza ki a dokumentum nevek közül azt, ami meg van nyitva.



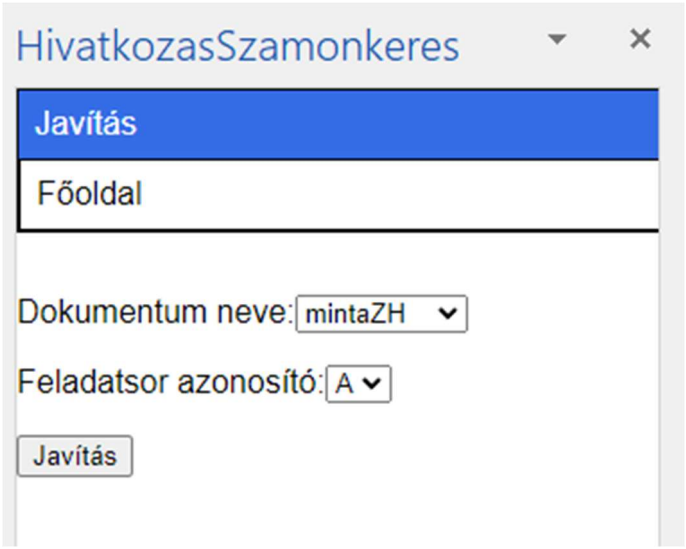

3. Adja neki egy feladatsor azonosítót.
4. Válasszon ki egy megjelölt hivatkozást, rontsa el, majd kattintson a Hiba helyének rögzítése gombra




5. Helyezzen 10 hibát. Itt is lehet törölni, de a szöveget manuálisan kell visszacsinálni.
6. Ha készen van, Kattintson a Rögzített hibák mentése gombra. Ha eltűntek a kijelölések akkor mentse el a dokumentumot, ez már mehet a hallgatóknak.
7. A Reset gomb segítségével visszaállíthatja a dokumentumot az előkészítés utáni állapotba.

# Javítás

1. A Javítás oldal a főoldalról érhető el.
2. Válasza ki a dokumentum nevét majd a feladatsor azonosítót (kb 3 másodperc míg meg jelennek a dropdownlistek)




3. Kattintson a Javítás gombra.

4. A javítás eredménye:

a. Zöld kiemelés: Hallgató által helyesen javította a feladatot
b. Zöld kiemelés és utána sárga kiemeléssel eredeti szöveg: Hallgató nem pontosan/rosszul/nem javította.
c. Piros kiemelés után sárga kiemeléssel eredeti szöveg: A hallgató olyat „javított” ami nem volt feladat